ACM模板

by - 繁凡さん

2020/10/8

https://fanfansann.blog.csdn.net/

声 明

本文中的所有的模板代码，虽然基本上都经过了少量测试，但仍可能会出现一些不可预知的 bug，在赛场上如果因为我的代码存在的一些 bug 产生的一切后续影响，我概不负责>_< ，所以打印、抄写代码需谨慎，尽量自己理解算法含义，以我的代码为参考，自行完成代码实现！！

繁凡

2021 年 11 月 13 日

目录

```cpp
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<iostream>
#include<string>
#include<vector>
#include<stack>
#include<cstdlib>
#include<cmath>
#include<set>
#include<list>
#include<deque>
#include<map>
#include<queue>
#include<bitset>
#include<limits.h>
//#pragma GCC optimize (2)
//#pragma G++ optimize (2)//手动开O2
#define ls (p<<1)//线段树左儿子
#define rs (p<<1|1)//线段树右儿子
#define mid (l+r>>1)//线段树mid
#define over(i,s,t) for(register int i=s;i≤t;++i)//for循环
#define lver(i,t,s) for(register int i=t;i≥s;--i)
//#define int __int128//水大数
using namespace std;
#undef mid
typedef long long ll;
typedef unsigned long long ull;//卡精度
typedef pair<int,int> PII;

const int N = 1e7+7;
const ll mod = 1e9+7;
const ll INF = 1e15+7;
const double EPS = 1e-10;//可近似为趋近为 0 可以根据定义求导时使用
const int base = 131;//base = 13331 // hash

template<typename T>inline T read(T &x)
{
    x=0;ll f=1;char c;
    while(!isdigit(c=getchar()))if(c=='-')f=-1;
    while(isdigit(c)){x=(x<<1)+(x<<3)+(c^48);c=getchar();}
    return x*f;
}

inline void write(int x)
{
    if(x<10)
    {
        putchar(x+48);
        return;
    }
    write(x/10),write(x%10);
}


```

一、C++头文件汇总

```cpp
#include <algorithm>      //STL 通用算法
#include <bitset>         //STL 位集容器
#include <cctype>         //字符处理
#include <cerrno>         //定义错误码
#include <cfloat>         //浮点数处理
#include <ciso646>          //对应各种运算符的宏
#include <climits>        //定义各种数据类型最值的常量
#include <clocale>        //定义本地化函数
#include <cmath>          //定义数学函数
#include <complex>        //复数类
#include <csignal>          //信号机制支持
#include <csetjmp>          //异常处理支持
#include <cstdarg>          //不定参数列表支持
#include <cstddef>          //常用常量
#include <cstdio>         //定义输入/输出函数
#include <cstdlib>        //定义杂项函数及内存分配函数
#include <cstring>        //字符串处理
#include <ctime>          //定义关于时间的函数
#include <cwchar>         //宽字符处理及输入/输出
#include <cwctype>        //宽字符分类
#include <deque>          //STL 双端队列容器
#include <exception>      //异常处理类
#include <fstream>        //文件输入/输出
#include <functional>     //STL 定义运算函数（代替运算符）
#include <limits>         //定义各种数据类型最值常量
#include <list>           //STL 线性列表容器
#include <locale>         //本地化特定信息
#include <map>            //STL 映射容器
#include <memory>         //STL通过分配器进行的内存分配
#include<new>             //动态内存分配
#include <numeric>          //STL常用的数字操作
#include <iomanip>        //参数化输入/输出
#include <ios>            //基本输入/输出支持
#include <iosfwd>         //输入/输出系统使用的前置声明
#include <iostream>       //数据流输入/输出
#include <istream>        //基本输入流
#include <iterator>         //STL迭代器
#include <ostream>        //基本输出流
#include <queue>          //STL 队列容器
#include <set>            //STL 集合容器
#include <sstream>        //基于字符串的流
#include <stack>          //STL 堆栈容器
#include <stdexcept>      //标准异常类
#include <streambuf>      //底层输入/输出支持
#include <string>         //字符串类
#include <typeinfo>         //运行期间类型信息
#include <utility>        //STL 通用模板类
#include <valarray>         //对包含值的数组的操作
#include <vector>         //STL 动态数组容器
```

```
字符测试是否字母和数字 isalnum
是否字母 isalpha
是否控制字符 iscntrl
是否数字 isdigit
是否可显示字符(除空格外) isgraph
是否可显示字符(包括空格) isprint
是否既不是空格，又不是字母和数字的可显示字符 ispunct
是否空格 isspace
是否大写字母 isupper
是否16进制数字(0—9，A-F)字符 isxdigit
字符大小写转换函数 转换为大写字母 toupper
转换为小写字母 tolower

```

```
头文件 math.h
数学函数：本分类给出了各种数学计算函数，
必须提醒的是ANSIC标准中的数据格式并不符合IEEE754标准，
一些C语言编译器却遵循IEEE754(例如frinklin C51)
————————————————————————————————

反余弦 acos
反正弦 asin
反正切 atan
反正切2 atan2
余弦 cos
正弦 sin
正切 tan

双曲余弦 cosh
双曲正弦 sinh
双曲正切 tanh

指数函数 exp
指数分解函数 frexp
乘积指数函数 fdexp
自然对数 log
以10为底的对数 log10
浮点数分解函数 modf

幂函数 pow
平方根函数 sqrt

求下限接近整数 ceil
绝对值 fabs
求上限接近整数 floor
求余数 fmod
```

```
头文件 string.h
字符串处理：本分类的函数用于对字符串进行合并、比较等操作
主要是C遗传过来的字符串函数
————————————————————————————————

字符串拷贝 块拷贝(目的和源存储区不可重叠) memcpy
块拷贝(目的和源存储区可重叠) memmove
串拷贝 strcpy
按长度的串拷贝 strncpy
字符串连接函数 串连接 strcat
按长度连接字符串 strncat
串比较函数 块比较 memcmp
字符串比较 strcmp
字符串比较(用于非英文字符) strcoll
```

```
按长度对字符串比较 strncmp
字符串转换 strxfrm
字符与字符串查找 字符查找 memchr
字符查找 strchr
字符串查找 strcspn
字符串查找 strpbrk
字符串查找 strspn
字符串查找 strstr
字符串分解 strtok
杂类函数 字符串设置 memset
错误字符串映射 strerror
求字符串长度 strlen

```

```
头文件 stdlib.h
实用工具函数：本分类给出了一些函数无法按以上分类，但又是编程所必须要的。
————————————————————————————————————
字符串转换函数
字符串转换为整数 atoi
字符串转换为长整数 atol
字符串转换为浮点数 strtod
字符串转换为长整数 strtol
字符串转换为无符号长整型 strtoul

伪随机序列产生函数
产生随机数 rand
设置随机函数的起动数值 srand

存储管理函数
分配存储器 calloc
释放存储器 free
存储器分配 malloc
重新分配存储器 realloc
```

```
序号 库类别 头文件
————————————————————————————————————
1    错误处理       errno.h
2    字符处理       ctyphe.
3    地区化          local.h
4    数学函数       math.h
5    信号处理       signal.h
6    输入输出        stdio.h
7    实用工具程序    stdlib.h
8    字符串处理      string.h

```

二、动态规划

$\oint$ 1. 背包

1.【01背包】

```cpp
#include<algorithm>
#include<cstdio>
int T,n,i,j,v[110],w[110],f[1010];
int main(){
    scanf("%d%d",&T,&n);
    for(i=1;i≤n;i++)scanf("%d%d",&v[i],&w[i]);
    for(i=1;i≤n;i++)
        for(j=T;j≥v[i];j--)
            f[j]=std::max(f[j],f[j-v[i]]+w[i]);
    printf("%d",f[T]);
}
```

2.【完全背包】

```cpp
#include<bits/stdc++.h>
using namespace std;
int T,n,ans,a[105],dp[30005];
int main(){
    scanf("%d",&T);
    while(T--){
        memset(dp,-127,sizeof(dp));
        scanf("%d",&n);dp[0]=ans=0;
        for(int i=1;i≤n;i++)scanf("%d",&a[i]);
        for(int i=1;i≤n;i++)
            for(int j=a[i];j≤25000;j++)
                dp[j]=max(dp[j],dp[j-a[i]]+1);
        for(int i=1;i≤n;++i)ans+=dp[a[i]]==1;
        printf("%d\n",ans);
    }
}
```

3.【多人背包】

> 他们一共有 K 个人，每个人都会背一个包。这些包 的容量是相同的，都是 V。可以装进背包里的一共有 N 种物品，每种物品都有
> 给定的体积和价值。 在 DD 看来，合理的背包安排方案是这样的： 每个人背包里装的物品的总体积恰等于包的容量。
> 每个包里的每种物品最多只有一件，但两个不同的包中可以存在相同的物品。 任意两个人，他们包里的物品清单不能完全相同。
> 在满足以上要求的前提下，所有包里的所有物品的总价值最大是多少呢?

```cpp
#include<algorithm>
#include<cstring>
#include<cstdio>
int ans,T,n,i,j,k,K,a,b,t,re[45],v[210],w[210],f[5010][45];
int main(){
    scanf("%d%d%d",&K,&T,&n);
    memset(f,-127,sizeof(f));f[0][1]=0;
    for(i=1;i≤n;i++)scanf("%d%d",&v[i],&w[i]);
    for(i=1;i≤n;i++)
        for(j=T;j≥v[i];j--){
            a=b=1,t=0;
            while(t≤K)
                if(f[j][a]≥f[j-v[i]][b]+w[i])re[++t]=f[j][a++];
                else re[++t]=f[j-v[i]][b++]+w[i];
            for(k=1;k≤K;k++)f[j][k]=re[k];
        }
    for(i=1;i≤K;i++)ans+=f[T][i];
    printf("%d",ans);
}
```

4.【分组背包】

```cpp
#include<algorithm>
```

```cpp
#include<cstdio>
using namespace std;
int T,n,i,j,a[65],s,t,r1,r2,r[65][3],v[65],w[65],f[32010];
int main(){
    scanf("%d%d",&T,&n);
    v[0]=0xffffffff;
    for(i=1;i≤n;i++){
        scanf("%d%d%d",&v[i],&s,&a[i]);
        if(a[i])r[a[i]][++r[a[i]][0]]=i;w[i]=v[i]*s;
    }
    for(i=1;i≤n;i++)
        for(j=T;j≥v[i]&&(!a[i]);j--){
            t=j-v[i],r1=r[i][1],r2=r[i][2];
            f[j]=max(f[j],f[t]+w[i]);
            if(v[r1]≤t)f[j]=max(f[j],f[t-v[r1]]+w[i]+w[r1]);
            if(v[r2]≤t)f[j]=max(f[j],f[t-v[r2]]+w[i]+w[r2]);
            if(v[r1]+v[r2]≤t)f[j]=max(f[j],f[t-v[r1]-v[r2]]+w[i]+w[r1]+w[r2]);
        }
    printf("%d",f[T]);
}
```

5.【多重背包】

```cpp
#include<algorithm>
#include<cstring>
#include<cstdio>
#define Re register int
using namespace std;
const int N=103,M=4e4+3;
int n,m,x,y,z,V,ans,v[N*17],w[N*17],dp[M];
inline void in(Re &x){
    Re fu=0;x=0;char ch=getchar();
    while(ch<'0'||ch>'9')fu|=ch=='-',ch=getchar();
    while(ch≥'0'&&ch≤'9')x=(x<<1)+(x<<3)+(ch^48),ch=getchar();
    x=fu?-x:x;
}
int main(){
    in(m),in(V);
    memset(dp,-127,sizeof(dp));
    dp[0]=0;
    while(m--){
        in(x),in(y),in(z);Re p=1;
        while(z≥p)v[++n]=y*p,w[n]=x*p,z-=p,p<<=1;
        if(z)v[++n]=y*z,w[n]=x*z;
    }
    for(Re i=1;i≤n;++i)
        for(Re j=V;j≥v[i];--j)
            dp[j]=max(dp[j],dp[j-v[i]]+w[i]);
    for(Re i=0;i≤V;++i)ans=max(ans,dp[i]);
    printf("%d\n",ans);
}
```

6.【混合背包】

```cpp
#include<algorithm>
#include<cstdio>
using namespace std;
int x,a,b,c,d,T,n,i,j,t,f[1010],v[100010],w[100010];
int main(){
    scanf("%d:%d%d:%d%d",&a,&b,&c,&d,&n);
    T=c*60+d-a*60-b;
```

```
    for(i=1;i≤n;i++){
        scanf("%d%d%d",&a,&b,&c);
        if(!c){
            for(j=0;j+a≤T;j++)
            f[j+a]=max(f[j+a],f[j]+b);
            continue;
        }
        x=1;
        while(c≥x){v[++t]=x*a,w[t]=x*b,c-=x,x<<=1;}
        v[++t]=c*a,w[t]=c*b;
    }
    for(i=1;i≤t;i++)
        for(j=T;j≥v[i];j--)
            f[j]=max(f[j],f[j-v[i]]+w[i]);
    printf("%d",f[T]);
}
```

7.【二维费用】

```
#include<algorithm>
#include<cstdio>
using namespace std;
int tmp,T1,T2,x,y,n,i,j,k,v1[105],v2[105],w[105],dp[105][105],ans[105][105];
int main(){
    scanf("%d",&n);
    for(i=1;i≤n;i++)scanf("%d%d%d",&v1[i],&v2[i],&w[i]);
    scanf("%d%d",&T1,&T2);
    for(i=1;i≤n;++i)
        for(j=T1;j≥v1[i];--j)
            for(k=T2;k≥v2[i];--k)
                if(dp[j][k]<(tmp=dp[j-v1[i]][k-v2[i]]+1)){
                    dp[j][k]=tmp;
                    ans[j][k]=ans[j-v1[i]][k-v2[i]]+w[i];
                }
                else if(dp[j][k]==tmp)ans[j][k]=min(ans[j][k],ans[j-v1[i]][k-v2[i]]+w[i]);
    printf("%d",ans[T1][T2]);
}
```

∮ 2. 树形DP

(1).【简单树形 DP】

```
#include<algorithm>
#include<cstring>
#include<cstdio>
#define R register int
using namespace std;
int head[6010],a,b,t,i,j,n,f[6010][2],pan[6010];
struct QAQ{int next,to;}Q[6010];
inline void add(int x,int y){Q[++t].to=y,Q[t].next=head[x],head[x]=t;}
inline void dfs(int w){
    for(R k=head[w];k;k=Q[k].next){
        int to=Q[k].to;
        dfs(to);
        f[w][0]+=max(f[to][0],f[to][1]);
        f[w][1]+=f[to][0];
    }
}
int main(){
```

```
    scanf("%d",&n);
    for(i=1;i≤n;i++)scanf("%d",&f[i][1]);
    for(i=1;i<n;i++)scanf("%d%d",&a,&b),pan[a]++,add(b,a);
    for(i=1;i≤n;i++)
        if(!pan[i]){
            dfs(i);
            printf("%d\n",max(f[i][0],f[i][1]));
            return 0;
        }
}
```

(2).【换根 DP】

> 每个奶牛居住在 N 个农场中的一个，这些农场由 N−1 条道路连接，并且从任意一个农场都能够到达另外一个农场。道路 i 连接农场
> A_i和B_i，长度为 L_i。集会可以在 N 个农场中的任意一个举行。另外，每个牛棚中居住着 C_i只奶牛。
> 在选择集会的地点的时候，Bessie 希望最大化方便的程度（也就是最小化不方便程度）。比如选择第 X
> 个农场作为集会地点，它的不方便程度是其它牛棚中每只奶牛去参加集会所走的路程之和（比如，农场 i 到达农场 X 的距离是 20，那么总路程就是
> $C_i \times 20$。帮助 Bessie 找出最方便的地点来举行大集会。

```
#include<algorithm>
#include<cstring>
#include<cstdio>
#define LL long long
#define Re register int
using namespace std;
const LL N=1e5+3,inf=1e18;
int n,m,x,y,z,o,A[N],S[N],dis[N],size[N],head[N];LL ans=inf,f[N],g[N];
struct QAQ{int w,to,next;}a[N<<1];
inline void add(Re x,Re y,Re z){a[++o].w=z,a[o].to=y,a[o].next=head[x],head[x]=o;}
inline void in(Re &x){
    int f=0;x=0;char c=getchar();
    while(c<'0'||c>'9')f|=c=='-',c=getchar();
    while(c≥'0'&&c≤'9')x=(x<<1)+(x<<3)+(c^48),c=getchar();
    x=f?-x:x;
}
inline void dfs1(Re x,Re fa){
    size[x]=1,S[x]=A[x];
    for(Re i=head[x],to;i;i=a[i].next)if((to=a[i].to)≠fa)
        dis[to]=a[i].w,dfs1(to,x),S[x]+=S[to],size[x]+=size[to],f[x]+=f[to]+(LL)S[to]*a[i].w;
}
inline void dfs2(Re x,Re fa){
    if(fa)g[x]=g[fa]+f[fa]-f[x]-(LL)S[x]*dis[x]+(LL)(S[1]-S[x])*dis[x];
    for(Re i=head[x],to;i;i=a[i].next)if((to=a[i].to)≠fa)dfs2(to,x);
    ans=min(ans,f[x]+g[x]);
}
int main(){
//    freopen("123.txt","r",stdin);
    in(n),m=n-1;
    for(Re i=1;i≤n;++i)in(A[i]);
    while(m--)in(x),in(y),in(z),add(x,y,z),add(y,x,z);
    dfs1(1,0),dfs2(1,0),printf("%lld\n",ans);
}
```

∮ 3. 数位DP

> 不含前导零且相邻两个数字之差至少为 2 的正整数被称为 windy 数。windy 想知道，在 a 和 b 之间，包括 a 和 b ，总共有多少个 windy 数?

1.【dfs】

```cpp
#include<cstring>
#include<cstdio>
#include<cmath>
#define R register int
using namespace std;
int a,b,l,num[12],dp[12][10];
inline int dfs(int len,int up,int ok){
    R i,ed,ans=0;
    if(len<1)return 1;
    if(!ok&&~dp[len][up]&&up≠-7)return dp[len][up];
    ed=ok?num[len]:9;
    for(i=0;i≤ed;i++)if(abs(i-up)≥2)
        ans+=dfs(len-1,(!i&&up==-7)?up:i,ok&&(i==ed));
    if(!ok&&up≠-7)dp[len][up]=ans;
    return ans;
}
inline int sovle(int x){
    l=0;
    while(x)num[++l]=x%10,x/=10;
    return dfs(l,-7,1);
}
int main(){
    scanf("%d%d",&a,&b);
    memset(dp,-1,sizeof(dp));
    printf("%d",sovle(b)-sovle(a-1));
}
```

2.【dp】

```cpp
#include<cstdio>
#include<cmath>
#define R register int
using namespace std;
int i,j,k,a,b,l,ans,num[12],dp[12][10];
inline int dpp(int len){
    ans=0;
    for(i=1;i<len;i++)
        for(j=1;j<10;j++)
            ans+=dp[i][j];
    for(j=1;j<num[len];j++)ans+=dp[len][j];
    for(i=len-1;i>0;i--){
        for(j=0;j<num[i];j++)
            if(abs(num[i+1]-j)≥2)ans+=dp[i][j];
        if(abs(num[i]-num[i+1])<2)break;
    }
    if(!i)ans++;
    return ans;
}
inline int sovle(int x){
    l=0;
    while(x)num[++l]=x%10,x/=10;
    return dpp(l);
}
inline int sakura(){
    scanf("%d%d",&a,&b);
    for(i=0;i<10;i++)dp[1][i]=1;
    for(i=2;i<11;i++)
        for(j=0;j<10;j++)
            for(k=0;k<10;k++)
```

```
                if(abs(j-k)≥2)
                    dp[i][j]+=dp[i-1][k];
    printf("%d",sovle(b)-sovle(a-1));
}
int QAQWQ=sakura();
int main(){}
```

∮ 4. DP 的优化

1.【二分】
最长公共子序列

```
#include<bits/stdc++.h>
using namespace std;
int pan[100010],b[100010],f[100010],n,i,a,len=1,l,r,mid;
inline int in(){
    int x=0,fh=1;char c=getchar();
    while(c<'0'||c>'9')c=getchar();
    while(c≥'0'&&c≤'9')x=x*10+c-'0',c=getchar();
    return x*fh;
}
int main(){
    n=in();
    for(i=1;i≤n;i++)a=in(),pan[a]=i,f[i]=0xfffffff;
    for(i=1;i≤n;i++)a=in(),b[i]=pan[a];
    f[1]=b[1];
    for(i=2;i≤n;i++){
        l=0;r=len;
        if(b[i]>f[len])f[++len]=b[i];
        else{
            while(l<r){
                mid=(l+r)/2;
                if(f[mid]<b[i])l=mid+1;
                else r=mid;
            }
            f[l]=min(b[i],f[l]);
        }
    }
    printf("%d",len);
}
```

2.【二进制优化多重背包】

```
#include<algorithm>
#include<cstdio>
#define Re register int
using namespace std;
const int N=103,M=4e4+3,logN=17;
int m,n,x,y,z,V,ans,v[N*logN],w[N*logN],dp[M];
inline void in(Re &x){
    int f=0;x=0;char c=getchar();
    while(c<'0'||c>'9')f|=c=='-',c=getchar();
    while(c≥'0'&&c≤'9')x=(x<<1)+(x<<3)+(c^48),c=getchar();
    x=f?-x:x;
}
int main(){
    in(m),in(V);
    for(Re i=1;i≤m;++i){
        in(x),in(y),in(z);
```

```
        for(Re j=1;j≤z;j<<=1)z-=j,w[++n]=x*j,v[n]=y*j;
        if(z)w[++n]=x*z,v[n]=y*z;
    }
    for(Re i=1;i≤n;++i)
        for(Re j=V;j≥v[i];--j)
            dp[j]=max(dp[j],dp[j-v[i]]+w[i]);
    for(Re j=0;j≤V;++j)ans=max(ans,dp[j]);
    printf("%d\n",ans);
}
```

3.【单调队列优化多重背包】

```
#include<cstdio>
#define Re register int
const int N=7003,M=7003;
int n,h,t,V,mp,tmp,v[N],w[N],c[N],Q[N],K[N],dp[M];
inline void in(Re &x){
    Re fu=0;x=0;char ch=getchar();
    while(ch<'0'||ch>'9')fu|=ch=='-',ch=getchar();
    while(ch≥'0'&&ch≤'9')x=(x<<1)+(x<<3)+(ch^48),ch=getchar();
    x=fu?-x:x;
}
inline int max(Re a,Re b){return a>b?a:b;}
inline int min(Re a,Re b){return a<b?a:b;}
int main(){
    in(n),in(V);
    for(Re i=1;i≤n;++i)in(v[i]),in(w[i]),in(c[i]);
    for(Re i=1;i≤n;++i)
        for(Re r=0;r<v[i];++r){
            h=1,t=0,mp=(V-r)/v[i];
            for(Re p=0;p≤mp;++p){
                tmp=dp[p*v[i]+r]-w[i]*p;
                while(h≤t&&Q[t]≤tmp)--t;
                Q[++t]=tmp,K[t]=p;
                while(h≤t&&p-K[h]>min(c[i],V/v[i]))++h;
                dp[p*v[i]+r]=max(dp[p*v[i]+r],Q[h]+p*w[i]);
            }
        }
    printf("%d",dp[V]);
}
```

4.【矩阵加速递推】

【学习笔记】动态规划—矩阵递推加速

5.【四边形不等式优化】

【模板】

(1).【分治】

```
#include<algorithm>
#include<cstdio>
#include<cmath>
#define Re register int
using namespace std;
const int N=5e5+3;
int i,j,n,h,t,a[N],Q[N],Poi[N];
double tmp,p[N],sqr[N];
inline void in(Re &x){
    int f=0;x=0;char c=getchar();
    while(c<'0'||c>'9')f|=c=='-',c=getchar();
    while(c≥'0'&&c≤'9')x=(x<<1)+(x<<3)+(c^48),c=getchar();
    x=f?-x:x;
}
```

```cpp
inline void sakura(Re l,Re r,Re L,Re R){
    if(l>r)return;
    Re mid=l+r>>1,j0;double mx=0;
    for(Re j=L;j≤mid&&j≤R;++j)
//暴力找p[i]的最优决策点j0，而其决策点j必须满足j≤i，即此处的j≤mid
        if((tmp=a[j]+sqr[mid-j])>mx)mx=tmp,j0=j;
    p[mid]=max(p[mid],mx);
    sakura(l,mid-1,L,j0),sakura(mid+1,r,j0,R);
}
int main(){
    in(n);
    for(Re i=1;i≤n;++i)in(a[i]),sqr[i]=sqrt(i);
    sakura(1,n,1,n);
    for(Re i=1;i<n-i+1;++i)swap(a[i],a[n-i+1]),swap(p[i],p[n-i+1]);
    sakura(1,n,1,n);
    for(Re i=n;i;--i)printf("%d\n",(int)ceil(p[i])-a[i]);
}
```

(2).【二分栈】

```cpp
#include<algorithm>
#include<cstdio>
#include<cmath>
#define Re register int
using namespace std;
const int N=5e5+3;
int i,j,n,h,t,a[N],Q[N],Poi[N];
double p[N],sqr[N];
inline void in(Re &x){
    int f=0;x=0;char c=getchar();
    while(c<'0'||c>'9')f|=c=='-',c=getchar();
    while(c≥'0'&&c≤'9')x=(x<<1)+(x<<3)+(c^48),c=getchar();
    x=f?-x:x;
}
inline double Y(Re i,Re j){return a[j]+sqr[i-j];}
inline int find_Poi(int j1,int j2){//找到两个直线的交点i
    Re l=j2,r=n,mid,ans=n+1;//为了处理两个直线没有交点的情况，用一个变量记录答案
    while(l≤r){
        mid=l+r>>1;
        if(Y(mid,j1)≤Y(mid,j2))ans=mid,r=mid-1;
//当前这个位置i使得直线j1的纵坐标小于直线j2的纵坐标，说明这个点i在交点的右方，所以右边界要缩小
        else l=mid+1;
    }
    return ans;
}
inline void sakura(){
    h=1,t=0;
    for(i=1;i≤n;++i){//由于i本身也是一个决策点，所以要先入队再取答案择优
        while(h<t&&Poi[t-1]≥find_Poi(Q[t],i))--t;//如果出现了上述可踢的情况，出队
        Poi[t]=find_Poi(Q[t],i),Q[++t]=i;
        while(h<t&&Poi[h]≤i)++h;
//找到第一个位置j使得直线Q[j]与直线Q[j+1]的交点大于i，
//那么直线Q[j]就是i前面在最上面的直线，即答案，自己画个图模拟一下就懂了
        p[i]=max(p[i],Y(i,Q[h]));
    }
}
int main(){
    in(n);
    for(Re i=1;i≤n;++i)in(a[i]),sqr[i]=sqrt(i);
    sakura();
```

```
    for(Re i=1;i<n-i+1;++i)swap(a[i],a[n-i+1]),swap(p[i],p[n-i+1]);
    sakura();
    for(Re i=n;i;--i)printf("%d\n",(int)ceil(p[i])-a[i]);
}
```

6.【斜率优化】

(1).【单调队列（x 与 k 均单调）】

```
#include<cstring>
#include<cstdio>
#define LL long long
#define Re register LL
const int N=5e4+5;
LL i,j,n,L,h=1,t=0,Q[N],S[N],dp[N];
//S[n]=∑C[i]+1, dp[i]=min(dp[j]+(S[i]-(S[j]+L+1))^2), ++L
//dp[i]=S[i]^2-2*S[i]*L+dp[j]+(S[j]+L)^2-2S[i]*S[j]
//(2*S[i]) * S[j] + (dp[i]-S[i]^2+2S[i]L)=(dp[j]+(S[j]+L)^2)
//   k     *  x   +           b          =        y
inline LL min(Re a,Re b){return a<b?a:b;}
inline LL X(Re j){return S[j];}
inline LL Y(Re j){return dp[j]+(S[j]+L)*(S[j]+L);}
inline long double slope(Re i,Re j){return (long double)(Y(j)-Y(i))/(X(j)-X(i));}//记得开long double
int main(){
    scanf("%lld%lld",&n,&L);++L;
    for(i=1;i≤n;S[i]+=S[i-1]+1,++i)scanf("%lld",&S[i]);
    Q[++t]=0;//重中之重
    for(i=1;i≤n;++i){
        while(h<t&&slope(Q[h],Q[h+1])≤2*S[i])++h;//至少要有两个元素 h<t。出队判断时尽量加上等号
        dp[i]=dp[j=Q[h]]+(S[i]-S[j]-L)*(S[i]-S[j]-L);
        while(h<t&&slope(Q[t-1],Q[t])≥slope(Q[t-1],i))--t;//至少要有两个元素 h<t。入队判断时尽量加上等号
        Q[++t]=i;
    }
    printf("%lld",dp[n]);
}
```

(2).【二分+单调队列（x 单调 k 不单调）】

```
#include<cstring>
#include<cstdio>
#define LL long long
#define Re register LL
const int N=3e5+5;
LL i,j,n,h=1,t=0,S,Q[N],ST[N],SF[N],dp[N];
//dp[p][i]=min(dp[p-1][j]+(ST[i]+S*p)*(SF[i]-SF[j]));
//dp[i]=dp[j]+ST[i]*(SF[i]-SF[j])+S*(SF[n]-SF[j]);
//(S+ST[i]) * SF[j] + (dp[i]-ST[i]*SF[i]-S*SF[i]) = (dp[j])
//    k     *   x   +              b              = y
//ti可小于0，所以ST[i]非递增，只可二分
//fi可等于0，所以SF[i](X)非严格递增，因此需要特判X(i)==X(j)的情况
inline LL min(Re a,Re b){return a<b?a:b;}
inline LL X(Re j){return SF[j];}
inline LL Y(Re j){return dp[j];}
inline long double slope(Re i,Re j){return X(j)==X(i)?(Y(j)≥Y(i)?1e18:-1e18):(long double)(Y(j)-
Y(i))/(X(j)-X(i));}
//由于需要二分查找，多了一些限制：队列里不能有在同一位置的点,返回inf还是-inf都影响着是否删除重点，平时不可不管，二分必须注意返回值
inline LL sakura(Re k){
    if(h==t)return Q[h];
    Re l=h,r=t;
    while(l<r){
```

```
        Re mid=l+r>>1,i=Q[mid],j=Q[mid+1];
        if(slope(i,j)<k)l=mid+1;
//        if( (Y(j) - Y(i)) < k * (X(j) - X(i)) )l=mid+1;//注意是(j)-(i)因为Q[mid+1]>Q[mid]s即j>i即SF[j]>SF[i]即X(j)>X(i)，如果是(i)-(j)的话乘过去要变号
        else r=mid;
    }
    return Q[l];
}
int main(){
    scanf("%lld%lld",&n,&S);
    for(i=1;i≤n;ST[i]+=ST[i-1],SF[i]+=SF[i-1],++i)scanf("%lld%lld",&ST[i],&SF[i]);
    Q[++t]=0;
    for(i=1;i≤n;++i){
        j=sakura(S+ST[i]);
        dp[i]=dp[j]+ST[i]*(SF[i]-SF[j])+S*(SF[n]-SF[j]);
        while(h<t&&slope(Q[t-1],Q[t])≥slope(Q[t-1],i))--t;//此处取等号作用出现，如果不取等会WA
        Q[++t]=i;
    }
    printf("%lld",dp[n]);
}
```

(3).【CDQ（x 与 k 均不单调）】

```
#include<algorithm>
#include<cstring>
#include<cstdio>
#define LD long double
#define LL long long
#define Re register int
#define S2(a) (1ll*(a)*(a))
using namespace std;
const LL N=1e5+3,inf=1e18;
int n,H[N],W[N],Q[N];LL S[N],dp[N];
inline void in(Re &x){
    int f=0;x=0;char c=getchar();
    while(c<'0'||c>'9')f|=c=='-',c=getchar();
    while(c≥'0'&&c≤'9')x=(x<<1)+(x<<3)+(c^48),c=getchar();
    x=f?-x:x;
}
//dp[i]=min(dp[i],dp[j]+(H[i]-H[j])*(H[i]-H[j])+S[i-1]-S[j]);
//dp[i]=dp[j]-2*H[i]*H[j]+H[j]*H[j]+H[i]*H[i]+S[i-1]-S[j]
//(2*H[i]) * H[j] + (dp[i]-S[i-1]-H[i]*H[i]) = (dp[j]+H[j]*H[j]-S[j])
//   k     *  x   +            b             =          y
#define X(j) (a[j].x)
#define Y(j) (a[j].y)
struct QAQ{
    int k,x,id;LL y;
    inline bool operator<(const QAQ &O)const{return x≠O.x?x<O.x:y<O.y;}
}a[N],b[N];
inline bool cmp(QAQ A,QAQ B){return A.k<B.k;}
inline LD slope(Re i,Re j){return X(i)==X(j)?(Y(j)>Y(i)?inf:-inf):(LD)(Y(j)-Y(i))/(X(j)-X(i));}
inline void CDQ(Re L,Re R){
    if(L==R){Re j=a[L].id;a[L].y=dp[j]+(LL)H[j]*H[j]-S[j];return;}
    Re mid=L+R>>1,p1=L,p2=mid+1,h=1,t=0;
    for(Re i=L;i≤R;++i)a[i].id≤mid?b[p1++]=a[i]:b[p2++]=a[i];
    for(Re i=L;i≤R;++i)a[i]=b[i];
    CDQ(L,mid);
    for(Re i=L;i≤mid;++i){
        while(h<t&&slope(Q[t-1],Q[t])≥slope(Q[t-1],i))--t;
        Q[++t]=i;
```

```
    }
    for(Re i=mid+1,j,id;i≤R;++i){
        while(h<t&&slope(Q[h],Q[h+1])≤a[i].k)++h;
        if(h≤t)id=a[i].id,j=Q[h],dp[id]=min(dp[id],a[j].y-(LL)a[i].k*a[j].x+S[id-1]+(LL)H[id]*H[id]);
    }
    CDQ(mid+1,R);
    Re w=L-1;p1=L,p2=mid+1;
    while(p1≤mid&&p2≤R)b[++w]=a[p1]<a[p2]?a[p1++]:a[p2++];
    while(p1≤mid)b[++w]=a[p1++];while(p2≤R)b[++w]=a[p2++];
    for(Re i=L;i≤R;++i)a[i]=b[i];
}
int main(){
//    freopen("123.txt","r",stdin);
    in(n);
    for(Re i=1;i≤n;++i)in(H[i]);
    for(Re i=1;i≤n;++i)in(W[i]);
    for(Re i=1;i≤n;++i)S[i]=S[i-1]+W[i],dp[i]=inf;
    for(Re i=1;i≤n;++i)a[i].k=(H[i]<<1),a[i].x=H[i],a[i].id=i;
    sort(a+1,a+n+1,cmp);
    dp[1]=0,CDQ(1,n);
    printf("%lld\n",dp[n]);
}
```

(4).【Splay 维护动态凸包（x 与 k 均不单调）】

$\oint$ 5. 插头 DP

1.【多回路插头 DP】

> 给出n*m的方格，有些格子不能铺线，其它格子必须铺，可以形成多个闭合回路。问有多少种铺法？ 从第2行到第n+1行，每行m个数字（1 or
> 0），1表铺线，0表不铺线

```
#include<algorithm>
#include<cstring>
#include<cstdio>
#define LL long long
#define Re register int
using namespace std;
const int N=15,M=8192+3;
int n,m,o,V,T,A[N][N];LL dp[2][M];
inline void in(Re &x){
    int f=0;x=0;char c=getchar();
    while(c<'0'||c>'9')f|=c=='-',c=getchar();
    while(c≥'0'&&c≤'9')x=(x<<1)+(x<<3)+(c^48),c=getchar();
    x=f?-x:x;
}
int main(){
//    freopen("123.txt","r",stdin);
    in(T);
    while(T--){
        in(n),in(m),V=(1<<m+1)-1;
        for(Re i=1;i≤n;++i)
            for(Re j=1;j≤m;++j)
                in(A[i][j]);
        for(Re j=0;j≤V;++j)dp[0][j]=dp[1][j]=0;
        dp[o][0]=1;
        for(Re i=1;i≤n;++i){
            o^=1;
```

```
            for(Re s=0;s≤V;++s)dp[o][s]=(s&1)?0:dp[o^1][s>>1];//把线拉下来
            for(Re j=1;j≤m;++j){
                o^=1;
                for(Re s=0;s≤V;++s)dp[o][s]=0;
                for(Re s=0;s≤V;++s)if(dp[o^1][s]){
                    Re L=(s>>j-1)&1,U=(s>>j+1-1)&1;
                    if(A[i][j]){//可以铺线
                        dp[o][s^(1<<j-1)^(1<<j+1-1)]+=dp[o^1][s];//横和竖和左上拐和右下拐
                        if(L≠U)dp[o][s]+=dp[o^1][s];//左下拐和右上拐
                    }
                    else{//不可以铺线
                        if(!L&&!U)dp[o][s]+=dp[o^1][s];//无插头才可以转移
                    }
                }
            }
        }
        printf("%lld\n",dp[o][0]);
    }
}
```

2.【单回路插头 DP】

(1).【 括号表示法】

```
#include<algorithm>
#include<cstring>
#include<cstdio>
#define LL long long
#define Re register int
using namespace std;
const int N=14,M=1594323+3;
int n,m,o,V,edx,edy,Mi[N],A[N][N];LL ans,dp[2][M];char s[N];
inline void in(Re &x){
    int f=0;x=0;char c=getchar();
    while(c<'0'||c>'9')f|=c=='-',c=getchar();
    while(c≥'0'&&c≤'9')x=(x<<1)+(x<<3)+(c^48),c=getchar();
    x=f?-x:x;
}
int main(){
//    freopen("123.txt","r",stdin);
    in(n),in(m),Mi[0]=1;
    for(Re i=1;i≤m+1;++i)Mi[i]=Mi[i-1]*3;V=Mi[m+1]-1;
    for(Re i=1;i≤n;++i){
        scanf("%s",s+1);
        for(Re j=1;j≤m;++j)if(!(A[i][j]=(s[j]=='*')))edx=i,edy=j;
    }
//    printf("edx=%d, edy=%d\n",edx,edy);
    dp[o][0]=1;
    for(Re i=1;i≤n;++i){
        o^=1;
        for(Re s=0;s≤V;++s)dp[o][s]=s%3?0:dp[o^1][s/3];
        for(Re j=1;j≤m;++j){
            o^=1;
            for(Re s=0;s≤V;++s)dp[o][s]=0;
            for(Re s=0;s≤V;++s)if(dp[o^1][s]){
                Re L=s/Mi[j-1]%3,U=s/Mi[j+1-1]%3;
                if(A[i][j]){if(!L&&!U)dp[o][s]+=dp[o^1][s];}
                else if(!L&&!U)dp[o][s+Mi[j-1]+2*Mi[j+1-1]]+=dp[o^1][s];
                else if(!L&&U)dp[o][s]+=dp[o^1][s],dp[o][s+U*Mi[j-1]-U*Mi[j+1-1]]+=dp[o^1][s];
                else if(L&&!U)dp[o][s]+=dp[o^1][s],dp[o][s-L*Mi[j-1]+L*Mi[j+1-1]]+=dp[o^1][s];
                else if(L==1&&U==1){
```

```
                    Re cnt=1;
                    for(Re k=j+2;k≤m;++k){
                        if(s/Mi[k-1]%3==1)++cnt;
                        else if(s/Mi[k-1]%3==2)--cnt;
                        if(!cnt){dp[o][s-Mi[j-1]-Mi[j+1-1]-Mi[k-1]]+=dp[o^1][s];break;}//2变1
                    }
                }
                else if(L==2&&U==2){
                    Re cnt=1;
                    for(Re k=j-1;k≥1;--k){
                        if(s/Mi[k-1]%3==1)--cnt;
                        else if(s/Mi[k-1]%3==2)++cnt;
                        if(!cnt){dp[o][s-2*Mi[j-1]-2*Mi[j+1-1]+Mi[k-1]]+=dp[o^1][s];break;}//1变2
                    }
                }
                else if(L==2&&U==1)dp[o][s-2*Mi[j-1]-Mi[j+1-1]]+=dp[o^1][s];
                else if(L==1&&U==2)if(i==edx&&j==edy&&!(s-Mi[j-1]-2*Mi[j+1-1]))ans+=dp[o^1][s];
            }
        }
    }
    printf("%lld\n",ans);
}
```

(2).【括号表示法+优化状态枚举】
【模板】 插头 dp

```
#include<algorithm>
#include<cstring>
#include<cstdio>
#define LL long long
#define Re register int
using namespace std;
const int N=14,M=1594323+3;
int n,m,o,V,edx,edy,Mi[N],A[N][N];LL ans,dp[2][M];char s[N];
inline void in(Re &x){
    int f=0;x=0;char c=getchar();
    while(c<'0'||c>'9')f|=c=='-',c=getchar();
    while(c≥'0'&&c≤'9')x=(x<<1)+(x<<3)+(c^48),c=getchar();
    x=f?-x:x;
}
int st[2][M];
inline void add(Re s,LL v){
    if(!dp[o][s])st[o][++st[o][0]]=s;
    dp[o][s]+=v;
}
inline void CL(){
    for(Re i=1;i≤st[o][0];++i)dp[o][st[o][i]]=0;
    st[o][0]=0;
}
int main(){
//    freopen("123.txt","r",stdin);
    in(n),in(m),Mi[0]=1;
    for(Re i=1;i≤m+1;++i)Mi[i]=Mi[i-1]*3;V=Mi[m+1]-1;
    for(Re i=1;i≤n;++i){
        scanf("%s",s+1);
        for(Re j=1;j≤m;++j)if(!(A[i][j]=(s[j]=='*')))edx=i,edy=j;
    }
//    printf("edx=%d, edy=%d\n",edx,edy);
    dp[o][st[o][++st[o][0]]=0]=1;
    for(Re i=1;i≤n;++i){
```

```
        o^=1,CL();
        for(Re I=1;I≤st[o^1][0];++I)if(st[o^1][I]*3≤V&&dp[o^1][st[o^1][I]])
            dp[o][st[o][++st[o][0]]=st[o^1][I]*3]=dp[o^1][st[o^1][I]];
        for(Re j=1;j≤m;++j){
            o^=1,CL();
            for(Re I=1;I≤st[o^1][0];++I)if(dp[o^1][st[o^1][I]]){
                Re s=st[o^1][I];LL v=dp[o^1][s];
                Re L=s/Mi[j-1]%3,U=s/Mi[j+1-1]%3;
                if(A[i][j]){if(!L&&!U)add(s,v);}
                else if(!L&&!U)add(s+Mi[j-1]+2*Mi[j+1-1],v);
                else if(!L&&U)add(s,v),add(s+U*Mi[j-1]-U*Mi[j+1-1],v);
                else if(L&&!U)add(s,v),add(s-L*Mi[j-1]+L*Mi[j+1-1],v);
                else if(L==1&&U==1){
                    Re cnt=1;
                    for(Re k=j+2;k≤m;++k){
                        if(s/Mi[k-1]%3==1)++cnt;
                        else if(s/Mi[k-1]%3==2)--cnt;
                        if(!cnt){add(s-Mi[j-1]-Mi[j+1-1]-Mi[k-1],v);break;}//2变1
                    }
                }
                else if(L==2&&U==2){
                    Re cnt=1;
                    for(Re k=j-1;k≥1;--k){
                        if(s/Mi[k-1]%3==1)--cnt;
                        else if(s/Mi[k-1]%3==2)++cnt;
                        if(!cnt){add(s-2*Mi[j-1]-2*Mi[j+1-1]+Mi[k-1],v);break;}//1变2
                    }
                }
                else if(L==2&&U==1)add(s-2*Mi[j-1]-Mi[j+1-1],v);
                else if(L==1&&U==2)if(i==edx&&j==edy&&!(s-Mi[j-1]-2*Mi[j+1-1]))ans+=v;
            }
        }
    }
    printf("%lld\n",ans);
}
```

∮ 6. 最长上升子序列与最长公共子序列

6.1 最长上升子序列（LIS）

给出一个长度为n的数字序列ai，我们需要找到一个最长的子序列， 使得子序列中的数字大小单调递增。 例如：$n=5$，原序列为$(1,4,2,3,5)$，那么我们选择$(1,2,3,5)$是最优的。

6.1.1 树状数组优化$O(nlogn)$

最长上升子序列是不能等于的，后面要大于前面，所以要正序，并且查询的时候要-1来保证只会小于不会等于

最长不上升子序列是可以等于的，其实就是求小于等于，所以后面的要小于等于前面的，由于树状数组维护的是最大值（大于等于），而这里要的是小于等于，所以要倒序。

要注意这里要用maxn也就是a数组里的最大值来作为树状数组的上限

因为树状数组里存的就是a数组的值，并维护其最大值

```
ll n,m,ans1,ans2,tree[N],a[N],maxn;
inline void update(ll k,ll val)//更新
{
    while(k≤maxn)//要注意这里是maxn也就是a数组里的最大值来作为树状数组的上限
    {
```

```
        tree[k]=max(tree[k],val);//维护最大值
        k+=k&(-k);
    }
}
inline ll query(ll k)
{
    ll res=0;
    while(k)
    {
        res=max(res,tree[k]);//求以小于等于x的数为结尾的最长不上升子序列的长度的最大值
        k-=k&(-k);
    }
    return res;
}
int main()
{
    while(scanf("%lld",&a[++n])≠EOF)maxn=max(maxn,a[n]);
    n--;
    for(int i=n;i≥1;--i)
    {
        ll q=query(a[i])+1;
        /*相当于朴素做法的：for(int j=i+1;j≤n;j++)
                            if(a[j]≤a[i])
        因为是按照顺序从后往前循环，所以当前的i所query到的所有的值都是在i后面的，
        解决了i<j的问题，树状数组维护最大值，就解决了a[j]≤a[i]的问题，
        所以这里求的就是以i为开头的最长不上升子序列的长度*/
        update(a[i],q);//这个最大值+1就是以当前这个数开头的最长不上升子序列的长度，丢到树状数组里面去更新后面的值
        ans1=max(ans1,q);
    }
    memset(tree,0,sizeof tree);
    for(int i=1;i≤n;++i)
    {
        ll q=query(a[i]-1)+1;
        /*查询以小于（没有等于！！！）x的数为结尾的最长上升子序列的长度的最大值
        因为不能等于所以要-1*/
        update(a[i],q);//这个最大值+1就是以当前这个数结尾的最长上升子序列的长度，丢到树状数组里面去
        ans2=max(ans2,q);
    }
    printf("%lld\n%lld\n",ans1,ans2);
    return 0;

}

```

6.2 最长公共子序列（LCS）

给出两个长度为n的序列$\{a_i\}$和$\{b_i\}$，求它们的公共子序列的最长长 度。 例如两个序列分别为$(1,2,4,8,16)$和$(2,4,6,8,10)$，那么它们的LCS就是$(2,4,8)$。

6.2.1 转换成LIS优化$O(nlogn)$

```
using namespace std;
typedef long long ll;//全用ll可能会MLE或者直接WA,试着改成int看会不会A
const ll N=100007;
const ll INF=1e10+9;
const ll mod=2147483647;
const double EPS=1e-10;//-10次方约等于趋近为0
const double Pi=3.1415926535897;
ll n,m,a[N],b[N],tree[N],mp[N];
```

```

inline void update(ll k,ll val)//树状数组维护的是mp数组的值的最大值
{
    while(k≤N)
    {
        tree[k]=max(tree[k],val);
        k+=k&(-k);
    }
}
inline ll query(ll k)//查到的k都是值小于k的节点的最大值，所以就保证了单调性
{
    ll res=0;
    while(k)
    {
        res=max(res,tree[k]);
        k-=k&(-k);
    }
    return res;
}

void advanced()
{
    over(i,1,n)
        mp[b[i]]=i;//mp函数就是一个简单的映射离散化
    over(i,1,n)//如果a里有b里没有那么p[a[i]]就是0，树状数组是没办法处理0的所以就不会任何影响
        update(mp[a[i]],query(mp[a[i]])+1);
    printf("%lld\n",query(n));
}
int main()
{
    scanf("%lld",&n);
    over(i,1,n)
        scanf("%lld",&a[i]);
    over(i,1,n)
        scanf("%lld",&b[i]);
    advanced();
    return 0;
}
```

三、字符串

∮ 1. KMP

```
// s[]是长文本，p[]是模式串，n是s的长度，m是p的长度
/*求模式串的Next数组：*/
for (int i = 2, j = 0; i ≤ m; i ++ )
{
    while (j && p[i] ≠ p[j + 1]) j = ne[j];
    if (p[i] == p[j + 1]) j ++ ;
    ne[i] = j;
}

// 匹配
for (int i = 1, j = 0; i ≤ n; i ++ )
{
    while (j && s[i] ≠ p[j + 1]) j = ne[j];
```

```
    if (s[i] == p[j + 1]) j ++ ;
    if (j == m)
    {
        j = ne[j];
        // 匹配成功后的逻辑
    }
}
```

给出两个字符串 s_1 和 s_2，若 s_1 的区间 $[l,r]$ 子串与 s_2 完全相同，则称 s_2 在 s_1 中出现了，其出现位置为 l。
现在请你求出 s_2 在 s_1 中所有出现的位置。

定义一个字符串 s 的 border 为 s 的一个非 s 本身的子串 t，满足 t 既是 s 的前缀，又是 s 的后缀。
对于 s_2，你还需要求出对于其每个前缀 s' 的最长 border t' 的长度。


```
const int N=1000007;
const int mod=1e9+7;
const ll INF=1e15+7;
const double EPS=1e-10;
const int p=131;//13331

int nex[N];
char a[N],b[N];
int n,m;
int j;
int main()
{
    scanf("%s%s",b+1,a+1);//从1开始输入
    //这里我让A向B匹配，跟书上的顺序一样
    int lena=strlen(a+1);
    int lenb=strlen(b+1);
    nex[1]=0;
    for(int i=2;i≤lena;++i){//A串自己匹配
        while(j>0&&a[i]≠a[j+1])j=nex[j];
        if(a[i]==a[j+1])j++;
        nex[i]=j;
    }
    for(int i=1,j=0;i≤lenb;++i){//A串向B串匹配
        while(j>0&&b[i]≠a[j+1])j=nex[j];
        if(b[i]==a[j+1])j++;
        if(j==lena)printf("%d\n",i-lena+1),j=nex[j];
    }
    over(i,1,lena){
        printf("%d%s",nex[i],i==lena?"\n":" ");
    }
    return 0;
}


```

§ 2. Trie树

2.1 Trie树模板

```
//#define int __int128
```

```cpp
using namespace std;
#undef mid
typedef long long ll;
typedef unsigned long long ull;
//typedef pair<int,int> PII;

const int N=1e6+7;
const ll mod=1e9+7;
const ll INF=1e15+7;
const double EPS=1e-10;
const int p=131;//13331


int n,m;
int trie[N][26],tot = 1;
int ed[N];
void inserts(char* str){
    int len = strlen(str),p = 1;
    for(int k = 0;k<len;++k){
        int ch = str[k]-'a';
        if(trie[p][ch] == 0)
            trie[p][ch] = ++tot;
        p = trie[p][ch];
    }
    ed[p]=true;
}

int val[N];

int searchs(char* str){
    int len = strlen(str),p=1;
    for(int k = 0;k<len;++k){
        p = trie[p][str[k]-'a'];
        if(p == 0)return 0;
        }
    if(!val[p]&&ed[p]){
        val[p]=true;
        return 1;
    }
    else if(!ed[p])return 0;
    else return 2;
}
char str[N];
int main()
{
    scanf("%d",&n);
    over(i,1,n){

        scanf("%s",str);//糟糕的 scanf 输入体验
        inserts(str);
    }
    scanf("%d",&m);
    over(i,1,m){

        scanf("%s",str);
        int ans = searchs(str);
        if(ans == 0)puts("WRONG");
        else if(ans == 1)puts("OK");
        else puts("REPEAT");
    }
```

```
62      return 0;
63  }
64
```

2.2 可持久化trie树、求最大区间异或和

题目描述

展开

给定一个非负整数序列 $\{a\}$，初始长度为n。

有 m 个操作，有以下两种操作类型：

1. `A x`：添加操作，表示在序列末尾添加一个数 x，序列的长度 $n+1$。
2. `Q l r x`：询问操作，你需要找到一个位置 p，满足$l \leq p \leq r$，使得： $a[p] \oplus a[p+1] \oplus \ldots \oplus a[N] \oplus x$ 最大，输出最大是多少。

我们维护一个前缀异或和:$s[i] = a[1]\ xor\ a[2]\ xor\ \ldots a[i-1]\ xor\ a[i]$

则$a[p]\ xor\ a[p+1]\ xor\ \ldots\ xor\ a[N]\ xor\ x$ 就相当于 $s[N]\ xor\ x\ xor\ s[p-1]$。

这样$S[N]\ xor\ x$就是一个定值val，则相当于求一个 p 满足$l-1<=p-1<=r-1$ 使得 $val\ xor\ s[p]$ 值最大。

因为是异或运算，我们可以利用 “最大异或对” 这道题的一些性质，使用 $trie$ 树。我们维护前缀异或和，找到一个 p ，贪心地使得$s[p]$的二进制最高位开始到最低位的每一个数字都尽量与 val 的二进制对应位相反（相同为0不同为1）
前缀异或和的每个版本都可以用持久化的 $trie$ 记录下来。
对持久化 $trie$ 的每个节点额外维护一个信息 max_id ，表示其所属的最大的持久化版本，显然一个节点的 max_id 等于其子节点中最大的版本号（因为子节点要么是连向之前的版本，要么创建了该节点后再创建子节点）。

因为要满足边界性所以要用可持久化trie树的保存历史版本来界定边界

- 边界$[l-1, r-1]$ 判断:

如果一个节点的 max_id 小于 $l-1$ ，说明这个节点是 $s[l-1]$ 插入之前就已经创建出来的节点，不应该考虑在内。

对持久化树的某一个版本的根节点开始往下访问，所能访问到的节点的版本不会超过该根节点的版本,所以该题只需要从 $r-1$ 版本开始访问即可满足取到的 p 小于等于 $r-1$。

```
1  //trie树维护的是前缀异或和的二进制01串
2  #include<cstdio>
3  #include<algorithm>
4  #include<cstring>
5  #include<cmath>
6  using namespace std;
7  typedef long long ll;
8  const int N = 500007, M = N * 25;
9  /*
10 template<typename T>inline T read(T &x)
11 {
12     x=0;ll f=1;char c;
13     while(!isdigit(c=getchar()))if(c=='-')f=-1;
14     while(isdigit(c)){x=(x<<1)+(x<<3)+(c^48);c=getchar();}
15     return x*f;
16 }
17 */
18 int n, m;
19 int sum[N];
20 int tr[M][2], max_id[M];
21 int root[N], idx;
22 //当前的区间位置（到时候[L,R]对比的就是这里的i）k表示当前是第几位，一共23位2^23，越高越大
23 void insert(int i, int k, int p, int q){//p上一个版本q下一个版本
24     if(k < 0){
25         max_id[q] = i;
```

```
        return ;
    }
    int v = sum[i] >> k & 1;
    if(p)//如果上个版本的当前结点是有东西的就继承一下
        tr[q][v ^ 1] = tr[p][v ^ 1];
    tr[q][v] = ++ idx;
    insert(i, k - 1, tr[p][v], tr[q][v]);
    max_id[q] = max(max_id[tr[q][0]], max_id[tr[q][1]]);
}

int query(int root, int C, int L){
    int p = root;
    for(int i = 23;i ≥ 0;i -- ){
        int v = C >> i & 1;
        //如果版本小于L-1，说明这个节点是s[l-1]插入之前就已经创建出来的节点，不应该考虑在内
        if(max_id[tr[p][v ^ 1]] ≥ L)//如果与当前这一位相反的数存在并且该数的位置在范围以内（求异或相反为1更大）
            p = tr[p][v ^ 1];
        else p = tr[p][v];
    }
    return C ^ sum[max_id[p]];
}

int main(){
    scanf("%d%d", &n, &m);
    max_id[0] = -1;//因为id从0开始这里要取一个更小的
    root[0] = ++ idx;
    insert(0, 23, 0, root[0]);

    for(int i = 1;i ≤ n; ++ i){
        int x;
        scanf("%d", &x);
        sum[i] = sum[i - 1] ^ x;//前缀异或和
        root[i] = ++ idx;
        insert(i, 23, root[i - 1], root[i]);//可持久化都是跟上一个版本比较

    }
    char op[2];

    int l, r, x;
    while(m -- ){
        scanf("%s", op);
        if(*op == 'A'){
            scanf("%d", &x);
            n ++ ;
            sum[n] = sum[n - 1] ^ x;
            root[n] = ++ idx;
            insert(n, 23, root[n - 1], root[n]);
        }
        else {
            scanf("%d%d%d", &l, &r, &x);
            //l≤p≤r,所以要查询r-1版本的
            printf("%d\n", query(root[r - 1], sum[n] ^ x, l - 1));
        }
    }
}

```

$\oint$ 3. AC自动机

AC自动机是一种多模匹配算法
AC自动机的关键是：**构建字典图实现自动跳转，构建失配指针实现多模式匹配。**

> 形式上，AC 自动机基于由若干模式串构成的 Trie 树，并在此之上增加了一些 fail 边；本质上，AC 自动机是一个关于若干模式串的DFA（确定有限状态自动机），接受且仅接受以某一个模式串作为后缀的字符串。 并且，与一般自动机不同的，AC 自动机还有关于某个模式串的接受状态，也就是与某个模式串匹配（以某个模式串为后缀）的那些状态，即某个模式串在 Trie树上的终止节点在 fail 树上的整个子树。

给定n个模式串和1个文本串，求有多少个模式串在文本串里出现过。

3.1 求有多少个模式串在文本串里出现过

```
/*luogu P3808 【模板】AC自动机（简单版）*/

struct Trie{//trie树
    int fail;//失配指针
    int vis[30];//子结点的位置编号
    //vis数组实际上就是trie树里的trie[p][ch],指向的是该结点的子结点的编号
    int End;//trie树里用于标记有几个单词是以这个结点结尾的
}AC[N];

int tot=0;//AC自动机里是从根节点0开始，trie树一般是从根节点1开始

inline void build(string s){//构造tried树//基本的trie树的操作
    int len = s.length();
    int p = 0;//trie树的当前指针（当前结点）从根节点0开始
    for(int i = 0;i < len;++i){
        if(AC[p].vis[s[i]-'a'] == 0)//tire树的老规矩（不存在这个结点）
            AC[p].vis[s[i]-'a'] = ++tot;//不存在就构造一个新结点
         p = AC[p].vis[s[i]-'a'];//往下走
    }
    AC[p].End++;//标记每一个单词的结尾//因为最后要求的是有多少个单词。要计算重复的
}

void Get_fail(){//构造失配指针利用bfs 遍历
    queue<int>Q;// 队列存
    for(int i= 0;i < 26;++i){//处理第二层的失配指针（26个字母遍历一遍）
        if(AC[0].vis[i] != 0){//若存在这个点
            AC[AC[0].vis[i]].fail = 0;//第二层都要指向根节点
            Q.push(AC[0].vis[i]);//并将该结点编号压入队列
        }
    }
    while(!Q.empty()){//遍历所有的结点
        int u = Q.front();//每次取出队头
        Q.pop();
        for(int i = 0;i < 26;++i){//枚举所有可能的子结点（26个字母）
            if(AC[u].vis[i]!=0){//若存在这个字母的子结点
                AC[AC[u].vis[i]].fail = AC[AC[u].fail].vis[i];//注意是vis[i],一定是指向与该结点字母相同的结点，若不存在就会是指向0（根节点）
                //让当前结点的子结点的失配指针fail指向，当前结点的失配指针fail所指向的
                //结点的，与当前结点的子结点字母相同的子结点
                Q.push(AC[u].vis[i]);
                //并把该结点压入队列里等待继续往下遍历
            }
            else//若不存在子结点
            AC[u].vis[i] = AC[AC[u].fail].vis[i];
            //因为不存在该结点，本身在trie树里会直接结束，但是会浪费时间
            //所以这里本身没有路可以走了，
            //但是这里让当前结点的子结点直接指向当前结点的失配指针fail指向的结点，
```

```
            //把路连上，下次遇上了可以直接走，这样将trie树构造成trie图，更加的优化、
            ////////////////////////////
            //其实就是相当于先预处理好，因为每一个点都要尽量连到另一个相同字母的结点，每一次遍历可能尽管这个结点没有这个字母
            //但是先把他连到结点的失配指针指向的那一个结点，说不定就有这个字母
        }
    }
}

int AC_Query(string s){//AC自动机匹配
    int len = s.length();
    int p = 0,ans = 0;
    for(int i = 0;i < len;++i){
        p = AC[p].vis[s[i]-'a'];//往下一层走
        for(int k = p;k&&AC[k].End!=-1;k = AC[k].fail){//往后走，若没有到走过的结点或者到达根节点，就一直往失配指针指向的地方走
            ans += AC[k].End;//加上路径上的值
            AC[k].End = -1;//标记已走过（已经加过一遍了）
        }
    }
    return ans;
}

int n;
string s;

int main()
{
    scanf("%d",&n);
    memset(AC, 0, sizeof AC);//memset可以初始化结构体哇
    over(i,1,n){
        cin>>s;
        build(s);//建trie树（图）
    }
    AC[0].fail = 0;//结束标志（根节点的失配指针是一定指向自己的）
    Get_fail();//求出失配指针
    cin>>s;
    cout<<AC_Query(s)<<endl;
    return 0;
}
```

3.2 建fail树dfs求每个模式串在文本串中的出现次数

暴力跳 fail 边，会被类似于 aaaaa……aaaaa 这样的串卡掉。

正确的做法是建出 fail 树，记录自动机上的每个状态被匹配了几次，最后求出每个模式串在 Trie 上的终止节点在 fail 树上的子树总匹配次数就可以了，我们dfs建立好的fail树，从当前结点的失配指针指向的结点向该结点连边建fail树，这样我们dfs该树的时候可以保证每一条边（u,v），u到根节点之间的串等于v到根节点之间的串，那么该串的出现次数就是所有的边（u,v），的size之和。不会加重， 因为fail树是由fail指针边构成的，然而所有的点只会建立一个fail指针。例如有三个相同的模式串，只会有两个fail指针。

```
const int N = 200010, S = 2000010;

queue<int>q;
struct trie{
    int fail;
    int vis[26];
    int ed;
}ac[N];

```

```
int match[N];
int n;
int tot, sizes[N];
int head[N], nex[N], ver[N], cnt, num;

void add(int x, int y){
    ver[cnt] = y;
    nex[cnt] = head[x];
    head[x] = cnt ++ ;
}

void insert(int x, char s[]){
    int len = strlen(s);
    int p = 0;//根从0开始
    for(int i = 0; i < len; ++ i){
        int ch = s[i] - 'a';
        if(!ac[p].vis[ch])
            ac[p].vis[ch] = ++ tot;
        p = ac[p].vis[ch];
    }
    ac[p].ed ++ ;
    match[x] = p;//记录每个模式串在 Trie 树上的终止节点
}

void get_fail(){

    for(int i = 0;i < 26; ++ i){
        if(ac[0].vis[i]){
            ac[ac[0].vis[i]].fail = 0;
            q.push(ac[0].vis[i]);
        }
    }

    while(q.size()){
        int p = q.front();
        q.pop();
        for(int i = 0;i < 26; ++ i){
            if(ac[p].vis[i]){
                ac[ac[p].vis[i]].fail = ac[ac[p].fail].vis[i];
                q.push(ac[p].vis[i]);
            }
            else ac[p].vis[i] = ac[ac[p].fail].vis[i];
        }
    }
}

void dfs(int x){
    for(int i = head[x]; ~i; i = nex[i]){
        int y = ver[i];
        dfs(y);
        sizes[x] += sizes[y];
    }
}

int ac_query(char s[]){

    int p = 0, ans = 0;
    int len = strlen(s);

    for(int i = 0;i < len;++i){//!计算有多少个单词在文本中出现
```

```
        p = ac[p].vis[s[i]-'a'];//往下一层走
        for(int k = p;k&&ac[k].ed≠-1;k = ac[k].fail){
                ans += ac[k].ed;
                ac[k].ed = -1;
        }
    }

    p = 0;

    for(int i = 0; i < len; ++ i){
        p = ac[p].vis[s[i] - 'a'];
        sizes[p] ++ ;//!记录匹配次数
    }

    for(int i = 1; i ≤ tot; ++ i)//!建fail树
        add(ac[i].fail, i);

    dfs(0);//!统计子树和

    for(int i = 1;i ≤ n; ++ i)
        printf("%d\n", sizes[match[i]]);

    return ans;
}

char s[S];

int main(){
    scanf("%d", &n);
    memset(head, -1, sizeof head);

    for(int i = 1;i ≤ n; ++ i){
        scanf("%s", s);
        insert(i, s);
    }
    //cout << "ok" << num ++  << endl;
    get_fail();
    //cout << "ok" << num ++  << endl;
    scanf("%s", s);

    //cout << "ok" << num ++  << endl;
    int res = ac_query(s);

    //cout << res << endl;
    return 0;
}
```

3.3 ac自动机fail树上dfs序建可持久化线段树

慕名而来

> 给定 n 个字符串 $s_{1...n}$。q 次询问 s_k在 $s_{l...r}$中出现了多少次。 $n, \sum |s| \leq 2 \times 10^5 q \leq 5 \times 10^5$

```
#include<bits/stdc++.h>
#define pb push_back
using namespace std;

const int N=2e5+10,M=N*40;
int n,Q;
```

```
7
8  namespace IO
9  {
10     int read()
11     {
12         int ret=0;char c=getchar();
13         while(!isdigit(c)) c=getchar();
14         while(isdigit(c)) ret=ret*10+(c^48),c=getchar();
15         return ret;
16     }
17     void write(int x){if(x>9)write(x/10);putchar(x%10^48);}
18     void writeln(int x){write(x);putchar('\n');}
19 }
20 using namespace IO;
21
22 namespace Tree
23 {
24     vector<int>e[N];
25     int ind,rt[N],st[N],en[N];
26     struct Segment
27     {
28         int sz,ls[M],rs[M],sum[M];
29         void copy(int x,int y){ls[x]=ls[y];rs[x]=rs[y];sum[x]=sum[y];}
30         void pushup(int x){sum[x]=sum[ls[x]]+sum[rs[x]];}
31         void update(int y,int &x,int l,int r,int p)
32         {
33             //printf("%d %d %d %d %d\n",y,x,l,r,p);
34             x=++sz;copy(x,y);
35             if(l==r){sum[x]++;return;}
36             int mid=(l+r)>>1;
37             if(p≤mid) update(ls[y],ls[x],l,mid,p);
38             else update(rs[y],rs[x],mid+1,r,p);
39             pushup(x);
40             //printf("%d %d %d\n",l,r,sum[x]);
41         }
42         int query(int y,int x,int l,int r,int L,int R)
43         {
44             if(L≤l && r≤R) return sum[x]-sum[y];
45             int mid=(l+r)>>1,res=0;
46             if(L≤mid) res+=query(ls[y],ls[x],l,mid,L,R);
47             if(R>mid) res+=query(rs[y],rs[x],mid+1,r,L,R);
48             return res;
49         }
50     }tr;
51     void dfs(int x)
52     {
53         st[x]=++ind;
54         for(int i=0;i<(int)e[x].size();++i) dfs(e[x][i]);
55         en[x]=ind;
56     }
57 }
58 using namespace Tree;
59
60 namespace ACM
61 {
62     int ed[N],len[N];
63     char s[N];
64     vector<int>ts[N];
65     queue<int>q;
66     struct ACM
```

```
    {
        int sz,rt,fail[N],ch[N][26];
        void init(){rt=sz=1;}
        void in(int id)
        {
            scanf("%s",s+1);len[id]=strlen(s+1);int now=rt;
            //printf("%d %d\n",id,len[id]);
            for(int i=1;i≤len[id];++i)
            {
                ts[id].pb(s[i]-'a');
                int x=s[i]-'a';
                if(!ch[now][x]) ch[now][x]=++sz;
                now=ch[now][x];
            }
            ed[id]=now;
        }
        void getfail()
        {
            q.push(rt);
            while(!q.empty())
            {
                int x=q.front();q.pop();
                for(int i=0;i<26;++i)
                {
                    if(!ch[x][i]) continue;
                    int t=fail[x],t1=ch[x][i];
                    while(t && !ch[t][i]) t=fail[t];
                    fail[t1]=t?ch[t][i]:rt;
                    q.push(t1);
                }
            }
            for(int i=1;i≤sz;++i) e[fail[i]].pb(i);
        }
    }S;
}
using namespace ACM;

void build(int y,int &x,int id)
{
    int now=S.rt,las=y;
    //printf("%d %d %d %d\n",id,len[id],st[ed[id]],en[ed[id]]);
    for(int i=0;i<len[id];++i)
    {
        now=S.ch[now][ts[id][i]];
        tr.update(las,x,1,ind,st[now]);las=x;
    }
}

int main()
{
#ifndef ONLINE_JUDGE
    freopen("CF547E.in","r",stdin);
    freopen("CF547E.out","w",stdout);
#endif
    n=read();Q=read();S.init();
    for(int i=1;i≤n;++i) S.in(i);
    S.getfail();dfs(S.rt);
    for(int i=1;i≤n;++i) build(rt[i-1],rt[i],i);
    while(Q--)
    {
```

```
        int l=read(),r=read(),id=read();
        printf("%d\n",tr.query(rt[l-1],rt[r],1,ind,st[ed[id]],en[ed[id]]));
    }
    return 0;
}
```

$\oint$ 4. 字符串哈希

4.1 字符串哈希

核心思想：将字符串看成P进制数，P的经验值是131或13331，取这两个值的冲突概率低
小技巧：取模的数用2^64，这样直接用unsigned long long存储，溢出的结果就是取模的结果

```
typedef unsigned long long ULL;
ULL h[N], p[N]; // h[k]存储字符串前k个字母的哈希值, p[k]存储 P^k mod 2^64

// 初始化
p[0] = 1;
for (int i = 1; i ≤ n; i ++ )
{
    h[i] = h[i - 1] * P + str[i];
    p[i] = p[i - 1] * P;
}

// 计算子串 str[l ~ r] 的哈希值
ULL get(int l, int r)
{
    return h[r] - h[l - 1] * p[r - l + 1];
}
```

如题，给定 N 个字符串（第 i 个字符串长度为 M_i，字符串内包含数字、大小写字母，大小写敏感），请求出 N 个字符串中共有多少个不同的字符串。

```
const int p=131;//13331
int n,m;
ull a[N];
char str[N];
ull get_hash(char s[]){
    ull res=0;
    int len=strlen(s);
    over(i,0,len-1){
        res=res*p+(ull)s[i];
    }
    return res;
}

int main()
{
    scanf("%d",&n);
    over(i,1,n){
        cin>>str;
        a[i]=get_hash(str);
    }
    int ans=1;
    sort(a+1,a+1+n);
    over(i,1,n-1){
```

```
        if(a[i]≠a[i+1])
            ans++;
    }
    printf("%d\n",ans);
    return 0;
}

```

4.2 二分+哈希

> 给定一个字符串，要求取出k个位置不相交的子串，且他们之间任意两个的最长公共前缀的长度均不小于x。现在给出k，求最大的x。
> 两个字符串str1，str2的公共前缀为x指str1和str2的长度均不小于x且这两个字符串的前x个字符对应相同。最长公共前缀即所有的公共前缀里最长的那个，如没有公共前缀则视为最长公共前缀长度为0。

很明显直接二分答案，用哈希来check是否有长度为k的字符串即可，check函数中用 `unordered_map` 节省时间，用类似滑动窗口的尺取操作来更新即可~~非常简单~~ 然后就没什么好说的了
时间复杂度 $O(nlogn)$

```
char s[N];
ll n,m,k,ans;
ll ha[N],p[N];
bool check(ll x)
{
    unordered_map<ll,pair<ll,ll> >mp;
    over(i,x,n)
    {
        ll tmp=ha[i]-ha[i-x]*p[x];
        if(i-mp[tmp].first≥x)mp[tmp].first=i,mp[tmp].second++;
        if(mp[tmp].second≥k)return true;
    }
    return false;
}
int main()
{
    scanf("%lld%lld",&n,&k);
    scanf("%s",s+1);
    p[0]=1;
    over(i,1,n)
    {
        p[i]=p[i-1]*2333;
        ha[i]=ha[i-1]*2333+s[i];
    }
    int l=1,r=n/k;
    while(l≤r)
    {
        ll mid=(l+r)/2;
        if(check(mid))l=mid+1,ans=mid;
        else r=mid-1;
    }
    printf("%lld\n",ans);
    return 0;
}

```

§ 5. 字符串的最小表示

> 字符串长度为n，旋转n次，取字典序最小的那一种，即为字符串的最小表示。

判断是否有相同雪花的方式就是直接暴力枚举就好
若只有旋转操作，可以用字符串的最小表示:
字符串长度为n，旋转n次，取字典序最小的那一种，即为字符串的最小表示。
现在有翻转操作，所以我们对原序列求最小表示，再对翻转后的序列求一个最小表示

```cpp
using namespace std;
typedef long long ll;
typedef pair<int,int> PII;
typedef pair<ll,ll> PLL;

const int N=1e5+7;
const int mod=1e9+7;
const ll INF=1e15+7;
const double EPS=1e-10;

int n;
int snows[N][6],id[N];

void get_min(int *b){//字符串的最小表示
    static int a[12];
    for(int i=0;i<12;i++)
        a[i]=b[i%6];
    int i=0,j=1,k;
    while(i<6&&j<6){
        for(k=0;k<6&&a[i+k]==a[j+k];k++);//每次直接找就完事了
        if(k==6)break;//说明全部相等
        if(a[i+k]>a[j+k]){//谁大往后跳，小的永远不动，也就是答案
            i+=k+1;//i+k是跑到字符相等的位置，要再加1往后移一位
            if(i==j)//如果相等就往后退
                i++;
        }
        else {
            j+=k+1;//一样
            if(i==j)
                j++;
        }
    }
    k=min(i,j);
    for(int i=0;i<6;++i)
        b[i]=a[i+k];
}

bool cmp_array(int a[],int b[]){//比较a和b字典序谁最小
    for(int i=0;i<6;++i){
        if(a[i]<b[i])
            return true;
        else if(a[i]>b[i])
            return false;
    }
    return false;//全部相等
}

bool cmp_val(int a,int b){//如果正反一比都是false说明全相等
    for(int i=0;i<6;++i){
        if(snows[a][i]<snows[b][i])
            return true;
        else if(snows[a][i]>snows[b][i])
            return false;
    }
    return false;
```

```
}
int main()
{
    scanf("%d",&n);
    int snow[6],rsnow[6];
    over(i,0,n-1){
        for(int j=0,k=5;j<6;j++,k--){
            scanf("%d",&snow[j]);//原序列
            rsnow[k]=snow[j];//翻转序列
        }
        get_min(snow);
        get_min(rsnow);
        if(cmp_array(snow,rsnow))memcpy(snows[i],snow,sizeof snow);
        else memcpy(snows[i],rsnow,sizeof rsnow);
        id[i]=i;
    }
    sort(id,id+n,cmp_val);
    for(int i=1;i<n;++i){
        if(!cmp_val(id[i],id[i-1])&&!cmp_val(id[i-1],id[i])){//如果相等（这里神）
            puts("Twin snowflakes found.");
            return 0;
        }
    }
    puts("No two snowflakes are alike.");
    return 0;
}
```

∮ 6. manacher算法（O(n)判断回文串）

```
char data[maxn<<1];
int p[maxn<<1],cnt,ans;
inline void qr(){
      char c=getchar();
      data[0]='~',data[cnt=1]='|';
      while(c<'a'||c>'z') c=getchar();
      while(c≥'a'&&c≤'z') data[++cnt]=c,data[++cnt]='|',c=getchar();
}
int main(){
      qr();
      for(int t=1,r=0,mid=0;t≤cnt;++t){
        if(t≤r) p[t]=min(p[(mid<<1)-t],r-t+1);
        while(data[t-p[t]]==data[t+p[t]]) ++p[t];
        //暴力拓展左右两侧,当t-p[t]==0时，由于data[0]是'~'，自动停止。故不会下标溢出。
        //假若我这里成功暴力扩展了，就意味着到时候r会单调递增，所以manacher是线性的。
        if(p[t]+t>r) r=p[t]+t-1,mid=t;
        //更新mid和r,保持r是最右的才能保证我们提前确定的部分回文半径尽量多。
        if(p[t]>ans) ans=p[t];
      }
      printf("%d\n",ans-1);
      return 0;
}
```

∮ 7. 后缀数组（SA）

给定一个字符串S，比如它是（abcad）,那么它的后缀有”abcad “ ,"bcad ", "cad", "ad", "d", ""。 讲这些后缀字符串按照字典序排序，得到的就是后缀数组。如果用普通的排序方法，排序要o（nlogn），但是每两个字符比较大小要O（n），所以是o（n × n log n）的复杂度。但是利用特殊的算法可以将其降到o（nlogn）。

该算法的思想是，先将每个后缀数组按照只考虑头一个字符，排序， 再将其按照只考虑头两个字符排序，然后是头4个，然后是8个，直到考虑n个。

后缀数组Suffix_Array：SA[]

将某个字符串的所有后缀按字典序排序后得到的数组；

SA[i] = k即表示排序后第i小的子串为S[k ... n]（为好理解暂用字符串下标1~n）；

名次数组：Rank[]

保存后缀S[i ... n]在排序中的名次；

rank[i] = k即表示子串S[i ... n]在所有后缀的字典序排序中为第k小；

可以看出SA[1] = 4, Rank[4] = 1; 即SA数组与Rank数组为**互逆关系**；

sa[i]：表示 排在第i位的后缀 起始下标

rank[i]：表示后缀 suffix(i)排在第几

height[i] : 表示 sa[i-1] 与 sa[i] 的LCP 值

h[i]: 表示 suffix(i)与其排名前一位的 LCP值

7.1 DA（倍增算法O(nlogn)）

```
const int N = 100005;
int wa[N],wb[N],wv[N],ws[N];
int cmp(int *r,int a,int b,int l)
{
    return r[a]==r[b]&&r[a+l]==r[b+l];
}
void da(int *r,int *sa,int n,int m)
{
    int i,j,p,*x=wa,*y=wb;
    // 下面四行是对第一个字母的一个基数排序：基数排序其实就是记录前面有多少个位置被占据了
    for(i=0;i<m;i++) ws[i]=0; // 将统计字符数量的数组清空
    for(i=0;i<n;i++) ws[x[i]=r[i]]++; // 统计各种字符的个数
    for(i=1;i<m;i++) ws[i]+=ws[i-1]; // 进行一个累加，因为前面的小字符集对后面字符的排位有位置贡献
    for(i=n-1;i≥0;i--) sa[--ws[x[i]]]=i; // 根据位置来排序，sa[x] = i，表示i位置排在第x位
    // wa[x[i]]就是字符集0-x[i]共有多少字符占据了位置，减去自己的一个位置剩下的就是自己的排名了，排名从0开始
    // 排名过程中主要的过程是对于处于相同字符的字符的排序，因为改变wa[x[i]]值得只会是本身，小于该字符的贡献值
    // 是不变的，对于第一个字符相同的依据是位置关系，在后面将看到通过第二个关键字来确定相同字符的先后关系


    // 这以后的排序都是通过两个关键字来确定一个串的位置，也即倍增思想
    // 通过将一个串分解成两部分，而这两部分的位置关系我们都已经计算出来
    for(j=1,p=1;p<n;j*=2,m=p)
    {
        for(p=0,i=n-j;i<n;i++) y[p++]=i; // 枚举的串是用于与i位置的串进行合并，由于i较大，因为匹配的串为空串
        // 由于枚举的是长度为j的串，那么i位置开始的串将凑不出这个长度的串，因此第二关键字应该最小，这其中位置靠前的较小
        for(i=0;i<n;i++) if(sa[i]≥j) y[p++]=sa[i]-j; // sa[i]-j开头的串作为第二关键字与编号为sa[i]的串匹配，
sa[i]<j的串不用作为第二关键字来匹配
        for(i=0;i<n;i++) wv[i]=x[y[i]]; // 取出这些位置的第一关键字
        for(i=0;i<m;i++) ws[i]=0;
        for(i=0;i<n;i++) ws[wv[i]]++;
        for(i=1;i<m;i++) ws[i]+=ws[i-1];
        for(i=n-1;i≥0;i--) sa[--ws[wv[i]]]=y[i]; // 按照第二关键字进行第一关键字的基数排序
        for(swap(x,y),p=1,x[sa[0]]=0,i=1;i<n;i++) // 对排好序的sa数组进行一次字符集缩小、常数优化
        x[sa[i]]=cmp(y,sa[i-1],sa[i],j)?p-1:p++;
```

```
    }
    return;
}

int rank[N],height[N];
void calheight(int *r,int *sa,int n) // 这里的n是原串的本来长度，即不包括新增的0
{
    int i,j,k=0;
    for(i=1;i≤n;i++) rank[sa[i]]=i; // 有后缀数组得到名次数组，排名第0的后缀一定是添加的0
    for(i=0;i<n;height[rank[i++]]=k) // 以 i 开始的后缀总能够从以 i-1 开始的后缀中继承 k-1 匹配项出来
    for(k?k--:0,j=sa[rank[i]-1];r[i+k]==r[j+k];k++); // 进行一个暴力的匹配，但是整个算法的时间复杂度还是O(n)的
    return;
}

int main() {

    return 0;
}
```

DC3 模板 (时间复杂度O(N),空间复杂度O(3N))

```
const int maxn = int(3e6)+10;
const int N = maxn;

    #define F(x) ((x)/3+((x)%3==1?0:tb))
    #define G(x) ((x)<tb?(x)*3+1:((x)-tb)*3+2)
    int wa[maxn],wb[maxn],wv[maxn],ws[maxn];
    int c0(int *r,int a,int b)
    {return r[a]==r[b]&&r[a+1]==r[b+1]&&r[a+2]==r[b+2];}
    int c12(int k,int *r,int a,int b)
    {if(k==2) return r[a]<r[b]||r[a]==r[b]&&c12(1,r,a+1,b+1);
    else return r[a]<r[b]||r[a]==r[b]&&wv[a+1]<wv[b+1];}
    void sort(int *r,int *a,int *b,int n,int m)
    {
        int i;
        for(i=0;i<n;i++) wv[i]=r[a[i]];
        for(i=0;i<m;i++) ws[i]=0;
        for(i=0;i<n;i++) ws[wv[i]]++;
        for(i=1;i<m;i++) ws[i]+=ws[i-1];
        for(i=n-1;i≥0;i--) b[--ws[wv[i]]]=a[i];
        return;
    }
    void dc3(int *r,int *sa,int n,int m) //涵义与DA 相同
    {
        int i,j,*rn=r+n,*san=sa+n,ta=0,tb=(n+1)/3,tbc=0,p;
        r[n]=r[n+1]=0;
        for(i=0;i<n;i++) if(i%3≠0) wa[tbc++]=i;
        sort(r+2,wa,wb,tbc,m);
        sort(r+1,wb,wa,tbc,m);
        sort(r,wa,wb,tbc,m);
        for(p=1,rn[F(wb[0])]=0,i=1;i<tbc;i++)
        rn[F(wb[i])]=c0(r,wb[i-1],wb[i])?p-1:p++;
        if(p<tbc) dc3(rn,san,tbc,p);
        else for(i=0;i<tbc;i++) san[rn[i]]=i;
        for(i=0;i<tbc;i++) if(san[i]<tb) wb[ta++]=san[i]*3;
        if(n%3==1) wb[ta++]=n-1;
        sort(r,wb,wa,ta,m);
        for(i=0;i<tbc;i++) wv[wb[i]=G(san[i])]=i;
        for(i=0,j=0,p=0;i<ta && j<tbc;p++)
```

```
        sa[p]=c12(wb[j]%3,r,wa[i],wb[j])?wa[i++]:wb[j++];
        for(;i<ta;p++) sa[p]=wa[i++];
        for(;j<tbc;p++) sa[p]=wb[j++];
        return;
    }
```

$\oint$ 8. 后缀自动机（SAM）

8.1 求子串最大出现次数

> 给定一个只包含小写字母的字符串S请你求出 S的所有出现次数不为 1 的子串的出现次数乘上该子串长度的最大值。

```
#include<cstdio>
#include<algorithm>
using namespace std;const int N=3*1e6+10;typedef long long ll;
char mde[N];int nl;ll res;
struct suffixautomation
{
    int mp[N][30];int fa[N];int ed;int ct;int len[N];int siz[N];
    suffixautomation(){ed=ct=1;}
    int v[N];int x[N];int al[N];int cnt;
    inline void add(int u,int V){v[++cnt]=V;x[cnt]=al[u];al[u]=cnt;}
    inline void ins(int c)
    {
        int p=ed;siz[ed=++ct]=1;len[ed]=nl;//先初始化size和len
        for(;p&&mp[p][c]==0;p=fa[p]){mp[p][c]=ed;}//然后顺着parent树的路径向上找
        if(p==0){fa[ed]=1;return;}int q=mp[p][c];//case1
        if(len[p]+1==len[q]){fa[ed]=q;return;}//case2
        len[++ct]=len[p]+1;//case 3
        for(int i=1;i≤26;i++){mp[ct][i]=mp[q][i];}
        fa[ct]=fa[q];fa[q]=ct;fa[ed]=ct;
        for(int i=p;mp[i][c]==q;i=fa[i]){mp[i][c]=ct;}
    }
    inline void bt(){for(int i=2;i≤ct;i++){add(fa[i],i);}}//暴力建树
    void dfs(int u)//dfs
    {
        for(int i=al[u];i;i=x[i]){dfs(v[i]);siz[u]+=siz[v[i]];}
        if(siz[u]≠1){res=max(res,(ll)siz[u]*len[u]);}
    }
}sam;
int main()
{
    scanf("%s",mde+1);//没啥好说的，建sam然后dfs
    for(nl=1;mde[nl]≠'\0';nl++){sam.ins(mde[nl]-'a'+1);}
    sam.bt();sam.dfs(1);printf("%lld",res);return 0;//拜拜程序~
}
```

8.2 求不同子串的种类

8.3 长度为k的字符串的个数

8.4 计算所有子串的和（0-9表示）

8.5 给定模式串 s，n 个匹配串 str

8.6 求每个匹配串的循环同构能够匹配的子串总数

```
#include <bits/stdc++.h>
#define LL long long
#define P pair<int, int>
#define lowbit(x) (x & -x)
#define mem(a, b) memset(a, b, sizeof(a))
#define rep(i, a, n) for (int i = a; i ≤ n; ++i)
const int maxn = 1000005;
#define mid ((l + r) >> 1)
#define lc rt<<1
#define rc rt<<1|1
using namespace std;
// __int128 read() {    __int128 x = 0, f = 1;  char c = getchar(); while (c < '0' || c > '9') {
if (c == '-') f = -1;       c = getchar();  }   while (c ≥ '0' && c ≤ '9') {      x = x * 10 + c -
'0';       c = getchar();  }   return x * f;}
// void print(__int128 x) { if (x < 0) {       putchar('-');       x = -x; }   if (x > 9)  print(x /
10);  putchar(x % 10 + '0');}
const LL mod = 1e9 + 7;
int len;
struct SAM{

    int trans[maxn<<1][26], slink[maxn<<1], maxlen[maxn<<1];
    // 用来求endpos
    int indegree[maxn<<1], endpos[maxn<<1], rank[maxn<<1], ans[maxn<<1];
    // 计算所有子串的和(0-9表示)
    LL sum[maxn<<1];
    int last, now, root;

    inline void newnode (int v) {
        maxlen[++now] = v;
        mem(trans[now],0);
    }

    inline void extend(int c) {
        newnode(maxlen[last] + 1);
        int p = last, np = now;
        // 更新trans
        while (p && !trans[p][c]) {
            trans[p][c] = np;
            p = slink[p];
        }
        if (!p) slink[np] = root;
        else {
            int q = trans[p][c];
            if (maxlen[p] + 1 ≠ maxlen[q]) {
                // 将q点拆出nq，使得maxlen[p] + 1 == maxlen[q]
                newnode(maxlen[p] + 1);
                int nq = now;
                memcpy(trans[nq], trans[q], sizeof(trans[q]));
                slink[nq] = slink[q];
                slink[q] = slink[np] = nq;
                while (p && trans[p][c] == q) {
                    trans[p][c] = nq;
                    p = slink[p];
                }
            }else slink[np] = q;
```

```
        }
        last = np;
        // 初始状态为可接受状态
        endpos[np] = 1;
    }


    inline void init()
    {
        root = last = now = 1;
        slink[root]=0;
        mem(trans[root],0);
    }
    // 计算所有子串的和（0-9表示）
    inline LL getSum() {
        // 拓扑排序
        for (int i = 1; i ≤ now; ++i) indegree[ maxlen[i] ]++;
        for (int i = 1; i ≤ now; ++i) indegree[i] += indegree[i-1];
        for (int i = 1; i ≤ now; ++i) rank[ indegree[ maxlen[i] ]-- ] = i;
        mem(endpos, 0);
        endpos[1] = 1; // 从根节点向后求有效的入度
        for (int i = 1; i ≤ now; ++i) {
            int x = rank[i];
            for (int j = 0; j < 10; ++j) {
                int nex = trans[x][j];
                if (!nex) continue;
                endpos[nex] += endpos[x]; // 有效入度
                LL num = (sum[x] * 10 + endpos[x] * j) % mod;
                sum[nex] = (sum[nex] + num) % mod; // 状态转移
            }
        }
        LL ans = 0;
        for (int i = 2; i ≤ now; ++i) ans = (ans + sum[i]) % mod;
        return ans;
    }
    inline void getEndpos() {
        // topsort
        for (int i = 1; i ≤ now; ++i) indegree[ maxlen[i] ]++; // 统计相同度数的节点的个数
        for (int i = 1; i ≤ now; ++i) indegree[i] += indegree[i-1];  // 统计度数小于等于 i 的节点的总数
        for (int i = 1; i ≤ now; ++i) rank[ indegree[ maxlen[i] ]-- ] = i;  // 为每个节点编号，节点度数越大
编号越靠后
        // 从下往上按照slik更新
        for (int i = now; i ≥ 1; --i) {
            int x = rank[i];
            endpos[slink[x]] += endpos[x];
        }
    }

    // 求不同的子串种类
    inline LL all () {
        LL ans = 0;
        for (int i = root+1; i ≤ now; ++i) {
            ans += maxlen[i] - maxlen[ slink[i] ];
        }
        return ans;
    }
    // 长度为K的字符串有多种，求出现次数最多的次数
    inline void get_Maxk() {
        getEndpos();
        for (int i = 1; i ≤ now; ++i) {
```

```
            ans[maxlen[i]] = max(ans[maxlen[i]], endpos[i]);
        }
        for (int i = len; i ≥ 1; --i) ans[i] = max(ans[i], ans[i+1]);
        for (int i = 1; i ≤ len; ++i) //cout << ans[i] << endl;
            printf("%d\n", ans[i]);
    }

}sam;

int main()
{
//    int n;scanf("%d",&n);
//    string T;
//
//    for(int i=0 ; i<n ; i++)
//    {
//
//          string str;
//          cin>>str;
//          T+=str;
//          T+=":";
//    }

    string T;cin>>T;
    sam.init();
   len=T.size();
    for(int i=0 ; i<len ; i++)
    sam.extend(T[i]-'a');
    cout<<sam.all()<<endl;

   //- sam.all();
}
```

8.7 SAM 转后缀树(子串排序重连后询问第k个字符)

给定一个长为 n字符串 S，现取出 S的所有子串，并按字典序从小到大排序，然后将这些排好序的字符串首尾相接，记为字符串 T。共有 Q次询问，每次询问 T中的第 K个字符。

操作强制在线，每次询问给出两个正整数 P,M，设 G为之前所有询问答案的 ASCII码之和（初始为 0），则对于该次询问：K=(P×G)modM 。
【输入格式】
第一行输入一个字符串 S，第二行一个正整数 Q。
接下来 Q行，每行两个正整数 P,M。
【输出格式】
对于每次询问，输出一行一个字符表示答案。

```
#define Re register int
using namespace std;
const int N=4e5+10;
int n,x,y,T;LL P,M,G,K;char s[N>>1];
inline void in(Re &x){
    int f=0;x=0;char c=getchar();
    while(c<'0'||c>'9')f|=c=='-',c=getchar();
    while(c≥'0'&&c≤'9')x=(x<<1)+(x<<3)+(c^48),c=getchar();
    x=f?-x:x;
}
struct Sakura2{
    int O,last,pos[N],siz[N],link[N],maxlen[N],trans[N][26];
    //siz[x]: 节点x的出现次数 (endpos的大小)
    //link[x]: x在parent树上的父亲
```

```
    //pos[x]: 节点x某一次出现位置（用于构建后缀树中的压缩边link[x]→x），
    ///翻转后的s中 倒序遍历pos[x]-maxlen[link[x]] → pos[x]-maxlen[x]+1得到的字符串 即为后缀树中的压缩边link[x]→x
    inline void insert(Re ch,Re id){//SAM新建节点
        Re z=++O,p=last;pos[z]=id,siz[z]=1,maxlen[z]=maxlen[last]+1;
        while(p&&!trans[p][ch])trans[p][ch]=z,p=link[p];
        if(!p)link[z]=1;
        else{
            Re x=trans[p][ch];
            if(maxlen[p]+1==maxlen[x])link[z]=x;
            else{
                Re y=++O;maxlen[y]=maxlen[p]+1,pos[y]=pos[x];
                for(Re i=0;i<26;++i)trans[y][i]=trans[x][i];
                while(p&&trans[p][ch]==x)trans[p][ch]=y,p=link[p];
                link[y]=link[x],link[z]=link[x]=y;
            }
        }
        last=z;
    }
    int t,ID[N],to[N][26];LL S[N];
    //to[x][ch]: 后缀树的trans数组
    //ID[x]: 顺序遍历后缀树的第x个节点编号
    //S[x]: 顺序遍历后缀树的前x个节点总共代表了多少个字符
    inline void dfs1(Re x){//遍历 SAM的parent树\后缀树 获取siz
        for(Re i=0;i<26;++i)
            if(to[x][i])dfs1(to[x][i]),siz[x]+=siz[to[x][i]];
    }
    inline void dfs2(Re x){//遍历 SAM的parent树\后缀树 获取节点顺序
        if(x^1)ID[++t]=x;//没有储存信息的根节点1就不要了
        for(Re i=0;i<26;++i)if(to[x][i])dfs2(to[x][i]);
    }
    inline LL calc(Re L,Re R){return 1ll*(L+R)*(R-L+1)/2;}//计算从L加到R的等差数列
    inline void build(){
        last=O=1;//根节点设为1
        for(Re i=1;i≤n/2;++i)swap(s[i],s[n-i+1]);//翻转原字符串
        for(Re i=1;i≤n;++i)insert(s[i]-'a',i);
        for(Re i=2;i≤O;++i)to[link[i]][s[pos[i]-maxlen[link[i]]]-'a']=i;//构建后缀树。
        //由于pos[x]是endpos[x]中的任意一个，所以获取边link[x]→x压缩掉的字符串信息时只能用pos[x]，不能用pos[link[x]]
        dfs1(1),dfs2(1);//遍历 SAM的parent树\后缀树
        for(Re i=1;i≤t;++i)x=ID[i],S[i]=S[i-1]+1ll*siz[x]*calc(maxlen[link[x]]+1,maxlen[x]);
    }
    inline char ask(LL K){
        Re l=1,r=t,x;
        while(l<r){//二分找到第一个大于等于K的位置
            Re mid=l+r>>1;
            if(S[mid]<K)l=mid+1;
            else r=mid;
        }
        x=ID[l],K-=S[l-1],l=maxlen[link[x]]+1,r=maxlen[x];
        while(l<r){//二分找到第一个大于等于K的位置
            Re mid=l+r>>1;
            if(1ll*siz[x]*calc(maxlen[link[x]]+1,mid)<K)l=mid+1;
            else r=mid;
        }
        K-=1ll*siz[x]*calc(maxlen[link[x]]+1,l-1),K=(K-1)%l+1;//注意取模的方式
        return s[pos[x]-K+1];//注意方向：往左数第K位
    }
    char ans;
    inline void sakura(){
```

```
        build();
        while(T--)scanf("%lld%lld",&P,&M),K=P*G%M+1,G+=(ans=ask(K)),putchar(ans),puts("");
    }
}T2;
int main(){
    freopen("letter.in","r",stdin);
    freopen("letter.out","w",stdout);
    scanf("%s",s+1),n=strlen(s+1);in(T);
    T2.sakura();
    fclose(stdin);
    fclose(stdout);
}
```

9. 编辑距离

编辑距离，又称Levenshtein距离（也叫做Edit Distance），是指两个字串之间，由一个转成另一个所需的最少编辑操作次数。许可的编辑操作包括将一个字符替换成另一个字符，插入一个字符，删除一个字符。

```
const int N = 1e3 + 5;

int T, cas = 0;
int n, m;
int dp[N][N];
char s[N], t[N];

int main()
{
    while (scanf("%s%s", s, t) ≠ EOF)
    {
        int n = (int)strlen(s), m = (int)strlen(t);
        for (int i = 0; i ≤ n; i++)
        {
            dp[i][0] = i;
        }
        for (int i = 0; i ≤ m; i++)
        {
            dp[0][i] = i;
        }
        for (int i = 1; i ≤ n; i++)
        {
            for (int j = 1; j ≤ m; j++)
            {
                dp[i][j] = min(dp[i - 1][j], dp[i][j - 1]) + 1;
                dp[i][j] = min(dp[i][j], dp[i - 1][j - 1] + (s[i - 1] ≠ t[j - 1]));
            }
        }
        printf("%d\n", dp[n][m]);
    }

    return 0;
}
```

四、数据结构

$\oint$ 1. 简单数据结构

1.1 单链表

```
// head存储链表头，e[]存储节点的值，ne[]存储节点的next指针，idx表示当前用到了哪个节点
int head, e[N], ne[N], idx;

// 初始化
void init()
{
    head = -1;
    idx = 0;
}

// 在链表头插入一个数a
void insert(int a)
{
    e[idx] = a, ne[idx] = head, head = idx ++ ;
}

// 将头结点删除，需要保证头结点存在
void remove()
{
    head = ne[head];
}
```

1.2 双链表

```
// e[]表示节点的值，l[]表示节点的左指针，r[]表示节点的右指针，idx表示当前用到了哪个节点
int e[N], l[N], r[N], idx;

// 初始化
void init()
{
    //0是左端点，1是右端点
    r[0] = 1, l[1] = 0;
    idx = 2;
}

// 在节点a的右边插入一个数x
void insert(int a, int x)
{
    e[idx] = x;
    l[idx] = a, r[idx] = r[a];
    l[r[a]] = idx, r[a] = idx ++ ;
}

// 删除节点a
void remove(int a)
{
    l[r[a]] = l[a];
    r[l[a]] = r[a];
}
```

1.3 栈

```cpp
// tt表示栈顶
int stk[N], tt = 0;

// 向栈顶插入一个数
stk[ ++ tt] = x;

// 从栈顶弹出一个数
tt -- ;

// 栈顶的值
stk[tt];

// 判断栈是否为空
if (tt > 0)
{

}
```

1.4 循环队列

```cpp
// hh 表示队头，tt表示队尾的后一个位置
int q[N], hh = 0, tt = 0;

// 向队尾插入一个数
q[tt ++ ] = x;
if (tt == N) tt = 0;

// 从队头弹出一个数
hh ++ ;
if (hh == N) hh = 0;

// 队头的值
q[hh];

// 判断队列是否为空
if (hh ≠ tt)
{

}
```

1.7 堆

```cpp
// h[N]存储堆中的值, h[1]是堆顶，x的左儿子是2x, 右儿子是2x + 1
// ph[k]存储第k个插入的点在堆中的位置
// hp[k]存储堆中下标是k的点是第几个插入的
int h[N], ph[N], hp[N], size;

// 交换两个点，及其映射关系
void heap_swap(int a, int b)
{
    swap(ph[hp[a]],ph[hp[b]]);
    swap(hp[a], hp[b]);
    swap(h[a], h[b]);
}

void down(int u)
{
    int t = u;
```

```
    if (u * 2 ≤ size && h[u * 2] < h[t]) t = u * 2;
    if (u * 2 + 1 ≤ size && h[u * 2 + 1] < h[t]) t = u * 2 + 1;
    if (u ≠ t)
    {
        heap_swap(u, t);
        down(t);
    }
}

void up(int u)
{
    while (u / 2 && h[u] < h[u / 2])
    {
        heap_swap(u, u / 2);
        u >>= 1;
    }
}

// O(n)建堆
for (int i = n / 2; i; i -- ) down(i);
```

1.8 一般哈希

(1) 拉链法

```
int h[N], e[N], ne[N], idx;

  // 向哈希表中插入一个数
  void insert(int x)
  {
      int k = (x % N + N) % N;
      e[idx] = x;//存到邻接表里
      ne[idx] = h[k];
      h[k] = idx ++ ;
  }

  // 在哈希表中查询某个数是否存在
  bool find(int x)
  {
      int k = (x % N + N) % N;
      for (int i = h[k]; i ≠ -1; i = ne[i])
          if (e[i] == x)
              return true;

      return false;
  }
```

(2) 开放寻址法

```cpp
int h[N];

// 如果x在哈希表中，返回x的下标；如果x不在哈希表中，返回x应该插入的位置
int find(int x)
{
    int t = (x % N + N) % N;
    while (h[t] ≠ null && h[t] ≠ x)
    {
        t ++ ;
        if (t == N) t = 0;
    }
    return t;
}
```

∮ 2. 并查集

并查集是一种树形的数据结构，顾名思义，它用于处理一些不交集的 **合并** 及 **查询** 问题。 它支持两种操作：

- 查找（Find）：确定某个元素处于哪个子集；
- 合并（Union）：将两个子集合并成一个集合。

2.1 朴素并查集

```cpp
int p[N]; //存储每个点的祖宗节点

// 返回x的祖宗节点
int find(int x)
{
    if (p[x] ≠ x) p[x] = find(p[x]);
    return p[x];
}

// 初始化，假定节点编号是1~n
for (int i = 1; i ≤ n; i ++ ) p[i] = i;

// 合并a和b所在的两个集合：
p[find(a)] = find(b);
```

2.2 维护size的并查集

```cpp
int p[N], size[N];
//p[]存储每个点的祖宗节点, size[]只有祖宗节点的有意义，表示祖宗节点所在集合中的点的数量
// 返回x的祖宗节点
int find(int x)
{
    if (p[x] ≠ x) p[x] = find(p[x]);
    return p[x];
}
// 初始化，假定节点编号是1~n
for (int i = 1; i ≤ n; i ++ )
{
    p[i] = i;
    size[i] = 1;
}
// 合并a和b所在的两个集合：
size[find(b)] += size[find(a)];
p[find(a)] = find(b);
```

2.3 维护到祖宗节点距离的并查集

```cpp
int p[N], d[N];
//p[]存储每个点的祖宗节点, d[x]存储x到p[x]的距离
// 返回x的祖宗节点
int find(int x){
    if (p[x] ≠ x)
    {
        int u = find(p[x]);
        d[x] += d[p[x]];
        p[x] = u;
    }
    return p[x];
}
// 初始化，假定节点编号是1~n
for (int i = 1; i ≤ n; i ++ ){
    p[i] = i;
    d[i] = 0;
}
// 合并a和b所在的两个集合：
p[find(a)] = find(b);
d[find(a)] = distance; // 根据具体问题，初始化find(a)的偏移量
```

2.4 路径压缩和按秩合并

```cpp
int fa[N];

void init(){
    for(int i = 1;i ≤ n;++i)
        fa[i] = i;
    memset(Rank,0, sizeof Rank);
}

int getfa(int x){//查询
    if(fa[x] == x) return x;
    return fa[x] = getfa(fa[x]);
}

inline void union( int x, int y){//合并
    int fx = getfa(x) , fy = get(y);
    fa[fx] = fy;
    return ;
}

inline bool same(int x , int y){ //判读是否在同一结合
    return getfa(x) == getfa(y) ;
}


//把深度当作秩的 按秩合并
inline void rank_union( int x, int y)
{
    fx = getfa(x) , fy = getfa(y);
    if(Rank[fx] < Rank[fy]) fa[fx] = fy;
    else {
        fa[fy] = fx;
        if(Rank[fx] == Rank[fy]) Rank[fx]++;
    }
```

```
    return ;
}
```

2.5 边带权

使用情景：

N个节点有M对关系(M条边)，每对关系(每条边)都有一个权值w，可以表示距离或划分成多个集合时的集合编号，问题依然是判断是否有冲突或者有多少条边是假的(冲突)等。

思路：

给N个节点虚拟一个公共的根节点，增加一个数组s[n]记录每个节点到虚拟根节点的距离，把x,y直接的权值w看为(x,y)的相对距离。

Union(x,y,w)时额外把x，y到虚拟根节点的距离(s值)的相对差值设置为w；Find(x)时，压缩路径的同时把当前s值加上之前父节点的s值，得到真实距离。

具体实现：

```
1. 初始化  init()
f[n]数组记录节点的父节点，s[n]数组记录节点到虚拟根节点的距离：    for(int i=0;i<n;i++) {  f[i]=i;  s[i]=0; }

2.  Find(x)
      if(x==f[x])return x;

      int t  = f[x];

      f[x] = Find(f[x]);

      s[x] += s[t];

      // s[[x] %= mod;    若s[x]表示划分成mod个集合时的集合编号等情况时，则需要求余。

      return f[x];

3. Union(x, y,w)
      int fx = Find(x), fy = Find(y);    //此时已经s[x]和s[y]都已经计算为真值。

      if(fx ≠ fy) {

         f [fx] = fy;

         s [fx] = (s[x] - s[y] + w + mod) % mod;

      }
```

4. 解决问题的算法步骤

   初始化后，遍历m对关系：若x,y的父节点不同，则Union(x,y,w)；否则，若x与y的差值为w，则说明正确，继续遍历，不为w时说明出现冲突。

当s[x]只是代表划分为mod个集合时的集合编号时，应该比较s[x]与s[y]的值是否相同，相同时说明出现冲突；不相同时说明之前已经解决了，正确可继续遍历。

拓展：加权并查集主要得赋予并理解s[x]值的意义，较难掌握且应用广泛

```
/*关押罪犯*/
const int gxs = 2;     //模2就只有0,1两个值，分别代表两个不同的集合
const int mod = 2;
int n, m;
int f[maxn], s[maxn];     //f记录父节点(前驱)，s记录到虚拟root点的距离

void init() {
    for (int i = 0; i < maxn; i++) f[i] = i, s[i] = 0;
```

```
}
//查找
int finds(int x) {
    if (x == f[x]) return x;
    int t = f[x];
    f[x] = finds(f[x]);
    s[x] += s[t];    //s[x]原来是与t的相对距离，现在是与root的相对距离
    s[x] %= gxs;    //s值求余后代表所属监狱(二选一)
    return f[x];
}

//新建关系
void unions(int x, int y, int w) {
    int fx = finds(x), fy = finds(y);
    if (fx ≠ fy) {
        f[fy] = fx;
        s[fy] = s[x] - s[y] + w + gxs;    //相对距离设置为w,解决这一对冲突
        s[fy] %= mod;        //求余直接赋予实际意义：所属的mod个集合的编号
    }
}

struct node {
    int a, b;
    ll val;
    bool operator < (const node &a)const {
        return val > a.val;
    }
};

vector<node> que;

int main() {
    cin >> n >> m;
    int a, b;
    ll v;
    for (int i = 0; i < m; i++) {
        cin >> a >> b >> v;
        que.push_back(node{ a,b,v });
    }
    sort(que.begin(), que.end());    //从大到小排序
    init();
    for (int i = 0; i < m; i++) {
        a = que[i].a;
        b = que[i].b;
        v = que[i].val;
        if (finds(a) == finds(b)) {    //在同一个集合就不能直接解决冲突
            if (s[a] == s[b]) {            //若s值相同就说明已经在同一个集合，冲突无法解决
                cout << v << endl;        //因为从大到小遍历，第一个解决不了的关系的val就是答案：最小化的最大冲突值
                return 0;
            }                            //否则说明解决之前的冲突后，当前冲突也被解决。
        }
        else {        //不在一个集合就可以通过设置s值解决冲突
            unions(a, b, 1);
        }
    }
    cout << 0 << endl;
    return 0;
}
```

```
/*d[x]表示x到根的距离*/
void init(){
    for(int i = 1;i≤N;++i)
        f[i] = i,Size[i] = 1;
}

int Get(int x){
    if(x == f[x])return x;
    int root = Get(f[x]);//往上走一层
    d[x] += d[f[x]];//因为是按原顺序，所以我要加上我前面的长度，这才是真正的距离
    //d[x]^=d[f[x]];前缀异或和
    return f[x] = root;
}

void Merge(int x,int y){
    int x = Get(x),y = Get(y);
    f[x] = y;
    d[x] = Size[y];//这里的x已经是原x的根了所以肯定是等于
    Size[y] += Size[x];
}
```

带权并查集求二分图最小环

有 n 个同学（编号为 1 到 n ）正在玩一个信息传递的游戏。在游戏里每人都有一个固定的信息传递对象，其中，编号为 i 的同学的信息传递对象是编号为 T_i的同学。
游戏开始时，每人都只知道自己的生日。之后每一轮中，所有人会同时将自己当前所知的生日信息告诉各自的信息传递对象（注意：可能有人可以从若干人那里获取信息， 但是每人只会把信息告诉一个人，即自己的信息传递对象）。当有人从别人口中得知自 己的生日时，游戏结束。请问该游戏一共可以进行几轮?

```
#include<cstdio>
#include<iostream>
using namespace std;
int f[200002],d[200002],n,minn,last;    //f保存祖先节点，d保存到其祖先节点的路径长。
int fa(int x)
{
    if (f[x]≠x)                         //查找时沿途更新祖先节点和路径长。
    {
        int last=f[x];                  //记录父节点（会在递归中被更新）。
        f[x]=fa(f[x]);                  //更新祖先节点。
        d[x]+=d[last];                  //更新路径长（原来连在父节点上）。
    }
    return f[x];
}
void check(int a,int b)
{
    int x=fa(a),y=fa(b);                //查找祖先节点。
    if (x≠y) {f[x]=y; d[a]=d[b]+1;}     //若不相连，则连接两点，更新父节点和路径长。
    else minn=min(minn,d[a]+d[b]+1);    //若已连接，则更新最小环长度。
    return;
}
int main()
{
    int i,t;
    scanf("%d",&n);
    for (i=1;i≤n;i++) f[i]=i;           //祖先节点初始化为自己，路径长为0。
    minn=0x7777777;
    for (i=1;i≤n;i++)
    {
        scanf("%d",&t);
```

```
31          check(i,t);                    //检查当前两点是否已有边相连接。
32      }
33      printf("%d",minn);
34      return 0;
35  }
```

2.6 扩展域

使用情景：

n个点有m对关系，把n个节点放入两个集合里，要求每对存在关系的两个节点不能放在同一个集合。问能否成功完成?

思路：

把每个节点扩展为两个节点（一正一反），若a与b不能在一起(在同一个集合)，则把a的正节点与b的反节点放一起，把b的正节点与a的反节点放一起，这样就解决了a与b的冲突。若发现a的正节点与b的正节点已经在一起，那么说明之前的某对关系与(a,b)这对关系冲突，不可能同时满足，即不能成功完成整个操作。

具体实现：

```
1. 初始化 init()

    n个点，每个点扩展为两个点(一正一反)，则需要一个容量为2*n的数组f[n]，值全部初始化为自己即可：for(int i=0;i<2*n;i++) f[i]=i;

    （注意初始编号，若编号为[1,n]，则初始化应该为：for(int i=1;i≤2*n;i++) f[i]=i;）

            一个点x的正点编号为x，反点编号为x+n（这样每个点的反点都是+n，规范、可读性强、不重复、易于理解）。

2. Find(x)和Union(x, y)不需要修改，含义和实现不变。
```

3. 解决问题的算法步骤

1）初始化2*n个节点的初始父节点，即它本身。

2）遍历m对关系，对每对(a,b)，先找到a和b的父节点，若相等则说明(a,b)的关系与之前的关系有冲突，不能同时解决，则得到结果：不能完成整个操作。

否则执行：Union(a, b+n), Union(b, a+n).　　(这时已经Find过了，直接执行f [fx] = fy这一句就等效与Union(x, y))

3）若m对关系都成功解决，则得到结果：能够完成整个操作。

拓展：

由于扩展域会直接使数组容量翻倍，所有一般只解决这种“二分”问题，只扩展为2倍即可。

优点在于：结构简单，并查集的操作也不需要做改变，非常易于理解。　　缺点显然就是：需要额外存储空间。

```
/*关押罪犯*/
struct edge {
    int a, b, c;
}e[maxm];

bool cmp(edge a, edge b) {
    return a.c > b.c;
}

int f[2 * maxn];
int Find(int x) {
    if (x == f[x])return x;
    return f[x] = Find(f[x]);
}

```

```cpp
int main() {
    int n, m;
    scanf("%d%d", &n, &m);
    for (int i = 0; i < m; i++)
        scanf("%d%d%d", &e[i].a, &e[i].b, &e[i].c);
    sort(e, e + m, cmp);    //按仇恨值从大到小排序
    for (int i = 1; i ≤ 2 * n; i++)f[i] = i;    //初始化并查集

    int i;    //从大到小依次把每对罪犯安排到不同监狱
    for (i = 0; i < m; i++) {
        int a = Find(e[i].a), b = Find(e[i].b);
        if (a == b)break;    //两人的正点已在同一个集合，无法解决，最大冲突出现
        f[a] = Find(e[i].b + n);    //把a和b的反点(敌人)合并
        f[b] = Find(e[i].a + n);    //把b和a的反点(敌人)合并(每个点都有一个正点和反点)
    }
    if (i == m)printf("0");
    else printf("%d", e[i].c);
    return 0;
}
```

2.7 集合中单个元素的删除

n个集合，第 i 个集合里为 $\{i\}$ 现在有 m 个操作。

- 1 x y 把 x 所在集合合并到 y 所在集合中去
- 2 x y把x元素移动到y集合中去。
- 3 x输出 x 元素所在集合中的元素个数和总和。

```cpp
/*给每一个点建一个虚拟点，所有的点都连虚拟点。例如i和i',以i为根的子结点都连i'，这样进行操作2时将i连到j就是将i连到j',互不影响*/
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<cmath>
using namespace std;

const int N = 500007;

int fa[N];
int n, m;
int cnt[N], sum[N];

int find(int x){
    if(fa[x] == x)return x;
    return fa[x] = find(fa[x]);
}

int main(){
    while(scanf("%d%d", &n, &m) ≠ EOF){
        for(int i = 1;i ≤ n; ++ i)
            fa[i] = i + n;//直接初始化成它的虚拟结点
        for(int i = n + 1; i ≤ 2 * n; ++ i)
            fa[i] = i, cnt[i] = 1, sum[i] = i - n;//这里虚拟结点千万别忘了初始化为自己
        int x, y, op;
        while(m -- ){
            scanf("%d", &op);
            if(op == 1){
                scanf("%d%d", &x, &y);
```

```
                int fx = find(x);
                int fy = find(y);
                if(fx ≠ fy){
                    fa[fx] = fy;
                    cnt[fy] += cnt[fx];
                    sum[fy] += sum[fx];
                    cnt[fx] = 0;
                    sum[fx] = 0;
                }
            }
            else if(op == 2){
                scanf("%d%d", &x, &y);
                int fx = find(x);
                int fy = find(y);
                if(fx ≠ fy){
                    cnt[fy] ++ ;
                    sum[fy] += x;
                    cnt[fx] -- ;
                    sum[fx] -= x;
                    fa[x] = fy;
                }
            }
            else {
                scanf("%d", &x);
                int fx = find(x);
                printf("%d %d\n", cnt[fx], sum[fx]);
            }
        }
    }
    return 0;
}
```

∮ 3. 树状数组

3.1 树状数组求逆序对

```
/*若i < j 且a[i] > a[j]，则称a[i]与a[j]构成逆序对*/

/*大体思路就是倒序遍历，每次树状数组里有多少个比自己小的即为自己的逆序对的
个数（因为是倒序，所以当前数组中的数都是比自己排名大的数，满足逆序对的性
质）*/

/*逆序对的个数 == 交换相邻两个数使得整个序列有序的最小交换次数*/

/*注意：这里的交换次数指的是交换两个数算作一次操作。
小朋友排队这道题是交换两个数，两个数各自会算一次交换，
会有一个代价，这时应该先正序求一次每个点前面有多少个
小于他的数，再倒序求一次每个点后面有多少个数小于他，
二者相加才为最终答案
*/
typedef long long ll;
const int N = 5000000;
int n,m;
struct node{
    int vis,id;
    bool operator<(const node &t)const{
        return vis < t.vis;
    }
```

```
}a[N];
int b[N];
int tree[N];
int lowbit(int x){
    return x & (-x);
}

void update(int x,int k){
    for(;x ≤ n;x += lowbit(x))
        tree[x] += k;
}

ll ask(int x){
    ll res = 0;
    for(;x;x -= lowbit(x))
        res += tree[x];
    return res;
}

int main(){
    scanf("%d",&n);
    for(int i = 1;i ≤ n;++i){
        scanf("%d",&a[i].vis);
        a[i].id = i;
    }
    stable_sort(a + 1,a + 1 + n);
    for(int i = 1;i ≤ n;++i)
        b[a[i].id] = i;
    ll ans = 0;
    for(int i = n;i > 0;i -- ){
        update(b[i],1);//一般先加再比
        ans += ask(b[i] - 1);
    }
    printf("%lld\n",ans);
    return 0;
}
/*
小朋友排队
n 个数。将这个数列变成不下降的序列，每个数都有另一个权值，开始为0。
如果某个数第一次交换，则该数权值加1，如果第二次要求他交换，则该数权值再加2依次类推。求最终最小各数权值和

int main()
{
    scanf("%d", &n);
    for(int i = 1; i ≤ n; ++ i){//前面比他大的
        scanf("%d", &a[i]);
        a[i] ++ ;

        add(a[i], 1);
        s[i] = i - query(a[i]);
        //s[i] = query(N + 1) - query(a[i]);
    }
    memset(tr, 0, sizeof tr);

    for(int i = n; i > 0; -- i){//后面比他小的
        add(a[i], 1);
        s[i] += query(a[i] - 1);


    }
```

```
    ll res = 0;
    for(int i = 1; i ≤ n; ++ i){
        res += 1ll * s[i] * (s[i] + 1) / 2;
    }
    cout << res << endl;
    return 0;
}

*/
```

3.2 区间修改、单点求值

```
int n,m;
int a[N];
ll tree[N];

int lowbit(int x){
    return  x & (-x);
}

void update(int x,int val){
    for(;x ≤ n;x += lowbit(x))
        tree[x] += val;
}

ll ask(int x){
    ll res = 0;
    for(;x;x -= lowbit(x))
        res += tree[x];
    return res;
}

int main(){
    scanf("%d%d",&n,&m);
    for(int i = 1;i ≤ n;++i)
        scanf("%d",&a[i]),update(i,a[i] - a[i - 1]);
    for(int i = 1;i ≤ m;++i){
        char s[2];
        int l,r,d;
        scanf("%s%d",s,&l);
        if(*s == 'C'){
            scanf("%d%d",&r,&d);
            update(l,d),update(r + 1,-d);
        }
        else printf("%lld\n",ask(l));
    }
    return 0;
}

```

3.3 区间修改、区间求和

```
int n,m;
int a[N];
ll tree[2][N],sum[N];

int lowbit(int x){
```

```
    return x & (-x);
}

ll ask(int k,int x){
    ll res = 0;
    for(;x;x -= lowbit(x))
        res += tree[k][x];
    return res;
}

void add(int k,int x,int val){
    for(;x ≤ n;x += lowbit(x))
        tree[k][x] += val;
}

int main(){
    scanf("%d%d",&n,&m);
    for(int i = 1;i ≤ n;++i){
        scanf("%d",&a[i]);
        sum[i] = sum[i - 1] + a[i];
    }
    while(m--){
        char op[2];
        int l,r,d;
        scanf("%s%d%d",op,&l,&r);
        if(op[0] == 'C'){
            scanf("%d",&d);
            add(0,l,d);
            add(0,r + 1,-d);

            add(1,l,l * d);
            add(1,r + 1,-(r + 1) * d);
        }
        else {
            ll ans = sum[r] + (r + 1) * ask(0,r) - ask(1,r);
            ans -= sum[l - 1] + l * ask(0,l - 1) - ask(1,l - 1);
            printf("%lld\n",ans);
        }
    }
    return 0;
}

```

3.4 单点修改、区间求最值

```
int n, m;
int tr[N];
int a[N];

int lowbit(int x){
    return x & -x;
}

void modify(int x, int v){
    for(;x ≤ N;x += lowbit(x)){
        tr[x] = max(tr[x], v);
    }
}

```

```
int query(int l, int r){
    int res = -INF;
    for(;r ≥ l && r - l + 1 ≥ lowbit(r); r -= lowbit(r)){
        res = max(res, tr[r]);
    }
    while(r ≥ l){
        res = max(res, a[r]);
        if(r - l + 1 < lowbit(r))r -- ;
        else {
            res = max(res, tr[r]);
            r -= lowbit(r);
        }
    }
    return res;
}

int main(){
    scanf("%d%d", &n, &m);
    for(int i = 1;i ≤ n;++ i)
        scanf("%d", &a[i]), modify(i, a[i]);
    while(m --){
        int l ,r;
        scanf("%d%d",&l, &r);
        printf("%d\n", query(l, r));
    }
    return 0;
}
```

3.5 实时求出剩余的数中的第k小的数（树状数组+二分）

```
/*AcWing 244. 谜一样的牛 《算法竞赛进阶指南》P209*/
int n,m;
int a[N];
int ans[N];
int tree[N];

int lowbit(int x){
    return x & -x;
}

void update(int x,int val){
    for(;x ≤ n;x += lowbit(x))
        tree[x] += val;
}

int ask(int x){
    int res = 0;
    for(;x;x -= lowbit(x))
        res += tree[x];
    return res;
}

int main(){
    scanf("%d",&n);

    for(int i = 2;i ≤ n;++i)
        scanf("%d",&a[i]);
    for(int i = 1;i ≤ n;++i)
        tree[i] = lowbit(i);
```

```
        //update(i,1);
    for(int i = n;i;i -- ){
        int k = a[i] + 1;//前面有k个，自己的答案就是k+1
        int l = 1,r = n;
        while(l < r){
                int mid = (l + r) >> 1;
            if(ask(mid) ≥ k)r = mid;
            else l = mid + 1;
        }
        ans[i] = r;
        update(r,-1);
    }
    for(int i = 1;i ≤ n;++i)
        printf("%d\n",ans[i]);
    return 0;
}

```

3.6 离线树状数组（查询区间数的种类）

> 给定一个数列，询问某个区间内不同的数有多少个。 $(1 \leq n, m \leq 10^6)$

```
struct node
{
    int l, r;
    int pos;
    bool operator<(const node& t)const {
        return r < t.r;
    }
}ask[N];
//树状数组维护的是区间颜色的种类
//数据范围大，所以我们进行离线操作，按照查询区间的r排序
//一个区间一个区间地进行add操作，
//每一个颜色我们都只存一次，所以我们如果遇见相同颜色的就把
//之前的add的那个位置-1，再在新的位置+1
//保证同一个颜色只会有一个为1，也就是只对答案贡献一次
int tr[N];

int lowbit(int x)
{
    return x & -x;
}

void add(int x, int k)
{
    for(;x ≤ N; x += lowbit(x))
        tr[x] += k;
}

int query(int x){
    int ans = 0;
    for(; x > 0; x -= lowbit(x))
        ans += tr[x];
    return ans;
}

int a[N];
int n, m;
```

```
int vis[N];
int nex[N];
int ans[N];

int main()
{
    scanf("%d", &n);
    for(int i = 1; i ≤ n; ++ i)
        scanf("%d", &a[i]);
    scanf("%d", &m);
    for(int i =1; i ≤ m; ++ i){
        int x, y;
        scanf("%d%d", &ask[i].l, &ask[i].r);
        ask[i].pos = i;
    }
    sort(ask + 1, ask + 1 + m);
    int Next = 1;
    for(int i = 1; i ≤ m; ++ i){
        for(int j = Next; j ≤ ask[i].r; ++ j){
            if(vis[a[j]])
                add(vis[a[j]], -1);//删掉之前的
            add(j, 1);
            //表示出现了一个新的数（哪怕原来有也删掉了）
            vis[a[j]] = j;
        }
        Next = ask[i].r + 1;//下一个区间，之前的已经不用在更新了，以此来降低复杂度
        ans[ask[i].pos] = query(ask[i].r) - query(ask[i].l - 1);
    }
    for(int i = 1; i ≤ m; ++ i)
        printf("%d\n", ans[i]);
    return 0;
}


```

$\oint$ 3.7 二维树状数组

3.7.1 单点修改，区间查询

> 一个矩阵，初始化为全0。有以下操作：
>
> 1）将 (x,y) 元素加 a。x为行坐标，y为列坐标。
>
> 2）求矩阵 [X,Y] 所有元素的和。该矩阵左上角为[l,b]，右下角为[r,t]。

```
int c[MAXN][MAXN],s;
int lowbit(int x){
    return x&-x;
}

void Add(int x,int y,int a)    //加数
{
    int i=x;
    while(i≤s){    //行
        int j=y;
        while(j≤s){    //列
            c[i][j]+=a;
            j+=lowbit(j);
```

```
        }
        i+=lowbit(i);
    }
}

int Sum(int l,int r,int b,int t)    //求和
{
    l--,b--;
    int suml=0,sumr=0;
    //求行矩阵和，l以上矩阵
    while(l≥1){
        int i=b,j=t;
        int sumb=0,sumt=0;
        //求列矩阵和
        while(i≥1){
            sumb+=c[l][i];
            i-=lowbit(i);
        }
        while(j≥1){
            sumt+=c[l][j];
            j-=lowbit(j);
        }
        suml+=sumt-sumb;
        l-=lowbit(l);
    }
    //求行矩阵和，r以上矩阵
    while(r≥1){
        int i=b,j=t;
        int sumb=0,sumt=0;
        //求列矩阵和
        while(i≥1){
            sumb+=c[r][i];
            i-=lowbit(i);
        }
        while(j≥1){
            sumt+=c[r][j];
            j-=lowbit(j);
        }
        sumr+=sumt-sumb;
        r-=lowbit(r);
    }
    return sumr-suml;
}

int main()
{
    int cmd,x,y,a,l,r,b,t;

    while(scanf("%d",&cmd)≠EOF){
        switch(cmd){
        case 0:    //初始化矩阵
            scanf("%d",&s);
            memset(c,0,sizeof(c));
            break;

        case 1:    //加数
            scanf("%d%d%d",&x,&y,&a);
            Add(x+1,y+1,a);
            break;

```

```
        case 2:     //求矩阵和
            scanf("%d%d%d%d",&l,&b,&r,&t);
            printf("%d\n",Sum(l+1,r+1,b+1,t+1));
            break;

        case 3:     //退出程序
            return 0;
        default:
            break;
        }
    }
    return 0;
}
```

3.7.2 区间修改，单点查询

```
struct BIT{
    LL C[N][N];
    inline void add(Re x,Re y,Re v){
        while(x≤n){Re j=y;while(j≤m)C[x][j]+=v,j+=j&-j;x+=x&-x;}
    }
    inline void add(Re x1,Re y1,Re x2,Re y2,Re z){
        add(x1,y1,z),add(x1,y2+1,-z),add(x2+1,y1,-z),add(x2+1,y2+1,z);
    }
    inline LL ask(Re x,Re y){
        LL ans=0;
        while(x){Re j=y;while(j)ans+=C[x][j],j-=j&-j;x-=x&-x;}
        return ans;
    }
}TR;
int main(){
    in(n),in(m),in(T);
    while(T--){
        in(op),in(x1),in(y1);
        if(op<2)in(x2),in(y2),in(x),TR.add(x1,y1,x2,y2,x);
        else printf("%lld\n",TR.ask(x1,y1));
    }
}
```

3.7.3 区间修改，区间查询

```
void add(int x,int y,int v)
{
    for(int i=x;i≤n;i+=lowbit(i))
        for(int j=y;j≤m;j+=lowbit(j))
        {
            t1[i][j]+=v;
            t2[i][j]+=v*x;
            t3[i][j]+=v*y;
            t4[i][j]+=v*x*y;
        }
}
void real_add(int x1,int y1,int x2,int y2,int v)
{
    add(x1,y1,v);
    add(x1,y2+1,-v);
    add(x2+1,y1,-v);
```

```
    add(x2+1,y2+1,v);
}
int ask(int x, int y)
{
    int res=0;
    for(int i=x;i;i-=lowbit(i))
        for(int j=y;j;j-=lowbit(j))
            res+=(x+1)*(y+1)*t1[i][j]-(y+1)*t2[i][j]-(x+1)*t3[i][j]+t4[i][j];
    return res;
}
int real_ask(int x1,int y1,int x2,int y2)
{
    return ask(x2,y2)-ask(x2,y1-1)-ask(x1-1,y2)+ask(x1-1,y1-1);
}
```

3.7.4 矩阵取反单点查询

> 题目大意：给定一个矩阵，初始化为0，现在可以进行两种操作，一种是查询某个点的值是 0 还是 1。另一种是让这个矩阵的一个子矩阵内的值取反。C是取反操作，Q是查询操作，给出子矩阵的左上角和右下角。
>
> 首先这题虽然看起来像是一个区间修改，单点查询的题，但是可以转化成单点修改，查询区间和。
>
> 首先考虑一维的情况，我要一段区间取反，假设是 [l, r]。那么我只需要book[l]+1，book[r+1]+1，假设查询 k 的时候，只需要查询前 k 的和 mod 2 的结果即可。
> 然后这种方法可以推广到二维，不过这里要用一下容斥原理。假设修改的子矩阵左上角和右下角分别为 `x1 y1 x2 y2`，首先 `book[x1][y1]+1`，`book[x2][y2]+1`，不过这时要 `book[x1][y2+1]+1`，`book[x2+1][y1]+1`，也可以减去 `1`，因为最后是按奇偶来判断是 `1` 还是 `0`.
>
> 可以用二维树状数组实现

```
const int N = 1000 + 10;
int c[N][N], n;
int lowbit(int x)
{
    return x & -x;
}
void update(int x, int y)
{
    for(int i = x; i ≤ n; i += lowbit(i))
    {
        for(int j = y; j ≤ n; j += lowbit(j))
        {
            c[i][j] += 1;
        }
    }
}
int query(int x, int y)
{
    int res = 0;
    for(int i = x; i > 0; i -= lowbit(i))
    {
        for(int j = y; j > 0; j -= lowbit(j))
        {
            res += c[i][j];
        }
    }
    return res;
}
int main()
{
    int T;
```

```
    scanf("%d", &T);
    while(T --)
    {
        int t;
        scanf("%d%d", &n, &t);
        memset(c, 0, sizeof(c));
        for(int i = 1; i ≤ t; i ++)
        {
            char Q[2];
            cin >> Q;
            if(Q[0] == 'C')
            {
                int x1, y1, x2, y2;
                scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
                update(x1, y1);
                update(x1, y2+1);
                update(x2+1, y2+1);
                update(x2+1, y1);
            }
            if(Q[0] == 'Q')
            {
                int x, y;
                scanf("%d%d", &x, &y);
                int ans = query(x,y)%2;
                printf("%d\n", ans);
            }
        }
        if(T > 0)
        {
            printf("\n");
        }
    }
    return 0;
}


```

$\oint$ 4. 线段树

4.1 懒惰标记

> 一行序列，每次操作把一个区间里的每个数都加上一个数，或者查询一个区间的和。

```
typedef long long ll;
const int N = 500010;
int n, m;
int a[N];
struct Tree{
    int l, r;
    ll sum;
    ll lz;
}tr[N * 4];

void pushup(int p){
    tr[p].sum = tr[p << 1].sum + tr[p << 1 | 1].sum;
}

```

```
void pushdown(int p){
    auto &root = tr[p], &left = tr[p << 1], &right = tr[p << 1 | 1];
    if(root.lz){
        left.lz += root.lz;
        left.sum += (ll)(left.r - left.l + 1) * root.lz;
        right.lz += root.lz;
        right.sum += (ll)(right.r - right.l + 1) * root.lz;
        root.lz = 0;
    }
}

void build(int p, int l, int r){
    tr[p] = {l, r};
    if(l == r){
        tr[p] = {l, r, a[r], 0};
        return ;
    }
    int mid = l + r >> 1;
    build(p << 1, l, mid);
    build(p << 1 | 1, mid + 1, r);
    pushup(p);
}

void modify(int p, int l, int r, int v){
    if(tr[p].l ≥ l && tr[p].r ≤ r){
        tr[p].sum += (tr[p].r - tr[p].l + 1) * v;
        tr[p].lz += v;
        return ;
    }
    pushdown(p);
    int mid = tr[p].l + tr[p].r >> 1;
    if(l ≤ mid)modify(p << 1, l, r, v);
    if(r > mid)modify(p << 1 | 1, l, r, v);
    pushup(p);
}

ll query(int p, int l, int r){
    if(tr[p].l ≥ l && tr[p].r ≤ r)
        return tr[p].sum;
    pushdown(p);//要在前面pushdown!
    int mid = tr[p].l + tr[p].r >> 1;
    ll res = 0;
    if(l ≤ mid)res = query(p << 1, l, r);
    if(r > mid)res += query(p << 1 | 1, l, r);
    return res;
}

int main(){
    scanf("%d%d", &n, &m);
    for(int i = 1; i ≤ n ;++ i)scanf("%d", &a[i]);
    build(1, 1, n);
    while(m -- ){
        char op[2];
        int l, r, d;
        scanf("%s%d%d", op, &l, &r);
        if(*op == 'C'){
            scanf("%d", &d);
            modify(1, l, r, d);
        }
        else printf("%lld\n", query(1, l, r));
```

```
    }
    return 0;
}

```

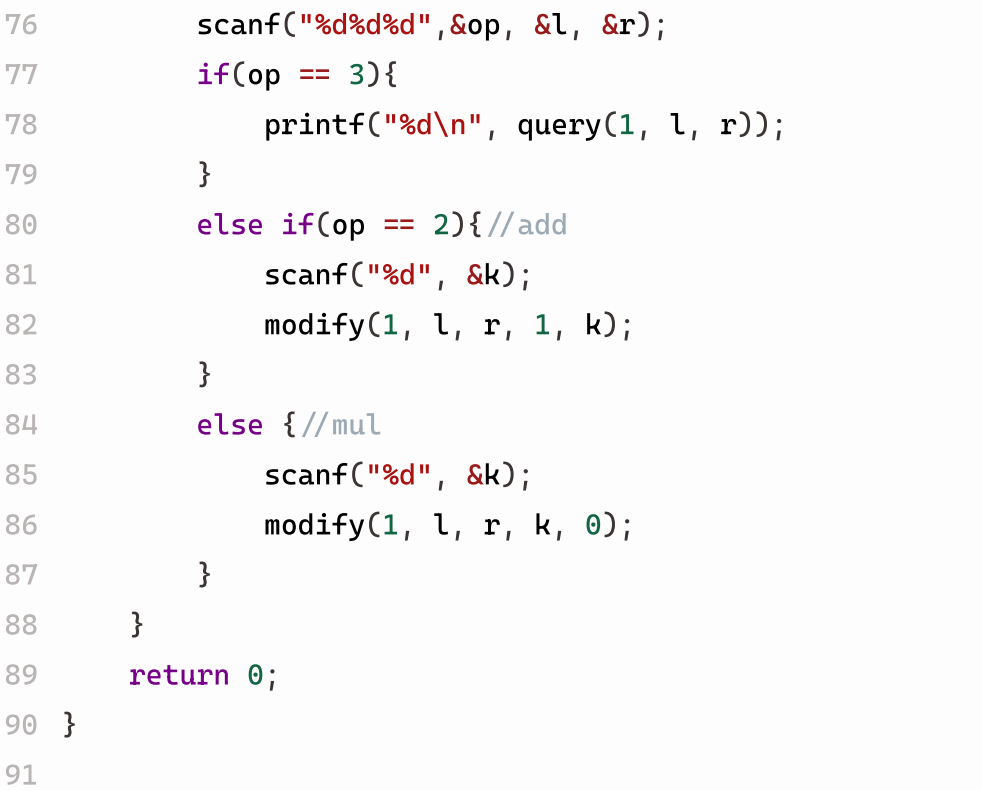
> 给你一个序列，支持三种操作：一个区间上所有的数乘以k，一个区间上所有的数加上k，求区间和膜掉mod。

```
typedef long long ll;
const int N = 500007,INF = 0x3f3f3f3f;
int n, m, mod;
int a[N];

struct Tree{
    int l, r;
    int sum;
    int mul, add;
}tr[N * 4];

inline void eval(Tree &t, int mul, int add){//先乘后加//evaluate v.评估
    t.sum = ((ll)t.sum * mul + (ll)(t.r - t.l + 1) * add) % mod;
    t.mul = ((ll)t.mul * mul) % mod;
    t.add = ((ll)t.add * mul + add) % mod;
}

inline void pushdown(int p){
    eval(tr[p << 1], tr[p].mul, tr[p].add);
    eval(tr[p << 1 | 1], tr[p].mul, tr[p].add);
    tr[p].mul = 1, tr[p].add = 0;
}

inline void pushup(int p){
    tr[p].sum = (tr[p << 1].sum + tr[p << 1 | 1].sum) % mod;
}

void build(int p, int l, int r){
    if(l == r){
        tr[p] = {l, r, a[r], 1, 0};
        return ;
    }
    tr[p] = {l, r, 0, 1, 0};
    int mid = l + r >> 1;
    build(p << 1, l, mid);
    build(p << 1 | 1, mid + 1, r);
    pushup(p);
}

void modify(int p, int l, int r, int mul, int add){
    if(tr[p].l ≥ l && tr[p].r ≤ r){
        eval(tr[p], mul, add);
        return ;
    }
    pushdown(p);
    int mid = tr[p].l + tr[p].r >> 1;
    if(l ≤ mid)modify(p << 1, l, r, mul, add);
    if(r > mid)modify(p << 1 | 1, l, r, mul, add);
    pushup(p);
}


int query(int p, int l, int r){
```

```
    if(tr[p].l ≥ l && tr[p].r ≤ r){
        return tr[p].sum;
    }
    pushdown(p);
    int mid = tr[p].l + tr[p].r >> 1;
    int res = 0;
    if(l ≤ mid)res = query(p << 1, l, r);
    if(r > mid)res = (res + query(p << 1 | 1, l, r)) % mod;
    return res;
}

int main(){

    scanf("%d%d", &n, &mod);
    for(int i = 1;i ≤ n; ++ i)
        scanf("%d", &a[i]);

    build(1, 1, n);

    scanf("%d",&m);
    for(int i = 1;i ≤ m; ++ i){
        int op, l, r, k;
        scanf("%d%d%d",&op, &l, &r);
        if(op == 3){
            printf("%d\n", query(1, l, r));
        }
        else if(op == 2){//add
            scanf("%d", &k);
            modify(1, l, r, 1, k);
        }
        else {//mul
            scanf("%d", &k);
            modify(1, l, r, k, 0);
        }
    }
    return 0;
}

```

4.2 扫描线法(面积)

求有重叠的多个矩形的总面积



```cpp
/*POJ 1151 Atlantis*/
typedef long long ll;
typedef pair<int,int> PII;
const int INF = 0x3f3f3f3f;
const int N = 1e5+7;
const int M = 2007;

struct line{
    double x,y1,y2;
    int f;
    bool operator<(const line &t)const{
        return x < t.x;
    }
}a[N<<1];

struct SegmentTree{
    int l,r;
    double cnt;
    double len;
    #define tl(x) tree[x].l
    #define tr(x) tree[x].r
    #define tcnt(x) tree[x].cnt
    #define tlen(x) tree[x].len
}tree[N<<3];

int n,T,tot,val[N<<1][2];
double raw[N<<1],ans;

inline void build(int p,int l,int r){
    tl(p) = l;
    tr(p) = r;
    if(l == r)return ;
    int mid = (l+r) / 2;
    build(ls,l,mid);
    build(rs,mid+1,r);
}

inline void push_up(int p){
    if(tcnt(p) > 0)tlen(p) = raw[tr(p) + 1] - raw[tl(p)];//raw存的是原y坐标
    else if(tl(p) == tr(p))tlen(p) = 0;
    else tlen(p) = tlen(ls) + tlen(rs);
}

void updata(int p,int l,int r,int v){
    if(l ≤ tl(p) && r ≥ tr(p)){
        tcnt(p) += v;//左或者右，+1或者-1
        push_up(p);//只需要往上更新，我们只用根节点的len
        return ;
    }
    int mid = (tl(p) + tr(p)) / 2;
    if(l ≤ mid)updata(ls,l,r,v);
    if(r > mid)updata(rs,l,r,v);
    push_up(p);
}

int main()
{
    while(cin>>n && n){
        ans = tot = 0;
        over(i,1,n){
```

```
            double x1,x2,y1,y2;
            scanf("%lf%lf%lf%lf",&x1,&y1,&x2,&y2);
            a[2*i-1] = (line){x1,y1,y2,1};//奇记左
            a[2*i] = (line){x2,y1,y2,-1};//偶记右
            raw[++tot] = y1;//离散化
            raw[++tot] = y2;
        }
        sort(a+1,a+1+2*n);//按照x排序
        sort(raw+1,raw+1+tot);
        tot = unique(raw+1,raw+1+tot) - (raw+1);//注意+1因为从1开始的
        over(i,1,2*n){//离散化
            val[i][0] = lower_bound(raw+1,raw+1+tot,a[i].y1) - raw;
            val[i][1] = lower_bound(raw+1,raw+1+tot,a[i].y2) - raw;
        }
        build(1,1,tot);
        over(i,1,2*n){//一共2n个x
            updata(1,val[i][0],val[i][1] - 1,a[i].f);
            ans += (a[i+1].x - a[i].x) * tlen(1);
        }
        printf("Test case #%d\nTotal explored area: %.2f\n\n",++T,ans);
    }
    return 0;
}
```

4.3 扫描线法（周长）

> 题目大意：N个矩形，求矩形周长的并。
>
> 解题思路：利用到线段数区间合并，记录有多少个连续块，还用到区间修改，每次对于一条边，除了要计算竖直方向，还要计算水平方向，而水平方向是修改后的增减量。

```
#define mem(a,x) memset(a,x,sizeof(a))
typedef long long ll;
const int N = 5007;
const int X = 20007;
const int inf = 1<<29;
struct Edge//扫描线
{
    int l,r;//左右端点的横坐标
    int h;//这条线的高度，即纵坐标
    int f;//标记这条边是上边(-1)还是下边(1)
}e[N*2];
bool cmp(Edge a,Edge b)
{

    return a.h < b.h;//高度从小到大排，扫描线自下而上扫
}
struct Node
{
    int l,r;//该节点代表的线段的左右端点坐标
    int len;//这个区间被覆盖的长度
    int s;//表示这个区间被重复覆盖了几次
    bool lc,rc;//表示这个节点左右两个端点是否被覆盖（0表示没有被覆盖，1表示有被覆盖）
    int num;//这个区间有多少条线段（这个区间被多少条线段覆盖）
    //len用来计算横线 num用来计算竖线
}q[4*X];
#define ls i<<1
#define rs i<<1|1
#define m(i) ((q[i].l + q[i].r)>>1)
void pushup(int i)//区间合并
```

```
{
    if (q[i].s)//整个区间被覆盖
    {
        q[i].len = q[i].r - q[i].l + 1;
        q[i].lc = q[i].rc = 1;
        q[i].num = 1;
    }
    else if (q[i].l == q[i].r)//这是一个点而不是一条线段
    {
        q[i].len = 0;
        q[i].lc = q[i].rc = 0;
        q[i].num = 0;
    }
    else //是一条没有整个区间被覆盖的线段，合并左右子的信息
    {
        q[i].len = q[ls].len + q[rs].len ;//长度之和
        q[i].lc = q[ls].lc;q[i].rc = q[rs].rc;//和左儿子共左端点，和右儿子共右端点
        q[i].num = q[ls].num + q[rs].num  - (q[ls].rc&q[rs].lc);
        //如果左子的右端点和右子的左端点都被覆盖了
    }
}
void build (int i,int l,int r)
{
    q[i].l = l,q[i].r = r;
    q[i].s = q[i].len = 0;
    q[i].lc = q[i].rc = q[i].num = 0;
    if (l == r) return;
    int mid = m(i);
    build(ls,l,mid);
    build(rs,mid+1,r);
}
void update(int i,int l,int r,int xx)
{
    if (l == q[i].l && q[i].r == r)
    {
        q[i].s += xx;
        pushup(i);
        return;
    }
    int mid = m(i);
    if (r ≤ mid) update(ls,l,r,xx);
    else if (l > mid) update(rs,l,r,xx);
    else
    {
        update(ls,l,mid,xx);
        update(rs,mid+1,r,xx);
    }
    pushup(i);
}
int main()
{
    int n;
    while (cin>>n)
    {
        int x1,x2,y1,y2,mx = -inf,mn = inf;
        int tot = 0;
        for (int i = 0;i < n;++i)
        {
            scanf("%d %d %d %d",&x1,&y1,&x2,&y2);
            mx = max(mx,max(x1,x2));
```

```
            mn = min(mn,min(x1,x2));
            Edge & t1 = e[tot];Edge & t2 = e[tot+1];
            t1.l = t2.l = x1,t1.r = t2.r = x2;
            t1.h = y1;t1.f = 1;
            t2.h = y2;t2.f = -1;
            tot += 2;
        }
        sort(e,e+tot,cmp);
        //数据小可以不离散化
        int ans = 0;//计算周长
        int last = 0;//保存上一次的总区间的被覆盖的长度
        build(1,mn,mx-1);
        //每两条横线之间才会有竖线
        for (int i = 0;i < tot;++i)
        {
            update(1,e[i].l,e[i].r-1,e[i].f);//根据扫描线更新
            //计算周长
            //横线：现在这次总区间被覆盖的程度和上一次总区间被覆盖的长度之差的绝对值
            ans += abs(q[1].len - last);
            //竖线：[下一条横线的高度-现在这条横线的高度]*2*num
            ans += (e[i+1].h - e[i].h)*2*q[1].num;
            last = q[1].len;//每次都要更新上一次总区间覆盖的长度
        }
        printf("%d\n",ans);
    }
    return 0;
}
```

4.4 二维线段树 - 矩形树

线段树是处理区间问题的，二维线段树就是处理平面问题

> 题意：给你一个n*n的矩阵，有m个询问，q x1 y1 x2 y2 查询这个子矩阵的最大值和最小值，c x y v是将x，y点的值改成c。
> 分析：二维线段树裸题，留作板子。二维线段树就是在一棵线段树的每一个节点，都保存着另一棵线段树的根节点编号。

```
#include<algorithm>
#include<cstdio>
using namespace std;
const int N = 1500;
const int inf=1e8;

int n,q,mx[N][N],mi[N][N];
int fg,xo,mians,mxans;

void upd2(int l,int r,int o,int y,int v) {
    if(l==r) {///如果找到了二维下标
        if(fg) mi[xo][o]=v,mx[xo][o]=v;///如果此时正好找到了一维下标，就赋值
        else {///否则更新一维的情况
            mi[xo][o]=min(mi[2*xo][o],mi[2*xo+1][o]);
            mx[xo][o]=max(mx[2*xo][o],mx[2*xo+1][o]);
        }
        return ;
    }
    int m=(l+r)/2;
    if(y≤m) upd2(l,m,2*o,y,v);
    else upd2(m+1,r,2*o+1,y,v);
    mi[xo][o]=min(mi[xo][2*o],mi[xo][2*o+1]);///递归回来更新二维情况
    mx[xo][o]=max(mx[xo][2*o],mx[xo][2*o+1]);
}

```

```cpp
///更新一维，x为要更新的一维下标，y为要更新的二维下标
void upd1(int l,int r,int o,int x,int y,int v) {
    if(l==r) {
        fg=1;///打上标记
        xo=o;
        upd2(1,n,1,y,v);
        return ;
    }
    int m=(l+r)/2;
    if(x≤m) upd1(l,m,2*o,x,y,v);
    else upd1(m+1,r,2*o+1,x,y,v);
    fg=0,xo=o;
    upd2(1,n,1,y,v);
}

void qy2(int l,int r,int o,int pre,int ly,int ry) {
    if(ly≤l&&ry≥r) {
        mians=min(mians,mi[pre][o]);
        mxans=max(mxans,mx[pre][o]);
        return ;
    }
    int m=(l+r)/2;
    if(ly≤m) qy2(l,m,2*o,pre,ly,ry);
    if(ry>m) qy2(m+1,r,2*o+1,pre,ly,ry);
}

void qy1(int l,int r,int o,int lx,int rx,int ly,int ry) {
    if(lx≤l&&rx≥r) {
        qy2(1,n,1,o,ly,ry);
        return ;
    }
    int m=(l+r)/2;
    if(lx≤m) qy1(l,m,2*o,lx,rx,ly,ry);
    if(rx>m) qy1(m+1,r,2*o+1,lx,rx,ly,ry);
}

int main() {
    scanf("%d",&n);
    int v;
    for(int i=1;i≤n;i++)
        for(int j=1;j≤n;j++)
            scanf("%d",&v),upd1(1,n,1,i,j,v);
    scanf("%d",&q);
    while(q--) {
        char op[2];
        scanf("%s",op);
        int x,y,x1,y1;
        if(op[0]=='q') {
            scanf("%d%d%d%d",&x,&y,&x1,&y1);
            mians=inf;
            mxans=0;
            qy1(1,n,1,x,x1,y,y1);
            printf("%d %d\n",mxans,mians);
        }else {
            scanf("%d%d%d",&x,&y,&v);
            upd1(1,n,1,x,y,v);
        }
    }
    return 0;
}
```

4.5 三维线段树 - 空间树

三维线段树

线段树-二叉树，二维线段树-四叉树，三维线段树自然就是八叉树了，分割的是空间，一般用于三维计算几何，当然也不一定用在实质的空间内的问题。

比如需要找出身高、体重、年龄在一定范围内并且颜值最高的女子，就可以用三维线段树（三维空间最值问题）

4.6 线段树上二分（查询某个点向一个方向连续值相同区间）

线段树上二分涉及查询某个点向一个方向连续值相同区间的时候，不需要在外面套一个二分。只需要讨论当前区间 [L,R]是否完整包含查询区间，然后进行转移。

```
int query_same(int o, int l, int r, int ql) { // 假设是向右查询均为 1 的区间
    maintain(o, l, r);
    if(ql ≤ l) { // 完全包含，直接二分，能返回马上返回
        if(l == r) return andv[o] == FULL ? l : ql - 1;

        if(andv[o] == FULL) return r;
        else {
            int ret = ql - 1, mid = (l + r) / 2;
            pushdown(o); // !
            maintain(lc(o), l, mid), maintain(rc(o), mid+1, r);
            if(andv[lc(o)] == FULL) {
                ret = max(ret, mid);
                ret = max(ret, query_same(rc(o), mid + 1, r, ql));
            } else {
                ret = max(ret, query_same(lc(o), l, mid, ql));
            }
            return ret;
        }
    } else { // 不完全包含，找往哪边走
        int ret = ql - 1, mid = (l + r) / 2;
        pushdown(o); //!

        if(ql ≤ mid) {
            ret = max(ret, query_same(lc(o), l, r, ql));
        } else maintain(lc(o), l, mid); // !
        if(ql > mid || ret == mid) {
            ret = max(ret, query_same(rc(o), mid + 1, r, ql));
        } else maintain(rc(o), mid + 1, r); // !

        return ret;
    }
}
```

4.7 离散数学应用（线段树）开闭区间表示

题目描述

展开

受校门外的树这道经典问题的启发，A君根据基本的离散数学的知识，抽象出 5 种运算维护集合 S（S 初始为空）并最终输出 S。现在，请你完成这道校门外的树之难度增强版——校门外的区间。

五种运算如下：

- `U T`：$S = S \cup T$
- `I T`：$S = S \cap T$
- `D T`：$S = S - T$
- `C T`：$S = T - S$
- `S T`：$S = S \oplus T$

集合的基本运算操作定义如下：

- $A \cup B$：$\{x|x \in A \vee x \in B\}$
- $A \cap B$：$\{x|x \in A \wedge x \in B\}$
- $A - B$：$\{x|x \in A \wedge x \notin B\}$
- $A \oplus B$：$(A - B) \cup (B - A)$

转化成线段树的区间覆盖+翻转问题

用线段树维护当前区间

并且维护两个标记cov, tag，分别代表覆盖和翻转的标记

- U：$A \cup B$ B区间覆盖1
- I：$A \cap B$ B区间的补集覆盖0
- D：$A - B$ B区间覆盖0
- C：$B - A$ 全集范围内01翻转，转到II操作
- S：$A \oplus B$ B区间范围内01翻转

细节处理：输入输出细节
开闭用2方法处理
覆盖和翻转的优先级

处理开闭区间的方法：2，判断边界是奇数还是偶数

开闭区间操作：

区间端点均为整数，不妨认为（l，r）为（$l+0.5, r-0.5$）
乘2就可以换算成整数区间

```cpp
#include<bits/stdc++.h>
#define ls p<<1
#define rs p<<1|1
using namespace std;
typedef long long ll;

const ll N=132010;
const ll mod=100000000;

struct segment
{
    ll l,r;
    ll cov;//覆盖标记
    ll tag;//翻转标记
}tree[N<<2];

ll ans[N],n=N;
```

```

char s[20];

inline void get(ll &l,ll &r)
{
    char c=getchar();
    while(c≠'('&&c≠'[')c=getchar();
    ll w=(c=='(');l=0;
    while(!isdigit(c))c=getchar();
    while(isdigit(c))l=(l<<1)+(l<<3)+(c^48),c=getchar();
    l<<=1;//乘2是为了区分开区间还是闭区间
    l+=w;r=0;//是开区间说明要+1
    while(!isdigit(c))c=getchar();
    while(isdigit(c))r=(r<<1)+(r<<3)+(c^48),c=getchar();
    r<<=1;
    while(c≠')'&&c≠']')c=getchar();
    r-=(c==')');//处理开闭区间//是开区间说明需要-1
}

inline void push_down(ll p)
{
    if(tree[p].cov≠-1)
    {
        tree[ls].cov=tree[rs].cov=tree[p].cov;
        tree[ls].tag=tree[rs].tag=0;//先都初始化为0
        tree[p].cov=-1;
    }
    if(tree[p].tag)
    {
        tree[ls].tag^=1,tree[rs].tag^=1;//相同为0不同为1，把区间翻转
        tree[p].tag=0;
    }
}

inline void build(ll p,ll l,ll r)
{
    tree[p].l=l,tree[p].r=r,tree[p].cov=-1;
    if(l==r)return;
    ll mid=(l+r)>>1;
    build(ls,l,mid),build(rs,mid+1,r);
}

inline void update(ll p,ll l,ll r,ll k)
{
    if(tree[p].l>r||tree[p].r<l||l>r)return;
    if(tree[p].l≥l&&tree[p].r≤r)
    {
        if(k≠2)tree[p].cov=k,tree[p].tag=0;
        else tree[p].tag^=1;//需要的话就翻转
        return;
    }
    push_down(p);
    update(ls,l,r,k),update(rs,l,r,k);
}

inline void query(ll p)
{
    if(tree[p].l==tree[p].r)
    {//如果这个点没有被修改过那它一直都是0，否则就是看它当前被修改成了什么（cov），然后再异或上tag，翻转过后的值
        ans[tree[p].l]=(tree[p].cov==-1)?0:tree[p].cov^tree[p].tag;
```

```
78          return;
79      }
80      push_down(p);
81      query(ls),query(rs);
82  }
83
84  inline void print()
85  {
86      query(1);
87      int flag=0,Empty=0;
88      for(int i=0;i≤n;++i)
89      {
90          if(ans[i]&&!flag)//遍历一遍，找到左边界
91          {
92              flag=Empty=1;
93              if(i&1)printf("(%d,",(i-1)>>1);//如果是奇数说明是开区间
94              else printf("[%d,",i>>1);
95          }
96          if(!ans[i]&&flag)
97          {
98              flag=0;
99              if(i&1)printf("%d]\n",(i-1)>>1);//如果是奇数说明是闭区间
100             else printf("%d)\n",i>>1);
101         }
102     }
103     if(!Empty)
104         puts("empty set");
105 }
106
107 int main()
108 {
109     build(1,0,n);
110     while(scanf("%s",s)≠EOF)//接受第一个字符
111     {
112         ll l,r;
113         get(l,r);
114         if(s[0]=='U')update(1,l,r,1);//U是求并集（区间内全部改为1）
115         else if(s[0]=='I')update(1,0,l-1,0),update(1,r+1,n,0);//I是求交集（区间以外全部为0，不相交就全为0嘛）
116         else if(s[0]=='D')update(1,l,r,0);//D是减去该区间，所以区间内全为0
117         else if(s[0]=='C')update(1,0,n,2),update(1,0,l-1,0),update(1,r+1,n,0);//全集范围内01翻转，并将区间
    的补集覆盖0
118         else update(1,l,r,2);//区间范围内01翻转
119     }
120     print();
121     return 0;
122 }
123
```

4.8 动态开点+线段树合并优化空间

> 首先村落里的一共有 n 座房屋，并形成一个树状结构。然后救济粮分 m 次发放，每次选择两个房屋 (x, y)，然后对于 x 到 y 的路径上(含 x 和 y)每座房子里发放一袋 z 类型的救济粮。然后深绘里想知道，当所有的救济粮发放完毕后，每座房子里存放的最多的是哪种救济粮。

本题是一个简单的树上差分的模板题，本来使用普通的线段树维护最大值即可。但是本题的空间只有128MB，直接按照全部的值域开一个线段树是存不下的，所以我们动态开点，每一个点（权值）开一颗线段树，最后将他们合并查询答案。每个节点维护一棵权值线段树，下标为救济粮种类，区间维护数量最多的救济粮编号（下标）。所以每个节点答案即为$tr[rot[x]]$。

- 线段树的动态开点要设置一个根数组并且需要预处理使得$root[i]=i$自己是自己的根

- TLE不一定都是因为死循环了，（死循环很可能是因为递归函数没有设置结束条件，遍历图的时候没有判断回环，以及for循环里sb的锅）TLE还有可能是数组开小了（线段树的数组，不是RE，而是TLE）
- 普通的单一线段树要开4倍空间，线段树合并的时候线段树要开50倍！不然就会TLE！

```
//树上对点差分 + 线段树动态开点 + 线段树合并
const int N = 1e5+7;
const int M = 2007;
struct Seg{
    int lc;
    int rc;
    int dat;//题目要求输出最多的类型，所以线段树维护的是最大值
    int pos;//有物品就是记录位置l，没有物品就是0因为最后输出的也是0
}tree[N * 20 * 4];

//离线操作
int fa[N][20],deep[N];//lca专用
int root[N],ans[N];
int ver[N<<1],nex[N<<1],head[N];//前向星专用
int X[N],Y[N],Z[N],val[N];//存点和点的离散化
int T,n,m,tot,t,num,cnt;
queue<int>q;//bfs专用

void add(int u,int v){
    ver[++tot] = v;
    nex[tot] = head[u];
    head[u] = tot;
}

void bfs(){//lca预处理
    q.push(1);
    deep[1] = 1;//根节点的深度为1
    while(q.size()){
        int u = q.front();
        q.pop();
        for(int i = head[u];i;i = nex[i]){
            int v = ver[i];
            if(deep[v])continue;
            deep[v] = deep[u] + 1;
            fa[v][0] = u;
            for(int j = 1 ;j ≤ t;++j)
                fa[v][j] = fa[fa[v][j-1]][j-1];
            q.push(v);
        }
    }
}

int lca(int x,int y){
    if(deep[x] > deep[y])swap(x,y);
    for(int i = t ;i ≥ 0;i--)
        if(deep[fa[y][i]] ≥ deep[x])
            y = fa[y][i];
    if(x == y)return x;
    for(int i = t ;i ≥ 0;i--)
        if(fa[x][i] ≠ fa[y][i])
            x = fa[x][i],y = fa[y][i];
    return fa[x][0];
}

//动态开点省空间，所以不需要建树
void Insert(int p,int l,int r,int val,int delta){
```

```
57      if(l == r){
58          tree[p].dat += delta;
59          tree[p].pos = tree[p].dat ? l : 0;
60          return ;
61      }
62      int mid = (l + r) >> 1;
63      if(val ≤ mid){
64          if(!tree[p].lc)tree[p].lc = ++num;
65          Insert(tree[p].lc,l,mid,val,delta);
66      }
67      else {
68          if(!tree[p].rc)tree[p].rc = ++num;
69          Insert(tree[p].rc,mid+1,r,val,delta);
70      }
71      tree[p].dat = max(tree[tree[p].lc].dat,tree[tree[p].rc].dat);
72      tree[p].pos = tree[tree[p].lc].dat ≥ tree[tree[p].rc].dat ? tree[tree[p].lc].pos :
    tree[tree[p].rc].pos;
73  }
74  //每一棵线段树都是一个差分数组的点
75  int Merge(int p,int q,int l,int r){
76      if(!p)return q;
77      if(!q)return p;
78      if(l == r){
79          tree[p].dat += tree[q].dat;
80          tree[p].pos = tree[p].dat ? l : 0;
81          return p;
82      }
83      int mid = (l + r) >> 1;
84      tree[p].lc = Merge(tree[p].lc,tree[q].lc,l,mid);
85      tree[p].rc = Merge(tree[p].rc,tree[q].rc,mid+1,r);
86      tree[p].dat = max(tree[tree[p].lc].dat,tree[tree[p].rc].dat);
87      tree[p].pos = tree[tree[p].lc].dat ≥ tree[tree[p].rc].dat ? tree[tree[p].lc].pos :
    tree[tree[p].rc].pos;
88      return p;//上一行的≥解决了题中如果相等就输出编号小的那一个的要求
89  }
90
91  void dfs(int u){
92      for(int i = head[u];i;i = nex[i]){
93          int v = ver[i];
94          if(deep[v] ≤ deep[u])//如果回来的就continue，本来应该是vis但是有预处理了deep也可以用deep
95              continue;
96          dfs(v);
97          root[u] = Merge(root[u],root[v],1,cnt);
98      }
99      ans[u] = tree[root[u]].pos;//存的是离散化后的下标
100 }
101
102 int main()
103 {
104     scanf("%d%d",&n,&m);
105     t = (int)(log(n) / log(2)) + 1;
106     over(i,1,n-1){
107         int x,y;
108         scanf("%d%d",&x,&y);
109         add(x,y);
110         add(y,x);
111     }
112     bfs();
113     over(i,1,n)root[i] = ++num;//根节点是它自己，但是这里是动态开点的编号
114     over(i,1,m){
```

```
        scanf("%d%d%d",&X[i],&Y[i],&Z[i]);
        val[i] = Z[i];//离散化
    }
    sort(val + 1,val + m + 1);
    cnt = unique(val+1,val+1+m) - (val+1);
    over(i,1,m){
        int x = X[i],y = Y[i];
        int z = lower_bound(val+1,val+1+cnt,Z[i]) - val;
        int p = lca(x,y);
        Insert(root[x],1,cnt,z,1);//维护差分数组
        Insert(root[y],1,cnt,z,1);
        Insert(root[p],1,cnt,z,-1);//对点差分
        if(fa[p][0])Insert(root[fa[p][0]],1,cnt,z,-1);
    }
    dfs(1);
    over(i,1,n)
        printf("%d\n",val[ans[i]]);
    return 0;
}
```

4.9 权值线段树区间第k小

```
#define Re register int
#define pl tree[p].PL
#define pr tree[p].PR
inline int ask(Re p,Re L,Re R,Re k){//查询第k小
    if(L==R)return L;//边界叶节点
    Re tmp=tree[pl].g;//计算左子树(数值范围在L~mid的数)共有多少个数字
    if(tmp≥k)return ask(pl,L,mid,k);
//左子树已经超过k个，说明第k小在左子树里面
    else return ask(pr,mid+1,R,k-tmp);
//左子树不足k个数字，应该在右子树中找到第(k-tmp)小
}
```

4.10 线段树分治（删除一些边询问是否为二分图）

有一个 n 个节点的图。在 k 时间内有 m 条边会出现后消失。
问出每一时间段内这个图是否是二分图。
输入格式
第一行三个整数 n,m,k。接下来 m 行，每行四个整数 x,y,l,r，表示有一条连接 x,y 的边在 l 时刻出现 r 时刻消失。
输出格式
k 行，第 i 行一个字符串 Yes 或 No，表示在第 i 时间段内这个图是否是二分图。

```
/*P5787 二分图 /【模板】线段树分治*/
const int N = 1e5 + 7, M = 2e5 + 7;
int n, m, k, u[M], v[M], f[N<<1], d[N<<1];
struct T {
    int l, r;
    vi e;
} t[N<<2];
stack< pi > s;

void build(int p, int l, int r) {
    t[p].l = l, t[p].r = r;
    if (l == r) return;
    build(ls, l, md), build(rs, md + 1, r);
}
```

```cpp

void ins(int p, int l, int r, int x) {
    if (t[p].l ≥ l && t[p].r ≤ r) return t[p].e.pb(x), void();
    if (l ≤ md) ins(ls, l, r, x);
    if (r > md) ins(rs, l, r, x);
}

inline int get(int x) {
    while (x ^ f[x]) x = f[x];
    return x;
}

inline void merge(int x, int y) {
    if (x == y) return;
    if (d[x] > d[y]) swap(x, y);
    s.push(mp(x, d[x] == d[y])), f[x] = y, d[y] += d[x] == d[y];
}

void dfs(int p, int l, int r) {
    bool ok = 1;
    ui o = s.size();
    for (ui i = 0; i < t[p].e.size(); i++) {
        int x = t[p].e[i], u = get(::u[x]), v = get(::v[x]);
        if (u == v) {
            for (int j = l; j ≤ r; j++) prints("No");
            ok = 0;
            break;
        }
        merge(get(::u[x] + N), v), merge(get(::v[x] + N), u);
    }
    if (ok) {
        if (l == r) prints("Yes");
        else dfs(ls, l, md), dfs(rs, md + 1, r);
    }
    while (s.size() > o) d[f[s.top().fi]] -= s.top().se, f[s.top().fi] = s.top().fi, s.pop();
}

int main() {
    rd(n), rd(m), rd(k), build(1, 1, k);
    for (int i = 1, l, r; i ≤ m; i++) {
        rd(u[i]), rd(v[i]), rd(l), rd(r);
        if (l ^ r) ins(1, l + 1, r, i);
    }
    for (int i = 1; i ≤ n; i++) f[i] = i, f[i+N] = i + N;
    dfs(1, 1, k);
    return 0;
}
```

$\oint$ 5. 平衡树

5.1 FQH - treap 非旋平衡树

operator 1 : 插入一个数
operator 2 : 删除一个数
operator 3 : 通过数值找排名
operator 4 : 通过排名找数值
operator 5 : 找到严格小于key的最大数（前驱）
operator 6 : 找到严格大于key的最小数（后继）

5.1.1 按权值分裂

```
const int N = 500007, M = 5000007, INF = 0x3f3f3f3f;
//num:节点编号
namespace treap{// 无旋treap (fhq-treap)
    const int N = 500007, M = 5000007, INF = 0x3f3f3f3f;

    struct fhq_treap{
        int l, r;
        int size;
        int fa;
        int val, fix;
        ll sum;
    }tr[N];

    int cnt;
    int x, y, z, root;

    inline void pushup(int p)
    {
        tr[p].size = tr[tr[p].l].size + tr[tr[p].r].size + 1;
        tr[p].sum = tr[tr[p].l].sum + tr[tr[p].r].sum + tr[p].val;
        tr[tr[p].l].fa = tr[tr[p].r].fa = p;
    }

    inline void split(int p, int k, int &x, int &y)//split_by_val
    {
        if(!p){x = y = 0; return ;}
        if(tr[p].val ≤ k)x = p, split(tr[p].r, k, tr[p].r, y);
        else y = p, split(tr[p].l, k, x, tr[p].l);
        pushup(p);
    }

    inline int merge(int x, int y)
    {
        if(!x || !y)return x + y;
        if(tr[x].fix > tr[y].fix){//小根堆
           tr[x].r = merge(tr[x].r, y);pushup(x);return x;
        }
        else {
            tr[y].l = merge(x, tr[y].l);pushup(y);return y;
        }
    }

    //!得到新节点
    inline int get_node(int val)
    {
        tr[++ cnt].size = 1;
        tr[cnt].fix = rand();
        tr[cnt].sum = tr[cnt].val = val;
        tr[cnt].fa = 0;
        return cnt;
```

```
51      }
52
53      //!树中插入新节点
54      inline void my_insert(int val)
55      {
56          split(root, val, x, y);
57          root = merge(merge(x, get_node(val)), y);
58      }
59
60      //!按值删除所有权值为k的所有点
61      inline void my_delet_all(int val)
62      {
63          split(root, val, x, z);
64          split(x, val - 1, x, y);
65          root = merge(x, z);
66          return ;
67      }
68
69      //!按值删除给定权值的一个点
70      inline void my_delet_one(int val)
71      {
72          split(root, val, x, z);
73          split(x, val - 1, x, y);
74          y = merge(tr[y].l, tr[y].r);
75          root = merge(merge(x, y), z);
76          return ;
77      }
78
79      //!查询指定排名的一个数,返回那个数的编号
80      inline int get_num_by_rank(int p, int k)
81      {
82          while(true){
83              if(k ≤ tr[tr[p].l].size)p = tr[p].l;
84              else if(k == tr[tr[p].l].size + 1)return p;
85              else k -= tr[tr[p].l].size + 1, p = tr[p].r;
86          }
87      }
88
89      //!按权值分裂时查询一个数的排名////////////////////////////////
90
91      //!根据权值查询一个数的排名
92      inline int get_rank_by_val(int val)
93      {
94          split(root, val - 1, x, y);
95          int res = tr[x].size + 1;
96          root = merge(x, y);
97          return res;
98      }
99
100     inline int get_val_by_rank(int k)
101     {
102         int res = tr[get_num_by_rank(root, k)].val;
103         return res;
104     }
105
106
107     //!查找前驱的编号
108     //!按值查找比它小的数中的最大的数的编号
109     inline int get_prev_of_num(int val)
110     {
```

```
111         split(root, val - 1, x, y);
112         int res = tr[get_num_by_rank(x, tr[x].size)].val;
113         root = merge(x, y);
114         return res;
115     }
116 
117     //!查找后继的编号
118     //!按值查找比它大的数中最小数的编号
119     inline int get_next_of_num(int val)
120     {
121         split(root, val, x, y);
122         int res = tr[get_num_by_rank(y, 1)].val;
123         root = merge(x, y);
124         return res;
125     }
126 
127     //!查找节点的祖先节点
128     inline int get_anc(int x)
129     {
130         while(tr[x].fa){
131             x = tr[x].fa;
132         }
133         return x;
134     }
135 
136 
137     inline void main()
138     {
139         srand((unsigned)time(NULL));
140         memset(tr, 0, sizeof tr);
141         cnt = 0;
142     }
143 }
144 
145 int n, m;
146 
147 int main()
148 {
149     scanf("%d", &n);
150     treap::main();
151 
152     while(n -- ){
153         int op, x;
154         scanf("%d%d", &op, &x);
155         if(op == 1)treap::my_insert(x);
156         else if(op == 2)treap::my_delet_one(x);
157         else if(op == 3)treap::get_rank_by_val(x);
158         else if(op == 4)printf("%d\n", treap::get_val_by_rank(x));
159         else if(op == 5)printf("%d\n", treap::get_prev_of_num(x));
160         else if(op == 6)printf("%d\n", treap::get_next_of_num(x));
161     }
162     return 0;
163 }
164 
165 
```

5.1.2 按排名分裂

```
    inline void split(int p, int k, int &x, int &y, int fx = 0, int fy = 0)
    {
        //传两个父亲节点的参数是为了防止记录父亲出问题.
        //因为父亲记录儿子的时候是通过取地址传参修改的
        //而两颗子树记录的父亲是分开的
        if(!p){x = y = 0; return ;}
        if(k ≤ tr[tr[p].l].size)tr[p].fa = fy, y = p, split(tr[p].l, k, x, tr[p].l, fx, p), pushup(p);
        else tr[p].fa = fx, x = p, split(tr[p].r, k - tr[tr[p].l].size - 1, tr[p].r, y, p, fy),
pushup(p);
    }
```

```
//!按排名分裂时查询一个数的排名/
    //!需要维护父节点

    //!根据编号查询这个数的排名
    inline int get_rank_by_num(int p)
    {
        int res = tr[tr[p].l].size + 1;
        while(p ≠ root){//一直回溯
            if(p == tr[tr[p].fa].r)res += tr[tr[tr[p].fa].l].size + 1;
            p = tr[p].fa;
        }
        return res;
    }
```

完整代码

> 书柜里的书是从上至下堆放成一列。用 1 到n 的正整数给每本书都编了号。
>
> - Top，则后有一个整数 s，表示把编号为 s 的书放在最上面。
>   （Top:提取该节点,放在树的最前面合并. ）
> - Bottom，则后有一个整数 s，表示把编号为 s 的书放在最下面。
>   （Bottom: 提取节点,放在树的最后面合并. ）
> - Insert，则后有两个整数 s,t，表示若编号为 s 的书上面有 x 本书，则放回这本书时他的上面有 x+t 本书。（t 等于1， -1，0）
>   （Insert:将它与前驱/后继从整棵树中分离出来,交换顺序合并. ）
> - Ask，则后面有一个整数 s，表示询问编号为 s 的书上面有几本书。
>   （Ask: 直接通过编号找到节点是中序遍历结果第几个. ）
> - Query，则后面有一个整数 s，询问从上面起第 s 本书的编号。
>   （Query:先找到节点是中序遍历第几个,然后split前k个,在分离出的第一颗子树中找最右边的节点.）

无旋treap可以维护一棵树的中序遍历结果.但是不支持通过编号来找节点.于是在无旋treap的基础上,我维护了每个节点的父亲,这样就可以求出一个节点是中序遍历中的第几个.

```

typedef long long ll;
const int N = 500007, M = 5000007, INF = 0x3f3f3f3f;

//num:节点编号
namespace treap{// 无旋treap (fhq-treap)
    const int N = 500007, M = 5000007, INF = 0x3f3f3f3f;

    struct fhq_treap{
        int l, r;
        int size;
        int fa;
        int val, fix;
        ll sum;
    }tr[N];

    int r1, r2, r3, r4;
    int cnt;
```

```
19      int x, y, z, root;
20      int id[N];
21
22      inline void pushup(int p)
23      {
24          tr[p].size = tr[tr[p].l].size + tr[tr[p].r].size + 1;
25          //tr[p].sum = tr[tr[p].l].sum + tr[tr[p].r].sum + tr[p].val;
26          //tr[tr[p].l].fa = tr[tr[p].r].fa = p;
27      }
28
29      inline void split(int p, int k, int &x, int &y, int fx = 0, int fy = 0)
30      {
31          //传两个父亲节点的参数是为了防止记录父亲出问题.
32          //因为父亲记录儿子的时候是通过取地址传参修改的
33          //而两颗子树记录的父亲是分开的
34          if(!p){x = y = 0; return ;}
35          if(k ≤ tr[tr[p].l].size)tr[p].fa = fy, y = p, split(tr[p].l, k, x, tr[p].l, fx, p), pushup(p);
36          else tr[p].fa = fx, x = p, split(tr[p].r, k - tr[tr[p].l].size - 1, tr[p].r, y, p, fy),
    pushup(p);
37      }
38
39      inline int merge(int x, int y)
40      {
41          if(!x || !y)return x + y;
42          if(tr[x].fix < tr[y].fix){//小根堆
43             tr[x].r = merge(tr[x].r, y);
44             tr[tr[x].r].fa = x;
45             pushup(x);return x;
46
47          }
48          else {
49              tr[y].l = merge(x, tr[y].l);
50              tr[tr[y].l].fa = y;
51              pushup(y);return y;
52          }
53      }
54
55      //!得到新节点
56      inline int get_node(int val)
57      {
58          tr[++ cnt].size = 1;
59          tr[cnt].fix = rand();
60          tr[cnt].val = val;
61          tr[cnt].fa = 0;
62          id[val] = cnt;
63          return cnt;
64      }
65
66      //!树中插入新节点
67      inline void my_insert(int k, int val)
68      {
69          split(root, k, x, y);
70          root = merge(merge(x, get_node(val)), y);
71      }
72
73      int get_rank_by_num(int p)
74      {
75          int res = tr[tr[p].l].size + 1;
76          int node = p;
77          while(node ≠ root && p){
```

```
            if(p == tr[tr[p].fa].r)
                res += tr[tr[tr[p].fa].l].size + 1;
            p = tr[p].fa;
        }
        return res;
    }

    //!查找节点的祖先节点
    inline int get_anc(int x)
    {
        while(tr[x].fa){
            x = tr[x].fa;
        }
        return x;
    }


    inline void main()
    {
        srand((unsigned)time(NULL));
        memset(tr, 0, sizeof tr);
        cnt = 0;
    }

    inline void solve_T(int x)
    {
        x = get_rank_by_num(id[x]);
        split(root, x, r1, r3);
        split(r1, x - 1, r1, r2);
        root = merge(r2, merge(r1, r3));
    }

    void solve_B(int x)
    {
        x = get_rank_by_num(id[x]);
        split(root, x, r1, r3, 0);
        split(r1, x - 1, r1, r2, 0);
        root = merge(r1, merge(r3, r2));
    }

    void solve_I(int x, int y)
    {
        x = get_rank_by_num(id[x]);
        if(y == 0)return ;

        if(y > 0){
            split(root, x + 1, r3, r4);
            split(r3, x, r2, r3);
            split(r2, x - 1, r1, r2);
            root = merge(r1, merge(r3, merge(r2, r4)));
        }
        else {
            split(root, x, r3, r4);
            split(r3, x - 1, r2, r3);
            split(r2, x - 2, r1, r2);
            root = merge(r1, merge(r3, merge(r2, r4)));
        }
    }

    void solve_A(int x){
```

```
        x = get_rank_by_num(id[x]);
        printf("%d\n", x - 1);
    }

    void solve_Q(int x)
    {
        //先找到节点是中序遍历第几个,然后split前k个,
        //在分离出的第一颗子树中找最右边的节点.
        split(root, x, r1, r2);
        int node = r1;
        while(tr[node].r) node = tr[node].r;
        printf("%d\n", tr[node].val);
        root = merge(r1, r2);
    }
}

int n, m;
int a[N];

int main()
{
    scanf("%d%d", &n, &m);
    treap::main();
    for(int i = 1; i ≤ n; ++ i)
        scanf("%d", &a[i]), treap::my_insert(i - 1, a[i]);
    for(int i = 1; i ≤ m; ++ i){
        char op[10];
        int x, y;

        scanf("%s%d", op, &x);
        if(op[0] == 'T'){
            treap::solve_T(x);
        }
        else if(op[0] == 'B'){
            treap::solve_B(x);
        }
        else if(op[0] == 'I'){
            scanf("%d", &y);
            treap::solve_I(x, y);
        }
        else if(op[0] == 'A'){
            treap::solve_A(x);
        }
        else if(op[0] == 'Q'){
            treap::solve_Q(x);
        }
    }
    return 0;
}
```

5.2 文艺平衡树

维护一个有序序列，经过m次区间操作：翻转一个区间，例如原有序序列是 $5\ 4\ 3\ 2\ 1$，翻转区间是 $[2,4]$的话，结果是 $5\ 2\ 3\ 4\ 1$，要求最后输出整个序列

```
/*平衡树实际上画出来是一颗乱糟糟的小根堆，但是满足小根堆的性质，
所以我们这里区间翻转，找到区间需要按照排名split，
只需要在这个区间里一直交换左右儿子即可实现区间翻转*/
/*最后中序遍历输出值即可*/
int n, m, root;
```

```
namespace treap{
    int n, m, tot;
    struct tree{
        int l, r;
        int fix;
        int val;
        int size;
        int lz;
    }tr[500007];

    inline int get_node(int v)
    {
        tr[++ tot].val = v;
        tr[tot].size = 1;
        tr[tot].fix = rand();
        return tot;
    }

    void pushdown(int p)//下传懒标记,注意先交换,再下传
    {
        if(tr[p].lz){
            swap(tr[p].l, tr[p].r);
            tr[tr[p].l].lz ^= 1;
            tr[tr[p].r].lz ^= 1;
            tr[p].lz = 0;
        }
    }

    void pushup(int p)
    {
        tr[p].size = tr[tr[p].l].size + tr[tr[p].r].size + 1;
    }

    void split(int p, int rnk, int &x, int &y)
    {
        if(!p){x = y = 0;return ;}
        pushdown(p);
        //合并左边，往右边走
        if(tr[tr[p].l].size + 1 ≤ rnk)x = p,split(tr[p].r, rnk - tr[tr[p].l].size - 1, tr[p].r, y);
        else y = p, split(tr[p].l, rnk, x, tr[p].l);
        pushup(p);
    }

    int merge(int x, int y)
    {
        if(!x || !y)return x + y;
        if(tr[tr[x].l].fix < tr[tr[y].l].fix)
        {
            pushdown(x);
            tr[x].r = merge(tr[x].r, y);
            pushup(x);
            return x;
        }
        else {
            pushdown(y);
            tr[y].l = merge(x, tr[y].l);
            pushup(y);
            return y;
        }
    }
```

```

    void print(int p)//必须要中序遍历才能输出答案
    {
        pushdown(p);

        if(tr[p].l)//左
            print(tr[p].l);
        printf("%d ",tr[p].val);//中
        if(tr[p].r)//右
            print(tr[p].r);
    }
}

int main()
{
    scanf("%d%d", &n, &m);
    for(int i = 1; i ≤ n; ++ i)
    {
        root = treap::merge(root, treap::get_node(i));
    }
    for(int i = 1, l, r, x, y, p; i ≤ m; ++ i)
    {
        scanf("%d%d", &l, &r);
        treap::split(root, r, x, y);
        treap::split(x, l - 1, x, p);
        treap::tr[p].lz ^= 1;
        root = treap::merge(treap::merge(x, p), y);
    }
    treap::print(root);
    return 0;
}


```

5.3 可持久化序列

求支持三个操作：

`1 l r`，翻转l到r的区间；

`2 l r`，询问l的到r的区间和；

`3 p`，回到p时刻。

每次修改新建点打翻转标记即可

```
typedef pair<int,int> Pair;
int read() {
    int x=0,f=1;
    char c=getchar();
    for (;!isdigit(c);c=getchar()) if (c=='-') f=-1;
    for (;isdigit(c);c=getchar()) x=x*10+c-'0';
    return x*f;
}
const int maxn=5e4+5;
const int nlogn=1.3e7+5;
struct node {
    int x,hp,l,r,sum,size;
    bool rev;
    void clear() {
        x=hp=l=r=sum=size=rev=0;
    }
};
```

```
18 struct TREAP {
19     int pool[nlogn];
20     int pooler;
21     node t[nlogn];
22     int now,all;
23     int root[maxn];
24     TREAP ():now(0),pooler(1) {
25         for (int i=1;i<nlogn;++i) pool[i]=i;
26         root[now]=pool[pooler++];
27     }
28     int newroot() {
29         int ret=pool[pooler++];
30         return ret;
31     }
32     int newnode(int x) {
33         int ret=pool[pooler++];
34         t[ret].hp=rand();
35         t[ret].size=1;
36         t[ret].x=t[ret].sum=x;
37         return ret;
38     }
39     void delnode(int x) {
40         t[x].clear();
41         pool[--pooler]=x;
42     }
43     void next() {
44         root[++all]=newroot();
45         t[root[all]]=t[root[now]];
46         now=all;
47     }
48     void back(int x) {
49         now=x;
50     }
51     void update(int x) {
52         t[x].sum=t[x].x+t[t[x].l].sum+t[t[x].r].sum;
53         t[x].size=t[t[x].l].size+t[t[x].r].size+1;
54     }
55     void pushdown(int x) {
56         if (!t[x].rev) return;
57         if (t[x].l) {
58             int tx=newnode(t[t[x].l].x);
59             t[tx]=t[t[x].l];
60             t[tx].rev^=true;
61             t[x].l=tx;
62         }
63         if (t[x].r) {
64             int tx=newnode(t[t[x].r].x);
65             t[tx]=t[t[x].r];
66             t[tx].rev^=true;
67             t[x].r=tx;
68         }
69         swap(t[x].l,t[x].r);
70         t[x].rev=false;
71     }
72     int merge(int x,int y) {
73         if (!x) return y;
74         if (!y) return x;
75         int now;
76         if (t[x].hp≤t[y].hp) {
77             now=newnode(t[x].x);
```

```
            t[now]=t[x];
            pushdown(now);
            t[now].r=merge(t[now].r,y);
        } else {
            now=newnode(t[y].x);
            t[now]=t[y];
            pushdown(now);
            t[now].l=merge(x,t[now].l);
        }
        update(now);
        return now;
    }
    Pair split(int x,int p) {
        if (t[x].size==p) return make_pair(x,0);
        int now=newnode(t[x].x);
        t[now]=t[x];
        pushdown(now);
        int l=t[now].l,r=t[now].r;
        if (t[l].size≥p) {
            t[now].l=0;
            update(now);
            Pair g=split(l,p);
            now=merge(g.second,now);
            return make_pair(g.first,now);
        } else if (t[l].size+1==p) {
            t[now].r=0;
            update(now);
            return make_pair(now,r);
        } else {
            t[now].r=0;
            update(now);
            Pair g=split(r,p-t[l].size-1);
            now=merge(now,g.first);
            pushdown(now);
            return make_pair(now,g.second);
        }
    }
    void rever(int l,int r) {
        ++l,++r;
        Pair g=split(root[now],l-1);
        Pair h=split(g.second,r-l+1);
        int want=h.first;
        int here=newnode(t[want].x);
        t[here]=t[want];
        t[here].rev^=true;
        int fi=merge(g.first,here);
        int se=merge(fi,h.second);
        root[now]=se;
    }
    int query(int l,int r) {
        ++l,++r;
        Pair g=split(root[now],l-1);
        Pair h=split(g.second,r-l+1);
        int want=h.first;
        int ret=t[want].sum;
        int fi=merge(g.first,want);
        int se=merge(fi,h.second);
        root[now]=se;
        return ret;
    }
```

```
    void insert(int x) {
        int k=newnode(x);
        root[now]=merge(root[now],k);
    }
} Treap;
int main() {
#ifndef ONLINE_JUDGE
    freopen("test.in","r",stdin);
    freopen("my.out","w",stdout);
#endif
    srand(time(0));
    int n=read(),m=read();
    for (int i=1;i≤n;++i) {
        int x=read();
        Treap.insert(x);
    }
    while (m--) {
        int op=read();
        if (op==1) {
            Treap.next();
            int l=read(),r=read();
            Treap.rever(l,r);
        } else if (op==2) {
            int l=read(),r=read();
            int ans=Treap.query(l,r);
            printf("%d\n",ans);
        } else if (op==3) {
            Treap.back(read());
        }
    }
    return 0;
}
```

∮ 6. 可持久化线段树(主席树)

6.1 查询区间第k大值

> 如题，给定 n 个整数构成的序列 a，将对于指定的闭区间 [l,r] 查询其区间内的第 k 小值。

可持久化线段树，不能用堆的方法存子结点了，所以用指针l表示左儿子r表示右儿子

可持久化线段树难以处理区间修改，因为很难处理懒惰标记，除非使用永久化标记线段树

新版本其他的都不变，只把新的点替换，其他的性质例如左右儿子是谁不变，所以每新加入一个数，也就是到了新版本，就完成与一次新老版本的迭代（`tr[q] = tr[p]` `q`是新版本`p`是老版本）

线段树维护的是整个值域。
在数值上建立一个线段树，维护`cnt`表示每个数值的区间上一共有几个数。

线段树和平衡树都是二叉树，所以我们需要做二分的时候可以考虑能否在树里做二分。

我们首先考虑`[1,R]`这个区间的个数。很明显我们可以直接用第R个版本即可。
但是我们要求的是`[L,R]`这个区间，有两个限制。所以我们利用类似前缀和的思想，用第`R`个版本个数`cnt1`减去第`L-1`个版本`cnt2`
即可得到`L`到`R`之间的个数：`cnt1-cnt2`

每次都是个上一个版本比较
l指的是左儿子，不是左边界

```
1  //线段树维护的是数值
```

```
int n, m;
int idx;
int a[N];
int root[N];
vector<int> nums;
struct Tree{
    int l, r;
    int cnt;//表示的是这个值域区间一共有多少个数
}tr[N * 4 + N * 17];//log(1e5) ≈ 17

int find(int x){
    return lower_bound(nums.begin(), nums.end(), x) - nums.begin();
}

int build(int l, int r){
    int p = ++ idx;
    if(l == r)return p;
    int mid = l + r >> 1;
    tr[p].l = build(l, mid);
    tr[p].r = build(mid + 1, r);
    return p;
}

int insert(int p, int l, int r, int x){//开始把n个数插入（迭代新版本）
    int q = ++ idx;
    tr[q] = tr[p];//新老版本交接
    if(l == r){
        tr[q].cnt ++ ;
        return q;
    }
    int mid = l + r >> 1;
    if(x ≤ mid)tr[q].l = insert(tr[p].l, l, mid, x);
    else tr[q].r = insert(tr[p].r, mid + 1, r, x);
    tr[q].cnt = tr[tr[q].l].cnt + tr[tr[q].r].cnt ;
    return q;
}

int query(int q, int p, int l, int r, int k){
    if(l == r)return r;
    int mid = l + r >> 1;
    int cnt = tr[tr[q].l].cnt - tr[tr[p].l].cnt;
    if(k ≤ cnt)return query(tr[q].l, tr[p].l, l, mid, k);
    else return query(tr[q].r, tr[p].r, mid + 1, r, k - cnt);
    //这里注意往右区间找的时候第k大已经变成了k-cnt大了
}

int main(){
    scanf("%d%d", &n, &m);

    for(int i = 1;i ≤ n; ++ i){
        scanf("%d", &a[i]);
        nums.push_back(a[i]);
    }

    sort(nums.begin(), nums.end());
    nums.erase(unique(nums.begin(), nums.end()), nums.end());
    //第一个版本
    root[0] = build(0, nums.size() - 1);
    //新版本是由旧版本继承过来的
    for(int i = 1;i ≤ n; ++ i)
```

```
        root[i] = insert(root[i - 1], 0, nums.size() - 1, find(a[i]));

    while(m -- ){
        int l, r, k;
        scanf("%d%d%d", &l, &r, &k);
        printf("%d\n", nums[query(root[r], root[l - 1], 0, nums.size() - 1, k)]);
    }
    return 0;
}


```

6.2 （带修改）动态查询区间第k大值

给定一个含有 n 个数的序列 $a_1, a_2 \dots a_n$，需要支持两种操作：
Q l r k 表示查询下标在区间 $[l, r]$ 中的第 k 小的数
C x y 表示将a_x改为 y

```
#include<cstdio>
#include<cstring>
#include<cmath>
#include<iostream>
#include<algorithm>
using namespace std;
const int MAX=10005;
struct segment_tree{int v;int ls,rs;}t[MAX*400];//线段树开nlogn大小
struct operation{bool b;int l,r,k;int pos,t;}q[MAX];//因为要离散花所以要把所有数据输进来离线搞
int n,m,a[MAX],o[MAX<<1],rt[MAX],len,tot,temp[2][20],cnt[2];
char opt;
void Modify(int &now,int l,int r,int pos,int val)
{
    if (!now) now=++tot;
    t[now].v+=val;
    if (l==r) return;
    int mid=l+r>>1;
    if (pos≤mid) Modify(t[now].ls,l,mid,pos,val);
    else Modify(t[now].rs,mid+1,r,pos,val);
}
void prepare_Modify(int x,int val)
{
    int k=lower_bound(o+1,o+len+1,a[x])-o;
    for (int i=x;i≤n;i+=i&-i) Modify(rt[i],1,len,k,val);//处理出需要修改哪log棵主席树
}
int Query(int l,int r,int k)
{
    if (l==r) return l;
    int mid=l+r>>1,sum=0;
    for (int i=1;i≤cnt[1];i++) sum+=t[t[temp[1][i]].ls].v;
    for (int i=1;i≤cnt[0];i++) sum-=t[t[temp[0][i]].ls].v;
    if (k≤sum)
    {
        for (int i=1;i≤cnt[1];i++) temp[1][i]=t[temp[1][i]].ls;
        for (int i=1;i≤cnt[0];i++) temp[0][i]=t[temp[0][i]].ls;
        return Query(l,mid,k);
    }
    else
    {
        for (int i=1;i≤cnt[1];i++) temp[1][i]=t[temp[1][i]].rs;
        for (int i=1;i≤cnt[0];i++) temp[0][i]=t[temp[0][i]].rs;
        return Query(mid+1,r,k-sum);
```

```
    }
}
int prepare_Query(int l,int r,int k)
{
    memset(temp,0,sizeof(temp));//同修改，处理出需要进行相减操作的是哪log棵主席树
    cnt[0]=cnt[1]=0;
    for (int i=r;i;i-=i&-i) temp[1][++cnt[1]]=rt[i];
    for (int i=l-1;i;i-=i&-i) temp[0][++cnt[0]]=rt[i];
    return Query(1,len,k);
}
int main()
{
    ios::sync_with_stdio(false);
    cin>>n>>m;
    for (int i=1;i≤n;i++) cin>>a[i],o[++len]=a[i];
    for (int i=1;i≤m;i++)
    {
        cin>>opt;
        q[i].b=(opt=='Q');
        if (q[i].b)    cin>>q[i].l>>q[i].r>>q[i].k;
        else cin>>q[i].pos>>q[i].t,o[++len]=q[i].t;
    }
    sort(o+1,o+len+1);
    len=unique(o+1,o+len+1)-o-1;//离散 -- 排序 + 去重
    for (int i=1;i≤n;i++) prepare_Modify(i,1);
    for (int i=1;i≤m;i++)
    {
        if (q[i].b)    printf("%d\n",o[prepare_Query(q[i].l,q[i].r,q[i].k)]);
        else
        {
            prepare_Modify(q[i].pos,-1);
            a[q[i].pos]=q[i].t;
            prepare_Modify(q[i].pos,1);
        }
    }
    return 0;
}
```

$\oint$ 7. 单调栈与单调队列

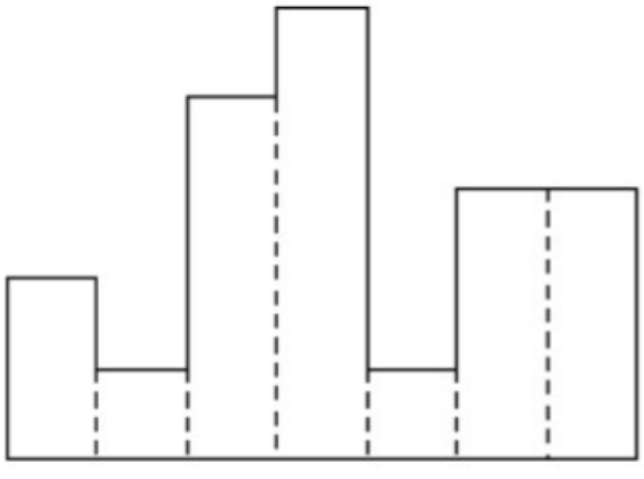
7.1 单调栈

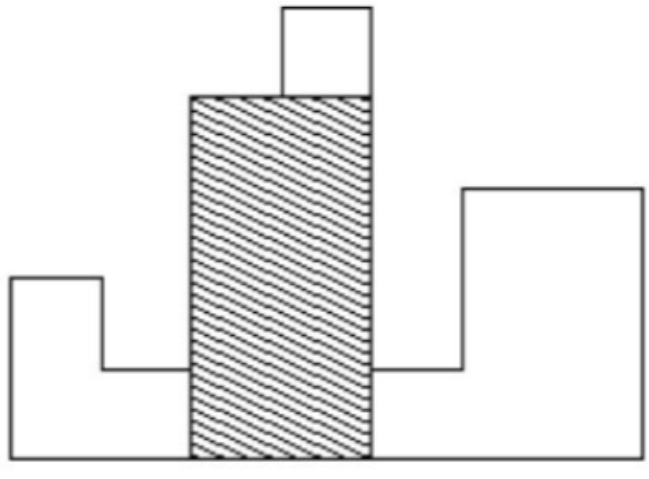
```
/*常见模型：找出每个数左边离它最近的比它大/小的数*/
int tt = 0;
for (int i = 1; i ≤ n; i ++ )
{
    while (tt && check(stk[tt], i)) tt -- ;
    stk[ ++ tt] = i;
}
```

直方图是由在公共基线处对齐的一系列矩形组成的多边形。

矩形具有相等的宽度，但可以具有不同的高度。

例如，图例左侧显示了由高度为2,1,4,5,1,3,3的矩形组成的直方图，矩形的宽度都为1：





通常，直方图用于表示离散分布，例如，文本中字符的频率。

现在，请你计算在公共基线处对齐的直方图中最大矩形的面积。

图例右图显示了所描绘直方图的最大对齐矩形。

```cpp
ll q[N],w[N],h;
int n;


int main()
{
    while(scanf("%d",&n)&&n){
        memset(q,-1,sizeof q);
        int top=0;
        ll ans=0;
        over(i,1,n+1){
            if(i≠n+1){
                scanf("%lld",&h);
            }
            else h=0;
            if(h>q[top])//高于栈顶元素，保持递增就入栈
                q[++top]=h,w[top]=1;
            else {
                ll cnt=0;
                while(h≤q[top]){//单调被破坏就pop，把所有低于这个元素的全部pop并更新答案。
                    ans=max(ans,(cnt+w[top])*q[top]);
                    cnt+=w[top--];
                }
                q[++top]=h;
                w[top]=cnt+1;
            }
        }
        printf("%lld\n",ans);
    }
    return 0;
}

```

有 n 个高楼排成一行，每个楼有一个高度 h_i。称可以从楼 i 跳到 楼 j，当 i, j (i<j)满足以下三个条件之一：

- $i+1=j$
- $\max(h_{i+1},h_{i+2},\cdots,h_{j-1})<\min(h_i,h_j)$
- $\min(h_{i+1},h_{i+2},\cdots,h_{j-1})>\max(h_i,h_j)$

```cpp
const int N = 500007, INF = 0x3f3f3f3f;
int n, m;
int stk_up[N], utop;
int stk_down[N], dtop;
int a[N];
int f[N];
int main()
{
    scanf("%d", &n);
    for(int i =1; i ≤ n;++ i)
        scanf("%d", &a[i]);
    memset(f, 0x3f, sizeof f);
    f[1] = 0;
    stk_down[++ dtop] = 1;
    stk_up[++ utop] = 1;
    for(int i = 2; i ≤ n; ++ i){
        f[i] = min(f[i], f[i - 1] + 1);//第一种方式
        //第二种方式，维护单减
        while(dtop && a[i] ≥ a[stk_down[dtop]]) {
            if(a[i] ≠ a[stk_down[dtop]])f[i] = min(f[i], f[stk_down[dtop - 1]] + 1);
            dtop -- ;
        }
        //第三种方式，维护单增
        while(utop && a[i] ≤ a[stk_up[utop]]) {
            if(a[i] ≠ a[stk_up[utop]])f[i] = min(f[i], f[stk_up[utop - 1]] + 1);
            utop -- ;
        }
        stk_down[++ dtop] = i;
        stk_up[++ utop] = i;
    }
    printf("%d\n",f[n]);
    return 0;
}
```

7.2 单调队列

```cpp
/*常见模型：找出滑动窗口中的最大值/最小值*/
int hh = 0, tt = -1;
for (int i = 0; i < n; i ++ )
{
    while (hh ≤ tt && check_out(q[hh])) hh ++ ;  // 判断队头是否滑出窗口
    while (hh ≤ tt && check(q[tt], i)) tt -- ;
    q[ ++ tt] = i;
}
```

7.3 单调队列求矩阵的和

> 题目大意：给出矩阵的行数n和列数m，矩阵 $A_{ij}=lcm(i,j)$,求每个大小为k*k的子矩阵的最大值的和。

> 我们不妨先把这个问题中二维的矩阵简化成一维的数列。那么现在的问题就变成了一个求连续的滑动窗口最值问题：给出一个长度为n的数列和一个长度为k(k<n)的窗口，记录滑动窗口位于每个位置下的下的最大值，求这些最大值和。如果我们先用单调队列，把每一列长度为k的滑动窗口的最大值先预处理出来，就像这样，先变成这么一个kn的矩阵，然后再用一次单调队列，把每一行中长度为k的滑动窗口中的最大值求出来做一下累加，本题的时间复杂度就降到了 O((n-k+1)k)

```
typedef long long ll;
typedef pair<int,int> PII;
const int INF = 0x3f3f3f3f;
const int N = 5e3+7;
int n,m,k;
int a[N][N],b[N][N];
int f[N];
int que[N];
void getmaxa(int x,int len){
    memset(que,0,sizeof que);
    int i,head = 1,tail = 1;
    for(i = 1;i < k;++i){
        while(tail ≥ head && f[i] > f[que[tail]])
            tail--;
        tail++;
        que[tail] = i;
    }
    for(i = k;i ≤ len;++i){
        while(tail ≥ head && i - que[head] ≥ k)
            head++;
        while(tail ≥ head && f[i] > f[que[tail]])
            tail--;
        tail++;
        que[tail] = i;
        b[i][x] = f[que[head]];
    }
}

ll ans;

void getmaxb(int len){
    memset(que,0,sizeof que);
    int i,head = 1,tail = 1;
    for(i = 1;i < k;++i){
        while(tail ≥ head && f[i] > f[que[tail]])
            tail--;
        tail++;
        que[tail] = i;
    }
    for(i = k;i ≤ len;++i){
        while(tail ≥ head && i - que[head] ≥ k)
            head++;
        while(tail ≥ head && f[i] > f[que[tail]])
            tail--;
        tail++;
        que[tail] = i;
        ans += f[que[head]];
    }
}

int GCD(int x,int y){
    return !y?x:GCD(y,x%y);
}

int LCM(int x,int y){
```

```
    return x/GCD(x,y)*y;
}
int main()
{
    scanf("%d%d%d",&n,&m,&k);
    for(int i = 1;i ≤ n;++i)
        for(int j = 1;j ≤ m;++j)
            a[i][j] = LCM(i,j);
    for(int i = 1;i ≤ m;++i){
        for(int j = 1;j ≤ n;++j)
            f[j] = a[j][i];
        getmaxa(i,n);
    }
    for(int i = k;i ≤ n;++i){
        for(int j = 1;j ≤ m;++j)
            f[j] = b[i][j];
        getmaxb(m);
    }
    cout≪ans≪endl;
    return 0;
}
```

$\oint$ 8. ST表（静态查询区间最值）

8.1 一维RMQ问题

> 给定一个长度为 N的数列，和 M 次询问，求出每一次询问的区间内数字的最大值。

$O(nlogn)$预处理，$O(1)$查询最值

```
ll n,m,f[N][21],a[N];

inline void ST()
{
    over(i,1,n)
        f[i][0]=a[i];//初始化,以自己为起点2^0=1长的区间就是他自己
    for(ll j=1;(1≪j)≤n;++j)//枚举区间长度
        for(ll i=1;i+(1≪j)-1≤n;++i)//枚举起点
            f[i][j]=max(f[i][j-1],f[i+(1≪(j-1))][j-1]);//取最大值
}

inline ll query(ll l,ll r)
{
    ll k=trunc(log2(r-l+1));
    return max(f[l][k],f[r-(1≪k)+1][k]);//因为已经用区间DP初始化过了，所以直接比较输出即可
}

int main()
{
    scanf("%lld%lld",&n,&m);
    over(i,1,n)scanf("%lld",&a[i]);
    ST();
    over(i,1,m)
    {
        ll x,y;
        scanf("%lld%lld",&x,&y);
        printf("%lld\n",query(x,y));
    }
    return 0;
```

```
}
```

8.2 二维RMQ问题

```
/*二维 RMQ，预处理复杂度 n*m*log*(n)*log(m) 3 * 数组下标从 1 开始*/
int val[310][310];
int dp[310][310][9][9];//最大值
int mm[310];//二进制位数减一，使用前初始化
void initRMQ(int n,int m){
    for(int i = 1;i ≤ n;i++)
    for(int j = 1;j ≤ m;j++)
        dp[i][j][0][0] = val[i][j];
    for(int ii = 0; ii ≤ mm[n]; ii++)
    for(int jj = 0; jj ≤ mm[m]; jj++)
        if(ii+jj)
        for(int i = 1; i + (1≪ii) - 1 ≤ n;i++)
        for(int j = 1; j + (1≪jj) - 1 ≤ m;j++){
            if(ii)dp[i][j][ii][jj] = max(dp[i][j][ii-1][jj],dp[i+(1≪(ii-1))][j][ii-1][jj]);
            else dp[i][j][ii][jj] = max(dp[i][j][ii][jj-1],dp[i][j+(1≪(jj-1))][ii][jj-1]);
    }
}
//查询矩形内的最大值 (x1≤x2,y1≤y2)
int rmq(int x1,int y1,int x2,int y2){
    int k1 = mm[x2-x1+1];
    int k2 = mm[y2-y1+1];
    x2 = x2 - (1≪k1) + 1;
    y2 = y2 - (1≪k2) + 1;
    return max(max(dp[x1][y1][k1][k2],dp[x1][y2][k1][k2]),max(dp[x2][y1][k1][k2],dp[x2][y2][k1][k2]));
}
int main(){
//在外面对 mm 数组进行初始化
    mm[0] = -1;
    for(int i = 1;i ≤ 305;i++)
    mm[i] = ((i&(i-1))==0)?mm[i-1]+1:mm[i-1];
    int n,m;
    int Q;
    int r1,c1,r2,c2;
    while(scanf("%d%d",&n,&m) == 2){
        for(int i = 1;i ≤ n;i++)
        for(int j = 1;j ≤ m;j++)
            scanf("%d",&val[i][j]);
        initRMQ(n,m);
        scanf("%d",&Q);
        while(Q--){
            scanf("%d%d%d%d",&r1,&c1,&r2,&c2);
            if(r1 > r2)swap(r1,r2);
            if(c1 > c2)swap(c1,c2);
            int tmp = rmq(r1,c1,r2,c2);
            printf("%d␣",tmp);
            if(tmp == val[r1][c1] || tmp == val[r1][c2] || tmp ==val[r2][c1] || tmp == val[r2][c2])
                printf("yes\n");
            else printf("no\n");
        }
    }
    return 0;
}
```

$\oint$ 9. 莫队

我们首先考虑双指针的暴力法，发现很容易就会被卡成$O(nm)$，这时候我们的莫队出现了，莫队说，我可以像变魔术一样，把$O(nm)$的算法通过一个神奇的排序方式，使得我们最坏的情况下，时间复杂度也会非常优秀：$O(n\sqrt{n})$。

莫队算法是一个离线的算法，我们先将所有的询问全部存下来，然后排序。我们的每一个询问都是一个左右区间，(l,r)

我们的排序方法为双关键字排序，我们将每个询问的左端点 l 分块。
第一关键字为左端点分块的编号从小到大，第二关键字为右端点的下标从小到大。

莫队算法把排序做了简单的修改，就把暴力法的复杂度从O(mn)提高到$O(n\sqrt{n})$。

（1）暴力法的排序：把查询的区间按左端点排序，如果左端点相同，再按右端点排序。

（2）莫队算法的排序：把数组分块（分成$\sqrt{n}$块），然后把查询的区间按左端点所在块的序号排序，如果左端点的块相同，再按右端点排序（注意不是按右端点所在的块排序，下一小节“莫队算法的几何解释”将说明原因）。

除了排序不一样，莫队算法和暴力法的其他步骤完全一样。

这个简单的修改是否真能提高效率？下面分析多种情况下莫队算法的复杂度。

（1）简单情况。区间交错，设区间[P_1,y_1]、[P_2,y_2]的关系是$P_1<P_2$，$y_1\le y_2$，其中P_1、P_2是左端点所在的块。L、R只需要从左到右扫描一次，m次查询的总复杂度是O(n)。

（2）复杂情况。区间包含，设两个区间查询[P_1,y_1]、[P_2,y_3]的关系是$P_1=P_2$，$y_2\le y_1$。，如下图所示。

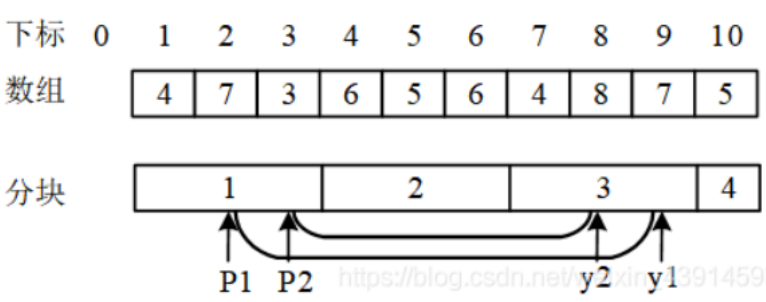


图5 按块排序后的区间包含

此时小区间[P_2,y_2]排在大区间[P_1,y_1]的前面，与暴力法正好相反。在区间内，右指针R从左到右单向移动，不再往复移动。而左指针L发生了回退移动，但是被限制在一个长为 的块内，每次移动的复杂度是O($\sqrt{n}$)的。m次查询，每次查询左端点只需要移动O($\sqrt{n}$)次，右端点R共单向移动O(n)次，总复杂度O(n$\sqrt{n}$)。

（3）特殊情况：m个询问，端点都在不同的块上，此时莫队算法和暴力法是一样的。但此时m小于 ，总复杂度O(mn) = O($\sqrt{n}$n)。

编码时，还可以对排序做一个小优化：奇偶性排序，让奇数块和偶数块的排序相反。例如左端点L都在奇数块，则对R从大到小排序；若L在偶数块，则对R从小到大排序（反过来也可以：奇数块从小到大，偶数块从大到小）。

1. 基础莫队

AcWing 2492. HH的项链

HH 有一串由各种漂亮的贝壳组成的项链。

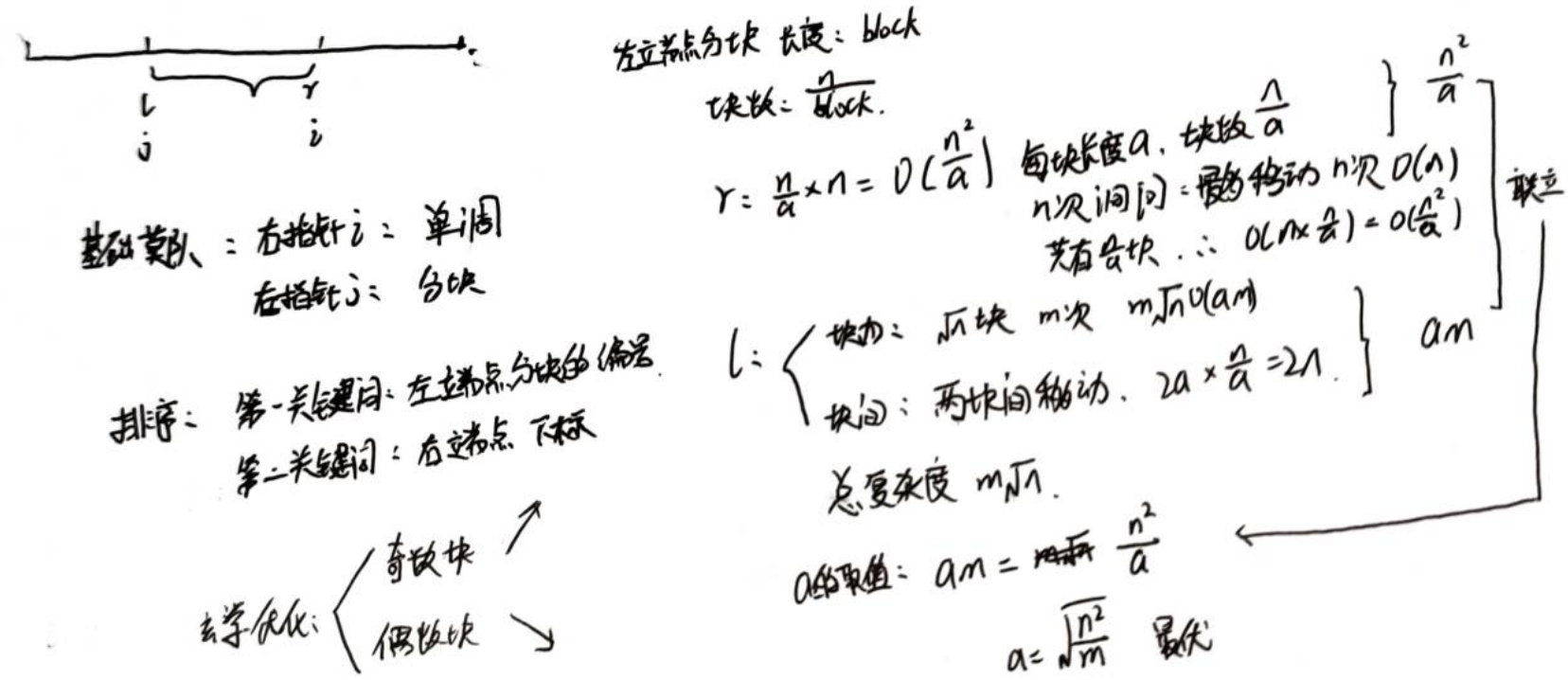
HH 相信不同的贝壳会带来好运，所以每次散步完后，他都会随意取出一段贝壳，思考它们所表达的含义。

HH 不断地收集新的贝壳，因此他的项链变得越来越长。

有一天，他突然提出了一个问题：某一段贝壳中，包含了多少种不同的贝壳？

这个问题很难回答，因为项链实在是太长了。

于是，他只好求助睿智的你，来解决这个问题。




```cpp
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <iostream>
#include <cmath>

using namespace std;
const int N = 50007, M = 200007, S = 1000007;

int n, m;
int w[N];
int block;
int cnt[S];
int ans[M];

struct Query{
    int id, l, r;
}q[M];

int get_block(int x){
    return x / block;//这里是从0开始
}

bool cmp(const Query& x, const Query& y){
    int a = get_block(x.l);
    int b = get_block(y.l);
    if(a != b)return a < b;
    return x.r < y.r;
}

void add(int x, int &res){
```

```
    if(cnt[x] == 0)res ++ ;
    cnt[x] ++ ;
}

void del(int x, int &res){
    cnt[x] -- ;
    if(cnt[x] == 0)res -- ;
}

int main()
{
    scanf("%d", &n);

    for(int i = 1; i ≤ n; ++ i) scanf("%d", &w[i]);

    scanf("%d", &m);
    block = sqrt((double)n * n / m);//1488 ms
    //block = sqrt(n);              //1700 ms

    for(int i = 0; i < m; ++ i){
        int l, r;
        scanf("%d%d", &l, &r);
        q[i] = {i, l, r};
    }
    sort(q, q + m, cmp);

    for(int k = 0, i = 0, j = 1, res = 0; k < m; ++ k){
        int id = q[k].id, l = q[k].l, r = q[k].r;
        while(i < r)add(w[ ++ i], res);
        while(i > r)del(w[i -- ], res);
        while(j < l)del(w[j ++ ], res);
        while(j > l)add(w[ -- j], res);//注意这里的细节，自己模拟一遍
        ans[id] = res;
    }

    for(int i = 0; i < m; ++ i)
        printf("%d\n", ans[i]);
    return 0;
}
```

玄学优化版，成功卡过了洛谷上的这道题

```
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <iostream>
#include <cmath>

using namespace std;
const int N = 1000007, M = 1000007, S = 1000007;

int n, m;
int w[N];
int block;
int cnt[S];
int ans[M];

inline int read()
{
    int x = 0, f = 1;
```

```
    char ch = getchar();
    while(ch > '9' || ch < '0'){if(ch == '-')f = -1;ch = getchar();}
    while(ch ≥ '0' && ch ≤ '9'){x = x * 10 + ch - '0';ch = getchar();}
    return x * f;
}

inline void write(int res){
    if(res<0){
        putchar('-');
        res=-res;
    }
    if(res>9)
        write(res/10);
    putchar(res%10+'0');
}

struct Query{
    int id, l, r;
}q[M];

inline int get_block(int x){
    return x / 2000;//这里是从0开始
}

bool cmp(const Query& x, const Query& y){
    int a = get_block(x.l);
    int b = get_block(y.l);
    //int a = x.l / block, b = y.l / block;
    if(a ≠ b)return a < b;
    if(a & 1)return x.r < y.r;
    return x.r > y.r;
}

inline void add(int x, int &res){
    if(cnt[x] == 0)res ++ ;
    cnt[x] ++ ;
}

inline void del(int x, int &res){
    cnt[x] -- ;
    if(cnt[x] == 0)res -- ;
}

int main()
{
    n = read();

    for(register int i = 1; i ≤ n; ++ i) w[i] = read();

    m = read();
    block = sqrt((double)n * n / m);//1488 ms
    //block = sqrt(n);              //1700 ms
    //block = 2000;
    for(register int i = 0; i < m; ++ i){
        int l = read(), r = read();
        q[i] = {i, l, r};
    }
    sort(q, q + m, cmp);

    for(register int k = 0, i = 0, j = 1, res = 0; k < m; ++ k){
```

```
79          int id = q[k].id, l = q[k].l, r = q[k].r;
80          while(i < r)add(w[ ++ i], res);
81          while(i > r)del(w[i -- ], res);
82          while(j < l)del(w[j ++ ], res);
83          while(j > l)add(w[ -- j], res);//注意这里的细节，自己模拟一遍
84          /*
85          while(i < r)res += ++ cnt[w[ ++ i]] == 1;
86          while(i > r)res -= -- cnt[w[i -- ]] == 0;
87          while(j < l)res -= -- cnt[w[j ++ ]] == 0;
88          while(j > l)res += ++ cnt[w[ -- j]] == 1;
89  */
90          ans[id] = res;
91      }
92
93      for(register int i = 0; i < m; ++ i)
94          write(ans[i]), puts("");
95      return 0;
96  }
```

2. 带修莫队

AcWing 2521. 数颜色

墨墨购买了一套 N 支彩色画笔（其中有些颜色可能相同），摆成一排，你需要回答墨墨的提问。

墨墨会像你发布如下指令：

1. `Q L R` 代表询问你从第 L 支画笔到第 R 支画笔中共有几种不同颜色的画笔。
2. `R P Col` 把第 P 支画笔替换为颜色 Col。

为了满足墨墨的要求，你知道你需要干什么了吗？

输入格式

第 1 行两个整数 N, M，分别代表初始画笔的数量以及墨墨会做的事情的个数。

第 2 行 N 个整数，分别代表初始画笔排中第 i 支画笔的颜色。

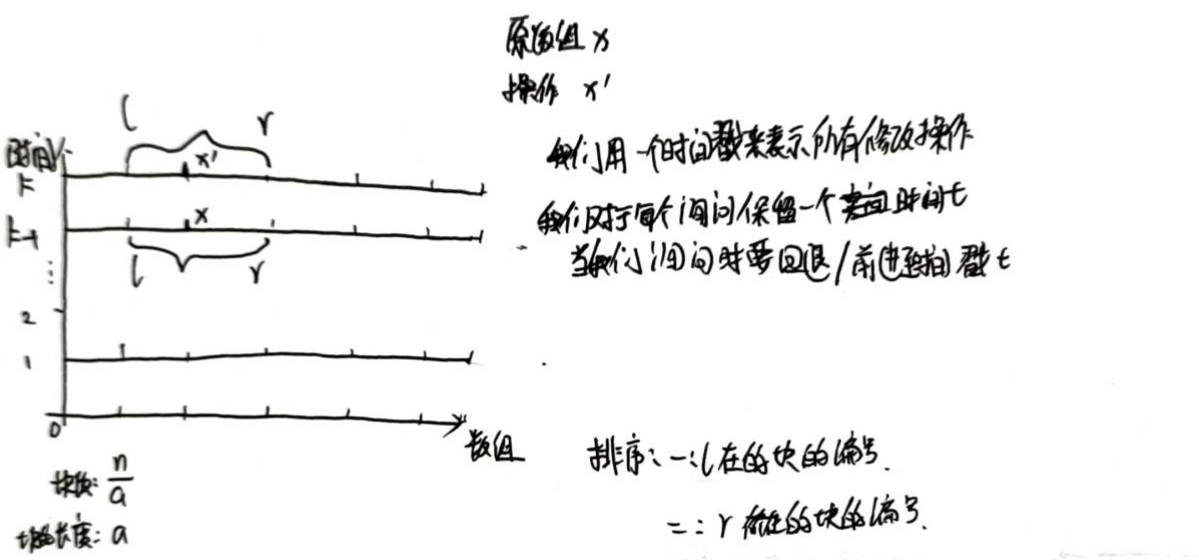
第 3 行到第 $2+M$ 行，每行分别代表墨墨会做的一件事情，格式见题干部分。

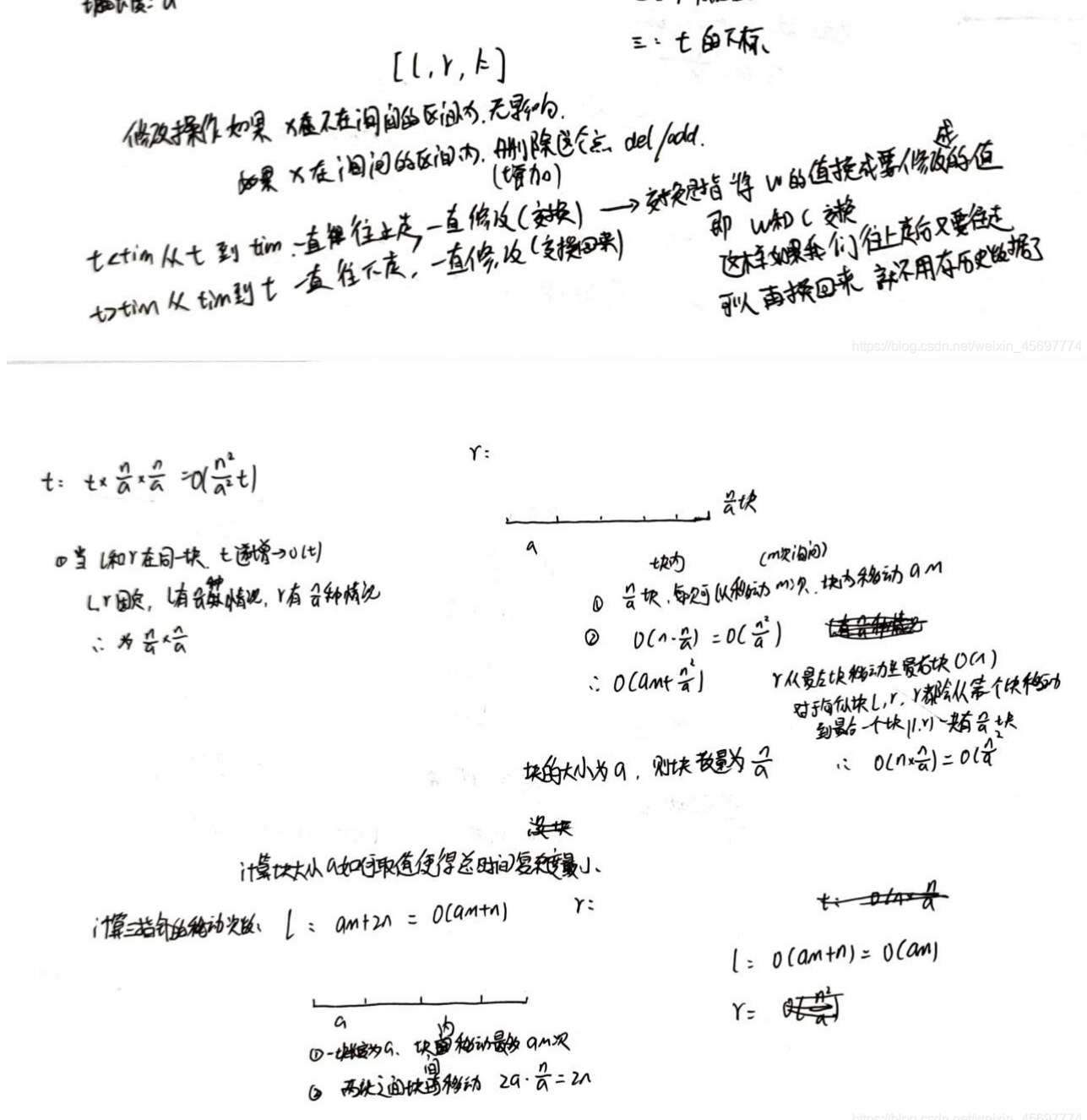
输出格式

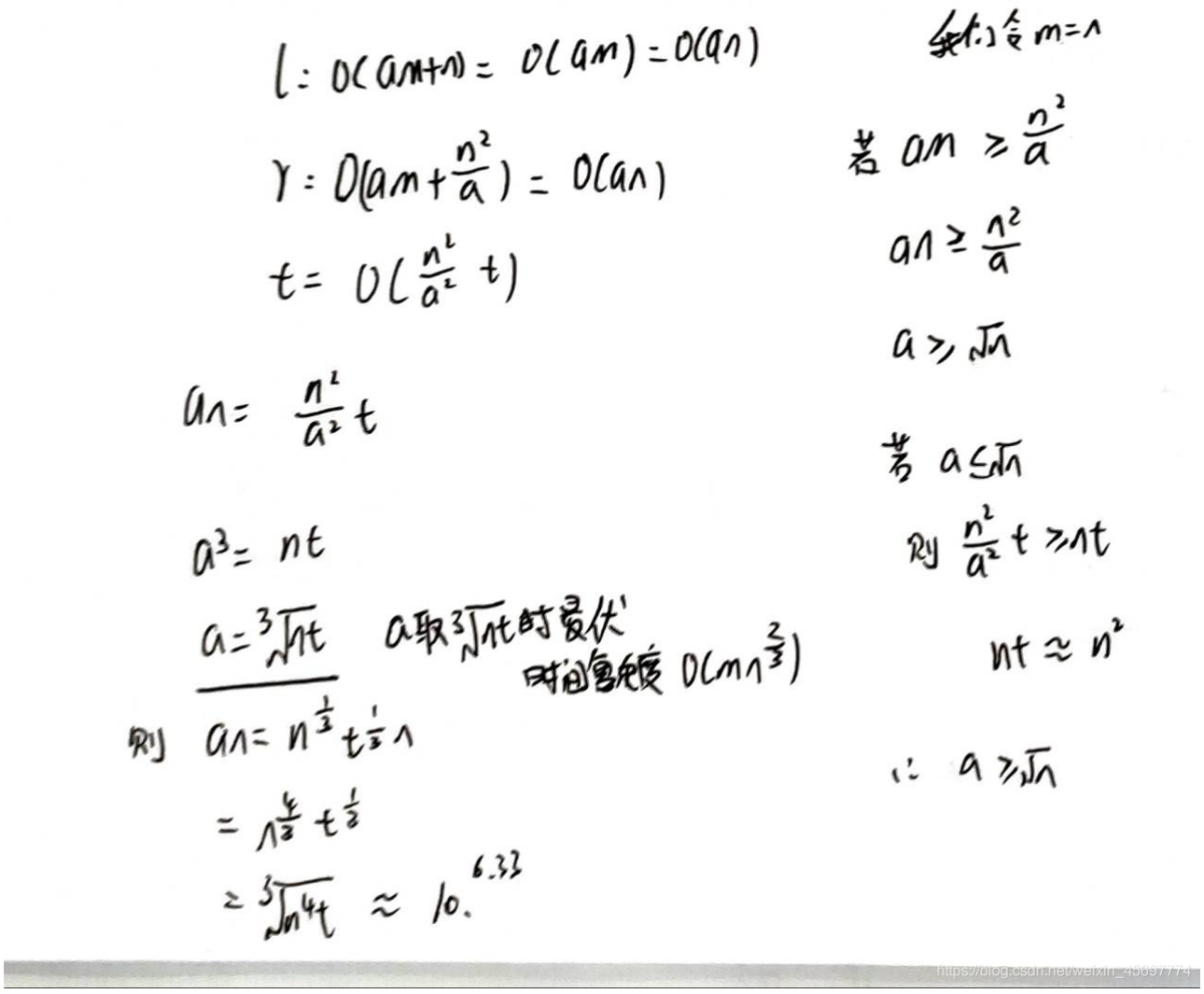
对于每一个 `Query` 的询问，你需要在对应的行中给出一个数字，代表第 L 支画笔到第 R 支画笔中共有几种不同颜色的画笔。

数据范围

$1 \leq N, M \leq 10000$,
修改操作不多于 1000 次，
所有的输入数据中出现的所有整数均大于等于 1 且不超过 10^6。







我发现直接把块的大小开成一个常数跑的最快...

```cpp
//#pragma GCC optimize(2)
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <iostream>
#include <cmath>
using namespace std;

const int N = 1000007, M = 1000007, S = 1000007;

inline int read()
{
    int x = 0, f = 1;
    char ch = getchar();
    while(ch > '9' || ch < '0'){if(ch == '-')f = -1;ch = getchar();}
    while(ch ≥ '0' && ch ≤ '9'){x = x * 10 + ch - '0';ch = getchar();}
    return x * f;
}

inline void write(int res){
    if(res<0){
        putchar('-');
        res=-res;
    }
    if(res>9)
        write(res/10);
    putchar(res%10+'0');
}

int n, m;
int block = 2589;//n ^ (2 / 3)
int w[N];
int cnt[S];
int ans[N];
int bi[N];
```

```cpp
struct Query{
    int id, l, r, t;
}q[M];

struct Modify{
    int pos, col, lst;
}c[M];

bool cmp(const Query &a, const Query &b){
    int al = bi[a.l], ar = bi[a.r];
    int bl = bi[b.l], br = bi[b.r];
    if(al ≠ bl)return a.l < b.l;
    if(ar ≠ br)return a.r < b.r;
    return a.t < b.t;
}

void add(int x, int& res){
    if(cnt[x] == 0)res ++ ;
    cnt[x] ++ ;
}

void del(int x, int& res){
    cnt[x] -- ;
    if(cnt[x] == 0)res -- ;
}

int main()
{
    n = read(), m = read();

    for(register int i = 1; i ≤ n; ++ i) w[i] = read();

    int mq = 0, mc = 0;
    for(register int i = 1; i ≤ m; ++ i){
        char op[2];
        int l, r;
        scanf("%s", op);
        l = read(), r = read();
        if(op[0] == 'Q'){
            q[ ++ mq] = (Query){mq, l, r, mc};
        }
        else {
            c[ ++ mc] = (Modify){l, r};
        }
    }
    //这里block一定要加1，可能出现0的情况导致除0发生浮点错误
    //block=ceil(exp((log(n)+log(mc))/3));//分块大小
    //block = cbrt(n * mc);
    //block = pow(n * n, 1.0 / 3);
    //block = pow(n, 2.0 / 3);
    for(int i = 1; i ≤ n; ++ i){
        bi[i] = (i - 1) / block;
    }
    sort(q + 1, q + 1 + mq, cmp);

    for(register int k = 1, i = 0, j = 1, t = 0, res = 0; k ≤ mq; ++ k){
        int id = q[k].id, l = q[k].l, r = q[k].r, tim = q[k].t;
        //先处理x轴
        /*
        while(i < r)add(w[ ++ i], res);
```

```
        while(i > r)del(w[i -- ], res);
        while(j < l)del(w[j ++ ], res);
        while(j > l)add(w[ -- j], res);
        */

        while(i < r)res += ++ cnt[w[ ++ i]] == 1;
        while(i > r)res -= -- cnt[w[i -- ]] == 0;
        while(j < l)res -= -- cnt[w[j ++ ]] == 0;
        while(j > l)res += ++ cnt[w[ -- j]] == 1;
        //再处理y轴
        while(t < tim){
            t ++ ;
            if(c[t].pos ≥ j && c[t].pos ≤ i){
                del(w[c[t].pos], res);
                add(c[t].col, res);
                //res -= !--cnt[w[c[t].pos]] - !cnt[c[t].col]++;
            }
            swap(w[c[t].pos], c[t].col);
        }
        while(t > tim){
            if(c[t].pos ≥ j && c[t].pos ≤ i){
                del(w[c[t].pos], res);
                add(c[t].col, res);
                //res -= !--cnt[w[c[t].pos]] - !cnt[c[t].col]++;
            }
            swap(w[c[t].pos], c[t].col);
            t -- ;
        }
        ans[id] = res;
    }

    for(register int i = 1; i ≤ mq; ++ i)
        write(ans[i]), puts("");
    return 0;
}
```

3. 回滚莫队

AcWing 2523. 历史研究

IOI 国历史研究的第一人——JOI 教授，最近获得了一份被认为是古代 IOI 国的住民写下的日记。

JOI 教授为了通过这份日记来研究古代 IOI 国的生活，开始着手调查日记中记载的事件。

日记中记录了连续 N 天发生的时间，大约每天发生一件。

事件有种类之分。第 i 天 $(1 \leq i \leq N)$ 发生的事件的种类用一个整数 X_i 表示，X_i 越大，事件的规模就越大。

JOI 教授决定用如下的方法分析这些日记：

1. 选择日记中连续的一些天作为分析的时间段
2. 事件种类 t 的重要度为 $t\times$ (这段时间内重要度为 t 的事件数)
3. 计算出所有事件种类的重要度，输出其中的最大值

现在你被要求制作一个帮助教授分析的程序，每次给出分析的区间，你需要输出重要度的最大值。

输入格式

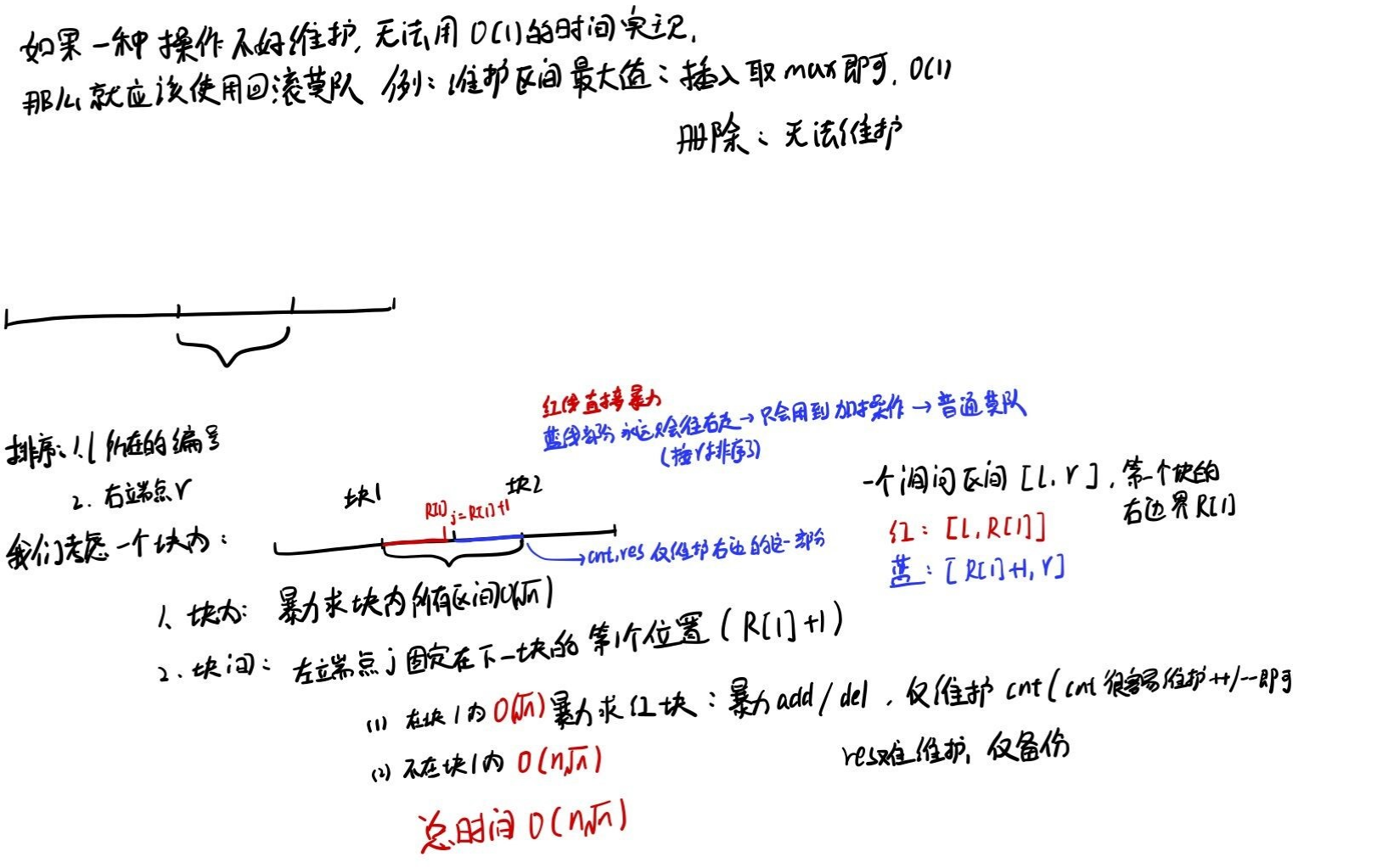
第一行两个空格分隔的整数 N 和 Q，表示日记一共记录了 N 天，询问有 Q 次。

接下来一行 N 个空格分隔的整数 $X_1 \dots X_N$，X_i 表示第 i 天发生的事件的种类。

接下来 Q 行，第 i 行 $(1 \leq i \leq Q)$ 有两个空格分隔整数 A_i 和 B_i，表示第 i 次询问的区间为 $[A_i, B_i]$。

输出格式

输出 Q 行，第 i 行 $(1 \leq i \leq Q)$ 一个整数，表示第 i 次询问的最大重要度。



左指针j，右指针i。

对于块外的情况，左端点一定是在块内的，如果有的询问的右端点是在块外，我们把i右定在right，j左端点定在right+1，我们i右端点一定只会一直向右走，因为我们是按照右端点升序排序的，块外的时候j往左走到实际位置左端点，只有增加操作，维护答案。

还是要删除的，但是删除的时候只需要维护cnt即可，res已经更新过了。

注意res只是当前块的右端点到中间的res，但是我们的答案是整个区间，所以我们备份一下中间到右端的res，往左走更新res，更新答案以后res回档为备份，相当于一个$O(1)$删除操作，我们每次更新所有左端点在一个块的好多个询问区间，所以每次cnt需要每次清零。

```cpp
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <iostream>
#include <vector>
#include <cmath>
```

```cpp
using namespace std;
typedef long long ll;
const int N = 200007;

int read()
{
    int x = 0, f = 1;
    char ch = getchar();
    while(ch > '9' || ch < '0') {if(ch == '-')f = -1;ch = getchar();}
    while(ch ≤ '9' && ch ≥ '0') {x = x * 10 + ch - '0';ch = getchar();}
    return x * f;
}

int n, m;
int bi[N];
int v[N];
int block;
ll ans[N];

struct Query
{
    int id, l, r;
}q[N];

bool cmp(const Query& x, const Query& y) {
    int a = bi[x.l], b = bi[y.l];
    if(a ≠ b) return a < b;
    return x.r < y.r;
}

vector<int>nums;
int cnt[N];

int get_block(int x)
{
    return x / block;
}

void add(int x, ll& res)
{
    cnt[x] ++ ;
    res = max(res, (ll)cnt[x] * nums[x]);
}

int main()
{
    n = read(), m = read();
    block = sqrt(n);
    for(int i = 1 ; i ≤ n; ++ i) {
        v[i] = read();
        nums.push_back(v[i]);
        bi[i] = i / block;
    }
    sort(nums.begin(), nums.end());
    for(int i = 1; i ≤ n; ++ i) {
        v[i] = lower_bound(nums.begin(), nums.end(), v[i]) - nums.begin();
    }

    for(int i = 0; i < m; ++ i) {
        int l = read(), r = read();
```

```
        q[i] = {i, l, r};
    }
    sort(q, q + m, cmp);

    for(int x = 0; x < m;) {
        //先找同一个块里的左右询问区间左x右y;
        int y = x;
        while(y < m && bi[q[y].l] == bi[q[x].l]) y ++ ;
        int right = bi[q[x].l] * block + block - 1; //当前块的右界
        while(x < y && q[x].r ≤ right) {
            ll res = 0;
            int id = q[x].id, l = q[x].l, r = q[x].r;
            for(int k = l; k ≤ r; ++ k)//O(sqrt(n))
                add(v[k], res);
            ans[id] = res;
            for(int k = l; k ≤ r; ++ k)
                cnt[v[k]] -- ;
            x ++ ;
        }
        ll res = 0;
        int i = right, j = i + 1;//j是左指针i是右指针
        while(x < y){
            int id = q[x].id, l = q[x].l, r = q[x].r;
            while(i < r) add(v[ ++ i], res);
            ll backup = res;//res只是当前块的右端点到右指针的答案，所以要备份
            while(j > l) add(v[ -- j], res);//这个询问区间的左端点一定在左块内部，我们就是这么排序的
            ans[id] = res;
            while(j < right + 1) cnt[v[j ++ ]] -- ;
            res = backup;
            x ++ ;
        }
        memset(cnt, 0, sizeof cnt);
    }

    for(int i = 0; i < m; ++ i)
        printf("%lld\n", ans[i]);
    return 0;
}
```

4. 树上莫队

AcWing 2534. 树上计数2

给定一棵 N 个节点的树，节点编号从 1 到 N，每个节点都有一个整数权值。

现在，我们要进行 M 次询问，格式为 `u v` ，对于每个询问你需要回答从 u 到 v 的路径上（包括两端点）共有多少种不同的点权值。

输入格式

第一行包含两个整数 N, M。

第二行包含 N 个整数，其中第 i 个整数表示点 i 的权值。

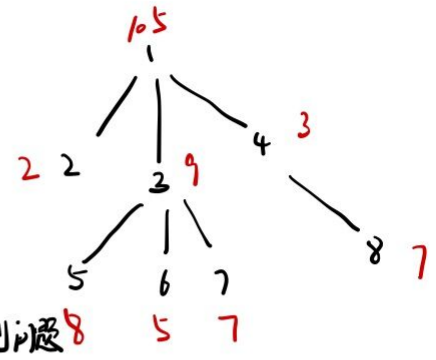
接下来 $N-1$ 行，每行包含两个整数 x, y，表示点 x 和点 y 之间存在一条边。

最后 M 行，每行包含两个整数 u, v，表示一个询问。

输出格式

共 M 行，每行输出一个询问的答案。

树上莫队　　树上一条路径有多少不同的颜色



将树上问题转化为数列问题

① 树链剖分

② 欧拉序列：对树的一次DFS序列.

(1 2 (2 3 5) 5 6 6 7 7 3 4 8) 8 4 1　　每个点会写两次.

例：1~8　1→4→8　欧拉序列中除了真正路径 1,4,8 以外其它点都出现了两次

例：2~5　2→1→3→5

任给 x,y，记录 first[u]：u第1次出现的位置

Last[u]：u最后一次出现的位置

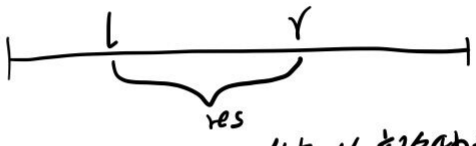
x,y. 满足 first[x] < first[y]　(不满足可以swap)

① LCA(x,y)=x

则对应在欧拉序列中 [first[x], first[y]] 中只出现一次的点

② Lca(x,y)≠x

则对应欧拉序列中 [Last[x], first[y]] 中只出现一次的点，以及 Lca(x,y)



维护数列中只出现一次的点的绝种类

莫队
- add
  - int cnt[ ];　add(x)
  - int vis[ ];　vis(x) ^=1
  - int res;　if(vis(x)==0) 删除x
- del
  - 我们发现删除一个数也可以 add(x)
  - 使得 vis(x) ^=1

① 离散化

② 求欧拉序列

③ 求 Lca (倍增) ← deep[u]
f[u][k]　(f[u][k] = f[f[u][k-1]][k-1])

④ 将 树上 问问转换为 序列中 i 问问

⑤ 莫队


```cpp
#include <cstdio>
#include <algorithm>
#include <cstring>
#include <iostream>
#include <vector>
#include <cmath>
//如果顺序有关系，涉及修改，需要用树链剖分
using namespace std;
//不涉及修改，没有顺序关系的可以用欧拉序列
const int N = 500007, M = 500007, INF = 0x3f3f3f3f;
int n, m;
int cnt[N], vis[N];
vector<int>v;
int f[N][20], dep[N];
int head[N], ver[M], nex[M], tot;
int w[M];
```

```cpp
int seq[N], top, first[N], last[N];
int block;
int ans[N];

void add_edge(int x, int y)
{
    ver[tot] = y;
    nex[tot] = head[x];
    head[x] = tot ++ ;
}

struct Query{
    int id, l, r, p;
}q[N];

int get_block(int x)
{
    return x / block;
}

bool cmp(Query &a, Query &b)
{
    int x = get_block(a.l);
    int y = get_block(b.l);
    if(x ≠ y)return x < y;
    return a.r < b.r;
}

void dfs(int x, int fa){
    seq[ ++ top] = x;
    first[x] = top;
    for(int i = head[x]; ~i; i = nex[i]){
        int y = ver[i];
        if(y == fa) continue;
        dfs(y, x);
    }
    seq[ ++ top] = x;
    last[x] = top;
}

int que[N];

void bfs()
{
    memset(dep, 0x3f, sizeof dep);
    int hh = 0, tt = 0;
    que[0] = 1;
    dep[0] = 0, dep[1] = 1;
    while(hh ≤ tt){
        int x = que[hh ++ ];
        if(hh == N) hh = 0;
        for(int i = head[x]; ~i; i = nex[i]){
            int y = ver[i];
            if(dep[y] > dep[x] + 1){
                dep[y] = dep[x] + 1;
                f[y][0] = x;
                for(int k = 1; k ≤ 15; ++ k){
                    f[y][k] = f[f[y][k - 1]][k - 1];
                }
                que[ ++ tt] = y;
```

```
                if(tt == N) tt = 0;
            }
        }
    }
}

int lca(int x, int y)
{
    if(dep[x] < dep[y]) swap(x, y);
    for(int k = 15; k ≥ 0; -- k){
        if(dep[f[x][k]] ≥ dep[y]){
            x = f[x][k];
        }
    }
    if(x == y) return x;
    for(int k = 15; k ≥ 0; -- k){
        if(f[x][k] ≠ f[y][k]){
            x = f[x][k];
            y = f[y][k];
        }
    }
    return f[x][0];
}

void add(int x, int &res)
{
    //!欧拉序列中出现两次就不是路径上的点了!要删掉
    //要删掉的点一定是只出现一次的，添加的时候add一次，删除的时候add一次，两次即为删除
    vis[x] ^= 1;//需要的是点的编号
    if(vis[x] == 0){
        cnt[w[x]] -- ;//需要的是点的权值（离散化过了）
        if(cnt[w[x]] == 0) res -- ;
    }
    else {
        cnt[w[x]] ++ ;
        if(cnt[w[x]] == 1) res ++ ;
    }
}

int main()
{
    scanf("%d%d", &n, &m);
    for(int i = 1; i ≤ n; ++ i){
        scanf("%d", &w[i]);
        v.push_back(w[i]);
    }
    sort(v.begin(), v.end());
    v.erase(unique(v.begin(), v.end()), v.end());
    for(int i = 1; i ≤ n; ++ i){
        w[i] = lower_bound(v.begin(), v.end(), w[i]) - v.begin();
    }

    memset(head, -1, sizeof head);

    for(int i = 1; i ≤ n - 1; ++ i){
        int x, y;
        scanf("%d%d", &x, &y);
        add_edge(x, y);
        add_edge(y, x);
    }
```

```

    dfs(1, -1);//得到欧拉序列

    bfs();//lca预处理

    for(int i = 0; i < m; ++ i){
        int a, b;
        scanf("%d%d", &a, &b);
        //a,b是树上的点
        //first[a], first[b], last[a], last[b]才是数列上的点，也是我们莫队要处理的点
        if(first[a] > first[b]) swap(a, b);
        int p = lca(a, b);
        if(a == p)
            q[i] = {i, first[a], first[b]};
        else q[i] = {i, last[a], first[b], p};
    }
    block = sqrt(top);//这里应该是欧拉序列里的点的个数
    sort(q, q + m, cmp);
    //右指针i左指针j，右指针先冲左指针跟上
    //左指针在1，右指针在0，初始状态形成一个空集
    for(int k = 0, i = 0, j = 1, res = 0; k < m; ++ k){
        int l = q[k].l, r = q[k].r, id = q[k].id, p = q[k].p;
        //这里走的应该是欧拉序列里的点了
        while(i < r) add(seq[ ++ i], res);//add
        while(i > r) add(seq[i -- ], res);//del
        while(j < l) add(seq[j ++ ], res);//del
        while(j > l) add(seq[ -- j], res);//add
        if(p) add(p, res);
        ans[id] = res;
        if(p) add(p, res);//最后一定要删除p，因为它不属于 i 到 i 这一连续序列中
    }

    for(int i = 0; i < m; ++ i)
        printf("%d\n", ans[i]);
    return 0;
}
```

五、图论

$\oint$ 1. 最短路

优先队列最短路

```
struct Edge
{
    ll v,w,next;//v:目的地,w:距离,next:下一个节点
}G[N];
ll head[N],cnt,n,m,s;
ll dis[N];//存距离
inline void addedge(ll u,ll v,ll w)//链式前向星存图
{
    cnt++;
    G[cnt].w=w;
    G[cnt].v=v;
    G[cnt].next=head[u];
    head[u]=cnt;
```

```
}
struct node
{
    ll d,u;//d是距离u是起点
    bool operator<(const node& t)const//重载运算符
    {
        return d>t.d;
    }
};
inline void Dijkstra()
{
    for(register int i=1;i≤n;++i)dis[i]=mod;//初始化
    dis[s]=0;
    priority_queue<node>q;//堆优化
    q.push((node){0,s});//起点push进去
    while(!q.empty())
    {
        node tmp=q.top();q.pop();
        ll u=tmp.u,d=tmp.d;
        if(d≠dis[u])continue;//松弛操作剪枝
        for(register int i=head[u];i;i=G[i].next)//链式前向星
        {
            ll v=G[i].v,w=G[i].w;
            if(dis[u]+w<dis[v])//符合条件就更新
            {
                dis[v]=dis[u]+w;
                q.push((node){dis[v],v});//沿着边往下走
            }
        }
    }
}
int main()
{
    scanf("%lld %lld %lld",&n,&m,&s);
    for(register int i=1;i≤m;++i)
    {
        ll x,y,z;
        scanf("%lld %lld %lld",&x,&y,&z);
        addedge(x,y,z);//建图
    }
    Dijkstra();
    for(register int i=1;i≤n;++i)
        printf("%lld ",dis[i]);
    printf("\n");
    return 0;
}

```

1.1 线段树优化的*Dijkstra*

时间和内存均是优先队列优化版本的 $\frac{1}{2}$

```
int n, m;
struct edge {
    int to, w, nxt;
    edge() {}
    edge(int t, int ww, int nn) {to = t, w = ww, nxt = nn;}
}e[maxn << 1];

int head[maxn], k = 0;
```

```cpp
void add(int u, int v, int w) {e[k] = edge(v, w, head[u]); head[u] = k++;}

ll ans[maxn];
struct node {
    ll dis; int x;
    node() {}
    node(ll d, int xx) {dis = d, x = xx;}
}dis[maxn << 2];

//建树初始化，主要是编号也要返回所以要先预处理一下
void build(int p, int l, int r) {
    if(l == r) {dis[p].x = l; return;}
    int mid = l + r >> 1;
    build(p << 1, l, mid); build(p << 1 | 1, mid + 1, r);
    dis[p].x = dis[p << 1].x;
}

void change(int p, int l, int r, int x, int y) {
    if(l == r) {dis[p].dis = y; return;}
    int mid = l + r >> 1;
    if(x ≤ mid) change(p << 1, l, mid, x, y);
    else change(p << 1 | 1, mid + 1, r, x, y);//单点修改的板子操作
    if(dis[p << 1].dis < dis[p << 1 | 1].dis) dis[p] = dis[p << 1];
    else dis[p] = dis[p << 1 | 1];
}

//因为用距离得到最小，但是需要的是编号，所以返回node
node ask(int p, int l, int r, int ls, int rs) {
    if(ls ≤ l && r ≤ rs) {return dis[p];}
    int mid = l + r >> 1; node ans = node(inf, 0), tmp;
    if(ls ≤ mid) ans = ask(p << 1, l, mid, ls, rs);
    if(rs > mid) {
        node tmp = ask(p << 1 | 1, mid + 1, r, ls, rs);
        if(ans.dis > tmp.dis) ans = tmp;
    }
    return ans;
}

int S;
void dij() {
    for(int k = 1; k < n; k++) {//n-1次够用的。虽然我也不知道为什么最后n次跑的比n-1次还要快……
        register int u = ask(1, 1, n, 1, n).x;
        for(int i = head[u]; ~i; i = e[i].nxt) {
            register int v = e[i].to;
            if(ans[u] + e[i].w < ans[v]) {//最短路更新
                ans[v] = ans[u] + e[i].w, change(1, 1, n, v, ans[v]);//单点修改
            }
        }
        change(1, 1, n, u, inf);//取出来过后要赋值INF，以免再次取用
    }
}

int main() {
    memset(head, -1, sizeof head);
    n = read(), m = read(), S = read();
    for(int u, v, w, i = 1; i ≤ m; i++) u = read(), v = read(), w = read(), add(u, v, w);

    //初始化
    for(int i = 1; i ≤ (n << 2); i++) dis[i].dis = inf;
    for(int i = 1; i ≤ n; i++) ans[i] = inf;
```

```
69
70      //线段树初始化，dis是线段树，ans是答案
71      build(1, 1, n);
72      change(1, 1, n, S, 0); ans[S] = 0;
73      dij();
74
75      for(int i = 1; i ≤ n; i++) printf("%lld ", ans[i]);
76      return 0;
77  }
78
```

1.1.1 路径还原

白书上的代码，没有优化是$O(n^2)$，正序输出最短路径

```
1  /*如果需要输出路径
2  可以用一个prev[ j ]来记录最短路上顶点 j 的前驱，
3  那么在o(|V|)的时间内完成最短路径的恢复。
4  在d[ j ]被d[ j ] = d[ k ] + cost[ k ][ j ]更新时，
5  修改prev[ j ] = k，这样就可以求出来prev的数组，
6  可以在以上算法中用路径前驱标记法来还原出来路径。*/
7
8  int cost[MAX_V][MAX_V];  //cost[u][v]表示边e=(u, v)的权值（不存在这条边时设为INF）
9  int d[MAX_V];            //从顶点s出发的最短距离
10 bool used[MAX_V];        //已经使用过的图中的点
11 int V;                   //顶点数
12 int prev[MAX_V];         //记录前驱点
13 //从s出发到各个顶点的距离
14 void dijkstra(int s)
15 {
16     fill(d, d + V, INF);             //初始化
17     fill(used, used + V, false);     //初始化
18     fill(prev, prev + V, -1);
19     d[s] = 0;
20     while(true)
21     {
22         int v = -1;
23         for(int u = 0; u < V; u++){
24             if(!used[u] && (v == -1 || d[u] < d[v])) v = u;
25             // 从尚未使用过的顶点中选择一个距离最小的顶点
26         }
27
28         if(v == -1) break; //没有可跟新的了，结束
29
30         used[v] = true;
31
32         for(int u = 0; u < V; u++){
33             d[u] = min(d[u], d[v] + cost[u][v]);   //因为加入了一个点V所以所有的d都要再更新一遍
34             prev[u] = v;
35         }
36     }
37 }
38 vector<int> get_path(int t) //到顶点t的最短路
39 {
40     vector<int> path;
41     for(; t ≠ -1; t = prev[t])
42         path.push_back(t);
43     reverse(path.begin(), path.end());
44     return path;
45 }
```

UVA10537 The Toll! Revisited
我们要注意去的时候交的n/20单位的货物，但是倒着回去求答案的时候我们要加上`ceil(n/19.0)`（自己手算）

字典序中大写字母比小写字母更小

注意看数据范围，别总用memset容易超时TLE

```cpp
//注意字典序中大写字母比小写字母小
typedef long long ll;
typedef pair<ll, int>PLI;
const int N = 100007, M = 5000007;
const ll INF = 1e17;

int n, m;
ll dist[N], num;
int head[N], ver[M], nex[M], tot;
bool vis[M];
int S, T;
int pre[N];
//直接把字符转成数字更加方便，最后输出的时候还可以强转回来
void add(int x, int y)
{
    ver[tot] = y;
    nex[tot] = head[x];
    head[x] = tot ++;
}

void dijkstra()
{
    for(int i = 1; i ≤ 150; ++ i)
        pre[i] = 0, dist[i] = INF, vis[i] = 0;
    dist[T] = num;
    priority_queue<PLI, vector<PLI>, greater<PLI> >q;
    q.push({dist[T], T});

    while(q.size())
    {
        int x = q.top().second;
        q.pop();
        if(vis[x])continue;
        vis[x] = 1;
        for(int i = head[x]; ~i; i = nex[i]){
            int y = ver[i];
            if(x > 95){//这里是x因为我们是倒着走的，所以每次从x出来我们得到的货物应该按照x是城镇还是村庄来定
                if(dist[y] > dist[x] + 1 || (dist[y] == dist[x] + 1 && pre[y] > x)){
                //找字典序最小的每一步都是越小越好所以判断:(dist[y] == dist[x] + 1 && pre[y] > x)
                    dist[y] = dist[x] + 1;
                    q.push({dist[y], y});
                    pre[y] = x;
                }
            }
            else {
                ll z = ceil((double)dist[x] / 19.0);
                if(dist[y] > dist[x] + z || (dist[y] == dist[x] + z && pre[y] > x)){
                    dist[y] = dist[x] + z;
                    q.push({dist[y], y});
                    pre[y] = x;
                }
            }
        }
```

```
54
55      }
56  }
57
58  void dfs(int x)
59  {
60      if(pre[x]){
61          printf("%c-", (char)x);
62          dfs(pre[x]);
63      }
64      return ;
65  }
66
67  int kcase;
68  int main()
69  {
70      //ios::sync_with_stdio(false);
71      //cin.tie(0);
72      while(scanf("%d", &n) ≠ EOF && n ≠ -1){
73          memset(head, -1, sizeof head);
74          tot = 0;
75          for(int i = 1; i ≤ n; ++i){
76              char a, b;
77              int x, y;
78              cin >> a >> b;
79              x = a, y = b;
80              add(x, y);
81              add(y, x);
82          }
83          char a, b ;
84          cin >> num >> a >> b;
85          S = a, T = b;
86          dijkstra();
87          printf("Case %d:\n", ++ kcase);
88          printf("%lld\n", dist[S]);
89          dfs(S);
90          printf("%c\n", (char)T);
91      }
92      return 0;
93  }
```

1.2 可以处理负权值的*SPFA*

SPFA要谨慎使用!

```
1  int nex[M],ver[M],head[N],edge[M],tot;
2
3  void add(int u,int v,int val){
4      ver[++tot] = v;
5      edge[tot] = val;
6      nex[tot] = head[u];
7      head[u] = tot;
8  }
9
10 queue<int>q;
11 bool vis[N];
12 int d[N];
13 int n,m;
14 void spfa(int s){
15     //memset(d,0x3f,sizeof d);
```

```
    for(int i = 1; i≤ n;++i)
        d[i] = 2147483647;
    memset(vis,0,sizeof vis);
    d[s] = 0;
    vis[s] = 1;
    q.push(s);
    while(q.size()){
        int x = q.front();
        q.pop();
        vis[x] = 0;
        //扫描所有出边
        for(int i = head[x];i;i = nex[i]){
            int y = ver[i];
            int z = edge[i];
            if(d[y] > d[x] + z){
                d[y] = d[x] + z;
                if(!vis[y])//不在堆里就入堆并标记一下
                    q.push(y),vis[y] = 1;
            }
        }
    }
}

int s;

int main()
{
    cin>>n>>m>>s;
    over(i,1,m){
        int x,y,z;
        scanf("%d%d%d",&x,&y,&z);
        add(x,y,z);
    }
    spfa(s);
    over(i,1,n){
        printf("%d ",d[i]);
    }
    return 0;
}
```

1.2.1 SPFA的SLF优化

```
void spfa(int s)
{
    memset(dis,0x3f,sizeof(dis));
    memset(vis,false,sizeof(vis));
    deque<int> q;
    dis[s]=0;
    vis[s]=true;
    q.push_back(s);
    while(q.size())
    {
        int now=q.front();
        vis[now]=false;
        q.pop_front();
        for(int i=head[now]; ~i; i=Next[i])
        {
            int j=edge[i];
            if (dis[j]>dis[now]+ver[i])
```

```
            {
                dis[j]=dis[now]+ver[i];
                if (!vis[j])
                {
                    vis[j]=true;
                    ////////////////////////SLF优化实际上只是增加了这两句判断
                    if (q.size() && dis[j]<dis[q.front()])
                        q.push_front(j);
                    ////////////////////////////////////////////////////////
                    else
                        q.push_back(j);
                }
            }
        }
    }
}
```

1.2.2 SPFA的LLL优化

```
void SPFA()
{
    memset(dis,inf,sizeof(dis));
    queue<int>q;
    q.push(1);dis[1]=0;vis[1]=1;
    while(q.size())
    {
        p=q.front();q.pop();
        if(dis[p]*cnt_2>sum)
        {
            q.push(p);
            continue;
        }
        sum-=dis[p];cnt_2--;
        vis[p]=0;
        for(int i=head[p];i;i=map[i].next)
        {
            s=map[i].to;
            if(dis[s]>dis[p]+map[i].value)
            {
                dis[s]=dis[p]+map[i].value;
                if(vis[s]==0)
                {
                    vis[s]==1;
                    q.push(s);
                    cnt_2++;
                    sum+=dis[s];
                }
            }
        }
    }
}
```

1.2.3 SPFA的DFS优化（适用于负环的判断）

```
inline bool spfa(int x)
{
    dfs[x]=1;
    for(int i=head[x];i;i=g[i].next)
```

```
    {
        int y=g[i].ver,z=g[i].edge;
        if(!v[y]||d[y]<d[x]+z){
            if(dfs[y]) return 0;
            v[y]=1;
            d[y]=d[x]+z;
            if(!spfa(y)) return 0;
        }
    }
    dfs[x]=0;
    return 1;
}
```

1.3. 两种分层图最短路

1.3.1 分层次

> Alice 和 Bob 现在要乘飞机旅行，他们选择了一家相对便宜的航空公司。该航空公司一共在nn个城市设有业务，设这些城市分别标记为 0 到
> n−1，一共有 m 种航线，每种航线连接两个城市，并且航线有一定的价格。 Alice 和 Bob
> 现在要从一个城市沿着航线到达另一个城市，途中可以进行转机。航空公司对他们这次旅行也推出优惠，他们可以免费在最多 k 种航线上搭乘飞机。那么
> Alice 和 Bob 这次出行最少花费多少?

分成$k+1$层图

```
int nex[N],ver[N],head[N],edge[N],tot;
int n,m,k,s,t;
int dist[N];
bool vis[N];

void init(){
    memset(head,-1,sizeof head);
    tot = 0;
    memset(dist,0x3f,sizeof dist);
    memset(vis,0,sizeof vis);

}

void add(int u,int v,int val){
    ver[++tot] = v;
    edge[tot] = val;
    nex[tot] = head[u];
    head[u] = tot;
}

void dijkstra(){
    priority_queue<PII,vector<PII>,greater<PII> >q;
    dist[1] = 0;
    //vis[s] = 1;dijkstra和spfa不一样！这里不用标记，不然就走不动啦
    q.push({dist[1],1});
    while(q.size()){
        PII now = q.top();//这种声明的变量后面不要用逗号运算符！！！
        q.pop();
        int u = now.second;
        if(vis[u])continue;
        vis[u] = 1;
        for(int i = head[u]; i =≠ -1 ;i = nex[i]){
            int v = ver[i],val = edge[i];
```

```
            if(!vis[v] && dist[v] > dist[u] + val){
                dist[v] = dist[u] + val;
                q.push({dist[v],v});
            }
        }
    }
}

int main()
{
    init();
    scanf("%d%d%d",&n,&m,&k);
    over(i,1,m){
        int x,y,z;
        scanf("%d%d%d",&x,&y,&z);
        for(int j = 0;j ≤ k;++j){
            add(x + j * n,y + j * n,z);//这一层和这一层连起来
            add(y + j * n,x + j * n,z);
            if(j ≠ k){//只要还有就这一层和下一层连起来，并且权值为0
                add(x + j * n,y + (j + 1) * n,0);
                add(y + j * n,x + (j + 1) * n,0);
            }
        }
    }
    dijkstra();
    int ans = INF;//dijkstra只是更新了一个dis数组，答案需要我们自己找
    over(i,0,k)
    ans = min(ans,dist[n + i * n]);
    printf("%d\n",ans);
    return 0;
}

```

1.3.2分维度

> 在郊区有 N 座通信基站，P 条 双向 电缆，第 i 条电缆连接基站Ai和Bi。特别地，1 号基站是通信公司的总站，N 号基站位于一座农场中。现在，农场主希望对通信线路进行升级，其中升级第 i 条电缆需要花费Li。 电话公司正在举行优惠活动。 农产主可以指定一条从 1 号基站到 N 号基站的路径，并指定路径上不超过 K 条电缆，由电话公司免费提供升级服务。农场主只需要支付在该路径上剩余的电缆中，升级价格最贵的那条电缆的花费即可。 求至少用多少钱可以完成升级。

这里是因为题目要求的答案是第k+1的权值最小，所以只求单条路径的权值即可，但是这样的更新并不能满足dijkstra的贪心性质，所以我们要选择SPFA算法。

```
/*POJ 3662 Telephone Lines*/
int d[N][N];
int n,m,k;
int nex[M],ver[M],head[N],edge[M],tot;

void add(int u,int v,int val){
    ver[++tot] = v;
    edge[tot] = val;
    nex[tot] = head[u];
    head[u] = tot;
}

queue<int>q;
bool vis[N];

void spfa(int s){
    memset(d,0x3f,sizeof d);
```

```
    memset(vis,0,sizeof vis);
    d[s][0] = 0;
    vis[s] = 0;
    q.push(s);
    while(q.size()){
        int x = q.front();
        q.pop();
        vis[x] = 0;
        for(int i = head[x];i;i = nex[i]){
            int y = ver[i],z = edge[i];
            int w = max(d[x][0],z);//不同于往常的是每一条路径的 最大边 是这条路径的花费
            if(d[y][0]>w){//先算普通的最短路
                d[y][0] = w;
                if(!vis[y])
                    q.push(y),vis[y] = 1;
            }
            //反正就是动规，暴力所有状态转移取最小，状态转移的方式就是可以免费。
            //每次对每种路径都取一次最优
            for(int p = 1;p <= k;++p){//然后再看免费的，不一定是几条边最优//不管怎么说肯定是从前往后地推出的
                int w = min(d[x][p-1],max(d[x][p],z));//使最大边最小//主要这里的答案都是一条边的权值
                //当前的状态是从x转移到y，其中这条边免费或者不免费
                if(d[y][p] > w){
                    d[y][p] = w;
                    if(!vis[y])
                        q.push(y),vis[y] = 1;
                }
            }
        }
    }

int main(){
    cin>>n>>m>>k;
    over(i,1,m){
        int x,y,z;
        scanf("%d%d%d",&x,&y,&z);
        add(x,y,z);
        add(y,x,z);
    }
    spfa(1);
    int ans = 1e9+7;
    over(i,0,k){//注意是从0开始，因为也可以不选
        //就真答案并不在掌握之中
        //其实就是分层图，应该是 k + 1 层
        ans = min(ans,d[n][i]);
    }
    if(ans == 1e9+7)
        puts("-1");
    else printf("%d\n",ans);
    return 0;
}
```

1.4 Floyd算法，传递闭包

Floyd算法

设$D[k,i,j]$表示“经过若干个编号不超过k的结点”从i到j的最短路长度。

Floyd算法的实质是动态规划。

k是阶段，所以必须放在最外层循环。

i和j是附加状态，应该放置于内层循环。

```cpp
int dis[1000][1000],n,m;
int main()
{
    cin>>n>>m;
    memset(dis,0x3f,sizeof dis);
    for(int i = 1;i ≤ n;++i)
        dis[i][i] = 0;
    for(int i = 1;i ≤ m;++i){
        int x,y,z;
        scanf("%d%d%d",&x,&y,&z);
        dis[x][y] = min(dis[x][y],z);//可能会重复
    }
    for(int k = 1;k ≤ n;++k)
        for(int i = 1;i ≤ n;++i)
            for(int j = 1;j ≤ n;++j)
                dis[i][j] = min(dis[i][j],dis[i][k] + dis[k][j]);
    for(int i = 1;i ≤ n;++i)
        for(int j = 1;j ≤ n;++j)
            printf("%d ",dis[i][j]);
    return 0;
}
```

使用Floyd实现的传递闭包

```cpp
int dis[1000][1000],n,m;

int main()
{
    cin>>n>>m;
    for(int i = 1;i ≤ n;++i)
        dis[i][i] = 1;
    for(int i = 1;i ≤ m;++i){
        int x,y;
        scanf("%d%d",&x,&y);
        dis[x][y] = dis[y][x] = 1;
    }
    for(int k = 1;k ≤ n;++k)
        for(int i = 1;i ≤ n;++i)
            for(int j = 1;j ≤ n;++j)
                dis[i][j] |= dis[i][k] & dis[k][j];
                //把整个关系梳理出来
}
```

1.5 最小环问题

给定一张无向图，求图中一个至少包含3个点的环，环上的节点不重复，并且环上的边的长度之和最小。
该问题称为无向图的最小环问题。
$1 \leq N \leq 100, 1 \leq M \leq 10000, 1 \leq l < 500$

你需要输出最小环的方案，若最小环不唯一，输出任意一个均可。
获取最小环时j应当大于i，由对称性可知最终可求出正确答案.
环的性质是dis[i][j]+a[j][k]+a[k][i]，即呈环状.
需要开long long.

```cpp
#include<iostream>
#include<algorithm>
#include<cstdio>
#include<cstring>
```

```cpp
#include<bitset>
#include<vector>

#define over(i,s,t) for(register int i = s;i ≤ t;++i)
#define lver(i,t,s) for(register int i = t;i ≥ s;--i)
//#define int __int128
#define lowbit(p) p&(-p)
using namespace std;

typedef long long ll;
typedef pair<int,int> PII;

const int INF = 0x3f3f3f3f;
const int N = 1e2+7;
const int M = 2e6+7;

vector<int>path;
int n,m;
int dis[N][N];//存最终的最短路长度
int a[N][N];//存单边的数据，因为要至少3条，所以应该更新的时候用单边
int pos[N][N];//存的是两点之间借的路

void get_path(int i,int j){
    if(!pos[i][j])return ;
    get_path(i,pos[i][j]);//可能左边还有，要按顺序
    path.push_back(pos[i][j]);
    get_path(pos[i][j],j);
}

int main()
{
    scanf("%d%d",&n,&m);
    memset(a,0x3f,sizeof a);
    over(i,1,n)a[i][i] = 0;
    over(i,1,m){
        int x,y,z;
        scanf("%d%d%d",&x,&y,&z);
        a[x][y] = a[y][x] = min(a[x][y],z);
    }
    int ans = INF;
    memcpy(dis,a,sizeof a);
    over(k,1,n){
        over(i,1,k-1)over(j,i+1,k-1){
            if(ans > dis[i][j] + a[j][k] + a[k][i]){//至少3条边
                ans = dis[i][j] + a[j][k] + a[k][i];//注意下标这里是一个环
                path.clear();
                path.push_back(i);//按顺序放入
                get_path(i,j);//得到从i到j为了得到这样的最短路中实际借用的那些边
                path.push_back(j);
                path.push_back(k);
            }
        }
        over(i,1,n)over(j,1,n){
            if(dis[i][j] > dis[i][k] + dis[k][j]){
                dis[i][j] = dis[i][k] + dis[k][j];
                pos[i][j] = k;
            }
        }
    }
    if(ans == INF){
```

```
65          puts("No solution.");
66          return 0;
67      }
68      over(i,0,path.size()-1)
69      printf("%d ",path[i]);
70      puts("");
71      return 0;
72
73  }
74
```

∮ 2. 最长路

最长路径算法

（简而言之不用Dijkstra，板子SPFA直接上）

1. 肯定不能用dijkstra算法，这是因为，Dijkstra算法的大致思想是每次选择距离源点最近的结点加入，然后更新其它结点到源点的距离，直到所有点都被加入为止。当每次选择最短的路改为每次选择最长路的时候，出现了一个问题，那就是不能保证现在加入的结 点以后是否会被更新而使得到源点的距离变得更长，而这个点一旦被选中将不再会被更新。例如这次加入结点u，最长路为10，下次有可能加入一个结点v，使得 u通过v到源点的距离大于10，但由于u在之前已经被加入到集合中，无法再更新，导致结果是不正确的。

如果取反用dijkstra求最短路径呢，记住，**dijkstra不能计算有负边的情况。**

2. 可以用 Bellman-Ford || SPFA算法求最长路径，。也可以用Floyd-Warshall 算法求每对节点之间的最长路经，因为最长路径也满足最优子结构性质，而Floyd算法的实质就是动态规划。但是，如果图中含有回路，Floyd算法并不能 判断出其中含有回路，且会求出一个错误的解；而Bellman-Ford算法则可以判断出图中是否含有回路。
3. 如果是有向无环图，先拓扑排序，再用动态规划求解。

> 设 G 为有 n 个顶点的带权有向无环图，G 中各顶点的编号为 1 到 n，请设计算法，计算图 G 中 <1,n> 间的最长路径。

```
int n,m;
struct Edge{//存边
    int next;
    int to;
    int dis;
}edge[maxn];
queue<int> q;//拓扑排序要用队列
int head[maxn];
int wei[maxn];//点的最大值
int deg[maxn];//点的入度
int a,b,p,num,x;
void add(int u,int v,int d){
    edge[++num].next=head[u];
    edge[num].to=v;
    edge[num].dis=d;
    head[u]=num;
}
int main() {
    scanf("%d%d",&n,&m);
    for(int i=1;i≤m;i++){
        scanf("%d%d%d",&a,&b,&p);
        add(a,b,p);
        deg[b]++;
    }
    memset(wei,-0x7f,sizeof(wei));//刚开始的时候，要把所有出了一以外的赋成负无穷，这样保证是从1开始
                                  //这个数据应该是比较水的，没有负边权
                                  //要是有的话，就在每个经过的点上打个标记来判断有没有一条路是到达n的
    for(int i=1;i≤n;i++){
        if(!deg[i])q.push(i);
```

```
        if(i==1)wei[i]=0;
        else wei[i]=-maxn;
    }
    while(!q.empty()){//拓扑排序
        x=q.front();
        q.pop();
        for(int i=head[x];i;i=edge[i].next){
            deg[edge[i].to]--;//入度减1
            wei[edge[i].to]=max(wei[edge[i].to],wei[x]+edge[i].dis);
            /*更新是这样的：
                这个点的值最大=max（这个点原来的值，他的前驱节点+连接前驱和他自己的边的值）
            */
            if(deg[edge[i].to]==0){//若入度为0则又入队
                q.push(edge[i].to);
            }
        }
    }
    if(wei[n]<0){//为什么不取等？因为有可能所有边的权值都是0
        printf("%d",-1);
        return 0;
    }
    printf("%d",wei[n]);
    return 0;
}
```

∮ 3. k短路

- Dijkstra求终点到其他点的最短路
- 正向边跑起点的BFS（以A*启发函数：f = x + h为排序，取出点）

```
#define INF 0x3f3f3f3f
typedef long long LL;
typedef pair<LL,int> P;
const int maxn = 1000 + 7;
struct Edge{//正向边
    int to,next;
    LL val;
}edge[maxn*100];
struct Line{//反向边
    int to,next;
    LL val;
}line[maxn*100];
int n,m,s,e,tot,k,head[maxn],revhead[maxn];
int tot2;
bool vis[maxn];
LL dist[maxn];//保存终点到其他点的最短路
inline void addEdge(int a,int b,LL c){//正向建边
    edge[tot].to = b;edge[tot].next =  head[a];edge[tot].val = c;head[a] = tot++;
}
inline void AddEdge(int a,int b,LL c){//反向建边
   line[tot2].to = b;line[tot2].next = revhead[a];line[tot2].val = c;revhead[a] = tot2++;
}
struct Node{//BFS保存状态
   int to;
   LL cost;
```

```
    bool operator <(const Node&another)const{//排序规则按照估价函数大小由小到大
         return cost + dist[to] > another.cost + dist[another.to];//估价= 当前 + 到终点最短
    }
    Node(int a,LL c):to(a),cost(c) {}
};
inline void Dijkstra(int a){//最短路
    dist[a] = 0;
    priority_queue<P,vector<P>,greater<P> >que;
    que.push(P(0,a));
    while(!que.empty()){
        P p = que.top();
        que.pop();
        if(vis[p.second])continue;
        vis[p.second] = 1;
        LL num = p.first;
        for(int i = revhead[p.second];~i;i = line[i].next){//跑反向边
            if(!vis[line[i].to]&&dist[line[i].to] > num + line[i].val){
                dist[line[i].to] = num + line[i].val;
                que.push(P(dist[line[i].to],line[i].to));
            }
        }
    }
}
inline LL BFS(int a){//BFS
    priority_queue<Node> que;
    que.push(Node(a,0));
    while(!que.empty()){
       Node node = que.top();
       que.pop();
       if(node.to==e){//到达终点次数
          k--;
          if(k==0){
             return node.cost;
          }
       }
       for(int i = head[node.to];~i;i = edge[i].next){//扩散（跑反向边）
            que.push(Node(edge[i].to,node.cost + edge[i].val));
       }
    }
    return -1;
}
int main()
{
    while(scanf("%d%d",&n,&m)≠EOF){
        tot = tot2 = 0;
        memset(dist,INF,sizeof(dist));
        memset(vis,0,sizeof(vis));
        memset(head,-1,sizeof(head));
        memset(revhead,-1,sizeof(revhead));
        for(int i = 0;i<m;i++){
            int a,b;
            LL v;
            scanf("%d%d%lld",&a,&b,&v);
            addEdge(a,b,v);
            AddEdge(b,a,v);
        }
        scanf("%d%d%d",&s,&e,&k);//起点 + 终点 + k短路
        Dijkstra(e);
        if(dist[s]==INF){
            printf("-1\n");
```

```
            continue;
        }
        if(s==e)k++;//起点终点重合，排除0距离
        LL ans = BFS(s);
        printf("%lld\n",ans);
    }
    return 0;
}
```

3.1 两点间 k 短路

```
const int MAXN = 1005;
const int MAXM = 100005;
struct Edge;
struct Node {
    Edge *e, *eo;
    int dist, times;
    bool vis;
} N[MAXN];
struct Edge {
    Node *u, *v;
    Edge *next;
    int w;

    Edge() {}
    Edge(Node *u, Node *v, int w, Edge *next) : u(u), v(v), w(w), next(next) {}
} _pool[MAXM << 1], *_curr = _pool;
void addEdge(int u, int v, int w) {
    N[u].e = new (_curr++) Edge(&N[u], &N[v], w, N[u].e);
    N[v].eo = new (_curr++) Edge(&N[v], &N[u], w, N[v].eo);
}
namespace Dijkstra {
    struct HeapNode {
        Node *u;
        int dist;

        HeapNode(Node *u, int dist) : u(u), dist(dist) {}

        bool operator<(const HeapNode &rhs) const {
            return dist > rhs.dist;
        }
    };
    void dijkstra(Node *s) {
        static std::priority_queue<HeapNode> q;
        s->dist = 0;
        q.emplace(s, 0);
        while (!q.empty()) {
            Node *u = q.top().u;
            q.pop();

            if (u->vis) continue;
            u->vis = true;

            for (Edge *e = u->eo; e; e = e->next) {
                if (e->v->dist > u->dist + e->w) {
                    e->v->dist = u->dist + e->w;
                    q.emplace(e->v, e->v->dist);
                }
            }
```

```
        }
    }
    void solve(int s, int n) {
        for (int i = 1; i ≤ n; i++) {
            N[i].dist = INT_MAX;
            N[i].vis = false;
        }

        dijkstra(&N[s]);
    }
}
namespace KthShortest {
    struct HeapNode {
        Node *u;
        int curr, last;
        HeapNode(Node *u, int curr, int last) : u(u), curr(curr), last(last) {}
        bool operator<(const HeapNode &rhs) const {
            return curr + last > rhs.curr + rhs.last;
        }
    };
    int astar(Node *s, Node *t, int k) {
        static std::priority_queue<HeapNode> q;
        q.emplace(s, 0, s->dist);
        while (!q.empty()) {
            Node *u = q.top().u;
            int curr = q.top().curr;
            int last = q.top().last;
            q.pop();
            ++u->times;
            if (u->times == k && u == t) return curr + last;
            if (u->times > k) continue;
            for (Edge *e = u->e; e; e = e->next) q.emplace(e->v, curr + e->w, e->v->dist);
        }
        return -1;
    }
    int solve(int s, int t, int k, int n) {
        Dijkstra::solve(t, n);
        return astar(&N[s], &N[t], k);
    }
}
int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    for (int i = 0, u, v, w; i < m; i++) {
        scanf("%d %d %d", &u, &v, &w);
        addEdge(u, v, w);
    }
    int s, t, k;
    scanf("%d %d %d", &s, &t, &k);
    if (s == t) ++k;
    int ans = KthShortest::solve(s, t, k, n);
    printf("%d\n", ans);
    return 0;
}
```

∮ 4. 欧拉回路

欧拉路就是给一个图，**存在一条回路从起点到终点把所边经过且每条边只经过一次。**“**一笔画问题**”

欧拉回路就是从起点遍历所有的边一次最后回到起点

1. 对于无向图，所有边都是连通的。
   (1) 存在欧拉路径的充分必要条件：度数为奇数的点只能有0或2个。
   (2) 存在欧拉回路的充分必要条件：度数为奇数的点只能有0个。

2. 对于有向图，所有边都是连通。
   (1) 存在欧拉路径的充分必要条件：要么所有点的出度均等于入度；要么除了两个点之外，其余所有点的出度等于入度，剩余的两个点：一个满足出度比入度多1（起点），另一个满足入度比出度多1（终点）.
   (2) 存在欧拉回路的充分必要条件：所有点的出度均等于入度。

4.1 非递归版

> **解决问题**：给定一张无向图，求出它的欧拉回路，若不止一条，随意输出一条即可

```
int head[N],nex[M],ver[M],tot = 1;
bool vis[M];
int stk[M],ans[M];//模拟系统栈、答案栈
int n,m,s,t,cnt,top;
queue<int>q;
void add(int x,int y){
    ver[++tot] = y;nex[tot] = head[x];head[x] = tot;
}
void euler(){//模拟dfs递归
    stk[++top] = 1;//起点
    while(top > 0){
        int x = stk[top],i = head[x];
        while(i && vis[i])i = nex[i];//找到一条尚未访问过的路
        // 与x相连的所有边均已访问，模拟回溯过程，并记录
        if(i){
            stk[++top] = ver[i];
            vis[i] = ver[i ^ 1] = true;//正常的欧拉回路模板
            head[x] = nex[i];
        }
        else {// 与x相连的所有边均已访问，模拟回溯过程，并记录
            top--;
            ans[++cnt] = x;
        }

    }
}
int main(){
    scanf("%d%d",&n,&m);
    tot = 1;
    over(i,1,m){
        int x,y;
        scanf("%d%d",&x,&y);
        add(x,y);add(y,x);
    }
    euler();
    over(i,1,cnt)
    printf("%d\n",ans[i]);
}
```

4.2 普通递归版

4.3 Hierholzers算法（输出字典序最小的答案）

此法输出的是所有合法方案中**字典序最小的答案**

```cpp
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cmath>
#include<cstring>

using namespace std;

const int N = 5000, M = 50007, INF = 0x3f3f3f3f;
typedef pair<int,int> PII;

int n, m;
int d[N];
int g[N][N];
int ans[M], cnt;

void dfs(int x){
    for(int i = 1;i ≤ 500;++i){
        if(g[x][i]){
            g[x][i] -- , g[i][x] -- ;
            dfs(i);
        }
    }
    ans[ ++ cnt] = x;
}

int main(){
    scanf("%d", &n);
    for(int i = 1;i ≤ n;++i){
        int x,y;
        scanf("%d%d",&x, &y);
        g[x][y] ++ , g[y][x] ++ ;
        d[y] ++, d[x] ++;
    }

    int start = 1;
    while(!d[start])start ++;

    for(int i = 1;i ≤ 500;++i){
        if(d[i] % 2){
            start = i;
            break;
        }
    }
    dfs(start);

    for(int i = cnt;i; -- i)
        printf("%d\n",ans[i]);
    return 0;
}

```

4.4 Fleury算法

```cpp
void dfs( int u ) {
```

```
    sta.push(u);
    for( register int i = head[u]; i; i = line[i].nxt ) {
        if( vis[i] ) continue;
        vis[i] = vis[ i ^ 1 ] = true;
        dfs( line[i].to );
        break;
    }
}
void fleury( int st ) {
    sta.push(st);
    while(!sta.empty() ) {
        int flag = 0, u = sta.top(); sta.pop();
        for( register int i = head[u]; i; i = line[i].nxt) {
            if( vis[i] ) continue;
            flag = 1; break;
        }
        if( !flag ) printf( "%d\n", u );
        else        dfs(u);
    }
}
```

4.5 究极优化版

> 给定一张图，请你找出欧拉回路，即在图中找一个环使得每条边都在环上出现恰好一次。
> type为1时图是无向图。
> type为0使图是有向图。

如果出题人丧心病狂（200000个1的自环），普通的代码T了可以用这个代码

```
const int N = 500010, M = 500007, INF = 0x3f3f3f3f;
int n, m;
int type;
int head[N], nex[M], edge[M], ver[M], tot;
int ans[M];
bool used[M];
int cnt;
int din[N], dout[N];

void add(int x,int y){
    ver[tot] = y;
    nex[tot] = head[x];
    head[x] = tot ++ ;
}

void dfs(int x)
{
    //特殊数据O(n)优化，固定i引用变量，这样head[x]也会变，达到删边的效果
    for(int &i = head[x]; ~i;){
        if(used[i]){
            i = nex[i];//删边，先删再dfs，防止遍历到祖先节点时该边未被删去
            continue;
        }
        used[i] = true;

        if(type == 1)
            used[i ^ 1] = true;

        int t;
```

```
30
31          if(type == 1){//无向图就是正反两条边
32              t = i / 2 + 1;//实际上只有一条，我们建图的时候无向图建了两条
33              if(i & 1)//奇数就是反向边,根据题意取负数
34                  t = -t;
35          }
36          //有向图就是下一条边
37          else t = i + 1;
38
39          int y = ver[i];
40          i = nex[i];//删边
41          dfs(y);
42
43          ans[++cnt] = t;
44      }
45  }
46  int main(){
47      scanf("%d",&type);
48      scanf("%d%d",&n,&m);
49      memset(head,-1,sizeof head);
50
51      for(int i = 1;i ≤ m;++i){
52          int x,y;
53          scanf("%d%d",&x,&y);
54          add(x,y);
55          if(type == 1)add(y,x);
56          din[y] ++ , dout[x] ++ ;
57      }
58      //先判断不成立的情况
59      if(type == 1){
60          for(int i = 1;i ≤ n;++i)
61              if((din[i] + dout[i]) & 1){/*求的是欧拉回路所以奇数度点要为0*/
62                  puts("NO");
63                  return 0;
64              }
65      }
66      else {
67          for(int i = 1;i ≤ n;++i){
68              if(din[i] ≠ dout[i]){
69                  puts("NO");
70                  return 0;
71              }
72          }
73      }
74      for(int i = 1;i ≤ n;++i)//找到一个可行的起点，因为本题中的1~n中的点可能没有连边
75          if(head[i] ≠ -1){
76              dfs(i);
77              break;
78          }
79
80      if(cnt < m){//如果不全部连通，则不存在欧拉路
81          puts("NO");
82          return 0;
83      }
84      puts("YES");
85      for(int i = cnt;i;i -- )//倒序输出
86          printf("%d ",ans[i]);
87      puts("");
88
89      return 0;
```

```
90 }
91
```

4.6 混合图欧拉回路

题目描述:给出一个 V（V<=100）个点和 E（E<=500）
条边的无向边与有向边的混合图，试打印出它的任意一条欧拉回路（无向边的两个方向只能从某个方向经过一次），如果没有输出 `No eulercircuit exist`。输入保证图连通。
由于有向图有欧拉回路当且仅当图是弱连通的并且所有点的 入度-出度=0，可以发现关键问题在于如何给无向边定向使得入度-出度=0.
先给所有无向边任意指定一个方向，并令 d'[i]=i的出度-入度（算上原图的有向边和定向了的无向边）。 如果某个 d'[i] 是奇数（相当于说它连接了奇数条边，包括有向边和无向边），那么显然无解，否则令 d[i]=d'[i]/2（为了方便，暂时称其为“度”）。
考虑如果有一条边 u->v 在原图中是无向边，那么将它反向，变成 v->u 之后，u的出度-1，入度+1，从而 d[u]-=1；同理，d[v]+=1. 可以发现这相当于说把 u 的一个度“流到”了 v 的一个度。 于是我们（在网络流中）从 u 到 v 连一条容量为 1 的边，表示可以在 d[u] 中流出一个给 d[v]。 另外，对每个点 u，如果 d[u]>0，那么要从 S 到 u 连一条容量为 d[u] 的边，表示 u 这里多出了 d[u] 的度； 否则 d[u]<=0，就从 u 到 T 连一条容量为 -d[u]的边，表示 u 这里缺少 -d[u] 的度。 执行一遍最大流之后如果 S出发的所有弧都满流，就说明所有多出来的度都流出去了（同时所有少的度都补上了，因为少的度的总数 这时候只需要看网络流中哪些 u->v的边流上了，就知道哪些边需要反向。把这些边反向之后找一遍有向图欧拉回路即可。

```cpp
#include <algorithm>
#include <cctype>
#include <cstdio>
#include <cstring>
#include <vector>

typedef long long LL;

int readInt() {
  int ans = 0, c, f = 1;
  while (!isdigit(c = getchar()))
    if (c == '-') f *= -1;
  do ans = ans * 10 + c - '0';
  while (isdigit(c = getchar()));
  return ans * f;
}

const int N = 105;
const int M = 1500;

int n, m, S, T, pre[N], nxt[M], to[M], ret[M], d[N], cnt, sum;
int u[M], v[M], id[M];
std::vector<int> V[N];

inline void addEdge(int x, int y, int r) {
  nxt[cnt] = pre[x];
  ret[cnt] = r;
  to[pre[x] = cnt++] = y;
  nxt[cnt] = pre[y];
  ret[cnt] = 0;
  to[pre[y] = cnt++] = x;
}

bool Init() {
  n = readInt(); m = readInt();
  S = 0; T = n + 1;
  memset(pre, -1, sizeof pre);
  cnt = 0;
  memset(d, 0, sizeof d);
  for (int i = 0; i < m; ++i) {
```

```
    u[i] = readInt(); v[i] = readInt();
    int c;
    while (!isalpha(c = getchar()));
    ++d[u[i]]; --d[v[i]];
    if (c == 'U') id[i] = cnt, addEdge(u[i], v[i], 1);
    else id[i] = -1;
  }
  sum = 0;
  for (int i = 1; i ≤ n; ++i) {
    if (d[i] & 1) return false;
    d[i] /= 2;
    if (d[i] > 0) {
      addEdge(S, i, d[i]);
      sum += d[i];
    } else addEdge(i, T, -d[i]);
  }
  return true;
}

int que[N], dis[N], cur[N];

bool BFS() {
  int hd = 0, tl = 0;
  memset(dis, -1, sizeof dis);
  dis[que[tl++] = S] = 0;
  while (hd < tl)
    for (int x = que[hd++], i = pre[x]; ~i; i = nxt[i])
      if (ret[i] ≠ 0 && dis[to[i]] == -1)
        dis[que[tl++] = to[i]] = dis[x] + 1;
  return dis[T] ≠ -1;
}

int DFS(int x, int maxf) {
  if (x == T) return maxf;
  int ans = 0;
  for (int &i = cur[x]; ~i; i = nxt[i])
    if (ret[i] ≠ 0 && dis[to[i]] == dis[x] + 1) {
      int t = DFS(to[i], std::min(ret[i], maxf - ans));
      ret[i] -= t;
      ret[i ^ 1] += t;
      if ((ans += t) == maxf) break;
    }
  return ans;
}

void Euler(int x) {
  while (!V[x].empty()) {
    int y = V[x].back(); V[x].pop_back();
    Euler(y);
    printf(" %d", x);
  }
}

void Solve() {
  int ans = 0;
  while (BFS())
    memcpy(cur, pre, sizeof cur), ans += DFS(S, 0x7fffffff);
  if (ans ≠ sum) puts("No euler circuit exist");
  else {
    for (int i = 1; i ≤ n; ++i) V[i].clear();
```

```
    for (int i = 0; i < m; ++i)
      if (id[i] ≥ 0 && !ret[id[i]])
        V[u[i]].push_back(v[i]);
      else
        V[v[i]].push_back(u[i]);
    printf("1");
    Euler(1);
    puts("");
  }
}

int main() {
  int T = readInt();
  while (T--) {
    if (!Init()) puts("No euler circuit exist");
    else Solve();
    if (T) puts("");
  }
  return 0;
}
```

4.7 中国邮递员问题（欧拉回路、状压DP）

> 给你n个点，m条无向边，每条边有一定的距离数值，构造成一个连通图。问从任意一点出发，遍历所有的边，每条边至少访问一次，再回到起点，求满足要求的方案中走过的距离之和的最小短值。
>
> 如果这个图是欧拉回路，那么很明显我们的答案就是欧拉回路的长度欧拉回路的充要条件是图中度数为奇数的点的个数为0。如果图不是欧拉回路，那么我们要做的就是添加一些边（实际上就是重复地经过一些边）使得它变成欧拉回路。邮递员要完成任务就必须在某些街道上重复走若干次。如果重复走一次，就加一条平行边，于是原来对应的图形就变成了多重图。只是要求加进的平行边的总权值最小就行了。于是，我们的问题就转化为，在一个有奇度数结点的赋权连通图中，增加一些平行边，使得新图不含奇度数结点，并且增加的边的总权值最小。

```
#include<cstdio>
#include<cstring>
#include<iostream>
#include<algorithm>

using namespace std;

const int N = 20, M = (1 << 20) + 5, INF = 0x3f3f3f3f;

int n, m;
int f[M];
int dist[N][N];
int q[N], top;
int d[N];
int sum;

void floyd()
{
    for(int k = 1; k ≤ n; ++ k)
        for(int i = 1; i ≤ n; ++ i)if(dist[i][k] < INF)
            for(int j = 1; j ≤ n; ++ j)
                dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
}

void dp()
{
    memset(f, 0x3f, sizeof f);
    f[0] = 0;
```

```
    for(int s = 0; s < (1 << top); ++ s){
        int i = 0;
        while(s & (1 << i))i ++ ;//第一对1和0
        for(int j = i + 1; j < top; ++ j){
            if((s & (1 << j)) == 0)//第二对1和0
                f[s | (1 << i) | (1 << j)] = min(f[s | (1 << i) | (1 << j)], f[s] + dist[q[i]][q[j]]);
        }
    }
}
int main()
{
    while(scanf("%d", &n) ≠ EOF && n)
    {
        scanf("%d", &m);
        top = sum = 0;
        memset(d, 0, sizeof d);
        memset(dist, 0x3f, sizeof dist);
        for(int i = 1; i ≤ m; ++ i){
            int x, y, w;
            scanf("%d%d%d", &x, &y, &w);
            dist[x][y] = dist[y][x] = min(dist[x][y], w);
            d[x] ++ , d[y] ++ ;
            sum += w;//要求所有的边都过一遍
        }



        for(int i = 1; i ≤ n; ++ i){
            if(d[i] & 1)
                q[top ++ ] = i;
        }
        floyd();
        //这里相当于是添加的边
        dp();

        printf("%d\n", f[(1 << top) - 1] + sum);
    }
    return 0;
}


```

∮ 5. 最小生成树

5.1 kruskal算法

给出一个无向图，求出最小生成树

```
typedef pair<int,int> PII;
const int N = 2e5+7;

struct node{
    int x,y,z;
    bool operator<(node &t)const{
        return z < t.z;
    }
}edge[N];
```

```

int fa[N],n,m,ans;

int Find(int x){
    if(x == fa[x])return x;
    return fa[x] = Find(fa[x]);
}

int main()
{
    cin>>n>>m;
    over(i,1,m)
    scanf("%d%d%d",&edge[i].x,&edge[i].y,&edge[i].z);
    sort(edge + 1,edge + 1 + m);
    over(i,1,n)
    fa[i] = i;
    over(i,1,m){
        int x = Find(edge[i].x);
        int y = Find(edge[i].y);
        if(x == y)continue;
        fa[x] = y;
        ans += edge[i].z;
    }
    printf("%d\n",ans);
}


```

5.2 prim算法

朴素写法，拿n-1条边，建议使用。
有时候普通的版本可以过，但是堆优化版本就会wa，所以尽量用kruskal，实在不行了就用朴素版本。

```
#include<cstdio>
#include<algorithm>
#include<cstring>

using namespace std;

const int N = 507, M = 5000007, INF = 0x3f3f3f3f;

int g[N][N], dist[M], n, m, ans;
bool vis[N];

int prim(){
    memset(dist, 0x3f, sizeof dist);
    memset(vis, 0, sizeof vis);
    dist[1] = 0;
    int res = 0;
    for(int i = 0;i < n; ++ i){
        int t = 0;
        for(int j = 1; j ≤ n; ++ j){
            if(!vis[j] && (t == 0 || dist[j] < dist[t])){
                t = j;
            }
        }

        if(i && dist[t] == INF)return INF;
        if(i)res += dist[t];
        vis[t] = 1;
```

```
        for(int y = 1; y ≤ n; ++ y){
            if(!vis[y])dist[y] = min(dist[y], g[t][y]);
        }
    }
    return res;
}

int main(){
    scanf("%d%d", &n, &m);
    memset(g, 0x3f, sizeof g);
    for(int i = 1; i ≤ n; ++ i)g[i][i] = 0;
    for(int i = 1; i ≤ m; ++ i){
        int x, y, z;
        scanf("%d%d%d", &x, &y, &z);
        g[x][y] = g[y][x] = min(g[x][y], z);
    }

    if(prim() == INF)puts("impossible");//不存在最小生成树
    else {
        for(int i = 2; i ≤ n; ++ i)
        ans += dist[i];
        if(ans == INF)puts("impossible");
        else printf("%d\n", ans);
    }
    return 0;
}

```

堆优化版本。

```
typedef long long ll;
typedef pair<int,int> PII;
const int N = 4e5+7;

int ver[N],nex[N],edge[N],head[N],tot;
int n,m,ans;
int dis[N];
int vis[N],cnt;
void add(int u,int v,int val){
    ver[++tot] = v;
    edge[tot] = val;
    nex[tot] = head[u];
    head[u] = tot;
}

priority_queue<PII,vector<PII>,greater<PII> >q;

void prim(){
    dis[1] = 0;
    q.push({0,1});
    while(q.size()&&cnt ≠ n){
        int d = q.top().first,u = q.top().second;
        q.pop();
        if(vis[u])continue;
        cnt++;
        ans += d;
        vis[u] = 1;
        for(int i = head[u];i;i = nex[i]){
            int v = ver[i];
            if(edge[i] < dis[v])
```

```
                dis[v] = edge[i],q.push({dis[v],v});
            }
        }
}

int main()
{
    memset(dis,0x3f,sizeof dis);
    scanf("%d%d",&n,&m);
    over(i,1,m){
        int x,y,z;
        scanf("%d%d%d",&x,&y,&z);
        add(x,y,z);
        add(y,x,z);
    }
    prim();
    if(cnt == n)
        printf("%d\n",ans);
    else puts("orz");
    return 0;
}
```

5.3 Boruvka算法

有 n 个点排成一行，在第 i,j 个点之间连边的代价为 $|i-j| \times D + A_i + A_j$，求将它们连成一棵树的最小代价。（$1 \leq n \leq 10^5$）

直接$O(n^2)$建图会T，而Boruvka算法正适用于此类题目，可以在$O(nlogn)$的时间复杂度内解决问题

> 考虑维护当前的连通块（初始每个点为独立的一个连通块）
> 对每个连通块，找到一条与该连通块相连的，且另一端点不在此连通块中的边权最小的边。
> 将所有的这些边都加入最小生成树，注意，当加入一条边时需判断该边的两端点是否在同一连通块。
> 重复若干遍上述操作，直到图连通。

```
/*Atcoder keyence2019 E*/
typedef long long ll;
int fa[MN];
int c1[MN], c2[MN];
int Min[MN];
ll D; int a[MN];
int X[MN], Y[MN];
int x[MN], y[MN];
ll ans = 0;
int bl;
int n;
int Abs(int a) {return a > 0 ? a : -a;}
ll F(int i) {return i ?  i * D + a[i] : 1e18;}
ll G(int i) {return i ? -i * D + a[i] : 1e18;}
ll H(int i, int j) {if(i == 0 || j == 0) return 1e18; return Abs(i - j) * D + a[i] + a[j];}
void add(int *c, int x, int v, int type)
{
    for(int i = x; i ≤ n; i += i & -i)
    {
        if(!type) if(F(v) < F(c[i])) c[i] = v;
        if( type) if(G(v) < G(c[i])) c[i] = v;
    }
}
int query(int *c, int x, int type)
{
    int ans = 0;
    for(int i = x; i; i -= i & -i)
```

```
    {
        if(!type) if(F(c[i]) < F(ans)) ans = c[i];
        if( type) if(G(c[i]) < G(ans)) ans = c[i];
    }
    return ans;
}
int Find(int x) {return fa[x] == x ? x : fa[x] = Find(fa[x]);}
void solve()
{
    memset(x, 0, sizeof x);
    memset(y, 0, sizeof y);
    memset(c1, 0, sizeof c1);
    memset(c2, 0, sizeof c2);
    memset(Min, 0, sizeof Min);
    for(int i = 1; i ≤ n; i++)
    {
        int u = Find(i);
        int x = query(c1, u - 1, 1);
        int y = query(c2, n - u, 1);
        if(G(x) > G(y)) x = y;
        if(H(x, i) ≤ H(Min[i], i)) Min[i] = x;
        add(c1, u, i, 1);
        add(c2, n - u + 1, i, 1);
    }
    memset(c1, 0, sizeof c1);
    memset(c2, 0, sizeof c2);
    for(int i = n; i ≥ 1; i--)
    {
        int u = Find(i);
        int x = query(c1, u - 1, 0);
        int y = query(c2, n - u, 0);
        if(F(x) > F(y)) x = y;
        if(H(x, i) ≤ H(Min[i], i)) Min[i] = x;
        add(c1, u, i, 0);
        add(c2, n - u + 1, i, 0);
    }
    for(int i = 1; i ≤ n; i++)
    {
        int u = Find(i);
        if(H(i, Min[i]) < H(x[u], y[u])) x[u] = i, y[u] = Min[i];
    }
    int tot = 0;
    for(int i = 1; i ≤ n; i++)
    {
        int u = Find(i);
        ++tot; X[tot] = x[u]; Y[tot] = y[u];
    }
    for(int i = 1; i ≤ tot; i++)
    {
        int x = Find(X[i]), y = Find(Y[i]);
        if(x == y) continue;
        ans += H(X[i], Y[i]);
        fa[x] = y;
        bl--;
    }
}
int main()
{
    scanf("%d%lld", &n, &D);
    for(int i = 1; i ≤ n; i++) scanf("%d", &a[i]), fa[i] = i;
```

```
88      bl = n;
89      while(bl > 1) solve();
90      printf("%lld\n", ans);
91  }
```

5.4 生成森林问题（K颗树）

数据结构：并查集 算法：改进Kruskal 复杂度：O(mlongm)
根据Kruskal算法思想，图中的生成树在连接完第n - 1条边前，都是一个最小生成森林，每次贪心的选择两个不属于同一连通分量的树（如果连接一个连通分量，因为不会减少块数，那么就是不合算的）且用最“便宜”的边连起来，连接n - 1次后就形成了一棵MST，n - 2次就形成了一个两棵树的最小生成森林，n - 3、……、n - k次后，就形成了k棵树的最小生成森林，也就是题目要求的解。
哪怕是森林也是直接一个kruskal算到底即可

5.5 最小生成树的唯一性

考虑最小生成树的唯一性。如果一条边 不在最小生成树的边集中 ，并且可以替换与其 权值相同、并且在最小生成树边集 的另一条边。那么，这个最小生成树就是不唯一的。
对于 Kruskal 算法，只要计算为当前权值的边可以放几条，实际放了几条，如果这两个值不一样，那么就说明这几条边与之前的边产生了一个环（这个环中至少有两条当前权值的边，否则根据并查集，这条边是不能放的），即最小生成树不唯一。
寻找权值与当前边相同的边，我们只需要记录头尾指针，用单调队列即可在$O(\alpha(m))$（m 为边数）的时间复杂度里优秀解决这个问题（基本与原算法时间相同）。

```
const int N = 500007;
int n, m;
int fa[N];
struct node{
    int x, y, z;
    bool operator<(const node &t)const {
        return z < t.z;
    }
}e[N];
int find(int x){
    if(fa[x] == x)return x;
    return fa[x] = find(fa[x]);
}

int main(){
    int t;
    scanf("%d", &t);
    while(t -- ){
        int cnt = 0;
        scanf("%d%d", &n, &m);
        for(int i = 0; i ≤ n; ++ i)
            fa[i] = i;

        for(int i = 1; i ≤ m; ++ i){
            int x, y, z;
            scanf("%d%d%d", &x, &y, &z);
            e[++ cnt] = (node){x, y, z};
        }
        sort(e + 1, e + 1 + cnt);
        //sum1为相同权值大小 可以 添加到最小生成树中的边数(当前还没放呢)
        //sum2为相同权值大小 实际 添加到最小生成树中的边数
        //如果最小生成树唯一，则sum1 == sum2
        int tail = 0, ans = 0, sum1 = 0, sum2 = 0, num = 0;
        bool flag = 1;
        for(int i = 1; i ≤ m + 1; ++ i){
            if(i > tail){//开始新的一个权值的边以后
```

```
                if(sum1 ≠ sum2){
                    flag = 0;
                    break;
                }
                sum1 = 0;
                //要到 m+1 才能把 m 条边都扫一遍
                for(int j = i; j ≤ m + 1; ++ j){
                    if(e[i].z ≠ e[j].z){
                        tail = j - 1;
                        break;
                    }
                    if(find(e[j].x) ≠ find(e[j].y))
                        sum1 ++ ;
                }
                sum2 = 0;
            }

            if(i > m)break;

            int x = find(e[i].x);
            int y = find(e[i].y);
            if(x ≠ y && num ≠ n - 1){
                fa[x] = y;
                sum2 ++ ;
                num ++ ;
                ans += e[i].z;
            }
        }
        if(flag)
            printf("%d\n", ans);
        else printf("Not Unique!\n");
    }
    return 0;
}
```

5.6 严格次小生成树

次小生成树

定义：给一个带权的图，把图的所有生成树按权值从小到大排序，第二小的称为次小生成树。

方法1：先求最小生成树，再枚举删去最小生成树中的边求解。时间复杂度。 O(mlogm + nm)

方法2：先求最小生成树，然后依次枚举非树边，然后将该边加入树中，同时从树中去掉一条边，使得最终的图仍是一棵树。则一定可以求出次小生成树。

设T为图G的一棵生成树，对于非树边a和树边b，插入边a，并删除边b的操作记为(+a, -b)。

如果T+a-b之后，仍然是一棵生成树，称(+a,-b)是T的一个可行交换。

称由T进行一次可行变换所得到的新的生成树集合称为T的邻集。

定理：次小生成树一定在最小生成树的邻集中。

不严格次小生成树只需要将 `w>t1` 改成 `w≥t1` 即可

5.7 LCA优化的次小生成树

```
int n,m;
int ver[M], nex[M], edge[M], head[N], tot;
int fa[N];
int deep[N];
int f[N][20];//lca
```

```cpp
int d1[N][20];//注意d1[i][k]是当前i这个点从自己到2^k祖先整个路径上所有边权的的最大值，这样就不会因为lca是跳着走而有漏掉的情况
int d2[N][20];
int q[M];//手写循环队列
struct node{
    int x,y,z;
    bool used;
    bool operator<(const node &t)const{
        return z < t.z;
    }
}e[M];
int Find(int x){
    if(fa[x] == x)return x;
    return fa[x] = Find(fa[x]);
}
void add(int x,int y,int z){
    ver[tot] = y;
    edge[tot] = z;
    nex[tot] = head[x];
    head[x] = tot ++ ;
}
void bfs(int rt){//bfs把所有的边都覆盖掉的
    deep[rt] = 1;
    int hh = 0, tt = 0;
    q[tt ++ ] = rt;
    while(hh != tt){
        int x = q[hh ++ ];
        if(hh == M)hh = 0;
        for(int i = head[x];~i;i = nex[i]){
            int y = ver[i],z = edge[i];
            if(!deep[y]){
                deep[y] = deep[x] + 1;
                q[tt ++ ] = y;
                if(tt == M)tt = 0;
                f[y][0] = x;//lca的正常处理
                d1[y][0] = z;//最大值，初始化为边权
                d2[y][0] = -INF;//次大值，最开始为-INF
                for(int k = 1;k <= 16;++k){
                    int anc = f[y][k - 1];
                    f[y][k] = f[anc][k - 1];
                    //左半边和右半边
                    int distance[4] = {d1[y][k - 1],d2[y][k - 1],d1[anc][k - 1],d2[anc][k - 1]};
                    //这两个要初始化为-INF，一会取最大值
                    d1[y][k] = d2[y][k] = -INF;
                    for(int o = 0;o < 4;++o){
                        int d = distance[o];
                        //老规矩，大于最大，最大给次大，更新最大
                        if(d > d1[y][k])d2[y][k] = d1[y][k],d1[y][k] = d;
                        else if(d != d1[y][k] && d > d2[y][k])//否则看次大，反正d是不可能小于他们
                            d2[y][k] = d;
                    }
                }
            }
        }
    }
}
int lca(int x,int y,int z){//返回差值
    static int distance[N * 2];//路上经过的全部放到这个数组里取最大值
    int cnt = 0;
    if(deep[x] < deep[y])swap(x,y);
```

```
    for(int k = 16;k ≥ 0;k--){
        if(deep[f[x][k]] ≥ deep[y]){
            distance[cnt ++ ] = d1[x][k];
            distance[cnt ++ ] = d2[x][k];
            x = f[x][k];
        }
    }
    if(x ≠ y){
        for(int k = 16;k ≥ 0;--k){
            if(f[x][k] ≠ f[y][k]){
                distance[cnt ++ ] = d1[x][k];
                distance[cnt ++ ] = d2[x][k];
                distance[cnt ++ ] = d1[y][k];
                distance[cnt ++ ] = d2[y][k];
                x = f[x][k],y = f[y][k];
            }
        }
        distance[cnt ++ ] = d1[x][0];
        distance[cnt ++ ] = d1[y][0];
        //此时x和y距离它们的lca只有一步，也就是只有的d1 = w,而d2 = -INF,不用加
    }
    int dist1 = -INF,dist2 = -INF;
    //按照老规矩更新
    for(int i = 0;i < cnt ;++i){
        int d = distance[i];
        if(d > dist1)dist2 = dist1,dist1 = d;
        else if(d ≠ dist1 && d > dist2)
            dist2 = d;
    }
    //能换最大值肯定是换最大值更优
    if(z > dist1)return z - dist1;
    if(z > dist2)return z - dist2;
    return INF;//换不了就返回INF，因为我们最后要的是最小值（严格次小生成树）
}
ll kruskal(){
    ll res = 0;
    for(int i = 1;i ≤ n;++i)//千万别往了并查集初始化！！！
        fa[i] = i;
    sort(e + 1,e + 1 + m);
    memset(head,-1,sizeof head);
    for(int i = 1;i ≤ m;++i){
        int x = e[i].x,y = e[i].y,z = e[i].z;
        int fx = Find(x);
        int fy = Find(y);
        if(fx ≠ fy){
            fa[fx] = fy;
            e[i].used = true;
            res += z;
            add(x,y,z);
            add(y,x,z);
        }
    }
    return res;
}
int main(){
    scanf("%d%d",&n,&m);
    for(int i = 1;i ≤ m;++i){
        int x,y,z;
        scanf("%d%d%d",&x,&y,&z);
        e[i] = {x,y,z};
```

```
    }

    ll sum = kruskal();

    bfs(1);//lca预处理

    ll res = 1e18;//会爆int

    for(int i = 1;i ≤ m;++i){
        if(!e[i].used){//对于所有非树边
            int x = e[i].x,y = e[i].y, z = e[i].z;
            res = min(res,sum + lca(x,y,z));
        }
    }
    printf("%lld\n",res);
    return 0;
}

```

5.8 瓶颈生成树

定义
无向图 G 的瓶颈生成树是这样的一个生成树，它的最大的边权值在 G 的所有生成树中最小。
性质
最小生成树是瓶颈生成树的充分不必要条件。 即最小生成树一定是瓶颈生成树，而瓶颈生成树不一定是最小生成树。

5.9 最小瓶颈路

最小瓶颈路问题是指在一张无向图中，询问一个点对(u,v)，需要找出从u到v的一条简单路径，使路径上所有边中最大值最小。

根据查询次数不同，最小瓶颈路问题可分为单次查询和多次查询。

5.9.1 单次查询

利用Kruskal的过程。将所有边升序排序，逐一枚举每条边尝试将边两端的点所在集合进行合并，如果合并之后u和v在同一个集合中，则说明此时u到v连通。由于边的枚举是由小到大，因此可以保证最后一次合并的边的权值就是答案。

5.9.2 多次查询

> 给定一张带权无向图，每次询问两个点，找出两点间的一条简单路径，使路径中权值最大的边权值最小。

可以证明该无向图最小生成树中u到v的路径一定是u到v的最小瓶颈路之一（因为最小瓶颈路很可能有多条）。因此，对这张无向图预先进行MST，然后就变成了对于每个询问(u,v)，回答u到v的路径上的权值最大值。这个可以用倍增LCA解决。

具体思路是在ST表预处理时，维护一个从该节点到其k级父亲中经过的所有边的最大权值。在查询中仍然将u和v往上跳，同时维护路径中最大值，到其LCA结束。

```
struct edge{int u,v,w;}e[maxm];
int heade[maxn],ev[maxm],ew[maxm],nexte[maxm];
int father[maxn],dep[maxn];
int fa[maxn][maxj],maxv[maxn][maxj];
int n,m,q,root,tot=0,num=0;
void add_edge(int u,int v,int w){tot++;ev[tot]=v;ew[tot]=w;nexte[tot]=heade[u];heade[u]=tot;}
int cmp(edge a,edge b){return a.w<b.w;}
int find(int x){return father[x]<0?x:father[x]=find(father[x]);}
void union_set(int u,int v,int w)
{
    int x=find(u),y=find(v);
    if(x==y){return;}
    if(-father[x]>-father[y]){father[x]+=father[y];father[y]=x;}
```

```
    else{father[y]+=father[x];father[x]=y;}
    add_edge(u,v,w);add_edge(v,u,w);
    num++;
}
void dfs(int ui)
{
    int i,vi,wi;
    for(i=heade[ui];~i;i=nexte[i])
    {
        vi=ev[i];wi=ew[i];if(vi==fa[ui][0]){continue;}
        fa[vi][0]=ui;maxv[vi][0]=wi;dep[vi]=dep[ui]+1;
        dfs(vi);
    }
}
void init()
{
    int i,j;
    for(j=1;(1<<j)≤n;j++){for(i=1;i≤n;i++){if(fa[i][j-1]){fa[i][j]=fa[fa[i][j-1]][j-1];maxv[i][j]=max(maxv[i][j-1],maxv[fa[i][j-1]][j-1]);}}}
}
int query(int u,int v)
{
    int i,j,t,tmp=-inf;
    if(dep[u]<dep[v]){swap(u,v);}
    t=(int)(log(dep[u])/log(2));
    for(j=t;j≥0;j--){if(dep[u]-(1<<j)≥dep[v]){tmp=max(tmp,maxv[u][j]);u=fa[u][j];}}
    if(u==v){return tmp;}
    for(j=t;j≥0;j--){if(fa[u][j]&&fa[u][j]≠fa[v][j]){tmp=max(tmp,max(maxv[u][j],maxv[v][j]));u=fa[u][j];v=fa[v][j];}}
    return max(tmp,max(maxv[u][0],maxv[v][0]));
}
int main()
{
    int i,j,u,v,w;
    //freopen("data.in","r",stdin);
    //freopen("test.out","w",stdout);
    cin>>n>>m>>q;root=(1+n)>>1;
    memset(heade,-1,sizeof(heade));memset(father,-1,sizeof(father));
    for(i=1;i≤m;i++){scanf("%d%d%d",&e[i].u,&e[i].v,&e[i].w);}
    sort(e+1,e+m+1,cmp);
    for(i=1;i≤m;i++){u=e[i].u;v=e[i].v;w=e[i].w;union_set(u,v,w);if(num≥n-1){break;}}
    dfs(root);init();
    for(i=1;i≤q;i++)
    {
        scanf("%d%d",&u,&v);
        printf("%d\n",query(u,v));
    }
    return 0;
}
```

5.10 Kruskal 重构树

> kruskal重构树是一个很常用的图论算法。主要用于解决u->v所有路径上最长边的最小值，就是找到u->v的一条路径，使路径上的最长边最小。

kruskal重构树有以下特质：

- 每个原图上的节点一一对应重构树上的叶子节点。
- 重构树上每一个其他节点（正方形）代表原图上的一个边，有点权。
- 重构树是一棵二叉树。

- 重构树是一个二叉堆。（所以两个叶子节点的LCA即为路径上的最大边）

```cpp
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<vector>
using namespace std;
#define maxn 25000
struct gg{
    int u,v,w;
}side1[maxn*2];
vector<int> side2[maxn*4];
bool cop(gg x,gg y){return x.w<y.w;}
int ncnt,num[maxn*4],n,m,k,head[maxn],cnt,dep[maxn*4],f[maxn*4][21],fa[maxn*4];
int get(int x){
    if(fa[x]==x)return x;
    fa[x]=get(fa[x]);
    return fa[x];
}
void uni(int x,int y,int w){
    int gx=get(x),gy=get(y);
    if(gx==gy)return;
    ncnt++;num[ncnt]=w;

 side2[ncnt].push_back(gx);side2[ncnt].push_back(gy);side2[gx].push_back(ncnt);side2[gy].push_back(ncnt)
;
    fa[gx]=fa[gy]=fa[ncnt]=ncnt;
    return;
}
void dfs(int u,int g){
    dep[u]=dep[g]+1;f[u][0]=g;
    for(int i=1;i≤20;i++)f[u][i]=f[f[u][i-1]][i-1];
    for(int i=0;i<(int)side2[u].size();i++){
        int v=side2[u][i];if(v==g)continue;
        dfs(v,u);
    }
    return;
}
int lca(int u,int v){
    if(dep[u]<dep[v])swap(u,v);
    for(int i=20;i≥0;i--)if(dep[f[u][i]]≥dep[v])u=f[u][i];
    if(u==v)return u;
    for(int i=20;i≥0;i--)if(f[u][i]≠f[v][i]){u=f[u][i];v=f[v][i];}
    return f[u][0];
}
int main(){
    scanf("%d%d%d",&n,&m,&k);
    for(int i=1;i≤m;i++){
        int u,v,w;scanf("%d%d%d",&u,&v,&w);
        side1[i]=(gg){u,v,w};
    }
    for(int i=0;i≤n;i++){fa[i]=i;}
    ncnt=n;
    sort(side1+1,side1+1+m,cop);
    for(int i=1;i≤m;i++){
        if(get(side1[i].u)==get(side1[i].v))continue;
        uni(get(side1[i].u),get(side1[i].v),side1[i].w);
    }
    dfs(ncnt,0);
    for(int i=1;i≤k;i++){
```

```
57          int a,b;scanf("%d%d",&a,&b);
58          printf("%d\n",num[lca(a,b)]);
59      }
60      return 0;
61  }
```

5.11 最小生成树计数

> 现在给出了一个简单无向加权图。你不满足于求出这个图的最小生成树，而希望知道这个图中有多少个不同的最小生成树。（如果两颗最小生成树中至少有一条边不同，则这两个最小生成树就是不同的）。

```
#define debug( ... ) fprintf(stderr,__VA_ARGS__)
#define DEBUG printf("Passing [%s] in LINE %lld\n",__FUNCTION__,__LINE__)
#define Debug debug("Passing [%s] in LINE %lld\n",__FUNCTION__,__LINE__)
#define all(x) x.begin(),x.end()
using namespace std;
const double eps=1e-8;
const double pi=acos(-1.0);
typedef long long ll;
typedef pair<ll,ll> pii;
template<class T>ll chkmin(T &a,T b){return a>b?a=b,1:0;}
template<class T>ll chkmax(T &a,T b){return a<b?a=b,1:0;}
template<class T>T sqr(T a){return a*a;}
template<class T>T mmin(T a,T b){return a<b?a:b;}
template<class T>T mmax(T a,T b){return a>b?a:b;}
template<class T>T aabs(T a){return a<0?-a:a;}
#define min mmin
#define max mmax
#define abs aabs
ll read(){
    ll s=0,base=1;
    char c;
    while(!isdigit(c=getchar()))if(c=='-')base=-base;
    while(isdigit(c)){s=s*10+(c^48);c=getchar();}
    return s*base;
}
struct edge{
    ll u,v,w;
};
edge e[102424];
ll fa[102424],use[102424];
ll is[102424];
ll find(ll x){return x==fa[x]?x:fa[x]=find(fa[x]);}
ll e_cmp(edge a,edge b){return a.w<b.w;}
const ll p=31011;
ll a[128][128];
ll inv[31015];
void kill(ll a,ll b,ll &aii,ll &aij,ll &aji,ll &ajj,ll &sign){
    sign=1;
    aii=ajj=1;
    aij=aji=0;
    while(b){
        aii-=a/b*aji;
        aij-=a/b*ajj;
        aij=(aij%p+p)%p;
        aii=(aii%p+p)%p;
        a%=b;
        swap(a,b);
        swap(aii,aji);
        swap(aij,ajj);
```

```
            sign=-sign;
        }
    }
    ll det(ll n){
        ll k,s=1,_a,b,c,d,pa,ans=1,nxa,nxb;
        for(ll i=1;i≤n;++i)for(ll j=1;j≤n;++j)if(a[i][j]<0)a[i][j]+=p;
        for(ll i=1;i≤n;++i){
            for(ll j=i+1;j≤n;++j){
                kill(a[i][i],a[j][i],_a,b,c,d,pa);
                s*=pa;
                for(ll k=1;k≤n;++k){
                    nxa=(a[i][k]*_a+a[j][k]*b)%p;
                    nxb=(a[i][k]*c+a[j][k]*d)%p;
                    a[i][k]=nxa;
                    a[j][k]=nxb;
                }
            }
            ans=ans*a[i][i]%p;
        }

        return ans*(s+p)%p;
    }
    int main(){
    #ifdef cnyali_lk
        freopen("4208.in","r",stdin);
        freopen("4208.out","w",stdout);
    #endif
        ll n,m;
        inv[1]=1;
        for(ll i=2;i<p;++i)inv[i]=(p-p/i)*inv[p%i]%p;
        n=read();m=read();
        for(ll i=1;i≤m;++i){
            e[i].u=read();
            e[i].v=read();
            e[i].w=read();
        }
        sort(e+1,e+m+1,e_cmp);
        for(ll i=1;i≤n;++i)fa[i]=i;
        ll g=n;
        for(ll i=1;i≤m&&g>1;++i)if(find(e[i].u)≠find(e[i].v)){
            use[i]=1;
            fa[find(e[i].u)]=find(e[i].v);
            --g;
        }
        if(g>1){printf("0\n");return 0;}
        ll u=1,v=1;
        ll ans=1;
        while(v≤m){
            for(ll i=1;i≤n;++i){
                fa[i]=i;
                is[i]=0;
            }
            for(ll i=1;i≤m;++i)if(use[i]&&e[i].w≠e[u].w){
                fa[find(e[i].u)]=find(e[i].v);
            }
            g=0;
            for(ll i=1;i≤n;++i){
                if(!is[find(i)]){
                    is[find(i)]=++g;
                }
```

```
            is[i]=is[find(i)];
        }
        for(ll i=1;i≤g;++i)for(ll j=1;j≤g;++j)a[i][j]=0;
        while(e[u].w==e[v].w){
            a[is[e[v].u]][is[e[v].v]]-=1;
            a[is[e[v].v]][is[e[v].u]]-=1;
            a[is[e[v].u]][is[e[v].u]]+=1;
            a[is[e[v].v]][is[e[v].v]]+=1;
            ++v;
        }

        --g;
        ans=ans*det(g)%p;
        u=v;
    }
    printf("%lld\n",ans);
    return 0;
}
```

5.12 曼哈顿最小生成树

题意：

平面上有n个点，现在把他们分成k个集合，使得每个集合中的每个点都至少有一个本集合的点之间的曼哈顿距离不大于X，求最小的X。

分析：

转化为求n个点生成完全图的最小生成树的第k大边。

> 如果暴力加边再用Kruskal，边太多会超时。这里用一个算法来减少有效边的加入。
> 边权值为点间曼哈顿距离，那么每个点的有效加边选择应该是和他最近的4个象限方向的点。这里用一个树状数组维护以y-x为索引的y+x的值，然后这个数组所储存的就是一个点的第一象限方向的距离他最近的点。这样我们每次查找只要看（i，N）这个区间是否有一个点的距离比现在的更小（因为以y-x为索引，所以l>i就表示l这个点在i的第一象限方向）。最后每个方向的边都加完后，只要用Kruskal算法加边，加到第K就输出权值。

```
#define ll long long
const int N = 10000+5;
const int INIT = 1061109567;
using namespace std;
int n,k,cnt,Id[N],v[N],f[N];    //id为v值的id序号，v为储存以y-x为索引的y+x值
struct node{
    int x,y,id;
    friend bool operator < (node a,node b){
        return a.x == b.x? a.y < b.y : a.x < b.x;
    }
}p[N];
struct edge{
    int u,v,w;
    friend bool operator < (edge a,edge b) {
        return a.w < b.w;
    }
}e[N<<2];
int lowbit(int x){
    return x&(-x);
}
void query(int id,int pos,int val){
    pos += 1000;
    int u = -1,ret = INT_MAX;
    for(int i = pos;i < N;i += lowbit(i)){
        if(v[i] < ret && v[i] ≠ INIT){ //找曼哈顿距离最短的
            ret = v[i];
            u = Id[i];
```

```
        }
    }
    if(u ≠ -1){     //找到这种点就加一个边
        e[cnt].u = id;
        e[cnt].v = u;
        e[cnt++].w = ret - val;
    }
}
void update(int id,int pos,int val){     //id,y-x,y+x
    pos += 1000;
    for(int i = pos;i > 0;i -= lowbit(i)){
        if(val < v[i]){ //更换最佳选择
            v[i] = val;
            Id[i] = id;
        }
    }
}
int find(int x){
    if(x == f[x]) return x;
    else return f[x] = find(f[x]);
}
void kruskal(){
    sort(e,e+cnt);
    for(int i = 0;i < N;i++) f[i] = i;
    int num = n;
    for(int i = 0;i < cnt;i++){
        int x = find(e[i].u);
        int y = find(e[i].v);
        if(x ≠ y){
            f[x] = f[y];
            num--;
            if(num == k){
                printf("%d\n",e[i].w);
                return;
            }
        }
    }
}
void addEdge(){
    memset(v,63,sizeof(v)); //v == 1061109567
    sort(p,p+n);
    for(int i = n-1;i ≥ 0;i--){
        int index = p[i].y - p[i].x;
        int val = p[i].y + p[i].x;
        query(p[i].id,index,val);
        update(p[i].id,index,val);
    }
}
int main(){
    cnt = 0;
    scanf("%d%d",&n,&k);
    for(int i = 0;i < n ;i++){
        scanf("%d%d",&p[i].x,&p[i].y);
        p[i].id = i;
    }

    addEdge();

    for(int i = 0;i < n;i++) swap(p[i].x,p[i].y);
    addEdge();
```

```
88
89      for(int i = 0;i < n;i++) p[i].y = -p[i].y;
90      addEdge();
91
92      for(int i = 0;i < n;i++) swap(p[i].x,p[i].y);
93      addEdge();
94
95      kruskal();
96      return 0;
97  }
98
99
```

$\oint$ 6. 最小树形图（朱刘算法）

6.1 给定一个根的有向图的最小树形图

> 给定包含 n 个结点，m 条有向边的一个图。试求一棵以结点 r 为根的最小树形图，并输出最小树形图每条边的权值之和，如果没有以 r为根的最小树形图，输出 -1。

```cpp
#include<bits/stdc++.h>
using namespace std;
const int N=106,M=10006,inf=2e8;
int n,m,rt,t,cnt=0,id[N],pre[N],ine[N],vis[N];
struct line{int x,y,c;}q[M];
inline int read(){
   int T=0,F=1; char ch=getchar();
   while(ch<'0'||ch>'9'){if(ch=='-') F=-1; ch=getchar();}
   while(ch≥'0'&&ch≤'9') T=(T<<3)+(T<<1)+(ch-48),ch=getchar();
   return F*T;
}
int zhuliu(){
    int ans=0;
    while(true){
         cnt=0;
         for(int i=1;i≤n;++i) ine[i]=inf,vis[i]=0,id[i]=0;//预处理
         for(int i=1;i≤m;++i) if(q[i].x≠q[i].y&&ine[q[i].y]>q[i].c)
ine[q[i].y]=q[i].c,pre[q[i].y]=q[i].x;//每个点的最短边
         for(int i=1;i≤n;++i) if(i≠rt&&ine[i]==inf) return -1;//有点无最短边
        for(int i=1;i≤n;++i){
            if(i==rt) continue;
            ans+=ine[i],t=i;
            while(vis[t]≠i&&!id[t]&&t≠rt) vis[t]=i,t=pre[t];
            //能走到环的点或者换上的点停下
            if(!id[t]&&t≠rt){
               id[t]=++cnt; //将环上的点标记为新的环
               for(int o=pre[t];o≠t;o=pre[o]) id[o]=cnt;
            }
        }//找环
        if(!cnt) break;//无环结束
        for(int i=1;i≤n;++i) if(!id[i]) id[i]=++cnt;
        for(int i=1;i≤m;++i){
            t=q[i].y,q[i].x=id[q[i].x],q[i].y=id[q[i].y];
            if(q[i].x≠q[i].y) q[i].c-=ine[t];
        }
        n=cnt,rt=id[rt];
        //去旧图,换新图
    }
```

```
    return ans;
}
int main(){
    n=read(),m=read(),rt=read();
    for(int i=1;i≤m;++i) q[i].x=read(),q[i].y=read(),q[i].c=read();
    printf("%d\n",zl());
    return 0;
}
```

6.2 为给定根的树形图

求没有确定的根的树形图:建立一个超级根r，以它为根跑算法,只要将r向原图每个点连接一条权值大于原图中所有边的边权的边，这样选这些边肯定不划算，因此只会选择一条。

6.3 判断无解的方法

判无解的奇技淫巧:从小到大依次枚举每个点i,加入边 $(i,(i+1)\%n+1,+\infty)$,这样如果你最后得到的答案为$+\infty$，那么就无解了。

∮ 7. 负环与01分数规划

7.1 判断负环

判正环求最长路，判负环求最短路

求负环的常用方法，基于SPFA：
(1) 统计每个点入队的次数，如果某个点入队n次，则说明存在负环；
(2) 统计当前每个点的最短路中所包含的边数，如果某点的最短路所包含的边数大于等于n，则也说明存在环；

```
int n;      // 总点数
int h[N], w[N], e[N], ne[N], idx;       // 邻接表存储所有边
int dist[N], cnt[N];     // dist[x]存储1号点到x的最短距离，cnt[x]存储1到x的最短路中经过的点数
bool st[N];     // 存储每个点是否在队列中
// 如果存在负环，则返回true，否则返回false
bool spfa()
{
    // 原理：如果某条最短路径上有n个点（除了自环），那么加上自己之后一共有n+1个点，由抽屉原理一定有两个点相同，所以存在环。
    memset(dis,0,sizeof dis);//如果求正环，则初始化为负无穷
    memset(st,false,sizeof st);
    memset(cnt,0,sizeof cnt);
    queue<int> q;
    for(int i=0;i≤n;i++) q.puh(i),st[i]=true;//如果有虚拟超级源点和所有的点相连，那么只需加入源点即可
    int count=0;
    while(q.size())
    {
        int t=q.front();
        st[t]=false;
        q.pop();
        for(int i=h[t];i≠-1;i=ne[i])
        {
            int j=e[i];
            if(dis[j]>dis[t]+w[i])
            {
                dis[j]=dis[t]+w[i];
                cnt[j]=cnt[t]+1;
                //if(++cont>4*N) return true;    当点的更新次数大于总点数的2~5倍时，就可认为存在环
                if(cnt[j]≥n) return true;
                if(!st[j])
```

```
                {
                    q.push(j);
                    st[j]=true;
                }
            }
        }
    }
    return false;
}
```

7.2 01分数规划

01分数规划问题可以用二分+负/正环判断解决。
我们要求的是平均长度，也就是每一条字符串的长度之和除以总的字符串个数，换成数学公式即为下式（个数对于每一个字符串来说都是1）：$\frac{\sum w_i}{\sum 1}$
要求这个值最大，满足单调性，很明显是一个01分数规划问题，自然使用二分答案。我们设答案为 `mid`
那么如果当前答案不够需要更大，上式即为$\frac{\sum w_i}{\sum 1} > mid$化简以后：

$$\sum w_i > mid * \sum 1$$
$$\sum w_i > \sum mid$$
$$\sum w_i - \sum mid > 0$$

那么也就是说对于每一个边，给它赋权值$w_i - mid$，如果存在正环，也就意味着有一个环的权值和大于0，也就是$\sum w_i - \sum mid > 0$，就意味着 `mid` 需要更大。

这里可以直接用权值建图，然后直接跑spfa，把不同的mid放到更新最短路时减去，因为每次只是-mid，与其他的任何条件都没有关系，只需要建一次图，如果权值经过化简得到的是$w_i - x_i * mid$之类的就需要每次二分对 `mid` 进行 `check` 的时候， `check` 函数里都新建一张权值为$w_i - x_i * mid$来判定是否有正/负环。具体代码在下面

7.3 判负环时TLE的优化技巧：

- 经验优化（如果总的入队次数超过一个挺大的值，我们根据经验判断一定存在负环才会导致运行时间慢，直接判定是负环，**有WA的风险**）
- 使用stack（能有效优化负环超时的问题，**有TLE的风险**）

7.4 经验 `trick`

一般总入队次数超过总边数的2~5倍或者边数较小次数大于1e6即可判定为负环，但是会有WA的风险。

```
const int N = 1000, M = 500007, INF = 0x3f3f3f3f;
double eps = 1e-4;

int ver[M],nex[M],edge[M],head[N],tot = 1;
int n;//这里的n是字符的个数
int m;
int q[N],cnt[N],vis[N];
double dist[N];

void add(int x,int y,int z){
    ver[++tot] = y;
    edge[tot] = z;
    nex[tot]  =head[x];
    head[x] = tot;
}

bool spfa(double mid){
    memset(cnt,0,sizeof cnt);
    memset(vis,0,sizeof vis);

    int hh = 0,tt = 0;
```

```

    for(int i = 0;i < 676;++i){
        q[tt ++ ] = i;
        vis[i] = true;
    }
    int counts = 0;//经验优化
    while(hh != tt){
        int x = q[hh ++ ];
        if(hh == N)hh = 0;
        vis[x] = false;

        for(int i = head[x];~i;i = nex[i]){
            int y = ver[i],z = edge[i];
            if(dist[y] < dist[x] + z - mid){
                dist[y] = dist[x] + z - mid;
                cnt[y] = cnt[x] + 1;
                counts ++ ;
                if(counts > 10000)return true;
                if(cnt[y] ≥ N)return true;
                if(!vis[y]){
                    vis[y] = true;
                    q[tt ++ ] = y;
                    if(tt == N)tt = 0;
                }
            }
        }
    }
    return false;
}

int main(){
    while(scanf("%d",&n) && n){
        memset(head,-1,sizeof head);
        tot = 1;
        for(int i = 1;i ≤ n;++i){
            char s[1010];
            scanf("%s",s);
            int len = strlen(s);

            if(len < 2)continue;
            int a = (s[0] - 'a' )* 26 + s[1] - 'a';
            int b = (s[len - 2] - 'a') * 26 + s[len - 1] - 'a';
            add(a,b,len);
        }
        if(!check(0))puts("No solution");
        else {
            double l = 0,r = 1000;
            while(r - l > eps){
                double mid = (l + r) / 2;
                if(spfa(mid))l = mid;
                else r = mid;
            }
            printf("%lf\n",r);
        }
    }
    return 0;
}
```

7.5 使用 stack

注意，`stack`只有在判断负环的情况下会有高效的优化作用，一般的spfa中队列比栈的性能更佳

```
bool spfa(double mid){
    memset(cnt,0,sizeof cnt);
    memset(vis,0,sizeof vis);

    int hh = 0,tt = 0;

    for(int i = 0;i < 676;++i){
        q[tt ++ ] = i;
        vis[i] = true;
    }
    int counts = 0;//经验优化
    while(hh ≠ tt){
        int x = q[-- tt];
        vis[x] = false;

        for(int i = head[x];~i;i = nex[i]){
            int y = ver[i],z = edge[i];
            if(dist[y] < dist[x] + z - mid){
                dist[y] = dist[x] + z - mid;
                cnt[y] = cnt[x] + 1;
                if(cnt[y] ≥ N)return true;
                if(!vis[y]){
                    vis[y] = true;
                    q[tt ++ ] = y;
                }
            }
        }
    }
    return false;
}


```

∮ 8. 树上问题

8.1 树的直径

树的直径满足如下性质：

- 若有多条直径，则所有的直径之间皆有公共点。
- 直径的两端一定是叶子。
- 树中距离某一直径端点最远的点，至少有一个是该直径的另一个端点。
- 对于树上任意一个点，与之距离最远的一个点，至少有一个直径的端点。

> 给你一个无权无向的树。编写程序以输出该树中最长路径（从一个节点到另一个节点）的长度。在这种情况下，路径的长度是我们从开始到目的地的遍历边数。

8.1.1 树形DP

$O(n)$推荐使用

```
const int N = 50007, M = 500007, INF = 0x3f3f3f3f;

int n, m, F[N], D[N], ans ;
int ver[N], nex[M], head[N], edge[N], tot;
```

```cpp
bool vis[N];
void add(int x, int y, int z)
{
    ver[tot] = y;
    edge[tot] = z;
    nex[tot] = head[x];
    head[x] = tot ++ ;
}

void dp(int x){
    vis[x] = 1;
    for(int i = head[x] ;~i; i =nex[i]){
        int y = ver[i], z = edge[i];
        if(vis[y])continue;
        dp(y);
        ans = max(ans, D[x] + D[y] + z);
        D[x] = max(D[x], D[y] + z);
    }
}

int main(){
    scanf("%d", &n);
    memset(head, -1, sizeof head);
    for(int i = 1; i < n; ++ i){
        int x, y;
        scanf("%d%d", &x, &y);
        add(x, y, 1), add(y, x, 1);
    }
    dp(1);
    printf("%d\n", ans);
    return 0;
}

```

8.1.2 两次DFS / BFS（找到直径的两个端点）

$O(n)$

```cpp
//两次dfs一次求P一次求Q
void dfs(int u,int &ed){
    if(dis[u] > ans)ans = dis[u],ed = u;
    vis[u] = 1;
    for(int i = head[u];~i;i = nex[i]){
        int v = ver[i],w = edge[i];
        if(vis[v])continue;
        dis[v] = dis[u] + w;
        dfs(v,ed);
    }
    return ;
}
int p,q;
void solve(){
    dfs(1,p);
    ans = dis[p] = 0;
    memset(vis,0,sizeof vis);
    dfs(p,q);
    cout << ans << endl;
}
```

> 在一棵无根树上，找A, B, C三个点，使得$AB + BC(AC > BC)$或$AC + AB(BC > AC)$最大。

> 我们假设$AC > BC$，那么答案就是找到最大$AB + BC$简单贪心一下，先令AB最大，然后再找一个相应的最大的BC，强调要满足(AC>AB)所以我们首先找出树的直径，然后枚举除了两个端点外的点C，使得$min(AC, BC)$最大，答案就是树的直径$+min(AC, BC)$

```cpp
typedef long long ll;
const int N = 500007, M = 5000007, INF = 0x3f3f3f3f;

int n, m;
ll d[N], copy_d[N];
bool vis[N];
int head[N], ver[M], nex[M], edge[M], tot;

void add(int x, int y, int z)
{
    ver[tot] = y;
    edge[tot] = z;
    nex[tot] = head[x];
    head[x] = tot ++ ;
}

int bfs(int S)
{
    queue<int>q;
    memset(d, 0, sizeof d);
    memset(vis, 0, sizeof vis);
    //while(q.size())q.pop();
    q.push(S), vis[S] = 1;

    int maxx = 0, maxid;
    while(q.size()){
        int x = q.front();
        q.pop();
        for(int i = head[x]; ~i; i = nex[i]){
            int y = ver[i], z = edge[i];
            if(vis[y])continue;
            vis[y] = 1;
            d[y] = d[x] + z;
            q.push(y);
            if(d[y] > maxx){
                maxx = d[y];
                maxid = y;
            }
        }
    }
    return maxid;
}

int main()
{
    scanf("%d%d", &n, &m);
    memset(head, -1, sizeof head);
    for(int i = 1; i ≤ m; ++ i){
        int x, y, z;
        scanf("%d%d%d", &x, &y, &z);
        add(x, y, z);
        add(y, x, z);
    }
    int p, q;
    p = bfs(1);
    q = bfs(p);
```

```
    ll ans = d[q];//the length of AB
    for(int i = 1; i ≤ n; ++ i){
        copy_d[i] = d[i];
    }
    bfs(q);
    ll res = 0;
    for(int i = 1; i ≤ n; ++ i){
        res = max(res, min(d[i], copy_d[i]));//find the longest edges among the remaining edges
    }
    ans += res;
    printf("%lld\n", ans);
    return 0;
}

```

8.2 动态修改树的边权并求每个时刻的直径（线段树）

> 有一个n个点的带权无向树，q次操作，每次修改一条边的权值，要求在每次修改后，输出树的直径大小，强制在线。

```
/*lmx,rmx,lmi,rmi,ld,rd,lrd,ans前缀/后缀的最大/最小和，前缀右减左最大，
后缀右减左最大，两端点必须选满右减左最大，任意子串右减左最大*/
#include<bits/stdc++.h>
#define ls(a) ((a)<<1)
#define rs(a) ((a)<<1|1)
#define N 200001
#define int long long
using namespace std;
typedef long long ll;
#define G if(++ip==ie)if(fread(ip=ibuf,1,LL,stdin))
const int LL=1<<19;
char ibuf[LL],*ie=ibuf+LL,*ip=ie-1;
inline char nc(void){G;return *ip;}
//#define getchar nc
inline ll read(void){
    char opt;ll flag=1,res=0;
    while((opt=getchar())<'0'||opt>'9')if(opt=='-')flag=-1;
    while(opt≥'0'&&opt≤'9'){res=(res<<3)+(res<<1)+opt-'0';opt=getchar();}
    return res*flag;
}
int n,m,st[N],top,q;
struct edge{int to;ll v;int x;};vector<edge>g[N];
ll
w,sum[N<<2],lmx[N<<2],rmx[N<<2],lmi[N<<2],rmi[N<<2],ld[N<<2],rd[N<<2],lrd[N<<2],ans[N<<2],lastans,p[N]
[2];
void pushup(int x){
    sum[x]=sum[ls(x)]+sum[rs(x)];
    lmx[x]=max(lmx[ls(x)],sum[ls(x)]+lmx[rs(x)]);
    rmx[x]=max(rmx[rs(x)],sum[rs(x)]+rmx[ls(x)]);
    lmi[x]=min(lmi[ls(x)],sum[ls(x)]+lmi[rs(x)]);
    rmi[x]=min(rmi[rs(x)],sum[rs(x)]+rmi[ls(x)]);
    ld[x]=max(ld[ls(x)],max(ld[rs(x)]-sum[ls(x)],lrd[ls(x)]+lmx[rs(x)]));
    rd[x]=max(rd[rs(x)],max(sum[rs(x)]+rd[ls(x)],lrd[rs(x)]-rmi[ls(x)]));
    lrd[x]=max(lrd[ls(x)]+sum[rs(x)],lrd[rs(x)]-sum[ls(x)]);
    ans[x]=max(max(ans[ls(x)],ans[rs(x)]),max(ld[rs(x)]-rmi[ls(x)],rd[ls(x)]+lmx[rs(x)]));
}
inline void Build(int pos,int l,int r){
    ll v;if(l==r)return sum[pos]=
(v=st[l]),lmx[pos]=rmx[pos]=max(v,0ll),lmi[pos]=rmi[pos]=min(v,0ll),ld[pos]=rd[pos]=lrd[pos]=ans[pos]=v,
void();
    int mid=(l+r)>>1;
```

```
    Build(ls(pos),l,mid),Build(rs(pos),mid+1,r);
    pushup(pos);
}
inline void Change(int pos,int l,int r,int x,ll v){
    if(l==r)return
sum[pos]=v,lmx[pos]=rmx[pos]=max(v,0ll),lmi[pos]=rmi[pos]=min(v,0ll),ld[pos]=rd[pos]=lrd[pos]=ans[pos]=v
,void();
    int mid=(l+r)>>1;
    (x≤mid)?Change(ls(pos),l,mid,x,v):Change(rs(pos),mid+1,r,x,v);
    pushup(pos);
}
inline void dfs(int x,int prt){
    int y;for(auto t:g[x])if((y=t.to)^prt)st[*p[t.x]=++top]=t.v,dfs(y,x),st[p[t.x][1]=++top]=-t.v;
}
signed main(void){
    int i,x;ll y,z;n=read(),q=read(),w=read();
    for(i=1;i<n;++i)x=read(),y=read(),z=read(),g[x].push_back({y,z,i}),g[y].push_back({x,z,i});
    dfs(1,0);Build(1,1,top);
    while(q--){
        x=(read()+lastans)%(n-1)+1,y=(read()+lastans)%w;
        Change(1,1,top,*p[x],y),Change(1,1,top,p[x][1],-y);
        printf("%lld\n",lastans=ans[1]);
    }
    return 0;
}
```

8.3 树的重心

重心的性质

- 以树的重心为根时，所有子树的大小都不超过整棵树大小的一半。
- 把两棵树通过一条边相连得到一棵新的树，那么新的树的重心在连接原来两棵树的重心的路径上。
- 在一棵树上添加或删除一个叶子，那么它的重心最多只移动一条边的距离。
- 树中所有点到某个点的距离和中，到重心的距离和是最小的；如果有两个重心，那么到它们的距离和一样。
- 一棵树最多有两个重心，且相邻。

> 给定一棵树,求树的重心的编号以及重心删除后得到的最大子树的节点个数size,如果size相同就选取编号最小的.

```
const int N = 50007, M = 500007, INF = 0x3f3f3f3f;
typedef long long ll;

int n, m;
int head[N], ver[M], nex[M], tot;
ll T;
int son[N], ans, pos;
bool vis[N];

void add(int x, int y)
{
    ver[tot] = y;
    nex[tot] = head[x];
    head[x] = tot ++ ;
}

void dfs(int x)
{
    vis[x] = 1, son[x] = 1;
    int res = 0;
    for(int i = head[x]; ~i; i = nex[i]){
        int y = ver[i];
        if(!vis[y]){
```

```
            dfs(y);
            son[x] += son[y];
            res = max(res, son[y]);//x下面的子树
        }
    }
    res = max(res, n - son[x]);//x上面的子树
    if(res < ans || (res == ans && x < pos)){
        ans = res, pos = x;
    }
}

int main()
{
    scanf("%lld", &T);
    while(T -- ){
        tot = 0;
        memset(head, -1, sizeof head);
        memset(vis, 0, sizeof vis);
        ans = INF;

        scanf("%d", &n);
        for(int i = 1; i < n; ++ i){
            int x, y;
            scanf("%d%d", &x, &y);
            add(x, y), add(y, x);
        }
        dfs(1);
        printf("%d %d\n", pos, ans);
    }
    return 0;
}
```

§ 9. 最近公共祖先

> 注意x和y的LCA可以是x或者y本身

9.1 LCA的在线倍增算法

(1) 向上标记法 O(n)

(2) 倍增

fa[i, j] 表示从i开始，向上走2^j步所能走到的节点。0 <= j <= logn

depth[i] 表示深度

哨兵：如果从i开始跳2^j步会跳过根节点，那么fa[i, j] = 0。depth[0] = 0

步骤：

[1] 先将两个点跳到同一层

[2] 让两个点同时往上跳，一直跳到它们的最近公共祖先的下一层。

预处理 O(nlogn)

查询 O(logn)


```
/*给定一棵包含 n个节点的有根无向树，有 m个询问，每个询问
给出了一对节点的编号 x和 y，询问 x与 y的祖孙关系。

注意要先算t，这里使用bfs防止爆栈，使用手写循环队列效率更高
*/
const int N = 50007, M = 500007 ,INF = 0x3f3f3f3f;
int t;
int n, m;
int ver[M], nex[M], edge[M], head[N],tot;
```

```cpp
int deep[N];
bool vis[N];
int q[M];
int f[N][30];

void add(int x,int y){
    ver[tot] = y;
    nex[tot] = head[x];
    head[x] = tot ++;
}

void bfs(int rt){
    int hh = 0,tt = 0;
    q[tt ++ ] = rt;
    deep[rt] = 1;
    while(hh != tt){
        int x = q[hh ++ ];
        if(hh == N)hh = 0;

        for(int i = head[x];~i;i = nex[i]){
            int y = ver[i];
            if(deep[y])continue;

            deep[y] = deep[x] + 1;
            //dist[y] = dsit[x] + edge[i];
            q[tt ++ ] = y;
            if(tt == N)tt = 0;
            f[y][0] = x;
            for(int j = 1;j ≤ t;++j)
                f[y][j] = f[f[y][j - 1]][j - 1];
        }
    }
}

int lca(int x,int y){
    if(deep[x] > deep[y])swap(x,y);
    for(int i = t;i ≥ 0;i --)
        if(deep[f[y][i]] ≥ deep[x])
            y = f[y][i];
    if(x == y)return x;
    for(int i = t;i ≥ 0;--i)
        if(f[x][i] ≠ f[y][i])
        x = f[x][i],y = f[y][i];
    return f[x][0];
}

int main(){
    scanf("%d",&n);

    int root = 0;
    t = (int)(log(n) / log(2)) + 1;

    memset(head,-1,sizeof head);

    for(int i = 1;i ≤ n;++i){
        int x,y;
        scanf("%d%d",&x,&y);
        if(y == -1)root = x;
        else add(x,y),add(y,x);
    }
```

```
70
71      bfs(root);
72
73      scanf("%d",&m);
74
75      while(m--){
76          int x,y;
77          scanf("%d%d",&x,&y);
78          int p = lca(x,y);
79          if(p == x)puts("1");
80          else if(p == y)puts("2");
81          else puts("0");
82      }
83      return 0;
84
85  }
86
```

9.2 LCA的离线Tarjan算法

tarjan算法实际上是对向上标记法的优化

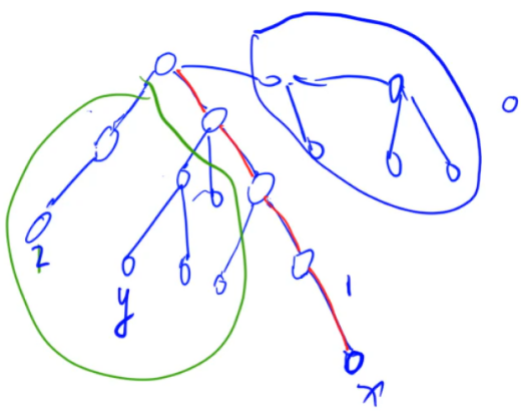
(3) Tarjan——离线求LCA　O(n + m)

在深度优先遍历时，将所有点分成三大类：

[1] 已经遍历过，且回溯过的点

[2] 正在搜索的分支

[3] 还未搜索到的点



2是已经搜索过且已回溯的点

1是正在搜索的点

0是还未搜索过的点

> 题目大意：n结点的树，输出任意两个结点间的最小距离
>
> 思路分析：求两个点的LCA，最小距离即$deep[a] + deep[b] - 2 * deep[lca]$

```
1   int head[N],ver[N<<1],tot,edge[N<<1],nex[N<<1];
2   int n,m,T,t;
3   int fa[N],dis[N],vis[N],ans[N];
4   vector<int>query[N],query_id[N];
5
6
7   void add(int u,int v,int w){
8       ver[tot] = v;
9       edge[tot] = w;
10      nex[tot] = head[u];
11      head[u] = tot ++ ;
12  }
13
14  void add_query(int x,int y,int id){
15      query[x].push_back(y);query_id[x].push_back(id);
16      query[y].push_back(x),query_id[y].push_back(id);
17  }
18
```

```cpp
int Get(int x){
    if(x == fa[x])return x;
    return fa[x] = Get(fa[x]);
}

void tarjan(int u){
    vis[u] = 1;
    for(int i = head[u];~i;i = nex[i]){
        int v = ver[i];
        if(vis[v])continue;
        dis[v] = dis[u] + edge[i];
        tarjan(v);
        fa[v] = u;
    }
    for(int i = 0;i < query[u].size();++i){
        int v = query[u][i],id = query_id[u][i];
        if(vis[v] == 2){
            int lca = Get(v);
            ans[id] = min(ans[id],dis[v] + dis[u] - 2 * dis[lca]);
        }
    }
    vis[u] = 2;
}

int main()
{
    cin>>T;
    while(T--){
        scanf("%d%d",&n,&m);
        for(int i = 1;i ≤ n;++i){
            head[i] = -1,fa[i] = i,vis[i] = 0;
            query[i].clear(),query_id[i].clear();
        }
        tot = 0;
        memset(dis,0,sizeof dis);
        for(int i = 1;i < n;++i){
            int x,y,z;
            scanf("%d%d%d",&x,&y,&z);
            add(x,y,z);add(y,x,z);
        }
        for(int i = 1;i ≤ m;++i){
            int x,y;
            scanf("%d%d",&x,&y);
            if(x == y)ans[i] = 0;
            else {
                add_query(x,y,i);
                ans[i] = INF;
            }
        }
        tarjan(1);
        for(int i = 1;i ≤ m;++i)
        printf("%d\n",ans[i]);
    }
    return 0;
}
```

9.3 树上差分

对边差分

- (u,v)上全部加上w，对于差分数组就是：
- u加上w，v加上w，lca减去2 × w
- 用子树中差分数组的和来还原信息
- 每个点的信息记录的是其到父亲的边的信息

对点差分

- (u,v)上全部加上w，对于差分数组就是：
- u加上w，v加上w，lca减去w，Fatherlca减去w
- 同样用子树中差分数组的和来还原信息

差分和数据结构结合

- 对于一个支持单点修改、区间求和的数据结构，如果使用差分， 就可以支持区间加法、单点查询
- 甚至可以支持区间加法、区间求和
- 一个经典的例子就是用树状数组来完成这些事情
- 用DFS序还可以把放到树上，区间变成子树

对点差分

> 题目大意：一颗树，有 n 个房间，并且有$n-1$根树枝连接。小熊维尼先去a_1，再去 a_2 最后到 a_n，每走到一个房间，他就可以从房间拿一块糖果吃，最后一个房间不用，求每个房间至少需要放多少个糖果。
> 思路分析：也就是他从a_1走到a_2，从a_1到a_2路径上的点权都要++，也就是对点差分。

```cpp
const int N = 500007,M = 1e6 + 7, INF = 0x3f3f3f3f;
int head[N<<1],ver[N<<1],tot,nex[N<<1];
int n,m,T,t;
int f[N][30],vis[N];
int deep[N];
int s[N];//差分数组
int a[N];

void add(int u,int v){
    ver[++tot] = v;
    nex[tot] = head[u];
    head[u] = tot;
}

queue<int>q;

void bfs(){//lca的预处理
    q.push(1);
    deep[1] = 1;//根的深度为1
    while(q.size()){
        int u = q.front();q.pop();
        for(int i = head[u];i;i = nex[i]){
            int v = ver[i];
            if(deep[v])continue;
            f[v][0] = u;
            deep[v] = deep[u] + 1;
            for(int j = 1;j ≤ t;++j)
                f[v][j] = f[f[v][j-1]][j-1];
            q.push(v);
        }
    }
}

int lca(int x,int y){
    if(deep[x] > deep[y])swap(x,y);
    lver(i,t,0)
    if(deep[f[y][i]] ≥ deep[x])
        y = f[y][i];
    if(x == y)return x;
```

```
    lver(i,t,0)
    if(f[x][i] ≠ f[y][i])
        x = f[x][i],y = f[y][i];
    return f[x][0];
}

void dfs(int u,int fa){
    for(int i = head[u];i;i = nex[i]){
        int v = ver[i];
        if(v == fa)continue;
        dfs(v,u);
        s[u] += s[v];
    }
}

int main()
{
    cin>>n;
    t = (int)(log(n) / log(2)) + 1;
    over(i,1,n)
    scanf("%d",&a[i]);
    over(i,1,n-1){
        int x,y;
        scanf("%d%d",&x,&y);
        add(x,y);add(y,x);
    }
    bfs();
    over(i,1,n-1){
        s[a[i]]++;
        s[a[i+1]]++;
        s[lca(a[i],a[i+1])]--;
        s[f[lca(a[i],a[i+1])][0]]--;
    }
    dfs(1,0);//从差分数组还原至原数组
    over(i,2,n)
    s[a[i]]--;//我们是直接正序循环一遍，这样从2开始每个点都是即当一次起点又当一次终点多加了一次
    over(i,1,n)
    printf("%d\n",s[i]);
    return 0;
}

```

对边差分

> 题目大意：有一颗n结点树，n-1条树边，另送m条附加边（非树边），要求先斩断一条树边，再斩断一条非树边（必须砍两条），使得该树不连通。求总方案数。
> 大致思路：对于每一条树边(x,y)，如果x到y的整个路径上连有一条非树边可以形成环，那么除了将x到y的这个树边砍断以外，必须把该形成环的非树边也砍断，整个棵树才不连通，方案数+1。如果x到y之间连有多条，那么在x到y之间不可能将其斩断方案数+0，如果没有环，则有m种方案，方案数+m。累加答案输出

```
int n,m;
int ver[M], edge[M], nex[M], head[M], tot;
int deep[N];
bool vis[N];
int d[N];
int q[N];
int ans;
int f[N][20];

void add(int x,int y){
    ver[tot] = y;
```

```
    nex[tot] = head[x];
    head[x] = tot ++ ;
}

void bfs(){
    deep[1] = 1;
    int hh = 0,tt = 0;
    q[tt ++ ] = 1;

    while(hh ≠ tt){
        int x = q[hh ++ ];
        if(hh == N)hh = 0;

        for(int i = head[x];~i;i = nex[i]){
            int y = ver[i];
            if(deep[y])continue;
            deep[y] = deep[x] + 1;
            q[tt ++ ] = y;
            if(tt == N)tt = 0;
            f[y][0] = x;
            for(int k = 1;k ≤ 16;++k)
                f[y][k] = f[f[y][k - 1]][k - 1];
        }
    }

}

int lca(int x,int y){
    if(deep[x] ≤ deep[y])swap(x,y);
    for(int k = 16;k ≥0;k --)
        if(deep[f[x][k]] ≥ deep[y])
            x = f[x][k];
    if(x == y)return x;

    for(int k = 16;k ≥ 0;k--)
        if(f[x][k] ≠ f[y][k])
            x = f[x][k],y = f[y][k];
    return f[x][0];
}

int dfs(int x ,int fa){
    int res = d[x];
    for(int i = head[x];~i;i = nex[i]){
        int y = ver[i];
        if(y == fa)continue;
        int s = dfs(y,x);
        if(s == 0)ans += m;
        else if(s == 1)ans ++;
        res += s;
    }
    return res;
}

int main(){
    memset(head,-1,sizeof head);
    scanf("%d%d",&n,&m);
    for(int i = 1;i < n;++i){
        int x,y;
        scanf("%d%d",&x,&y);
        add(x,y);
```

```
72          add(y,x);
73      }
74
75      bfs();
76
77      for(int i = 1;i ≤ m;++i){
78          int x,y;
79          scanf("%d%d",&x,&y);
80          int p = lca(x,y);
81          d[x] ++ ,d[y] ++ ,d[p] -=2;
82      }
83
84      dfs(1,-1);
85
86      printf("%d\n",ans);
87      return 0;
88  }
```

∮ 10. 有向图的连通性

tarjan缩点以后的点（强连通分量）

- 要么出度和入度都为0
- 要么有出度，入度为0
- 要么有入度，出度为0
- 要么入度和出度都不为 0，但是对于同一个点的入度和出度不 **同时** 非0

10.1 tarjan模板

题目大意

> 被所有奶牛喜欢的奶牛就是一头明星奶牛。所有奶牛都是自恋狂，每头奶牛总是喜欢自己的。奶牛之间的“喜欢”是可以传递的——如果A 喜欢 B，B 喜欢 C，那么 A 也喜欢 C。牛栏里共有 N 头奶牛，给定一些奶牛之间的爱慕关系，请你算出有多少头奶牛可以当明星。

大体思路

> 受欢迎的奶牛只有可能是图中唯一的出度为零的强连通分量中的所有奶牛，所以若出现两个以上出度为0的强连通分量则不存在明星奶牛，因为那几个出度为零的分量的爱慕无法传递出去。那唯一的分量能受到其他分量的爱慕同时在分量内相互传递，所以该分量中的所有奶牛都是明星。

```
1  int n, m;
2  int dfn[N], low[N];
3  int ind;
4  bool ins[N];
5  int stk[N], top;
6  int head[N], nex[M], ver[M], edge[M], tot;
7  int cnt_scc;
8  vector<int>scc[N];
9  int scc_num[N];//个数
10 int scc_id[N];//id
11
12 void add(int x, int y)
13 {
14     ver[tot] = y;
15     nex[tot] = head[x];
16     head[x] = tot ++ ;
17 }
```

```
18
19 void tarjan(int x)
20 {
21     dfn[x] = low[x] = ++ ind;
22     stk[++ top] = x;
23     ins[x] = 1;
24     for(int i = head[x]; ~i; i = nex[i]){
25         int y = ver[i];
26         if(!dfn[y]){
27             tarjan(y);
28             low[x] = min(low[x], low[y]);
29         }
30         else if(ins[y])
31             low[x] = min(low[x], dfn[y]);
32     }
33     if(dfn[x] == low[x]){//说明回来了
34         cnt_scc ++ ;
35         int y ;
36         do{
37             y = stk[top -- ], ins[y] = 0;
38             scc_id[y] = cnt_scc;
39             scc[cnt_scc].push_back(y);
40             scc_num[cnt_scc] ++ ;
41         }while(x ≠ y);
42     }
43 }
44
45 int out[M];
46
47 int main()
48 {
49     memset(head, -1, sizeof head);
50     scanf("%d%d", &n, &m);
51     for(int i = 1; i ≤ m;++ i){
52         int x, y;
53         scanf("%d%d", &x, &y);
54         add(x, y);
55     }
56
57     for(int i = 1; i ≤ n; ++ i){
58         if(!dfn[i])
59             tarjan(i);
60     }
61     //找有多少个出度为0的点
62     for(int i = 1; i ≤ n; ++ i){//注意缩点的时候是对每一个点都过一遍
63         for(int j = head[i]; ~j; j = nex[j]){
64             int k = ver[j];
65             if(scc_id[i] ≠ scc_id[k]){
66                 out[scc_id[i]] ++ ;//该强连通分量缩点以后的这个点的出度 ++
67             }
68         }
69     }
70     int ans = 0;
71     for(int i = 1; i ≤ cnt_scc; ++ i){
72         if(!out[i]){//有一个出度为0 的既是答案
73             if(!ans)ans = i;
74             else {
75                 puts("0");
76                 return 0;
77             }
```

```
78          }
79      }
80      printf("%d\n", scc_num[ans]);
81      return 0;
82  }
83
```

一般用到tarjan算法的题目步骤都非常相似：

1. **tarjan算法**
2. **缩点，建图**
3. **按照拓扑序递推（这里缩点以后就已经是逆拓扑序了）/ 循环遍历新图求解答案。**

10.2 最大半连通子图

这一道是我认为非常经典的一道有向图 `tarjan` 算法模板例题，包含了：

- **有向图的tarjan模板**
- **去重缩点模板**
- **按拓扑序递推模板**

一个**有向**图 $G = (V, E)$ 称为半连通的 (Semi-Connected)，如果满足：$\forall u, v \in V$，满足 $u \to v$ 或 $v \to u$，即对于图中任意两点 u, v，存在一条 u 到 v 的有向路径或者从 v 到 u 的有向路径。

若 $G' = (V', E')$ 满足，E' 是 E 中所有和 V' 有关的边，则称 G' 是 G 的一个导出子图。

若 G' 是 G 的导出子图，且 G' 半连通，则称 G' 为 G 的半连通子图。

若 G' 是 G 所有半连通子图中包含节点数最多的，则称 G' 是 G 的最大半连通子图。

给定一个有向图 G，请求出 G 的最大半连通子图拥有的节点数 K，以及不同的最大半连通子图的数目 C。

由于 C 可能比较大，仅要求输出 C 对 X 的余数。

输入格式

第一行包含三个整数 N, M, X。N, M 分别表示图 G 的点数与边数，X 的意义如上文所述；

接下来 M 行，每行两个正整数 a, b，表示一条有向边 (a, b)。

图中的每个点将编号为 1 到 N，保证输入中同一个 (a, b) 不会出现两次。

输出格式

应包含两行。


```
1  /*[ZJOI2007]最大半连通子图*/
2  int mod;
3  int n, m;
4  int dfn[N], low[N], num;
5  int ver[M], edge[M], head[N], nex[M], tot;
6  int h[N];
7
8  int Size[N], scc_cnt, scc_id[N];
9  int stk[N], top, ins[N];
10
11 int f[N], g[N];//顶点个数和方案数
12
13 void add(int h[],int x,int y){
14     ver[tot] = y;
15     nex[tot] = h[x];
16     h[x] = tot ++ ;
17 }
18 //!有向图的tarjan模板
19 void tarjan(int x){
20     dfn[x] = low[x] = ++ num;
21     stk[ ++ top] = x;
22     ins[x] = true;
```

```
23
24      for(int i = head[x];~i;i = nex[i]){
25          int y = ver[i];
26          if(!dfn[y]){
27              tarjan(y);
28              low[x] = min(low[x],low[y]);
29          }
30          else if(ins[y]){
31              low[x] = min(low[x], dfn[y]);
32          }
33      }
34
35      if(dfn[x] == low[x]){
36          int y;
37          ++ scc_cnt;
38          do{
39              y = stk[top -- ];
40              ins[y] = false;
41              scc_id[y] = scc_cnt;
42              Size[scc_cnt] ++ ;
43          }while(x ≠ y);
44      }
45  }
46
47  int main()
48  {
49      memset(head,-1,sizeof head);
50      memset(h,-1,sizeof h);
51      scanf("%d%d%d",&n,&m,&mod);
52      while(m -- ){
53          int x,y;
54          scanf("%d%d",&x,&y);
55          add(head,x,y);
56      }
57
58      for(int i = 1;i ≤ n;++i)
59          if(!dfn[i])
60          tarjan(i);
61
62      unordered_set<ll>st;
63      //(u,v) → u * 1000000 + v;
64      //因为一共只有100000个点
65      for(int i = 1;i ≤ n;++i){
66          for(int j = head[i];~j;j = nex[j]){
67              int k = ver[j];
68              int a = scc_id[i], b = scc_id[k];
69              ll hash = a * 100000ll + b;
70              //!去重缩点模板
71              //建新图并判重
72              if(a ≠ b && !st.count(hash)){
73                  add(h,a,b);
74                  st.insert(hash);
75              }
76          }
77      }
78
79  //!按拓扑序递推模板
80      for(int i = scc_cnt;i ≥ 1;-- i){//建完的图是逆拓扑序,按照拓扑序递推,所以必须倒着来
81          if(!f[i]){
82              f[i] = Size[i];
```

```
            g[i] = 1;
        }
        for(int j = h[i];~j;j = nex[j]){
            int k = ver[j];
            if(f[k] < f[i] + Size[k]){
                f[k] = f[i] + Size[k];
                g[k] = g[i];
            }
            else if(f[k] == f[i] + Size[k]){
                g[k] = (g[k] + g[i]) % mod;
            }
        }
    }

    ll maxx = 0,sum = 0;
    for(int i = 1;i ≤ n;++i){
        if(f[i] > maxx){
            maxx = f[i];
            sum = g[i];
        }
        else if(f[i] == maxx){
            sum = (sum + g[i] ) % mod;
        }
    }
    printf("%lld\n%lld\n",maxx, sum);
    return 0;
}

```

10.3 tarjan缩点+记忆化搜索

```
int n, m;
int dfn[M], low[M], num;
int head[N], ver[M], nex[M], tot;
int hc[N], vc[M], nc[M], tc;
int scc_cnt;
int scc_id[N];
bool vis[N];

void add(int x, int y)
{
    ver[tot] = y;
    nex[tot] = head[x];
    head[x] = tot ++ ;
}

void add_c(int x, int y)
{
    vc[tc] = y;
    nc[tc] = hc[x];
    hc[x] = tc ++ ;
}

int w[N];
int val[N];
int stk[N], top;
bool ins[N];
int f[M];
void tarjan(int x)
```

```
{
    dfn[x] = low[x] = ++ num;
    stk[++ top] = x;
    ins[x] = true;

    for(int i = head[x]; ~i; i = nex[i]){
        int y = ver[i];
        if(!dfn[y]){
            tarjan(y);
            low[x] = min(low[x], low[y]);
        }
        else if(ins[y])
            low[x] = min(low[x], dfn[y]);
    }
    if(low[x] == dfn[x]){
        int y;
        scc_cnt ++ ;
        do{
            y = stk[top -- ];
            ins[y] = false;
            scc_id[y] = scc_cnt;
            val[scc_cnt] += w[y];
        }while(x ≠ y);
    }
    return ;
}

//记忆化搜索
void dfs(int x)
{
    if(f[x])return ;
    f[x] = val[x];
    int res = 0;
    for(int i = hc[x] ;~i;i = nc[i]){
        int y = vc[i];
        if(!f[y])dfs(y);
        res = max(res, f[y]);
    }
    f[x] += res;//选择一个最大的分支（因为只能走一条路径）
}

int main()
{
    scanf("%d%d", &n, &m);
    for(int i = 1; i ≤ n; ++ i)
        scanf("%d", &w[i]);

    memset(head, -1, sizeof head);
    memset(hc, -1, sizeof hc);

    for(int i = 1; i≤ m; ++i){
        int x, y;
        scanf("%d%d", &x, &y);
        add(x, y);
    }

    for(int i = 1; i ≤ n; ++ i)
        if(!dfn[i])
            tarjan(i);

```

```
    for(int x = 1; x ≤ n; ++ x){
        for(int i = head[x]; ~i; i = nex[i]){
            int y = ver[i];
            if(scc_id[x] ≠ scc_id[y])
                add_c(scc_id[x], scc_id[y]);
        }
    }
    memset(vis, 0, sizeof vis);

    for(int i = 1; i ≤ scc_cnt; ++ i)
        if(!f[i])dfs(i);

    int ans = 0;

    for(int i = 1; i ≤ scc_cnt;++ i)
        ans = max(ans, f[i]);

    printf("%d\n", ans);
    return 0;
}

```

```
int dp(int x)
{
    if(f[x])return f[x];//记忆化搜索
    int res = 0;

    for(int i = hc[x]; ~i; i = nc[i]){
        int y = vc[i];
        res = max(res, dp(y));//找一条最长链
    }
    f[x] = res + scc_size[x];
    return f[x] ;
}
...
        int ans = 0;
        for(int i = 1; i ≤ scc_cnt; ++ i){
            if(!in[i]){
                memset(f, 0 ,sizeof f);
                //memset(vis, 0, sizeof vis);
                ans = max(ans, dp(i));
            }
        }
```

10.4 几个有向图连通性的结论

有向图

- **一个有向图最少给多少个点提供资源（有连边的点之间可以互相传达）使得所有的点都拥有资源**

结论：缩点以后入度为0的点的个数（因为入度为0则它不能被其他的点支援，所以只能直接给它资源）

- **连多少条边使得整个有向图变为强连通图**

结论：$max\{p,q\}$（缩点以后，入度为0的点的数量记为p，出度为0的点的数量记为q）

- **最大可以增加多少条边使得这个图仍然不是强连通图**

结论：$n*(n-1)-m-min*(n-minv)$（其中n为点数、m为边数、minv为缩点以后的点中入度和出度至少有一个为0的包含节点个数最少的点）

```
int n, m;
int dfn[N], low[N], num;
```

```
int ins[N], stk[N], top;
int head[N], edge[M], nex[N], ver[N], tot;
int hs[N];
int scc_cnt, Size[N], scc_id[N];
int out[N], in[N];

void add(int h[], int x, int y )
{
    ver[tot] = y;
    nex[tot] = h[x];
    h[x] = tot ++ ;
}

void tarjan(int x)
{
    dfn[x] = low[x] = ++num;
    stk[++ top] = x;
    ins[x] = true;
    for(int i = head[x];~i ;i = nex[i]){
        int y = ver[i];
        if(!dfn[y]){
            tarjan(y);
            low[x] = min(low[x], low[y]);
        }
        else if(ins[y])
            low[x] = min(low[x], dfn[y]);
    }

    if(dfn[x] == low[x]){
        ++scc_cnt;
        int y;
        do{
            y = stk[top -- ];
            ins[y] = false;
            scc_id[y] = scc_cnt;
            Size[scc_cnt] ++;
        }while(x =/= y);
    }
}

int main()
{
    memset(head,-1,sizeof head);
    memset(hs,-1,sizeof hs);
    scanf("%d",&n);
    for(int i = 1;i ≤ n;++i){
        int b;
        while(~scanf("%d",&b) && b){
            add(head,i,b);
        }
    }

    for(int i = 1;i ≤ n;++i)
        if(!dfn[i])
            tarjan(i);

    for(int x = 1;x ≤ n;++x)
    {
        for(int i = head[x];~i;i = nex[i])
        {
```

```
            int y = ver[i];
            int a = scc_id[x],b = scc_id[y];
            if(a ≠ b){
                add(hs,a,b);
            }
        }
    }

    for(int x = 1;x ≤ scc_cnt;++x){
        for(int i = hs[x];~i;i = nex[i])
        {
            int y = ver[i];
            out[x] ++ ;
            in[y] ++ ;
        }
    }

    int maxo = 0;
    int maxi = 0;
    for(int x = 1;x ≤ scc_cnt;++x)
    {
        if(!out[x])maxo ++ ;
        if(!in[x])maxi ++ ;
    }
    printf("%d\n",maxi);
    if(scc_cnt == 1)
        puts("0");
    else printf("%d\n",max(maxo,maxi));

    return 0;
}
```

$\oint$ 11. 无向图的连通性

- 在一张连通的无向图中，对于两个点 u和 v，如果无论删去哪条边（只能删去一条）都不能使它们不连通d，我们就说 u和v 边双连通 。
- 在一张连通的无向图中，对于两个点u和v，如果无论删去哪个点（只能删去一个，且不能删u和v自己）都不能使它们不连通，我们就说u和v点双连通 。
- **边双连通具有传递性**，即，若x, y边双连通， y, z边双连通，则 x, z边双连通。
- **点双连通不具有传递性。**

> 经验总结：
>
> - 注意可能由孤立点
> - 看好数据范围点若从0开始输入，则需要从0开始遍历
> - 孤立点也是双连通分量

线性算法tarjan

11.1 tarjan算法求无向图的桥、边双连通分量并缩点

```
// tarjan算法求无向图的桥、边双连通分量并缩点
const int SIZE = 100010;
int head[SIZE], ver[SIZE * 2], Next[SIZE * 2];
int dfn[SIZE], low[SIZE], c[SIZE];
int n, m, tot, num, dcc, tc;
bool bridge[SIZE * 2];
int hc[SIZE], vc[SIZE * 2], nc[SIZE * 2];

```

```cpp
void add(int x, int y) {
    ver[++tot] = y, Next[tot] = head[x], head[x] = tot;
}

void add_c(int x, int y) {
    vc[++tc] = y, nc[tc] = hc[x], hc[x] = tc;
}

void tarjan(int x, int in_edge) {//注意in_edge 是前向星的编号，所以tarjan的时候要用 i 而不是x
    dfn[x] = low[x] = ++num;
    for (int i = head[x]; i; i = Next[i]) {
        int y = ver[i];
        if (!dfn[y]) {
            tarjan(y, i);//!tarjan(y,  i ); , is y ans i !!!
            low[x] = min(low[x], low[y]);
            if (low[y] > dfn[x])
                bridge[i] = bridge[i ^ 1] = true;
        }
        else if (i != (in_edge ^ 1))
            low[x] = min(low[x], dfn[y]);
    }
}

void dfs(int x) {
    c[x] = dcc;
    for (int i = head[x]; i; i = Next[i]) {
        int y = ver[i];
        if (c[y] || bridge[i]) continue;
        dfs(y);
    }
}

int main() {
    cin >> n >> m;
    tot = 1;
    for (int i = 1; i <= m; i++) {
        int x, y;
        scanf("%d%d", &x, &y);
        add(x, y), add(y, x);
    }
    for (int i = 1; i <= n; i++)
        if (!dfn[i]) tarjan(i, 0);
    for (int i = 2; i < tot; i += 2)
        if (bridge[i])
            printf("%d %d\n", ver[i ^ 1], ver[i]);

    for (int i = 1; i <= n; i++)
        if (!c[i]) {
            ++dcc;
            dfs(i);
        }
    printf("There are %d e-DCCs.\n", dcc);
    for (int i = 1; i <= n; i++)
        printf("%d belongs to DCC %d.\n", i, c[i]);

    tc = 1;
    for (int i = 2; i <= tot; i++) {
        int x = ver[i ^ 1], y = ver[i];
        if (c[x] == c[y]) continue;
        add_c(c[x], c[y]);
```

```
    }
    printf("缩点之后的森林，点数 %d，边数 %d\n", dcc, tc / 2);
    for (int i = 2; i < tc; i += 2)
        printf("%d %d\n", vc[i ^ 1], vc[i]);
}
```

11.2 tarjan算法求无向图的割点、点双连通分量并缩点

```

// tarjan算法求无向图的割点、点双连通分量并缩点
#include<iostream>
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<vector>
using namespace std;
const int SIZE = 100010;
int head[SIZE], ver[SIZE * 2], Next[SIZE * 2];
int dfn[SIZE], low[SIZE], stack[SIZE], new_id[SIZE], c[SIZE];
int n, m, tot, num, root, top, cnt, tc;
bool cut[SIZE];
vector<int> dcc[SIZE];
int hc[SIZE], vc[SIZE * 2], nc[SIZE * 2];

void add(int x, int y) {
    ver[++tot] = y, Next[tot] = head[x], head[x] = tot;
}

void add_c(int x, int y) {
    vc[++tc] = y, nc[tc] = hc[x], hc[x] = tc;
}

void tarjan(int x) {
    dfn[x] = low[x] = ++num;
    stack[++top] = x;
    //if (x == root && head[x] == -1) { // 孤立点
    if (x == root && head[x] == 0) { // 孤立点
    //这里head初始化的是0所以写0，
    //!但是我一般初始化为-1， 所以要写成-1！！！！！
    //这里是lyd的模板，但是和我习惯的格式有点不同，要注意不然会WA
        dcc[++cnt].push_back(x);
        return;
    }
    int flag = 0;
    for (int i = head[x]; i; i = Next[i]) {
        int y = ver[i];
        if (!dfn[y]) {
            tarjan(y);
            low[x] = min(low[x], low[y]);
            if (low[y] ≥ dfn[x]) {
                flag++;
                if (x ≠ root || flag > 1) cut[x] = true;
                cnt++;
                int z;
                do {
                    z = stack[top--];
                    dcc[cnt].push_back(z);
                } while (z ≠ y);
                dcc[cnt].push_back(x);
```

```
            }
        }
        else low[x] = min(low[x], dfn[y]);
        //注意这里不要放错位置了，要和if(!dfn[y])交相呼应
    }
}

int main() {
    cin >> n >> m;
    tot = 1;
    for (int i = 1; i ≤ m; i++) {
        int x, y;
        scanf("%d%d", &x, &y);
        if (x == y) continue;
        add(x, y), add(y, x);
    }
    for (int i = 1; i ≤ n; i++)
    //这里注意n要正确，如果是自己计算的n错了以后dcc_cnt就会出错，导致WA，
    //要求自己算max求点数的时候一定记得每次都要初始化
        if (!dfn[i]) root = i, tarjan(i);
    for (int i = 1; i ≤ n; i++)
        if (cut[i]) printf("%d ", i);
    puts("are cut-vertexes");
    for (int i = 1; i ≤ cnt; i++) {
        printf("v-DCC #%d:", i);
        for (int j = 0; j < dcc[i].size(); j++)
            printf(" %d", dcc[i][j]);
        puts("");
    }
    // 给每个割点一个新的编号(编号从cnt+1开始)
    num = cnt;
    for (int i = 1; i ≤ n; i++)
        if (cut[i]) new_id[i] = ++num;
    // 建新图，从每个v-DCC到它包含的所有割点连边
    tc = 1;
    for (int i = 1; i ≤ cnt; i++)
        for (int j = 0; j < dcc[i].size(); j++) {
            int x = dcc[i][j];
            if (cut[x]) {
                add_c(i, new_id[x]);
                add_c(new_id[x], i);
            }
            else c[x] = i; // 除割点外，其它点仅属于1个v-DCC
        }
    printf("缩点之后的森林，点数 %d，边数 %d\n", num, tc / 2);
    printf("编号 1~%d 的为原图的v-DCC，编号 >%d 的为原图割点\n", cnt, cnt);
    for (int i = 2; i < tc; i += 2)
        printf("%d %d\n", vc[i ^ 1], vc[i]);
}

```

11.3 结论：变成边双连通分量所需要新建的边数

无向图

连多少条边使得整个无向图变成任意两个点之间都有两条完全不相同的路径。

- 结论1：任意两个点之间都有两条完全不相同的路径 => 没有割边 => 边双连通图
- 结论2： 需要新建的边数：$\frac{cnt+1}{2}$ (下取整)（其中cnt为将所有的边双连通图缩点以后有割桥形成树的叶子节点，也就是度数为1的结点数）

> 首先我们可以对图进行边连通分量缩点， 缩点后图就会变成一颗树， 代表任意2点之间的路径是唯一的。 这时候题目转化为添加最少的边使树上任意
> 2点的路径至少有2条。

```cpp
/* AcWing 395. 冗余路径 */
const int N = 5007, M = 50007, INF = 0x3f3f3f3f;

int n, m;
int dfn[N], low[N], num;
int head[N], nex[M], ver[M], tot;
int ans;
int stk[N], top;
bool bridge[M];
int d[M];
int dcc_cnt,dcc_id[M];

void add(int x, int y){
    ver[tot] = y;
    nex[tot] = head[x];
    head[x] = tot ++ ;
}

void tarjan(int x, int in_edge){
    dfn[x] = low[x] = ++ num;
    stk[++ top] = x;
    for(int i = head[x];~i; i = nex[i]){
        int y = ver[i];
        if(!dfn[y]){
            tarjan(y,i);
            low[x] = min(low[x], low[y]);

            if(dfn[x] < low[y])//x是y的父节点y无法在不经过(x,y)的前提下到达x或者到达比x更早访问的结点,存在割桥
                bridge[i] = bridge[i ^ 1] = true;
        }
        else if(i != (in_edge ^ 1))
        low[x] = min(low[x], dfn[y]);
    }

    if(dfn[x] == low[x]){
        int y ;
        ++ dcc_cnt;
        do{
            y = stk[top -- ];
            dcc_id[y] = dcc_cnt;
        }while(x != y);
    }
}

int main(){
    memset(head,-1,sizeof head);
    scanf("%d%d",&n, &m);
    for(int i = 1;i ≤ m;++i){
        int x,y;
        scanf("%d%d",&x, &y);
        add(x,y), add(y, x);
    }
    //因为都连通所以只需要跑一次即可
    tarjan(1, -1);

    for(int i = 0;i < tot; i += 2)
```

```
        if(bridge[i]){
            int x = ver[i], y = ver[i ^ 1];
            d[dcc_id[x]] ++ ;
            d[dcc_id[y]] ++ ;
        }

    int cnt = 0;
    for(int i = 1;i ≤ dcc_cnt;++i)
        if(d[i] == 1)
            cnt ++;
    printf("%d\n", (cnt + 1) / 2);
    return 0;
}

```

```
/*给定一个由 n个点 m条边构成的无向图，请你求出该图删除一个点之后，连通块最多有多少。*/
const int N = 50007, M = 500007, INF = 0x3f3f3f3f;
int n, m;
int dfn[N], low[N], num;
int ver[M], nex[M], edge[M], head[N], tot;
int ans;

void add(int x,int y){
    ver[tot] = y;
    nex[tot] = head[x];
    head[x] = tot ++ ;
}

void tarjan(int x, int rt){//计算该割点连接了多少个连通块
    dfn[x] = low[x] = ++ num;
    int cnt = 0;//(割开这个点会将图分成cnt个连通块)
    for(int i = head[x]; ~i;i = nex[i]){
        int y = ver[i];
        if(!dfn[y]){
            tarjan(y, rt);
            low[x] = min(low[x], low[y]);
            if(dfn[x] ≤ low[y]){//该割点返回了一条独立的路
                cnt ++ ;
            }
        }
        else low[x] = min(low[x], dfn[y]);
    }
    if(x ≠ rt)cnt ++ ;//如果不是根的话说明该点的上面应该可以割出来一个连通块

    ans = max(ans,cnt);
}

int main(){
    while(~scanf("%d%d",&n,&m) && n || m){
        memset(head, -1, sizeof head);
        memset(dfn, 0, sizeof dfn);
        tot = num = 0;
        for(int i = 1;i ≤ m;++i){
            int x,y;
            scanf("%d%d",&x,&y);
            add(x,y);add(y,x);
        }
        ans = 0;
        int cnt = 0;
        for(int i = 0;i < n;++ i){//数据从0到n - 1
```

```
46              if(!dfn[i]){
47                  cnt ++ ;//可能有孤立点(孤立连通块)
48                  tarjan(i, i);
49              }
50          }
51          printf("%d\n", ans + cnt - 1);//减去多算的这个点(ans是由这个点割开之后能分散开的连通块个数)
52      }
53  }
54
55
```

11.4 动态求解当前图中的桥的数目（割边缩点+并查集+lca+优化 $O(m+qlogn)$）

```
1  //这道题代码虽然比较长，但是每个函数各司其职，井井有条，写起来非常舒服
2  #include<iostream>
3  #include<algorithm>
4  #include<cstdio>
5  #include<math.h>
6  #include<cstring>
7  #include<bitset>
8  #include<vector>
9  #include<queue>
10 #define ls (p<<1)
11 #define rs (p<<1|1)
12 #define over(i,s,t) for(register int i = s;i ≤ t;++i)
13 #define lver(i,t,s) for(register int i = t;i ≥ s;--i)
14 //#define int __int128
15 #define lowbit(p) p&(-p)
16 using namespace std;
17
18 typedef long long ll;
19 typedef pair<int,int> PII;
20 const int INF = 0x3f3f3f3f;
21 const int N = 2e5+7;
22 const int M = 5e5+7;
23
24 int head[N],nex[M],ver[M],tot = 1;//原图
25 int hc[N],nc[M],vc[M],tc = 1;//缩点后建图
26 int dfn[N],low[N],num;//tarjan专用
27 int dcc,cnt;//强连通专用
28 int n,m,t,T;//输入专用
29 bool bridge[M];//割边专用
30 int deep[N],f[N][20];//LCA专用
31 int c[N];//缩点专用
32 int fa[N];//并查集专用
33
34 void add(int x,int y){
35     ver[++tot] = y;nex[tot] = head[x];head[x] = tot;
36 }
37
38 void add_c(int x,int y){
39     vc[++tc] = y;nc[tc] = hc[x];hc[x] = tc;
40 }
41
42 void tarjan(int x,int in_edge){
43     dfn[x] = low[x] = ++num;
44     for(int i = head[x];i;i = nex[i]){
45         int y = ver[i];
```

```
        if(!dfn[y]){
            tarjan(y,i);
            low[x] = min(low[x],low[y]);
            if(low[y] > dfn[x])
                bridge[i] = bridge[i ^ 1] = true;//成对变换
        }
        else if(i ≠ (in_edge ^ 1))//i和in_edge都是前向星的指针编号
            low[x] = min(low[x],dfn[y]);
    }
}

void dfs(int x){
    c[x] = dcc;
    for(int i = head[x];i;i = nex[i]){
        int y = ver[i];
        if(c[y] || bridge[i])continue;
        dfs(y);
    }
}

queue<int>q;

void bfs(){//求的是缩点后的图
    deep[1] = 1;//根节点的深度是1
    q.push(1);
    while(q.size()){
        int x = q.front();
        q.pop();
        for(int i = hc[x];i;i = nc[i]){
            int y = vc[i];
            if(deep[y])continue;
            deep[y] = deep[x] + 1;
            f[y][0] = x;
            over(j,1,19)
                f[y][j] = f[f[y][j - 1]][j - 1];
            q.push(y);
        }
    }
}

int lca(int x,int y){
    if(deep[x] > deep[y])swap(x,y);
    lver(i,18,0)
        if(deep[f[y][i]] ≥ deep[x])
            y = f[y][i];
    if(x == y)return x;
    lver(i,18,0)
    if(f[x][i] ≠ f[y][i])
        x = f[x][i],y = f[y][i];
    return f[x][0];
}

int Get(int x){
    if(x == fa[x])return x;
    return fa[x] = Get(fa[x]);
}

int main()
{
    while(scanf("%d%d",&n,&m) ≠ EOF && n){
```

```
106
107         tot = 1;dcc = num = 0;
108         over(i,1,n)
109             head[i]  = hc[i] = dfn[i] = low[i] = deep[i] = c[i] = 0;
110         over(i,1,2 * m + 1)
111             bridge[i] = 0;
112         over(i,1,m){
113             int x,y;
114             scanf("%d%d",&x,&y);
115             add(x,y);add(y,x);
116         }
117         over(i,1,n)
118             if(!dfn[i])
119                 tarjan(i,0);
120         //两个套路都一样
121         over(i,1,n)
122             if(!c[i])
123                 ++dcc,dfs(i);
124         //缩完点该建图了
125         tc = 1;
126         over(i,2,tot){//建图就用到了成对变换
127                 int x = ver[i ^ 1],y = ver[i];
128                 if(c[x] == c[y])continue;
129                 add_c(c[x],c[y]);
130                 //因为之前是无向图转有向图是双边，这里已经是有向图了只需要建单边即可
131         }
132         //lca预处理
133         bfs();
134         //并查集初始化
135         over(i,1,dcc)
136             fa[i] = i;
137         scanf("%d",&t);
138         int ans = dcc - 1;//dcc是缩点后的点数，而ans是边数，所以最开始ans，树的边数等于点数 - 1
139         printf("Case %d:\n", ++T);
140         while(t--){
141             int x,y;
142             scanf("%d%d",&x,&y);
143             x = c[x],y = c[y];
144             int p = lca(x,y);
145             x = Get(x);
146             while(deep[x] > deep[p]){
147                 fa[x] = f[x][0];//先给fa赋值
148                 ans--;
149                 x = Get(x);//这样get的时候x也能往上走
150             }
151             y = Get(y);
152             while(deep[y] > deep[p]){
153                 fa[y] = f[y][0];
154                 ans--;
155                 y = Get(y);
156             }
157             printf("%d\n",ans);
158         }
159         cout<<endl;
160     }
161     return 0;
162 }
163
164
```

$\oint$ 12. 2 - SAT问题

~~注意一个坑，2SAT问题中如果要求你输出方案，如果你的代码输出的跟样例不一样，不要着急，因为2SAT 问题本来就是有多解，结果我样例不过，交上去就A了~~

方案输出时，color[x]谁小选谁

> 有 n 个布尔变量 $x_1 \sim x_n$，另有 m 个需要满足的条件，每个条件的形式都是 「x_i为 true / false 或 x_j为 true / false」。比如 「x_1为真或 x_3为假」、「x_7为假或 x_2为假」。2-SAT 问题的目标是给每个变量赋值使得所有条件得到满足。

我们在2-sat问题中会得到若干个关系式形如：
p∨q，也就是p||q，p或q，p为true或者q为true

而我们离散数学中学逻辑关系式中有这么一个转化关系：
`p∨q == ¬p → q ∧ ¬q → p`

也就是：

`p || q == -p ->q && -q -> p`

根据p和q 的真假性我们可以得到下列表格

原关系式	建图方式
$p \vee q$	$\neg p \to q \wedge \neg q \to p$
$\neg p \vee q$	$p \to q \wedge \neg q \to \neg p$
$p \vee \neg q$	$\neg p \to \neg q \wedge q \to p$
$\neg p \vee \neg q$	$p \to \neg q \wedge q \to \neg p$

我们按照箭头建有向边，构成一个有向图。

每一条有向边，`u->v`表示如果选择u，那么v也必须选择，不然就违反了关系式。

因此我们对这张图求强连通分量

那么，对于这张图中的每个强连通分量中的点一定要么同时选，要么同时不选。

判断无解：如果$x_{i,0}$和$x_{i,1}$在同一个强连通分量中，那么明显无解，因为二者对立不可能同时存在（$x_{i,0} = 1 \sim n$ 指的是编号为i的点选择`false`，$x_{i,1} = n + 1 \sim 2 * n$指的是编号为i 的点选择`true`）

这张拓扑图中，如果u可以到达v，那么u选择则v也必须选择。

tarjan算法 dfs 式地走一遍就是有向图的拓扑序，因为是用的栈存的节点，所以是拓扑逆序（对于每一个强连通分量都是一个拓扑逆序，块间也是拓扑序，因为每一块之间至多只有一条边，多了就连一块了）。

选择方案：对于每一个点来说，在$x_{i,0}$和$x_{i,1}$中选择拓扑序**较大的点**。这样就可以避免产生冲突了。并且这种方法一定可以构造出解。

当 x 所在的强连通分量的拓扑序在 $\neg x$所在的强连通分量的拓扑序之后取 x 为真 就可以了。在使用 Tarjan 算法缩点找强连通分量的过程中，已经为每组强连通分量标记好顺序了——不过是反着的拓扑序。所以一定要写成 `color[x] < color[-x]`

其中我们一般在2-sat问题中用`i`表示`false`，用`i+n`表示`true`

时间复杂度为：$O(n + m)$

12.1 2 - SAT模板

```cpp
//时间复杂度O(n+m)
//当 x 所在的强连通分量的拓扑序在 ¬x 所在的强连通分量的拓扑序之后取 x
//为真 ,注意我们得到的是拓扑逆序，所以要写成color[x] < color[¬x]
// 其中 i 表示 ¬x，用 i+n 表示 x
p ∨ q == ¬p → q  ∧ ¬q → p
p || q == -p →q && -q → p
typedef long long ll;

```

```cpp
const int N = 2000007, M = 5000007, INF = 0x3f3f3f3f;

int n, m;
int dfn[N], low[N], num;
bool vis[N], ins[N];
int a[N], scc_cnt;
int scc_id[N], color[N];
int stk[N], top;
int ver[M], nex[M], head[N], tot;

void add(int x, int y){
    ver[tot] = y;
    nex[tot] = head[x];
    head[x] = tot ++ ;
}

void tarjan(int x)
{
    dfn[x] = low[x] = ++ num;
    stk[++ top] = x;
    ins[x] = true;
    for(int i = head[x]; ~i; i = nex[i]){
        int y = ver[i];
        if(!dfn[y]){
            tarjan(y);
            low[x] = min(low[x], low[y]);
        }
        else if(ins[y])
            low[x] = min(low[x], dfn[y]);
    }
    if(low[x] == dfn[x]){
        int y;
        ++ scc_cnt;
        do{
            y = stk[top -- ];
            ins[y] = false;
            scc_id[y] = scc_cnt;
            color[y] = scc_cnt;
        }while(x ≠ y);
    }
}

int main()
{
    memset(head, -1, sizeof head);
    scanf("%d%d", &n, &m);
    for(int i = 1; i ≤ m; ++ i){
        int p, q ,c ,d;
        scanf("%d%d%d%d", &p, &c, &q, &d);
        if(c && d){// p 且 q  -p→ q && -q → p
            add(p, q + n);
            add(q, p + n);
        }
        else if(!c && d){//p → q && -q → -p
            add(p + n, q + n);
            add(q, p);
        }
        else if(c && !d){//-p → -q && q → p
            add(p, q);
            add(q + n, p + n);
```

```
        }
        else if(!c && !d){//p -> -q && q -> -p
            add(p + n, q);
            add(q + n, p);
        }
    }
    for(int i = 1; i <= 2 * n; ++ i){
        if(!dfn[i])
            tarjan(i);
    }

    for(int i = 1 ;i <= n; ++ i) {
        if(color[i] == color[i + n]){
            puts("IMPOSSIBLE");
            return 0;
        }
    }
    puts("POSSIBLE");
    //谁小选谁，这里i + n表示的是x (true)
    for(int i = 1; i <= n; ++ i)
        printf("%d ", (color[i] > color[i + n]));
    puts("");
    return 0;
}

```

位运算版

```
    g[p + n * c].push_back(q + n * (d ^ 1));
    g[q + n * d].push_back(p + n * (c ^ 1));
```

12.2 最小字典序解

> 题意：给你n个组，m条规则，每组有俩个人，这两个人不能同时出现，然后m条规则代表着有两个人，这两个人也不能同时出现，问你是否存在每组都能出现一人的选择方案
>
> 解题思路：因为这个需要字典序输出，所以只能用暴力的方法解决，如果x，y在同一条规则里面，那么建立一条边由x指向和y同一组的另一个人，y也这样做，然后开始暴力dfs

注意此模版求的就是最小字典序解

```
#define ll double
#define eps 1e-5

using namespace std;

inline ll Max(ll a,ll b){return a>b?a:b;}
inline ll Min(ll a,ll b){return a<b?a:b;}


#define N 8010*2
#define M 40000+5

struct Edge{
    int to, nex;
}edge[M];

int head[N], edgenum;
void addedge(int u, int v){
    Edge E = {v, head[u]};
    edge[edgenum] = E;
```

```
    head[u] = edgenum ++;
}

bool mark[N];
int Stack[N], top;
void init(){
    memset(head, -1, sizeof(head)); edgenum = 0;
    memset(mark, 0, sizeof(mark));
}

bool dfs(int x){
    if(mark[x^1])return false;//一定是拆点的点先判断
    if(mark[x])return true;

    mark[x] = true;
    Stack[top++] = x;

    for(int i = head[x]; i != -1; i = edge[i].nex)
        if(!dfs(edge[i].to)) return false;

    return true;
}

bool solve(int n){
    for(int i = 0; i < n; i+=2)
        if(!mark[i] && !mark[i^1])
        {
            top = 0;
            if(!dfs(i))//dfs(i) 假设i成立
            {//当i不成立时，把所有因i成立的点都取消标记
                while( top ) mark[ Stack[--top] ] = false;
                if(!dfs(i^1))
                    return false;//若i的对立面也不成立则i点无解
            }
        }
        return true;
}

int main(){
    int n, i, j, m;
    while(~scanf("%d%d",&n,&m)){
        n <<= 1;
        init();

        while(m--)
        {
            int u,v;
            scanf("%d %d",&u,&v); u--, v--;
            addedge(u,v^1);
            addedge(v,u^1);
        }

        if(solve(n))
        {
            for(i=0;i<n;i++)
                if(mark[i])printf("%d\n",i+1);
        }
        else
            printf("NIE\n");

```

```
    }
    return 0;
}
```

需要注意的是，如果题目中有重边的话，使用链式前向星就会导致 `RE` ，改成 `vector` 即可
如下题

12.3 2 - SAT + 二分答案

```

/*ACM-ICPC 2004 Europe - Southwestern） Now or later*/
typedef long long ll;
const int N = 5000007, M = 5000007, INF = 0x3f3f3f3f;

int n, m;
int dfn[N], low[N], num;
int head[N], ver[M], nex[M], tot;
vector<int>g[N];
int a[N][2];
int stk[N], top, scc_cnt;
bool ins[N];
int color[N];

void tarjan(int x)
{
    dfn[x] = low[x] = ++ num;
    stk[++ top] = x;
    ins[x] = true;
    for(size_t i = 0; i < g[x].size(); ++ i){
        int y = g[x][i];
        if(!dfn[y]){
            tarjan(y);
            low[x] = min(low[x], low[y]);
        }
        else if(ins[y])
            low[x] = min(low[x], dfn[y]);
    }
    if(low[x] == dfn[x]){
        int y;
        ++ scc_cnt;
        color[x] = scc_cnt;
        do{
            y = stk[top -- ];
            ins[y] = false;
            color[y] = scc_cnt;
        }while(x ≠ y);
    }
    return ;
}

inline bool check(int x){
    //memset(head, -1, sizeof head);
    for(int i = 1; i ≤ 2 * n; ++ i)
        g[i].clear();
    memset(dfn, 0, sizeof dfn);
    memset(low, 0, sizeof low);
    memset(color, 0, sizeof color);
    memset(ins, 0, sizeof ins);
    tot = num = scc_cnt = 0;
```

```
51
52      //0 : -x : E 早 : i
53      //1 : x : L 晚 : i + n
54
55      //比较之后的两个相差的时间是否小于x
56      //如果小于x的话那么两者就不能同时出现（同时为true）
57      for(int p = 1;p ≤ n;++ p){
58          for(int c = 0; c ≤ 1; ++ c){
59              for(int q = p + 1; q ≤ n; ++ q){
60                  for(int d = 0; d ≤ 1; ++ d){
61                      if(abs(a[p][c] - a[q][d]) < x){
62                          /*if(c && d){//11 → 01, 01
63                              add(p, q + n);
64                              add(q, p + n);
65                          }
66                          else if(!c && d){//01 → 11, 00
67                              add(p + n, q + n);
68                              add(q, p);
69                          }
70                          else if(c && !d){//10 → 00, 11
71                              add(p, q);
72                              add(q + n, p + n);
73                          }
74                          else if(!c && !d){//00 → 10, 01
75                              add(p + n, q);
76                              add(q + n, p);
77                          }*/
78                          g[p + n * c].push_back(q + n * (d ^ 1));
79                          g[q + n * d].push_back(p + n * (c ^ 1));
80                      }
81                  }
82              }
83          }
84      }
85
86      for(int i = 1; i ≤ 2 * n; ++ i)
87          if(!dfn[i])
88              tarjan(i);
89
90      for(int i = 1; i ≤ n; ++ i){
91          if(color[i] == color[i + n])
92              return false;
93      }
94      return true;
95  }
96
97  void solve(){
98      int l = 0, r = 0, ans = -1;
99
100     for(int i =1; i ≤ n; ++ i)
101             scanf("%d%d", &a[i][0], &a[i][1]), r = max(max(a[i][1], a[i][0]), r);
102
103
104     while(l ≤ r){
105         int mid = l + r >> 1;
106         if(check(mid))ans = mid, l = mid + 1;
107         else r = mid - 1;
108     }
109     printf("%d\n", ans);
110     return ;
```

```
}

int main()
{
    while(scanf("%d", &n) ≠ EOF){
        solve();
    }
    return 0;
}
```

∮ 13. 拓扑排序

给定一张有向无环图，若一个序列A满足图中的任意一条边(x,y)x都在y的前面呢么序列A就是图的拓扑排序

实际上拓扑排序就是满足所有的边x指向y，x一定在y的前面。这样按照拓扑排序递推，就可以满足每一个状态都没有循环依赖 －＞没有后效性－＞可以满足DP递推

有向无环图（DAG）<=> 拓扑图

也就是：有向无环图 => 拓扑排序 => 拓扑图 =>每一个状态都没有循环依赖=>没有后效性=>可以DP / 递推求解答案（比如最短 / 长路）

13.1 toposort模板

> 题目信息：有个人的家族很大， 给出每个人的孩子的信息。 输出一个序列，使得每个人的孩子都比那个人后列出。
> 思路分析：因为是家谱图本题一定有解，很明显就是一个topo模板题

```
int ver[M], nex[M], edge[M], head[N], tot;
int n, m;
int d[N];//入度
bool vis[N];
int q[M];
//int ans[M];
int cnt;

void add(int x,int y){
    ver[tot] = y;
    nex[tot] = head[x];
    head[x] = tot ++ ;
    d[y] ++ ;
}

void toposort(){
    int hh = 0, tt = -1;
    for(int i = 1;i ≤ n;++i)
        if(!d[i]){
            q[++ tt] = i;
            if(tt == M)tt = 0;
        }

    while(hh ≤ tt){
        int x = q[hh ++ ];

        //ans[++ cnt] = x;//手写队列可以直接用q数组输出，但是有可能因为数据过大把手写的队列循环挤掉了前面的几项，所以注意一下

        if(hh == M )hh = 0;
        for(int i = head[x];~i;i = nex[i]){
```

```
            int y = ver[i];
            if(-- d[y] == 0){
                q[ ++ tt] = y;
                if(tt == M)tt = 0;//手写的双端队列一定要
                //各项都齐全，如果不全部都加上这句话的话变成普通队列可能因为数据过大而WA/RE
            }
        }
    }
    //return tt == n - 1;//如果要求判断是否有解
}

int main(){
    scanf("%d",&n);
    memset(head, -1, sizeof head);

    for(int i = 1;i ≤ n;++i)
    {
        int son;
        while(~scanf("%d",&son) && son)
            add(i, son);
    }
    toposort();
    for(int i = 0;i < n;++i)
    printf("%d ",q[i]);
}
```

13.2 拓扑排序判断是否有环

> 给定一个由有向边与无向边组成的图，现在需要你把所有的无向边变成有向边，使得形成的图中没有环。如果可以做到请输出该图，否则直接输出"NO"。

> 只需要遵循拓扑排序的规则，给无向图指定方向的时候对于每条边，从拓扑序小的点指向拓扑序大的点，就一定保证无环。

```
int n, m;
int head[N], ver[M], nex[M], edge[M], tot;
int topo[N], din[N];
int cnt;

void add(int x, int y){
    ver[tot] = y;
    nex[tot] = head[x];
    head[x] = tot ++ ;
    din[y] ++ ;
}

bool toposort(int x)
{
    queue<int>q;
    for(int i = 1; i ≤ n; ++ i)
        if(din[i] == 0)q.push(i);//入度为0的先入队
    while(q.size())
    {
        int x = q.front();
        q.pop();
        topo[x] = ++ cnt;//拓扑序
        for(int i = head[x]; ~i; i = nex[i]){
            int y = ver[i];
            if( -- din[y] == 0){
                q.push(y);
            }
```

```
        }
    }
    if(cnt < n)return false;
    return true;
}

int main()
{
    int t ;
    scanf("%d", &t);
    while(t -- ){
        scanf("%d%d", &n, &m);
        memset(head, -1, sizeof head);
        memset(din, 0, sizeof din);
        tot = cnt = 0;
        vector<PII> v;
        for(int i = 1; i ≤ m; ++ i){
            int x, y, op;
            scanf("%d%d%d", &op, &x, &y);
            if(op == 1)add(x, y);
            else v.push_back({x, y});
        }
        if(!toposort(n)){
            puts("NO");
            continue;
        }
        puts("YES");
        for(int x = 1; x ≤ n; ++ x)
        for(int i = head[x]; ~i; i = nex[i]){
            int y = ver[i];
            printf("%d %d\n", x, y);
        }
        for(int i = 0; i < (int)v.size(); ++ i){
            int x = v[i].first, y = v[i].second;
            if(topo[x] < topo[y])
                printf("%d %d\n", x, y);
//只要满足拓扑序，从拓扑序小的指向拓扑序大的就一定没有环
            else printf("%d %d\n", y, x);
        }
    }
    return 0;
}

```

- 题目已给定每个点编号的求拓扑序按字典序最小输出
- 自己去给所有点分配编号的拓扑序按字典序最小输出
  这两个是不一样的

13.3 已有编号求字典序最小的拓扑序（优先队列）

> 题目大意：给定n个人的m组相对排序关系，求这n个人的拓扑排序（有多种排序情况时，取字典序最小的那种）

将队列换成优先队列直接跑拓扑排序即可

```

void toposort(){
    priority_queue<int, vector<int>, greater<int> >q;
    for(int i = 1;i ≤ n;++i)
        if(!din[i])
        q.push(i);
```

```
7
8       while(q.size()){
9           int x = q.top();
10          q.pop();
11          ans[++cnt] = x;
12          for(int i = head[x];~i;i = nex[i]){
13              int y = ver[i];
14              if(-- din[y] == 0)
15              q.push(y);
16          }
17
18      }
19  }
```

13.4 给所有点分配编号使得拓扑序的字典序最小（反图、优先队列）

相当于有多组约束条件，要求输出字典序最小的解

> **题目大意**：给出N个节点和M条边的有向图，要你为这N个节点编号为1~N
>
> - **所有边的子节点的编号要比其父节点大**
> - 最后的编号要是字典序最小。

也就是需要在满足约束条件的同时尽可能的使序号小排在前面，注意是满足约束条件条件的情况下。

> **解题思路**：本题是要求我们亲自给所有的点分配新的编号， 使得最后的拓扑序字典序最小。也就是原来的结点编号最后是没有用的，我们这里是先对点进行拓扑排序，因为要最后分配出来的字典序最小，所以我们可以将所有的边全部反向建图，这样会按照逆拓扑序**先给子结点分配编号**，这里我们贪心地从大到小地给子结点分配编号，即大数能往后就往后，也就是子结点把大的编号都占了，前面的父节点编号就会尽可能的小。

```
1   int n, m;
2   int din[N];
3   int ver[M], nex[M], head[N], tot;
4   int ans[N], cnt;
5   void add(int x,int y){
6       ver[tot] = y;
7       nex[tot] = head[x];
8       head[x] = tot ++ ;
9       din[y] ++ ;
10  }
11  void toposort(){
12      priority_queue<int>q;
13      for(int i = 1;i ≤ n;++i)
14          if(!din[i])
15              q.push(i);
16      while(q.size()){
17          int x = q.top();
18          q.pop();
19          ans[x] = cnt -- ;
20          for(int i = head[x];~i;i = nex[i]){
21              int y = ver[i];
22              if(-- din[y] == 0)
23                  q.push(y);
24          }
25      }
26  }
27  int main(){
28      memset(head,-1,sizeof head);
29      scanf("%d%d",&n,&m);
30      cnt = n;
31      for(int i = 1;i ≤ m;++i){
32          int x,y;
```

```
        scanf("%d%d",&x,&y);
        add(y,x);
    }
    toposort();
    for(int i = 1;i ≤ n;++i)
        printf("%d ",ans[i]);
    puts("");
    return 0;
}
```

13.5 可达性统计（拓扑排序+bitset优化）

1.1 DAG中从每个点出发能到达的点的数量

1.2 DAG最多删除多少条边仍可保证其连通性

```
bitset<N>A[N], B[N];
int n, m;
int head[N], ver[M], edge[M], nex[M], tot;
int in[N];
int ans[M], top;

void add(int x, int y)
{
    ver[tot] = y;
    nex[tot] = head[x];
    head[x] = tot ++ ;
    in[y] ++ ;
}

void toposort()
{
    queue<int>q;
    for(int i = 1; i ≤ n; ++ i)
        if(in[i] == 0)
            q.push(i);

    while(q.size())
    {
        int x = q.front();
        q.pop();
        ans[++ top] = x;
        for(int i = head[x]; ~i; i = nex[i]){
            int y = ver[i];
            in[y] -- ;
            if(in[y] == 0)
                q.push(y);
        }
    }
    return ;
}

void solve()
{
    for(int k = top ;k ≥ 1; -- k){
        int x = ans[k];
        for(int i = head[x]; ~i; i = nex[i]){
            int y = ver[i];
```

```
            //我们倒序从叶子节点开始
            B[x] |= (A[y] & A[x]);
            //y能到，x也能到，并且x可以到y说明x到y至少有2条路径
            A[x] |= A[y];
        }
    }
}

int main()
{
    memset(head, -1, sizeof head);
    scanf("%d%d", &n, &m);
    for(int i = 1; i ≤ m; ++ i){
        int x, y;
        scanf("%d%d", &x, &y);
        add(x, y);
    }
    for(int i = 1; i ≤ n; ++ i)
        A[i][i] = 1;

    toposort();
    solve();
    int ans = 0;
    for(int x = 1; x ≤ n; ++ x){
        for(int i = head[x]; ~i; i = nex[i]){
            int y = ver[i];
            if(B[x][y] == 1)ans ++ ;//这里应该判断x到y==1
        }
    }
    //for(int i = 1; i ≤ n; ++ i)//可以直接交可达性统计那一道题
    //    printf("%d\n", A[i].count());//输出DAG中从每个点出发能到达的点的数量

    printf("%d\n", ans);
    return 0;
}
```

13.6 (2019年南京C题)拓扑排序递推DP经典

> 题目大意就是一个n*m的矩阵，每一个格子上都有一个数字，我们可以从任意的点出发，每次向上下左右四个方向走，只能到有公共边的格子上去，求一共有多少条完整路径，其中完整路径的定义是整条路上数字从1连续地走到n，其中如果到n了以后他的四个方向都没有n+1也就是不能继续往下走了才算到终点。（还有一个要求就是路径的长度必须大于等于4才算）

```
int n, m;
int head[N], ver[M], nex[M], tot;
int f[N][5];
int in[N];
queue<int>q;
int ans;
int G[maxn][maxn];
void add(int x, int y)
{
    ver[tot] = y;
    nex[tot] = head[x];
    head[x] = tot ++ ;
    in[y] ++ ;
}
void toposort()
{
    for(int i = 1; i ≤ n * m; ++ i){
        if(!in[i])
```

```
            q.push(i), f[i][1] = 1;//所有起点，赋值为1
    }
    while(q.size())
    {
        int x = q.front();
        q.pop();
        bool flag = true;
        for(int i = head[x]; ~i; i = nex[i]){
            int y = ver[i];
            in[y] -- ;
            f[y][2] = (f[y][2] + f[x][1]) % mod;
            f[y][3] = (f[y][3] + f[x][2]) % mod;
            f[y][4] = (f[y][4] + f[x][3] + f[x][4]) % mod;
            if(in[y] == 0)
                q.push(y);
            flag = false;//没有到终点
        }
        if(flag)ans = (ans + f[x][4]) % mod;
    }
}
int main()
{
    memset(head, -1, sizeof head);
    memset(G, 0x3f, sizeof G);//千万注意这个初始化
    scanf("%d%d", &n, &m);
    for(int i = 1; i ≤ n; ++ i)
        for(int j = 1; j ≤ m; ++ j)
            scanf("%d", &G[i][j]);
    for(int i = 1; i ≤ n; ++ i)
    for(int j = 1; j ≤ m; ++ j){
        int x, y = (i - 1) * m + j;
        if(i > 1)
            if(G[i - 1][j] == G[i][j] - 1)
                x = (i - 2) * m + j, add(x, y);
        if(j > 1)
            if(G[i][j - 1] == G[i][j] - 1)
                x = y - 1, add(x, y);
        if(i < n)
            if(G[i + 1][j] == G[i][j] - 1)
                x = y + m, add(x, y);
        if(j < m)
            if(G[i][j + 1] == G[i][j] - 1)
                x = y + 1, add(x, y);
    }
    toposort();
    printf("%d", ans % mod);
    return 0;
}
```

$\oint$ 14. 二分图（包含全套常用定理性质）

二分图：
如果一张无向图 (V, E)存在点集A, B，满足 $|A|, |B| \geq 1$，$A \cap B = \emptyset$，$A \cup B = V$，且对于 $x, y \in A$ 或$x, y \in B$，$(x, y) \notin E$，则称这张无向图为一张二分图，A, B 分别为二分图的左部和右部

一张无向图是二分图，当且仅当图中无奇环。可用染色法判断。

14.1 染色法判断二分图

时间复杂度： $O(n)$。

```cpp
bool find(int x, int col){
    color[x] = col;
    for(int i = head[x];~i;i = nex[i]){
        int y = ver[i], z = edge[i];
        if(color[y]){
            if(color[y] == col)return false;//染色失败不存在二分图.
        }//这个大括号不能不加
        else if(!find(y, -col))return false;
    }
    return true;
}
/*main函数中*/
    for(int i = 1;i ≤ n;++i){
        if(color[i] == 0)
            if(!find(i, 1, x)){
                puts("NO");
                return 0;
            }
    }
    puts("YES");
    return 0;
}
```

应用实例：
二分+染色法判定

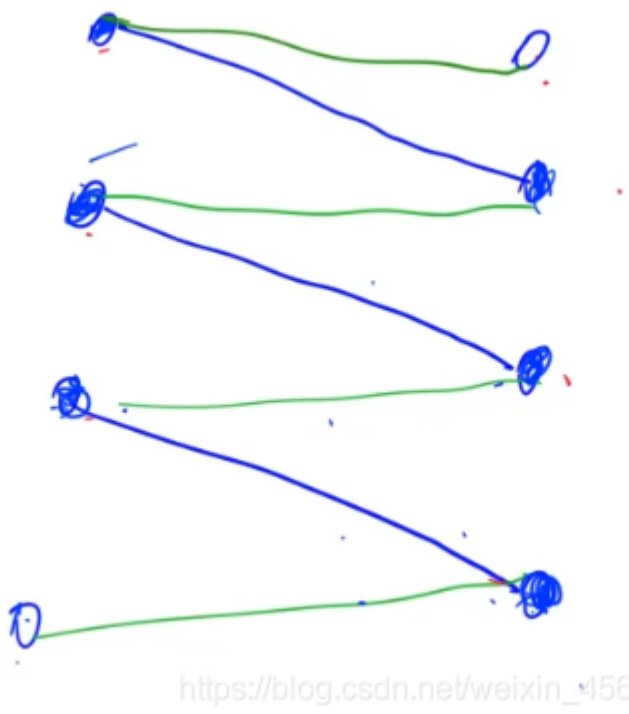
> 有两个监狱，n个罪犯，有m组罪犯之间有怒气，求让所有罪犯都关进监狱的最小的怒气值（最小怒气值为两个监狱的最大怒气值）。注意：有两个罪犯x,y，有且仅有x,y在同一个监狱时才有怒气值。

求满足边权大于mid的边可以构成二分图（题意为将一些点分成两部分互不干涉），求最大的mid。

```cpp
/*P1525 关押罪犯*/
const int N = 50007, M = 500007;
int n, m;
int color[N];
int ver[M], nex[M], edge[M], head[M], tot;
void add(int x,int y, int z){
    ver[tot] = y;
    edge[tot] = z;
    nex[tot] = head[x];
    head[x] = tot ++ ;
}
//染色法模板
bool find(int x, int col, int limit){
    color[x] = col;
    for(int i = head[x];~i;i = nex[i]){
        int y = ver[i], z = edge[i];
        if(z ≤ limit)continue;//所有大于limit的能建成二分图,那么大于它的都不取,答案也就是limit

        if(color[y]){
            if(color[y] == col)return false;//染色失败不存在二分图.
        }
        else if(!find(y, -col, limit))return false;
    }
    return true;
}
```

```
bool check(int x){
    memset(color, 0,sizeof color);

    for(int i = 1;i ≤ n;++i){
        if(color[i] == 0)
            if(!find(i, 1, x))
               return false;
    }
    return true;
}
int main(){
    memset(head,-1,sizeof head);
    scanf("%d%d",&n,&m);
    for(int i = 1; i ≤ m;++i){
        int x,y,z;
        scanf("%d%d%d",&x,&y,&z);
        add(x,y,z);add(y,x,z);
    }
    int l = 0, r = 1e9;
    while(l < r){
        int mid = l + r >> 1;
        if(check(mid))r = mid;
        else l = mid + 1;
    }
    printf("%d\n",l);
    return 0;
}
```

> 增广路：若P是图G中一条连通两个未匹配顶点的路径，并且属于M的边和不属于M的边(即已匹配和待匹配的边)在P上交替出现，则称P为相对于M的一条增广路径
> 如下图，蓝边为匹配边，绿边为未匹配边，蓝绿交替的一条路径即为一条增广路



> 枚举所有未匹配点，找增广路径，直到找不到增广路径。根据 Berge's lemma 当找不到增广路的时候，得到最大匹配。

14.2 增广路的性质

- 长度为奇数。
- 奇数边是非匹配边，偶数边是匹配边。
- 如果把路径上所有边的状态（是否为匹配边）取反，那么得到的新的边集 S’ 仍然是一组匹配，并且匹配的边数增加了 1。

重要结论 ： 二分图的一组匹配 S 是最大匹配，当且仅当图中不存在 S 的增广路。

14.3 一些二分图的概念和定理

- **最大匹配数：最大匹配的匹配边的数目**

- **最小点覆盖数：选取最少的点，使任意一条边至少有一个端点被选择**
- **最大独立集：选取最多的点，使任意所选两点均不相连（独立点的点集）**
- **图的最大团：“任意两点之间都有一条边相连”的子图被称为无向图的团，点数最多的团为图的最大团**
- **最小路径覆盖数：对于一个 DAG（有向无环图），选取最少条路径，使得每个顶点属于且仅属于一条路径。路径长可以为 0（即单个点）。**
- **定理1：最大匹配数 = 最小点覆盖数（ Konig 定理）**
- **定理2：最大独立集 = 定点数 - 最大匹配数**
- **定理3：无向图 G 的最大团，等于补图 G'的最大独立集。**
- **定理4：最小路径覆盖数 = 顶点数 - 最大匹配数**
- **最大环=最大独立集**

> 对于一般无向图，最大团、最大独立集是 NPC 问题，所以最大团问题应该建补图转化为二分图求解。

霍尔定理

设二分图的两部分为X、Y，且 $|X| \leq |Y|$。则定理描述为：二分图存在完美匹配，等价于对于 X 的任意子集 $X\prime$，与它们中任意点相连的Y的结点个数$\geq |X\prime|$。

14.4 增广路定理

> 未覆盖点出发，依次经过非匹配边，匹配边，非匹配边，匹配边...所得的路称为交替路。如果交替路的终点是一个未覆盖点，则称这条交替路为一条增广路，非匹配边比匹配边多一条。
> 如果有一条增广路，呢么把此路上的匹配边和非匹配边互换，得到的匹配边比刚才多一条。反之，若找不到增广路，则当前为最大匹配注意

一个匹配是最大匹配的充分必要条件是不存在增广路，注意在任意图中都成立，不止是二分图。

14.5 二分图最大匹配

二分图的最大匹配可用匈牙利算法或者直接跑最大流解决。

14.6 二分图完美匹配

完美匹配：如果一个图的某个匹配中，所有的顶点都是匹配点，那么它就是一个完美匹配。

14.7 匈牙利算法

算法实际时间复杂度 $O(nm)$。
实际应用的时间复杂度$O(n^2m^2)$
因此数据小跑匈牙利，好写。数据大跑Dinic，稳过。

```
int n, m, k;
int match[N];
int g[N][N];
bool vis[N];
bool find(int x){//匈牙利算法模板
    //for (int i = head[x]; ~i; i = nex[i])根据建图方式自行选择遍历
    for(int i= 1;i ≤ m;++ i){
        if(!vis[i] && g[x][i]){
            vis[i] = true;
            int t = match[i];
            if(t == -1 || find(t)){
                match[i] = x;
                return true;
            }
        }
    }
    return false;
```

```
}

int main(){
    while(~scanf("%d%d%d",&n, &m, &k) && n){
        memset(g,0,sizeof g);
        for(int i = 1;i ≤ k;++ i){
            int t,a,b;
            scanf("%d%d%d",&t, &a,&b);
            g[a][b] = true;
        }
        int res = 0;
        memset(match, -1, sizeof match);
        for(int i = 0;i < n;++ i){
            memset(vis,0,sizeof vis);
            if(find(i))
            res ++ ;
        }
        printf("%d\n",res);
    }
    return 0;
}
```

14.8 二分图匹配模型的两个要素

- 点能分成两个集合，每个集合内部有 0 条边。简称 0 要素。
- 每个点只能与 1 条匹配边相连。简称 1 要素。

14.9 二分图最小点覆盖的一个要素

- 每条边有两个端点，二者至少选择一个。简称二要素。

14.10 DAG的最小路径点覆盖

给定一张有向无环图，用尽量少的不相交的简单路径覆盖所有点（也就是每个点恰好被覆盖一次），这样的路径集合被称为最小路径点覆盖。

求解方法：拆点转化为二分图
把每个点拆成编号为 x 和 x ‘（x+n） 的两个点。
建立一张新的二分图，1~n 是左部点，n+1~2n 是右部点。对原图的每条有向边 (x,y)，在二分图的左部点 x 与右部点 y+n 之间连边。
最终得到的二分图成为原图的拆点二分图。
最小路径覆盖数 = 顶点数 - 最大匹配数

14.11 DAG的最小可重复路径点覆盖

给定一张有向无环图，用尽量少的可相交的简单路径覆盖所有点（也就是每个顶点可以被覆盖多次），这样的路径集合被称为有向无环图的最小可重复路径点覆盖。

解题方法：对原图进行传递闭包，即可将最小可重复路径点覆盖转化为最小路径点覆盖即可。

14.12 最小可重复路径点覆盖模板

```
/*最小路径可重复覆盖*/
/*传递闭包→最小路径覆盖*/

/*BZOJ2718 Vani和Cl2捉迷藏*/
/* from 《算法竞赛进阶指南》*/
/*求出最小路径可重复点覆盖包含的路径条数并输出藏身方案*/
const int N = 507;
```

```cpp
int n,m;
bool g[N][N];
int match[N];
int hide[N];
bool vis[N], succ[N];
int cnt;

bool Find(int x){
    for(int i = 1;i ≤ n;++i){//n * n
        if(g[x][i] && !vis[i]){
            vis[i] = true;
            int t = match[i];
            if(t == -1 || Find(t)){
                match[i] = x;
                return true;
            }
        }
    }
    return false;
}

int main(){
    scanf("%d%d",&n,&m);
    for(int i = 1;i ≤ m;++i){
        int x,y;
        scanf("%d%d",&x,&y);
        g[x][y] = true;
    }
    //Floyd
    for(int i = 1; i ≤ n;++i)//initialization Floyd
        g[i][i] = true;

    for(int k = 1;k ≤ n;++k)
        for(int i = 1;i ≤ n;++i)
            for(int j = 1;j ≤ n;++j)
                g[i][j] |= g[i][k] && g[k][j];

    for(int i = 1;i ≤ n;++i)//remember to reset
        g[i][i] = false;
    //have the biggest pair in the double graph
    //在拆点二分图上求最大匹配
    int res = 0;
    memset(match, -1, sizeof match);
    for(int i = 1;i ≤ n;++i){
        memset(vis,0,sizeof vis);
        if(Find(i))
            res ++ ;
    }

    //the first question
    printf("%d\n", n - res);
    int ans = n - res;

    //the second question
    //构造方案，先把所有路径终点（左部非匹配点）作为藏身点
    for(int i = 1;i ≤ n;++i)
        succ[match[i]] = true;
    for(int i = 1;i ≤ n;++i)
        if(!succ[i])hide[ ++ cnt] = i;
    memset(vis, 0, sizeof vis);
```

```
    bool modify = true;
    while(modify){
        modify = false;
        //求出next(hide)
        for(int i = 1;i ≤ ans ;++i)
            for(int j = 1;j ≤ n;++j)
                if(g[hide[i]][j])vis[j] = true;
        for(int i = 1;i ≤ ans;  ++ i){
            if(vis[hide[i]]){
                modify = true;
                //不断向上移动
                while(vis[hide[i]])
                    hide[i] = match[hide[i]];
            }
        }
    }
    for(int i = 1;i ≤ ans; ++ i)
        printf("%d\n",hide[i]);
    cout << endl;
    return 0;
}
```

14.13 二分图的多重匹配

在二分图最大匹配中，每个点（不管是X方点还是Y方点）最多只能和一条匹配边相关联，然而，我们经常遇到这种问题，即二分图匹配中一个点可以和多条匹配边相关联，但有上限，或者说，Li表示点i最多可以和多少条匹配边相关联。

二分图多重匹配分为二分图多重最大匹配与二分图多重最优匹配两种，分别可以用最大流与最大费用最大流解决。

（1）二分图多重最大匹配：
在原图上建立源点S和汇点T，S向每个X方点连一条容量为该X方点L值的边，每个Y方点向T连一条容量为该Y方点L值的边，原来二分图中各边在新的网络中仍存在，容量为1（若该边可以使用多次则容量大于1），求该网络的最大流，就是该二分图多重最大匹配的值。

（2）二分图多重最优匹配：
在原图上建立源点S和汇点T，S向每个X方点连一条容量为该X方点L值、费用为0的边，每个Y方点向T连一条容量为该Y方点L值、费用为0的边，原来二分图中各边在新的网络中仍存在，容量为1（若该边可以使用多次则容量大于1），费用为该边的权值。求该网络的最大费用最大流，就是该二分图多重最优匹配的值。

> 有N(N<100,000)个人要去M(M<10)个星球，每个人只可以去一些星球，一个星球最多容纳Ki个人。请问是否所有人都可以选择自己的星球...

直接建立二分图模型，上匈牙利算法....

匈牙利算法可以解决多重匹配.... 注意不能把可以匹配多个的点分割然后按照正常的二分匹配来做，那样肯定会挂的....

解决多重匹配就是记录一下多重匹配的点(简称Y方点)已经匹配了Pi个点。如果Pi<Ki那么就直接上了，否则的话继续搜索Yi已经匹配的每一个点并将Yi染色...

因为Yi搜一次就需要染色了，而且Y方点最多是10个，所以每次找增广路的深度最多是10... 这样就很快了...

```
#include<iostream>
#include<cstdio>
using namespace std;
#define re(i,n) for(int i=0;i<n;i++)
const int N = 100005;
const int M = 11;
int yM[M][N];
int tem[M],num[M],chk[M];
int G[N][M];
int n,m;
bool dfs(int u){
    re(v,m) if(G[u][v] && !chk[v]){
```

```
        chk[v] =1;
        if(tem[v] < num[v]){
            yM[v][tem[v]++] = u;
            return 1;
        }
        else re(i,tem[v])
            if(dfs(yM[v][i])){
                yM[v][i] = u;
                return 1;
            }
    }
    return 0;
}
int main(){
    while(~scanf("%d%d",&n,&m)){
        re(i,n) re(j,m) scanf("%d",&G[i][j]);
        re(i,m) scanf("%d",&num[i]);
        re(i,m) tem[i] = 0;
        int flag = 1;
        re(i,n) {
            re(j,m) chk[j] = 0;
            if(!dfs(i)) { flag = 0; break;}
        }
        puts(flag?"YES":"NO");
    }
}

```

∮ 15. 一般图最大匹配（带花树）

15.1 一般图最大匹配

> 题目描述 给出一张 n 个点 m 条边的无向图，求该图的最大匹配。

> 总结一下带花树算法的流程
> 1.每次找一个未匹配的点出来增广
> 2.在增广过程中，如果相邻点是白点，或者是同一朵花中的节点，则直接跳过这个点
> 3.如果相邻点是一个未被匹配过的白点，证明找到了增广路，沿着原有的pre和match路径，对这一次的匹配结果进行更新
> 4.如果相邻点是一个被匹配过的白点，那么把这个点的匹配点丢进队列中，尝试能否让这个点的匹配点找到另外一个点进行匹配，从而可以增广。
> (以上步骤同匈牙利算法)
> 5.如果相邻点是一个被匹配过的黑点，证明此时出现了奇环，我们需要将这个环缩成一个黑点。具体的实现过程是：找到他们的最近花公共祖先，也就是他们的花根，同时，沿着当前这两个点一路到花根，将花上的所有节点全部染成黑点（因为一朵花都是黑点），将原来的白点丢进栈中。同时，修改花上所有点的pre，此时，只剩下花根并不与花内的节点相匹配。

```
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<iostream>
#include<queue>

using namespace std;
const int N = 5007, M = 500007, INF = 0x3f3f3f3f;

int n, m;
int dfn[M];
int low[N], fa[N];
```

```
int tim;
int pre[N];//前驱
int match[N];//匹配点
queue<int>q;
int head[N], ver[M], nex[M], tot;
int vis[N];

int ans;

void add(int x, int y)
{
    ver[tot] = y;
    nex[tot] = head[x];
    head[x] = tot ++ ;
}

int Find(int x)
{
    if(fa[x] == x)return x;
    return fa[x] = Find(fa[x]);
}



//朴素的lca，根据建图的方式暴力跳
inline int lca(int x, int y)
{
    ++ tim;//时间戳 / 缩点的编号
    while(dfn[x] ≠ tim){
        dfn[x] = tim;//每走一步就标记一下， 如果遇见一个点已经被标记过了就说明这个点已经被另一个点走过了，也就说明是lca
        x = Find(pre[match[x]]);
        if(y)swap(x, y);
    }
    return x;//lca
}
//开花（缩环）
//x和y是要开花的两个点（此时x和y都是黑点刚找到奇环的时候）
//w是他们的lca
//整个花缩完之后就是一个黑点

//!这里虽然是两个点但是只是从x走到了x和y的lca：w，y到w路径上的点并没有染色或者丢到队列里去，所以要开两次，一次x一次y。
inline void blossom(int x, int y, int w)
{
    while(Find(x) ≠ w){
        pre[x] = y;
        y = match[x];
        //如果是白点就直接染成黑色，然后丢到队列里面
        vis[y] = 1, q.push(y);
        if(Find(x) == x)fa[x] = w;
        if(Find(y) == y)fa[y] = w;
        x = pre[y];
    }
}

inline bool aug(int s)
{
    if((ans + 1) * 2 > n)return 0;

    for(int i = 1; i ≤ n; ++ i)
```

```
        fa[i] = i, pre[i] = vis[i] = 0;
    while(!q.empty())q.pop();
    q.push(s);
    vis[s] = 1;//从黑点开始
    //队列中都是黑点，所以遇到相邻未染色的点都染成白点
    while(!q.empty()){
        int x = q.front();
        q.pop();
        for(int i = head[x] ;~i; i = nex[i]){
            int y = ver[i];
            //如果相邻点是白点，或者是同一朵花中的节点，则直接跳过这个点
            if(Find(x) == Find(y) || vis[y] == 2) continue;
            if(!vis[y]){//未染色（匹配）点，说明找到了增广路
                vis[y] = 2;//没染色就把它染成白色
                pre[y] = x;
                if(!match[y]){
                    int u = y;
                    while(u){
                        int v = pre[u], z = match[v];
                        match[u] = v;match[v] = u;
                        u = z;
                    }
                    return 1;
                }
                vis[match[y]] = 1, q.push(match[y]);
            }
            else {
                int w = lca(x, y);
                blossom(x, y, w);
                blossom(y, x, w);
            }
        }
    }
    return 0;
}

int main()
{
    memset(head, -1, sizeof head);

    scanf("%d%d", &n, &m);
    for(int i = 1; i ≤ m; ++ i){
        int x, y;
        scanf("%d%d", &x, &y);
        add(x, y), add(y, x);
    }

    for(int i = 1; i ≤ n; ++ i){
        if(!match[i])
            ans += aug(i);
    }

    printf("%d\n", ans);

    for(int i = 1; i ≤ n; ++ i)
        printf("%d%s", match[i], i == n ? "\n" : " ");
    return 0;
}
```

15.2 一般图最大权匹配

> 给定一张有 n 个顶点的无向带权图，有 m 条带权边。
> 求一种匹配的方案，使得最终匹配边的边权之和最大。
> 输出：
> 第一行一个数，最大边权和。
> 接下来一行 n 个整数，描述一组最优方案。第 v 个整数表示点 v 匹配的点的编号。如果 v 号点没有匹配，请输出 0。

```cpp
#include <bits/stdc++.h>
#define N 810
using namespace std;
typedef long long ll;
inline int read()
{
    int x=0,f=1; char ch=getchar();
    while(ch<'0'||ch>'9'){if(ch=='-')f=-1; ch=getchar();}
    while(ch≥'0'&&ch≤'9'){x=x*10+ch-'0'; ch=getchar();}
    return x*f;
}
struct edge{int u,v,w;}mps[N][N];
int n,m,mat[N],pre[N],bl[N],fa[N],tim;ll totw=0;
int sign[N],lab[N],slacku[N],blofm[N][N],tot,mx;
vector <int> leaves[N];
int q[N],hd;
inline int calc_e(const edge &e)
{
    return lab[e.u]+lab[e.v]-mps[e.u][e.v].w*2;
}
inline void updata(int u,int x)
{
    if(!slacku[x]||calc_e(mps[u][x])<calc_e(mps[slacku[x]][x]))
        slacku[x]=u;
}
inline void calc_slack(int x)
{
    slacku[x]=0;
    for(int i=1;i≤n;i++)
        if(mps[i][x].w>0&&fa[i]≠x&&sign[fa[i]]==0)
            updata(i,x);
}
inline void q_push(int x)
{
    if(x≤n) q[++hd]=x;
    else for(int i=0;i<(int)leaves[x].size();i++)
        q_push(leaves[x][i]);
}
inline int get_lca(int x, int y)
{
    if(tim=100000000)
        memset(bl,0,sizeof bl),tim=0;
    for(++tim;x||y;swap(x,y)) if(x)
    {
        if(bl[x]==tim) return x;
        bl[x]=tim; x=fa[mat[x]];
        if(x) x=fa[pre[x]];
    }
    return 0;
}
inline void set_fa(int x,int y)
{
```

```
    fa[x]=y; if(x>n)
    for(int i=0;i<(int)leaves[x].size();i++)
        set_fa(leaves[x][i],y);
}
inline void set_mat(int x,int y)
{
    mat[x]=mps[x][y].v;
    if(x≤n) return ;
    int xr=blofm[x][mps[x][y].u];
    int pr=find(leaves[x].begin(),leaves[x].end(),xr)-leaves[x].begin();
    if(pr%2==1)
        reverse(leaves[x].begin()+1, leaves[x].end()),
        pr=(int)leaves[x].size()-pr;
    for(int i=0;i<pr;i++)
        set_mat(leaves[x][i],leaves[x][i^1]);
    set_mat(xr,y);
    rotate(leaves[x].begin(),leaves[x].begin()+pr,leaves[x].end());
}
inline void blossom_blooms(int x)
{
    for(int i=0;i<(int)leaves[x].size();i++)
    {
        if(leaves[x][i]>n&&!lab[leaves[x][i]])
            blossom_blooms(leaves[x][i]);
        else set_fa(leaves[x][i],leaves[x][i]);
    }
    fa[x]=0;
}
inline void blossom_make(int u,int lca,int v)
{
    int x=n+1; while(x≤tot&&fa[x]) x++;
    if(x>tot) tot++;
    lab[x]=sign[x]=0;
    mat[x]=mat[lca]; leaves[x].clear();
    leaves[x].push_back(lca);
    for(int i=u;i≠lca;i=fa[pre[fa[mat[i]]]])
        leaves[x].push_back(i),leaves[x].push_back(fa[mat[i]]),q_push(fa[mat[i]]);
    reverse(leaves[x].begin()+1, leaves[x].end());
    for(int i=v;i≠lca;i=fa[pre[fa[mat[i]]]])
        leaves[x].push_back(i),leaves[x].push_back(fa[mat[i]]),q_push(fa[mat[i]]);
    set_fa(x,x);
    for(int i=1;i≤tot;i++)
        mps[x][i].w=mps[i][x].w=0, blofm[x][i]=0;
    for(int i=0;i<(int)leaves[x].size();i++)
    {
        int xs=leaves[x][i];
        for(int j=1;j≤tot;j++)
            if(!mps[x][j].w||calc_e(mps[xs][j])<calc_e(mps[x][j]))
                mps[x][j]=mps[xs][j],mps[j][x]=mps[j][xs];
        for(int j=1;j≤tot;j++)
            if(blofm[xs][j]) blofm[x][j]=xs;
    }
    calc_slack(x);
}
inline void link(int x,int y)
{
    while(1)
    {
        int xx=fa[mat[x]];
        set_mat(x,y);
```

```
        if(!xx) return ;
        set_mat(xx,fa[pre[xx]]);
        x=fa[pre[xx]]; y=xx;
    }
}
inline int deal_edge(const edge &e)
{
    int u=fa[e.u],v=fa[e.v];
    if(sign[v]==-1) //unsigned
    {
        pre[v]=e.u; // cause we bfs all vertices tegother,we dont' need to discuss two situation
        sign[v]=1; sign[fa[mat[v]]]=0;
        slacku[v]=slacku[fa[mat[v]]]=0;
        q_push(fa[mat[v]]);
    }
    else if(!sign[v]) //S signed vertex
    {
        int lca=get_lca(u,v);
        if(!lca)
        {
            link(u,v); link(v,u); //connected! new argument.
            for(int i=n+1;i≤tot;i++)
                if(fa[i]==i&&lab[i]==0)
                    blossom_blooms(i); // flower may not be a flower any more so we blossom blooms!
            return 1;
        }
        else blossom_make(u,lca,v); // form a new flower!
    }
    return 0;
}
inline void blossom_bloom_1(int x)
{

    for(int i=0;i<(int)leaves[x].size();i++)
        set_fa(leaves[x][i],leaves[x][i]);
    int xr=blofm[x][mps[x][pre[x]].u];
    int pr=find(leaves[x].begin(), leaves[x].end(),xr)-leaves[x].begin();
    if(pr%2==1)
        reverse(leaves[x].begin()+1, leaves[x].end()),
        pr=(int)leaves[x].size()-pr;
    for(int i=0;i<pr;i+=2)
    {
        int u=leaves[x][i],v=leaves[x][i+1];
        pre[u]=mps[v][u].u;
        sign[u]=1; sign[v]=0;
        slacku[u]=0; calc_slack(v); q_push(v);
    }
    sign[xr]=1; pre[xr]=pre[x];
    for(int i=pr+1;i<(int)leaves[x].size();i++)
    {
        int u=leaves[x][i];
        sign[u]=-1; calc_slack(u);
    }
    fa[x]=0;
}
inline int match()
{
    for(int i=1;i≤tot;i++) slacku[i]=0,sign[i]=-1;
    hd=0; for(int i=1;i≤tot;i++)
        if(fa[i]==i&&!mat[i])
```

```
            slacku[i]=pre[i]=sign[i]=0,q_push(i);
    if(!hd) return 0;
    while(1)
    {
        for(int i=1;i≤hd;i++)
        {
            int lx=q[i]; for(int j=1;j≤n;j++)
                if(mps[lx][j].w>0&&fa[lx]≠fa[j])
                {
                    if(!calc_e(mps[lx][j]))
                    {
                        if(deal_edge(mps[lx][j]))
                            return 1;
                    }
                    else if(sign[fa[j]]≠1) updata(lx,fa[j]);
                }
        }
        int d=0x3fffffff;
        for(int i=1;i≤n;i++) if(!sign[fa[i]])
            d=min(d,lab[i]);
        for(int i=n+1;i≤tot;i++)
            if(fa[i]==i&&sign[i]==1)
                d=min(lab[i]/2,d);
        for(int i=1;i≤tot;i++) if(fa[i]==i&&slacku[i])
        {
            if(sign[i]==-1) d=min(calc_e(mps[slacku[i]][i]),d);
            else if(sign[i]==0) d=min(calc_e(mps[slacku[i]][i])/2,d);
        }
        for(int i=1;i≤n;i++)
            if(sign[fa[i]]==0) lab[i]-=d;
            else if (sign[fa[i]]==1) lab[i]+=d;
        for(int i=n+1;i≤tot;i++)
            if(fa[i]==i)
            {
                if(sign[i]==0) lab[i]+=d*2;
                else if(sign[i]==1) lab[i]-=d*2;
            }
        hd=0;
        for(int i=1;i≤n;i++) if(!lab[i]) return 0; //all vetices matched,single vetices's label = 0
        for(int i=1;i≤tot;i++)
            if(fa[i]==i&&slacku[i]&&fa[slacku[i]]≠i&&calc_e(mps[slacku[i]][i])==0)
                /*new edge*/ if(deal_edge(mps[slacku[i]][i])) return 1;
        for(int i=n+1;i≤tot;i++)
            if(fa[i]==i&&sign[i]==1&&!lab[i])
                blossom_bloom_1(i);
    }
    return 0;
}
inline void solve()
{
    for(int i=1;i≤n;i++) mat[i]=0;
    tot=n; hd=totw=0;
    for(int i=0;i≤n;i++) fa[i]=i,leaves[i].clear();
    for(int i=1;i≤n;i++) for(int j=1;j≤n;j++)
        blofm[i][j]=(i==j? i:0);
    for(int i =1;i≤n;i++) lab[i]=mx; //init label
    while(match());
    for(int i=1;i≤n;i++) if(mat[i]&&mat[i]<i)
        totw+=mps[i][mat[i]].w;
}
```

```
int main()
{
    n=read(); m=read();
    for(int i=1;i≤n;i++)
        for(int j=1;j≤n;j++)
            mps[i][j]=(edge){i,j,0};
    for(int i=1;i≤m;i++)
    {
        int u=read(), v=read(), w=read();
        mps[u][v].w=mps[v][u].w=w; mx=max(mx,w);
    }
    solve(); printf("%lld\n",totw);
    for(int i=1;i≤n;i++)
        printf("%d ",mat[i]);
    puts("");
    return 0;
}
```

$\oint$ 16. KM算法（二分图最大权完美匹配）

> 题目描述
> 给定一张二分图，左右部均有 n 个点，共有 m 条带权边，且保证有完美匹配。
> 求一种完美匹配的方案，使得最终匹配边的边权之和最大。
> 第一行一个整数 ans 表示答案。
> 第二行共 n 个整数 $a_1, a_2, a_3 \cdots a_n$，其中 a_i表示完美匹配下与右部第 i 个点相匹配的左部点的编号。如果存在多种方案，请输出任意一种。

匈牙利算法又称为 KM 算法，可以在 $O(n^3)$ 时间内求出二分图的 最大权完美匹配 。
考虑到二分图中两个集合中的点并不总是相同，为了能应用 KM 算法解决二分图的最大权匹配，需要先作如下处理：将两个集合中点数比较少的补点，使得两边点数相同，再将不存在的边权重设为 0，这种情况下，问题就转换成求最大权完美匹配问题 ，从而能应用 KM 算法求解。
其实就是普通的匈牙利算法求二分图最大匹配的拓展版本

- 顶标：两边点都有的标记（左 a_i右 b_j）满足 $a_i + b_j \geq w_{i,j}$，不唯一。
- 相等边：$a_i + b_j = w_{i,j}$的边 (i, j)。
- 相等子图：相等边构成的子图。
- 交错树：增广路径形成的树。
- KM 算法核心定理 : 对于某组可行顶标，如果其相等子图存在完美匹配，那么，该匹配就是原二分图的最大权完美匹配。

具体流程：
(1) 分配可行顶标，并对每个节点执行 (2),(3),(4)。
(2) 匈牙利算法找增广。
(3) 找不到增广路（相等子图匹配）就调整顶标。
(4) 重复 (2),(3) 直到找到增广路。
$O(n^4)$的DFS版本《算法竞赛进阶指南》P431

$O(n^3)$的BFS迭代版本

把 DFS 换成 BFS。本质上没有什么区别
但是每个左边的点只会进队、搜索一次。
用p 数组记录的是增广交错树。

迭代版本的BFS ， 不需要 queue。

```
/*好难懂啊，还是抄板子吧*/
typedef long long ll;
typedef unsigned long long ull;//卡精度
typedef pair<int,int> PII;

```

```cpp
const int N = 507, M = 5e3 +7, maxn = 1007;
const int mod = 1e9+7;
const ll INF = 1e15+7;

ll w[N][N];//边权
ll la[N], lb[N];//左、右部点的顶标
bool va[N], vb[N];//访问标记，是否在交错树中
int match[N];//右部点匹配的左部点（一个只能匹配一个嘛）
int n;
ll delta, upd[N];
int p[N];
ll c[N];

void bfs(int x)
{
    int a, y = 0, y1 = 0;

    for(int i = 1; i ≤ n; ++ i)
        p[i] = 0, c[i] = INF;

    match[y] = x;

    do{
        a = match[y], delta = INF, vb[y] = true;
        for(int b = 1; b ≤ n; ++ b){
            if(!vb[b]){
                if(c[b] > la[a] + lb[b] - w[a][b])
                    c[b] = la[a] + lb[b] - w[a][b], p[b] = y;
                if(c[b] < delta)//Δ还是取最小的
                    delta = c[b], y1 = b;
            }
        }
        for(int b = 0; b ≤ n; ++ b)
            if(vb[b])
                la[match[b]] -= delta, lb[b] += delta;
            else c[b] -= delta;
        y = y1;
    }while(match[y]);
    while(y)match[y] = match[p[y]], y = p[y];
}

ll KM()
{
    for(int i = 1; i ≤ n; ++ i)
        match[i] = la[i] = lb[i] = 0;
    for(int i = 1; i ≤ n; ++ i){
        for(int j = 1; j ≤ n; ++ j)
            vb[j] = false;
        bfs(i);
    }
    ll res = 0;
    for(int y = 1; y ≤ n; ++ y)
        res += w[match[y]][y];
    return res;
}

int m;
int main()
{
    scanf("%d%d", &n, &m);
```

```
66      for(int i = 1; i ≤ n; ++ i)
67          for(int j = 1; j ≤ n; ++ j)
68          w[i][j] = -INF;
69      for(int i = 1 ;i ≤ m; ++ i){
70          int x, y;
71          ll z;
72          scanf("%d%d%lld", &x, &y, &z);
73          w[x][y] = max(w[x][y], z);//最大匹配嘛
74      }
75      printf("%lld\n", KM());
76      for(int i = 1; i ≤ n; ++ i)
77          printf("%d ", match[i]);
78      puts("");
79      return 0;
80  }
81
```

题目大意：a[i]表示对手的每个队伍战斗力p[i]表示打败对手后获得的分数b[i]表示我方第一种人的战斗力c[i]表示我方第二种人的战斗力，定义我方一组选手的战斗力为b[i]+c[j]，第一种选手与第二种选手某种顺序两两组队后，与对方进行pk，共有 $n!$ 种pk顺序，求最大期望×n

期望就是加权平均，期望*n实际上就是所有方案中的最大权值，也就是二分图最大加权匹配，所以这是一个板子题，但是数据卡$O(n^4)$的dfs版本的KM算法并且卡一切费用流算法，所以我们要用bfs版本的$O(n^3)$的KM板子才能过。

```
1  int main()/*写这个主要是想记录一下最大期望×n是什么意思*//*2019nanjing J spy*/
2  {
3      scanf("%d", &n);
4      for(int i = 1; i ≤ n; ++ i)scanf("%lld", &A[i]);
5      for(int i = 1; i ≤ n; ++ i)scanf("%lld", &P[i]);
6      for(int i = 1; i ≤ n; ++ i)scanf("%lld", &B[i]);
7      for(int i = 1; i ≤ n; ++ i)scanf("%lld", &C[i]);
8      for(int i = 1; i ≤ n; ++ i)
9      for(int j = 1; j ≤ n; ++ j){
10         ll sum = 0;
11         for(int k = 1; k ≤ n; ++ k){
12             if(A[k] < B[i] + C[j])sum += P[k];
13         }
14         w[i][j] = sum;
15     }
16     printf("%lld\n", KM());
17     return 0;
18 }
```

$\oint$ 17. 最小斯坦纳树(求联通k个关键点最小的代价)

> 给定一个包含 n 个结点和 m 条带权边的无向连通图 $G = (V, E)$。再给定包含 k 个结点的点集 S，选出 G 的子图 $G' = (V', E')$，使得：$S \subseteq V'$G'为连通图；E'中所有边的权值和最小。
> 你只需要求出 E'中所有边的权值和。

一个有权的无向图上有k个关键点，求联通k个关键点最小的代价。
答案的子图一定是树。

斯坦纳树是这样一类问题：带边权无向图上有几个（一般约10个）点是【关键点】，要求选择一些边使这些点在同一个联通块内，同时要求所选的边的边权和最小。

时间复杂度：$O(n \times 3^k + m \log m \times 2^k)$

```
1  #include <bits/stdc++.h>
```

```cpp
using namespace std;
const int MAXN=510;
int n,m,k,x,y,z,eg,p[MAXN],hd[MAXN],ver[2*MAXN],vis[MAXN],nx[2*MAXN],edge[2*MAXN],dp[MAXN][4200];
priority_queue < pair<int,int> > q;
void add_edge (int x,int y,int z) {
    ver[++eg]=y;
    nx[eg]=hd[x],edge[eg]=z;
    hd[x]=eg;
    return;
}
void dijkstra (int s) {
    memset(vis,0,sizeof(vis));
    while (!q.empty()) {
        pair <int,int> a=q.top();
        q.pop();
        if (vis[a.second]) {continue;}
        vis[a.second]=1;
        for (int i=hd[a.second];i;i=nx[i]) {
            if (dp[ver[i]][s]>dp[a.second][s]+edge[i]) {
                dp[ver[i]][s]=dp[a.second][s]+edge[i];
                q.push(make_pair(-dp[ver[i]][s],ver[i]));
            }
        }
    }
    return;
}
int main () {
    freopen("st010.in","r",stdin);
    freopen("st010.out","w",stdout);
    memset(dp,0x3f,sizeof(dp));
    scanf("%d%d%d",&n,&m,&k);
    for (int i=1;i≤m;i++) {
        scanf("%d%d%d",&x,&y,&z);
        add_edge(x,y,z),add_edge(y,x,z);
    }
    for (int i=1;i≤k;i++) {
        scanf("%d",&p[i]);
        dp[p[i]][1<<(i-1)]=0;
    }
    for (int s=1;s<(1<<k);s++) {
        for (int i=1;i≤n;i++) {
            for (int subs=s&(s-1);subs;subs=s&(subs-1)) {
                dp[i][s]=min(dp[i][s],dp[i][subs]+dp[i][s^subs]);
            }
            if (dp[i][s]≠0x3f3f3f3f) {q.push(make_pair(-dp[i][s],i));}
        }
        dijkstra(s);
    }
    printf("%d\n",dp[p[1]][(1<<k)-1]);
    return 0;
}
```

$\oint$ 18. 基环树

基环树（~~基于环的树~~），也是环套树，是一种有 n 个点 n 条边的图，简单地讲就是树上在加一条边。它形如一个环，环上每个点都有一棵子树的形式。

基环内向树：每个点出度为1（因此每个环上点的子树，儿子指向父亲）

基环外向树：每个点入度为1（因此每个环上点的子树，父亲指向儿子）

基环树的关键就是找到环，可以先把环当作这个无根树的 “根” ，也就是把环当成一个点（先不管它），这样一颗基环树就变成了一个普通的树，然后我们先按照解决普通树的方法对“根”的所有子树依次处理求解答案，最后在单独对环上所有的点进行操作求解最终答案即可。

18.1 找环

拓扑排序

处理无向图

可以找出环上的所有点。

```
void topsort(){
    int l=0,r=0;
    for (int i=1;i≤n;i++)
      if(in[i]==1) q[++r]=i;
    while(l<r) {
        int now=q[++l];
        for (int i=ls[now];i;i=a[i].next){
            int y=a[i].to;
            if(in[y]>1){
                in[y]--;
                if(in[y]==1) q[++r]=y;
            }
        }
    }
}
```

之后入度 $>=2$ 的点就是环上的点

dfs*

处理有向图，码量小

```
void check_c(int x)
{
    v[x]=true;
    if(v[d[x]]) mark=x;//环上的一点
    else check_c(father[x]);
    return;
}
```

我们稍加修改就可以保存所有的环上的节点了。

```
inline bool dfs(int x, int in_edge)
{
    if(v[x] == 1){//再次访问说明到了环的链接点，标记它是环(v[x] = 2)
        v[x] = 2, loop[++ cnt] = x, v2[x] = 1;
        return true;
    }
    v[x] = 1;
    for(int i = head[x]; ~i; i = nex[i]){
        int y = ver[i];
        //如果当前边不是上一条边(没有往回走)并且当前节点在环上
        if(i != ((in_edge) ^ 1) && dfs(y, i)){//每次存一个点，靠继续dfs遍历整个环
            if(v[x] != 2)//当前节点不是衔接点,把环里的点存起来
                loop[++ cnt] = x, v2[x] = 1, s[cnt] = s[cnt - 1] + edge[i];
            else {//环的链接点
```

```
                s[st - 1] = s[st] - edge[i];//又转回来（破环成链）
                return false;
            }
            return true;
        }
    }
    return false;
}
```

18.2 求基环树森林的直径

题目描述

展开

你准备浏览一个公园，该公园由 N 个岛屿组成，当地管理部门从每个岛屿 i 出发向另外一个岛屿建了一座长度为 L_i 的桥，不过桥是可以双向行走的。同时，每对岛屿之间都有一艘专用的往来两岛之间的渡船。相对于乘船而言，你更喜欢步行。你希望经过的桥的总长度尽可能长，但受到以下的限制：

- 可以自行挑选一个岛开始游览。
- 任何一个岛都不能游览一次以上。
- 无论任何时间，你都可以由当前所在的岛 S 去另一个从未到过的岛 D。从 S 到 D 有如下方法：
  - 步行：仅当两个岛之间有一座桥时才有可能。对于这种情况，桥的长度会累加到你步行的总距离中。
  - 渡船：你可以选择这种方法，仅当没有任何桥和以前使用过的渡船的组合可以由 S 走到 D (当检查是否可到达时，你应该考虑所有的路径，包括经过你曾游览过的那些岛)。

注意，你不必游览所有的岛，也可能无法走完所有的桥。

请你编写一个程序，给定 N 座桥以及它们的长度，按照上述的规则，计算你可以走过的桥的长度之和的最大值。

由于这个公园有n座岛屿和n座桥，就相当于一个由N个节点和N条边构成的无向图。显然，这样的一个无向图就是一个基环树森林。（题目中的渡船是没有用的干扰项，因为渡船使用的条件是两点之间没有路，但是所有的点之间都有一条无向边..）

我们要求的最长的路径实际上就是基环树森林的直径。我们考虑先找到环，然后把环当作当前这颗基环树的“根节点”，我们发现直径有两种情况：

- 在“根节点”的某一棵子树中
- 经过“根节点”，在“根节点”的某两棵子树中

情况一我们直接以环的每一个点为根找它的子树的直径，这里我们直接用树形dp即可，如果用dfs会爆栈。最后取最大值。

情况二我们需要找到环上的两个节点i,j，使得$d_i+d_j+dist_{i,j}$最大，其中$dist_{i,j}$表示节点i,j在环上的距离。如果一个个枚举的话，显然时间复杂度太大，不可行。这里可以断开环将断开后的链复制到两倍的长度，用单调队列进行优化.
因为需要O(1)得到换上任意两点之间的距离，所以我们可以直接维护一个距离的前缀和s数组。
我们要找的是最大的$d_i+d_j+dist_{i,j}$，其中$dist_{i,j}$即为$s_i-s_j(j<i)$。整理后即为：$d_i+s_i+d_j-s_j$。
其中单调队列中

> 对于每个节点i，显然如果一个节点$j(j<i)$满足$d_j-s_j\le d_i-s_i$，那么根据最优性，j 是一定不会被选作最终决策的，因此可以将其从答案队列中删除。

所以本题就是基环树：dfs找环+树形DP求树的直径+破环成链+单调队列优化

```
typedef long long ll;
const int N = 2000007, M = 5000007, INF = 0x3f3f3f3f;

int n, m;
int head[N], ver[M], nex[M], edge[M], tot;
int loop[N], cnt;
int v[N];//表示dfs时走过这个点
int v2[N];//表示这个点所在的基环树已经被走过了，在dfs里标记环中的点，在dp中标记非环点
```

```cpp
ll ans, ans1, ans2, ans3, res, st;
ll dist[N], dp[N];
ll s[N];//前缀和，方便计算两点距离

//ans存储当前子树的直径，ans2存储第一种情况的答案
void add(int x, int y, int z)
{
    ver[tot] = y;
    nex[tot] = head[x];
    edge[tot] = z;
    head[x] = tot ++ ;
}
//dfs找环
inline bool dfs(int x, int in_edge)
{
    if(v[x] == 1){//再次访问说明到了环的链接点，标记它是环(v[x] = 2)
        v[x] = 2, loop[++ cnt] = x, v2[x] = 1;
        return true;
    }
    v[x] = 1;
    for(int i = head[x]; ~i; i = nex[i]){
        int y = ver[i];
        //如果当前边不是上一条边(没有往回走)并且当前节点在环上
        if(i != ((in_edge) ^ 1) && dfs(y, i)){//每次存一个点，靠继续dfs遍历整个环
            if(v[x] != 2)//当前节点不是衔接点,把环里的点存起来
                loop[++ cnt] = x, v2[x] = 1, s[cnt] = s[cnt - 1] + edge[i];
            else {//环的链接点
                s[st - 1] = s[st] - edge[i];//又转回来（破环成链）
                return false;
            }
            return true;
        }
    }
    return false;
}

inline void tree_dp(int x){//树形DP处理第一种情况——不经过环的树的直径
    v2[x] = 1;//标记整棵基环树的非环部分
    for(int i = head[x]; ~i; i = nex[i]){
        int y = ver[i];
        if(v2[y])continue;
        tree_dp(y);
        ans = max(ans, dist[x] + dist[y] + edge[i]);
        dist[x] = max(dist[x], dist[y] + edge[i]);
    }
}

//对于这一颗基环树来说
inline ll brt(int root){
    //这里的cnt是一只累加的，表示的是整个基环树森林的所有环的边数
    //所以我们每次从上一颗基环树的编号+1开始一颗新的基环树
    st = cnt + 1, ans2 = 0, ans3 = 0;
    dfs(root, 0);
    //dfs找到环以后，从st开始走，跑当前基环树的所有的环
    for(int i = st; i ≤ cnt; ++ i){
        ans = 0;
        tree_dp(loop[i]);
        ans2 = max(ans2, ans);//找出第一种情况的最大答案
        //我们破环成链，2*n的链表示整个环，初始化dp数组
        dp[i + cnt - st + 1] = dp[i] = dist[loop[i]];
```

```
        s[i + cnt - st + 1] = s[i + cnt - st] + (s[i] - s[i - 1]);
        //s[i] - s[i - 1]表示的是i和i-1两点之间的距离 == edge[i];
        //因为cnt没有重置，一直用的一个数，所以
        //环上的节点个数为 cnt - st + 1

    }
    deque<int>q;
    for(int i = st; i ≤  2 * cnt - st + 1; ++ i){//对复制后的链进行遍历
        while(q.size() && q.front() ≤ i - cnt + st - 1)
            q.pop_front();//排除超出范围的决策(越界的pop)
        if(q.size())
            ans3 = max(ans3, dp[i] + dp[q.front()] + s[i] - s[q.front()]);
        while(q.size() && dp[q.back()] - s[q.back()] ≤ dp[i] - s[i])
            q.pop_back();
        //排除过时的决策(单调队列维护单增，要是还没有新来的强，那就永远追不上了，就pop吧)
        q.push_back(i);//将当前决策插入队列
    }//处理第二种情况
    return max(ans2, ans3);//取两种情况的最大值
}

int main()
{
    memset(head, -1, sizeof head);
    scanf("%d", &n);
    for(int i = 1; i ≤ n; ++ i){
        int y, z;
        scanf("%d%d", &y, &z);
        add(i, y, z);
        add(y, i, z);
    }
    res = 0;
    for(int i = 1; i ≤ n; ++ i){
    //求整个基环树森林的直径，一次dfs不一定能遍历所有的树，因为不一定都相连
        if(!v2[i])//如果没走过就走
            res += brt(i);//加上这颗树的直径
    }
    printf("%lld\n", res);
    return 0;
}
```

$\oint$ 19. 支配树/灭绝树

> 题目描述 给定一张有向图，求从1号点出发，每个点能支配的点的个数（包括自己）

灭绝树/支配树
用于$O((n+m)logn)$或者$O((n+m)\alpha)$求解一类如下问题：

在一张捕食图上（从捕食者向被捕食者有连边），若某生物的所有食物都灭绝了，则该生物灭绝。 灭绝树便是此图的一种生成树，使得满足灭绝树上某点灭绝，该点子树内所有点都灭绝

对于一棵可以存在环的**有向图**

给定一个起点r 对于一个终点x 如果r到达x的所有路径都会经过一个点y那么我们就称之为**y支配x**我们令支配x的点集为$S=p_1,p_2,p_3,.....,p_k$那么从起点r到达x的任意一条路径排列为$r.....p_1.....p_2.....p_k...x$支配树可以对于每一个点$x(x!=r)$求出$p_k$也就是点集S当中距离x最近的支配点,这个点我们称之为$idom[x]$

如果每一个点连出指向$idom[x]$ 那么就构成了一颗内向树 我们称之为支配树

支配点： 从u到v的所有路径如果都经过一个点p，那么我们称这个点p为v的支配点，如果把p切掉，那么u将无法到达v，显然一个点的支配点可以有很多个

半支配点： 存在从一个点u到v的路径中(不包括u，v)，所有的点pi的dfn都大于v的dfn(下面两个点的比较大小都指的是他们的dfn)(我们在做题前是要扣出一个dfs树的)，当然如果u是v在dfs树上的父节点，那么u也是v的半支配点

```cpp
# include <iostream>
# include <cstdio>
# include <cstring>
# include <queue>
# define sz 20
# define FOR(i,st,ed) for(int i=st;i≤ed;++i)
# define _FOR(tu,u) for(int v,i=tu.hd[u];i;i=tu.e[i].nt)
# define _(tu) v=tu.e[i].to;
using namespace std;
int re(){
    int s=0,f=1;char ch=getchar();
    while(!isdigit(ch)){if(ch=='-')f=-1;ch=getchar();}
    while(isdigit(ch)){s=(s<<1)+(s<<3)+ch-'0';ch=getchar();}
    return s*f;
}//快读就不解释了

const int N=2e5+7;
struct MAP{
    struct edge{
        int to,nt;
    }e[N<<1];int cnt,hd[N];
    void link(int x,int y){e[++cnt]=(edge){y,hd[x]};hd[x]=cnt;}
}xx,a,ra,t,rt,z;//排除xx，依次表示原图，原图的反图，dfs树
//dfs树的反图，和最后的支配树
int n,m;

int dfn[N],dji,id[N],ff[N];//ff表示dfs树上的父节点
void dfs(int u){
    id[dfn[u]=++dji]=u;
    _FOR(a,u){
        _(a);
        if(!dfn[v]) dfs(v),ff[v]=u,t.link(u,v);
    }
}//扣出一个dfs树

int anc[N],semi[N],mn[N];//这个mn[v],表示v点的dfs树上的祖先
//的semi最小的那个祖先，所以semi[mn[x]]=semi[x];
void init(){
    FOR(i,1,n) anc[i]=semi[i]=mn[i]=i;
}
int find(int u){
    if(u≠anc[u]){
        int t=anc[u];anc[u]=find(anc[u]);
        if(dfn[semi[mn[u]]]>dfn[semi[mn[t]]]) mn[u]=mn[t];
    }
    return anc[u];
}//以上是带权并查集(路径压缩)
void len_tarjan(){
    init();
    for(int j=n;j≥2;--j){
        int u=id[j],res=j;
        if(!u) continue;
        _FOR(ra,u){
            _(ra);
            if(!dfn[v]) continue;
```

```
            if(dfn[v]<dfn[u]) res=min(res,dfn[v]);
            else find(v),res=min(res,dfn[semi[mn[v]]]);
        }
        semi[u]=id[res];anc[u]=ff[u];t.link(semi[u],u);
    }
}//我们的tarjan大佬的算法

int ru[N],dep[N],fa[N][25];
int lca(int x,int y){
    if(dep[x]<dep[y]) swap(x,y);
    int del=dep[x]-dep[y];
    FOR(i,0,sz)
        if((1<<i)&del) x=fa[x][i];
    if(x==y) return x;
    for(int i=sz;i>=0;--i)
        if(fa[x][i]!=fa[y][i]) x=fa[x][i],y=fa[y][i];
    return fa[x][0];
}/lca
void work(int u){
    int t=rt.e[rt.hd[u]].to;
    _FOR(rt,u){
        _(rt);t=lca(t,v);
    }
    dep[u]=dep[t]+1;
    fa[u][0]=t;
    z.link(t,u);
    FOR(i,1,sz)
        fa[u][i]=fa[fa[u][i-1]][i-1];
}//建立支配树
void tpsort(){
    FOR(j,1,n)
        _FOR(t,j){
            _(t);
            ru[v]++;rt.link(v,j);
        }
    FOR(i,1,n) if(!ru[i]) t.link(0,i),rt.link(i,0);
    queue <int> q;q.push(0);
    while(q.size()){
        int u=q.front();q.pop();
        _FOR(t,u){
            _(t);
            if((--ru[v])<=0) q.push(v),work(v);
        }
    }
}

int siz[N];
void adfs(int u){
    siz[u]=1;
    _FOR(z,u){
        _(z);adfs(v);siz[u]+=siz[v];
    }
}//最后adfs用来搜答案
int main(){
    n=re();m=re();int x,y;
    FOR(i,1,m){
        x=re();y=re();
        a.link(x,y);ra.link(y,x);
    }
    dfs(1);
```

```
116     len_tarjan();
117     tpsort();
118     adfs(0);
119     FOR(i,1,n) cout<<siz[i]<<' ';
120     return 0;
121 }
```

$\oint$ 20. 树的最大匹配

题意：给定一棵根结点为 的树，一开始全白，alice和bob轮流玩，Alice选择一个点，将其染黑，接下来Bob将这个点的某个祖先点或其子树点染黑，2个人轮流来，双方都采取最优策略，问最终谁获胜
题解：dp表示当前点剩下的最少未匹配点，cnt为$\sum dp[v]$，则有当cnt=0时，dp[u]=1;cnt>0时，dp[u]=cnt-1。当dp[1]为0时，说明该树完美匹配，完美匹配则Bob胜利，否则Alice胜利。

解法

对于一个点，可以到达的点有所有祖先和所有孩子。如果u可以到达v节点，那么v节点显然也可以到达u节点。
此时我们可以考虑将树上博弈转化成图。n个节点，每个点与可到达的节点连边，就会形成一张图。
手玩几个例子，大概可以发现当最大匹配是完美匹配的时候后手必胜，否则先手必胜。下面给出证明。

1.最大匹配是完美匹配时，后手必胜。

因为完美匹配，所以n个点必可以分成n/2组,$(v_1,v_2),(v_3,v_4)\dots(v_{n-1},v_n)$，那么先手每选一个点，后手就选与之同组的一个点。那么先手必将在某一步无路可走。后手必胜。

2.最大匹配不是完美匹配时，先手必胜。

设最大匹配$E=(v_1,v_2),(v_3,v_4)\dots(v_{k-1},v_k)$。那么先手选一个不在匹配内的点，会发生以下两种情况：
(1)后手选择匹配内的点。
此时先手如同1进行操作就可以必胜。因为如果后手在某一轮选择一个匹配外的点，例如在v_4后选择一个匹配外的点，我们就会发现$(v_a,v_1),(v_2,v_3)(v_4,v_b)$是可达的，那么$(v_a,v_1),(v_2,v_3)(v_4,v_b)\dots(v_{k-1},v_k)$就是一个更大的匹配，与假设不符。所以后手无法再次选择匹配外的点，游戏如1进行，先手此时必胜。
(2)后手也选择匹配外的点。
那么很显然，(v_a,v_b)是可达的，可以加入最大匹配，与假设不符。不可能。
所以，最大匹配不是完美匹配时，先手必胜。

于是，题目就转换成了求由这棵树构成的图是否具有完美匹配。树DP即可。


```
1  const int N = 500007;
2
3  int f[N];
4  vector<int>edge[N];
5
6  void dfs(int x, int fa){
7      for(auto y : edge[x]){
8          if(y == fa)continue;
9          dfs(y, x);
10         f[x] += f[y];
11     }
12     if(f[x])f[x] -- ;
13     else f[x] = 1;
14 }
15 int n;
16
17 int main(){
18     scanf("%d", &n);
19     for(int i = 1; i ≤ n - 1; ++ i){
20         int x, y;
21         scanf("%d%d", &x, &y);
22         edge[x].push_back(y);
23         edge[y].push_back(x);
24     }
25     dfs(1, 0);
26     if(!f[1])printf("Bob\n");
27     else printf("Alice\n");
```

```
28
29      return 0;
30  }
31
```

20.1 树上 DP - 求树的最大匹配数

设树的根为1。设 f[i] 为 i 为覆盖点时以 i 为根的子树的最大匹配数，g[i] 为 i 为未盖点时以 i 为根的子树的最大匹配数。则有：

$$g[i] = \sum max\{f[son], g[son]\}$$

$$f[i] = max\{g[i] - max(f[son], g[son]) + g[son] + 1\}$$


```
1  INT f[maxn], g[maxn];
2  void dfs(INT node)
3  {
4      f[node] = g[node] = 0;
5      for(int i = head[node]; i; i = edges[i].next)
6      {
7          INT to = edges[i].to;
8          dfs(to);
9          g[node] += std::max(f[to], g[to]);
10     }
11     for(int i = head[node]; i; i = edges[i].next)
12     {
13         INT to = edges[i].to;
14         f[node] = std::max(f[node], g[node] - std::max(f[to], g[to]) + g[to] + 1);
15     }
16 }
17
```

$\oint$ 21. 最大团

21.1 Bron-Kerbosch 算法爆搜求最大团

如果不是二分图的话就是NP完全问题，只能爆搜，关键是剪枝的技巧不同

> 题意：给出具 n 个点的无向图的邻接矩阵，求这个图的最大团的点数

```
1  using namespace std;
2  int n;
3  int G[N][N];
4  int cnt[N];//cnt[i]为≥i的最大团点数
5  int group[N];//最大团的点
6  int vis[N];//记录点的位置
7  int res;//最大团的数目
8  bool dfs(int pos,int num){//num为已取的点数
9      for(int i=pos+1;i≤n;i++){
10         if(cnt[i]+num≤res)//剪枝，若取i但cnt[i]+已经取了的点数仍<ans
11             return false;
12
13         if(G[pos][i]){//与当前团中元素比较，取Non-N(i)
14             int j;
15             for(j=0;j<num;j++)
16                 if(!G[i][vis[j]])
17                     break;
```

```
            if(j==num){//若为空，则皆与i相邻，则此时将i加入到最大团中
                vis[num]=i;
                if(dfs(i,num+1))
                    return true;
            }
        }


    if(num>res){//每添加一个点最多使最大团数+1,后面的搜索就没有意义了
        res=num;//最大团中点的数目
        for(int i=1;i≤num;i++)//最大团的元素
            group[i]=vis[i-1];
        return true;
    }
    return false;
}
void maxClique(){
    res=-1;
    for(int i=n;i≥1;i--){//枚举所有点
        vis[0]=i;
        dfs(i,1);
        cnt[i]=res;
    }
}
int main(){
    while(scanf("%d",&n)≠EOF&&(n)){
        for(int i=1;i≤n;i++)
            for(int j=1;j≤n;j++)
                scanf("%d",&G[i][j]);
        maxClique();

       printf("%d\n",res);//最大团的个数
    }
    return 0;
}
```

21.2 最大独立集转反图最大团

题意：n 个点 m 条边，点的序号从 1 到 n，边为无向边，现在给图中的点染色，只能染黑色或者白色，且要求相邻结点不能同时为黑色，问图中染黑色的结点的最大个数以及结点位置

思路：

由于要求相邻结点不能同时为黑色，那么黑色就变成了一个独立集，那么实质就是要计算图中的最大独立子集

由于最大独立集=补图最大团，那么直接采用 Bron-Kerbosch 算法建立反图计算最大团即可

```
const int MOD=1E9+7;/*POJ-1419 Graph Coloring*/
const int N=1000+5;
const int dx[]= {-1,1,0,0};
const int dy[]= {0,0,-1,1};
using namespace std;

int n,m;
bool G[N][N];
int cnt[N];//cnt[i]为≥i的最大团点数
int group[N];//最大团的点
int vis[N];//记录点的位置
int res;//最大团的数目
bool dfs(int pos,int num){//num为当前独立集中的点数
    for(int i=pos+1;i≤n;i++){
```

```
        if(cnt[i]+num≤res)//剪枝，若取i但cnt[i]+已经取了的点数仍<ans
            return false;

        if(G[pos][i]){//与当前团中元素比较，取Non-N(i)
            int j;
            for(j=0;j<num;j++)
                if(!G[i][vis[j]])
                    break;
            if(j==num){//若为空，则皆与i相邻，则此时将i加入到最大团中
                vis[num]=i;
                if(dfs(i,num+1))
                    return true;
            }
        }
    }

    if(num>res){//每添加一个点最多使最大团数+1,后面的搜索就没有意义了
        for(int i=0;i<num;i++)//最大团的元素
            group[i]=vis[i];
        res=num;//最大团中点的数目
        return true;
    }
    return false;
}
void maxClique(){
    res=-1;
    for(int i=n;i>0;i--){//枚举所有点
        vis[0]=i;
        dfs(i,1);
        cnt[i]=res;
    }
}
int main(){
    int T;
    scanf("%d",&T);
    while(T--){
        memset(G,0,sizeof(G));

        scanf("%d%d",&n,&m);
        for(int i=0;i<m;i++){
            int x,y;
            scanf("%d%d",&x,&y);
            G[x][y]=1;
            G[y][x]=1;
        }

        //建立反图
        for(int i=1;i≤n;i++){
            for(int j=1;j≤n;j++){
                if(i==j)
                    G[i][j]=0;
                else
                    G[i][j]^=1;
            }
        }
        maxClique();

        if(res<0)
            res=0;
        printf("%d\n",res);//最大团的个数
```

```
        for(int i=0;i<res;i++)//最大团中的顶点
            printf("%d ",group[i]);
        printf("\n");
    }
    return 0;
}
```

$\oint$ 22. 弦图

对于弦图：最大团 = 最小染色

> 子图：点集和边集都是原图的子集的图
> 诱导子图：是子图，不含其它边
> 团：子图，并且是完全图
> 极大团：不是任何一个团的子图
> 最大团：点数最多的团
> 最小染色：用最少的颜色染给每个点，使相邻点不同色
> 最大独立集：不相邻的最大点集
> 最小团覆盖：最少的团覆盖所有点
> 显然的结论：团数≤色数，最大独立集≤最小团覆盖
> 弦：连接环内两个不相邻的点的边
> 弦图：任意大于三的环都至少有一条弦
> 结论1：弦图的诱导子图是弦图
> 单纯点：设N(v)为v相邻的点集，v为单纯点当且仅当v+N(v)的诱导子图为团
> 结论2:任何一个弦图至少一个单纯点，不是完全图的弦图至少两个
> 完美消除序列
> 每个点正好出现一次的序列v1,v2,...,vn，满足vi在vi,vi+1,...,vn的诱导子图中为单纯点
> 结论3：一个无向图为弦图当且仅当它有一个完美消除序列
> 结论4：团数=色数，最大独立集=最小团覆盖

定理：一个无向图是弦图当且仅当它有一个完美消除序列。

用最大势算法$(MaximumCardinalitySearch)$可以在$O(n+m)$内求出一个消除序列的反序。
只要倒过来就可以了。

```
for (int i=n,now;i;--i)
{
    bool fg=0;
    while (!fg)
    {
        for (int j=v[best].size()-1;j>=0;--j)
            if (!vis[v[best][j]]) {fg=1;now=v[best][j];break;}
            else v[best].pop_back();
        if (!fg) --best;
    }
    seq[i]=now;rk[now]=i;vis[now]=1;
    for (int e=head[now];e;e=nxt[e])
        if (!vis[to[e]])
        {
            v[++label[to[e]]].push_back(to[e]);
            best=max(best,label[to[e]]);
        }
}
```

22.1 弦图的判定

朴素算法$O(nm)=O(n^3)$
优化算法：设$\{vi+1...vn\}$中所有与vi相邻的点依次为$v_{j_1}...v_{j_k}$。
只需判断vj1是否与vj2...vjk相邻即可。
时间复杂度: $O(n+m)=O(n^2)$

```
for (int i=1;i≤n;++i)
{
    top=0;
    for (int e=head[seq[i]];e;e=nxt[e])
        if (rk[to[e]]<i) s[++top]=to[e];
    for (int j=2;j≤top;++j)
        if (!g[s[1]][s[j]]) ans=0;
}
```

最小色数问题
等于最小团数，也就是$max_{i=1}^{n}label[i]+1$

最大独立集
等于最小团覆盖数。按照完美消除序列一个个贪心选取即可。

22.2 弦图的最小染色问题

> K国是一个热衷三角形的国度,连人的交往也只喜欢三角原则.他们认为三角关系:即AB相互认识,BC相互认识,CA相互认识,是简洁高效的.为了巩固三角关系,K国禁止四边关系,五边关系等等的存在.
> 所谓N边关系,是指N个人 A1A2...An之间仅存在N对认识关系:(A1A2)(A2A3)...(AnA1),而没有其它认识关系.比如四边关系指ABCD四个人 AB,BC,CD,DA相互认识,而AC,BD不认识.全民比赛时,为了防止做弊，规定任意一对相互认识的人不得在一队，国王相知道，最少可以分多少支队。

最多只有三角关系，很明显就是一个弦图（弦图就是任意大于三的环都至少有一个弦这样就不会构成四边、五边关系）将弦图分成多组的问题可以看做给弦图上的点染色且两个有直接边相连的点不能同色，这样就转化成了弦图的最小染色问题。直接跑优先队列优化的$O(nlogn+m)$的mcs算法即可。

```
int n,m,cnt=-1;
const int MAX=2000015;
int head[MAX];
bool vis[MAX],used[MAX];
int seq[MAX],label[MAX],color[MAX];
struct edge{
    int nxt;
    int to;
}e[MAX];
void add(int u,int v){
    e[++cnt].nxt=head[u];
    e[cnt].to=v;
    head[u]=cnt;
}
typedef pair<int,int>p;
priority_queue<p>q;
void mcs(){
    for(rg int i=1;i≤n;++i) q.push(p(0,i));
    for(rg int i=n;i≥1;--i){
        while(vis[q.top().second]) q.pop();
        int u=q.top().second;
        q.pop();
        seq[i]=u;
        vis[u]=1;
        for(rg int i=head[u];~i;i=e[i].nxt){
            if(!vis[e[i].to]) q.push(p(++label[e[i].to],e[i].to));
        }
    }
}
```

```
int solve(){
    int res=0;
    for(rg int i=n;i≥1;--i){
        memset(used,0,sizeof(used));
        for(rg int j=head[seq[i]];~j;j=e[j].nxt){
            used[color[e[j].to]]=1;
        }
        for(color[seq[i]]=1;used[color[seq[i]]];++color[seq[i]]);
        res=max(res,color[seq[i]]);
    }
    return res;
}
int main(){
    memset(head,-1,sizeof(head));
    n=read(),m=read();
    for(rg int i=1;i≤m;++i){
        int u=read(),v=read();
        add(u,v);
        add(v,u);
    }
    mcs();
    printf("%d\n",solve());
    return 0;
}
```

23. 竞赛图（有向完全图）

竞赛图也叫有向完全图。每对顶点之间都有一条边相连的有向图称为竞赛图

竞赛图的一些简单的性质：

- 竞赛图没有自环，没有二元环；若竞赛图存在环，则一定存在三元环。（如果存在一个环大于三元，那么一定存在另一个三元的小环。）
- 任意竞赛图都有哈密顿路径（经过每个点一次的路径，不要求回到出发点）。
- 图存在哈密顿回路的充要条件是强联通。
- 哈密顿问题中，对于 n 阶竞赛图，当 n 大于等于 2 时一定存在哈密顿通路

设 D 为 n 阶有向简单图，若 D 的基图为 n 阶无向完全图，则 D 为 n 阶竞赛图。

简单来说，竞赛图就是将完全无向图的无向边给定了方向。

23.1 兰道定理

兰道定理（Landau’ s Theorem）是用来判定竞赛图的定理。

将一个竞赛图的每一个点的出度从小到大排序后得到的序列称为竞赛图的比分序列

定理内容

一个长度为n的序列$s=(s_1\leq s_2,\leq \ldots \leq s_n)$，$n\geq 1$，是合法的比分序列当且仅当：

$$\forall 1\leq k\leq n,\sum_{i=1}^{k}s_i\geq \binom{k}{2}$$

且$k=n$时这个式子必须取等号。

定理证明

首先这个定理的必要性是显然的：即任一n阶竞赛图都满足这个条件。

现在我们只需要证明这个定理的充分性。

在这里，我们的证明是一个构造**算法**。思路是从一个一般竞赛图开始，每次改变两条边的方向，构造出一个比分序列是给定序列的竞赛图。

假设有一个一个满足定理条件的序列s。现在我们构造一个及其平凡的n阶竞赛图T_n，这个竞赛图的第i个节点向所有$j<i$的j节点都连有向边，因此其比分序列是$(0,1,\ldots,n-1)$，我们要从这个平凡竞赛图开始构造。

考虑当前构造到了一个竞赛图U，它的比分序列u满足

$$\forall 1\leq k\leq n,\sum_{i=1}^{k}s_i\geq \sum_{i=1}^{k}u_i$$

当$k=n$时显然要取等号。

显然初始时T_n是满足这个条件的。

例题HDU 5873 Football Games

题目大意：现在有比赛，所有队伍两两进行比赛，赢的积2分，输的积0分，如果平局的话就各自都积1分，现在给出每只队伍的得分情况，判断是否合法。

思路：给出的 m 个队伍可以构成一个竞赛图，问得分情况是否合法实质是在问竞赛图是否合法
根据竞赛图的兰道定理，将得分视为比分序列，将所有得分进行排序，然后依次处理：由于每次比赛胜利都会使得总分 +2，那么前 i 只队伍的得分情况必须大于等于 i(i-1)，当判断到最后一只队伍时，有前 n-1 只队伍的得分必须大于 n(n-1)

```
int a[N];
int main(){
    int t;
    while(scanf("%d",&t)≠EOF){
        while(t--){
            int n;
            scanf("%d",&n);
            for(int i=1;i≤n;i++)
                scanf("%d",&a[i]);
            sort(a+1,a+1+n);

            int sum=0;
            bool flag=true;
            for(int i=1;i≤n;i++){
                sum+=a[i];
                if(i≤n-1){
                    if(sum<i*(i-1)){
                        flag=false;
                        break;
                    }
                }
                else{
                    if(sum≠i*(i-1)){
                        flag=false;
                        break;
                    }
                }
            }

            if(flag)
```

```
                printf("T\n");
            else
                printf("F\n");
        }
    }
    return 0;
}
```

23.2 求竞赛图的任意三元环

> 一个tournament是一个没有自环的有向图，同时，每两个点之间有一条边连接。这就是说，对于两个点u，v（u≠v），有一条从u到v的边或一条从v到u的边。
>
> 给你一个tournament，请找出一个长度为3的环。

因为竞赛图（有向完全图）如果存在环的话就一定存在三元环
我们直接暴力dfs即可，如果x到y，y能到x的fa，那么fa，x，y三个点就形成了一个三元环。如果找到三元环就直接退出。

```
const int N = 5007, M = 5000007, INF = 0x3f3f3f3f;
int n, m;
char s[N][N];

bool vis[N];

bool dfs(int x , int fa)
{
    vis[x] = 1;
    for(int i = 1; i ≤ n; ++ i){
        if(s[x][i] - '0'){
            if(s[i][fa] - '0'){
                printf("%d %d %d\n", fa, x, i);
                return true;
            }
            if(!vis[i]){
                if(dfs(i, x))
                    return true;
            }
        }
    }
    return false;
}

int main()
{
    scanf("%d", &n);
    for(int i = 1; i ≤ n; ++ i)
    scanf("%s", s[i] + 1);

    for(int i =1; i ≤ n; ++ i)
        if(!vis[i])
        if(dfs(i, i))return 0;
    puts("-1");
    return 0;
}

```

23.3 求竞赛图的哈密顿回路数量的期望

如果一个竞赛图含有哈密顿回路，则称这张竞赛图为值得记录的。

从所有含有 n 个顶点（顶点互不相同）的，值得记录的竞赛图中等概率随机选取一个。

求选取的竞赛图中哈密顿回路数量的期望值。

由于答案可能过大/丢失精度，只需要输出答案除以 998244353 的余数。

即：设答案为 $\frac{q}{p}$，则你需要输出一个整数 x，满足 $px \equiv q \mod 998244353$ 且 $0 \leqslant x < 998244353$，可以证明恰好存在一个这样的 x。

若不存在这样的竞赛图，输出 `-1` 。

竞赛图存在哈密顿回路的充要条件就是强连通

设$f(n)$ 表示 n 个点形成强连通竞赛图的方案数，一个简单的容斥就是

$$f(n)=2^{\binom{n}{2}}-\sum_{i=1}^{n-1}\binom{n}{i}f(i)2^{\binom{n-i}{2}}$$

考虑分子怎么求。我们先钦定一个环，然后剩余边随便连，可以发现这样实际上是在算所有哈密顿回路出现次数之和。

于是答案就是

$$\frac{(n-1)!2^{\binom{n}{2}-n}}{f(n)}$$

求f的话直接分治 `ntt` 就可以了

题解来源

```
#define rep(i,st,ed) for (int i=st;i≤ed;++i)
typedef long long LL;
const int MOD=998244353;
const int N=800005;

LL f[N],g[N],A[N],B[N],fac[N],inv[N];
LL wn1[N],wn2[N];
int rv[N],n;

void upd(LL &x,LL v) {
    x+=v,(x≥MOD)?(x-=MOD):0;
}

LL ksm(LL x,LL dep) {
    LL res=1;
    for (;dep;dep>>=1,x=x*x%MOD) {
        (dep&1)?(res=res*x%MOD):0;
    }
    return res;
}

void NTT(LL *a,int n,int f) {
    for (int i=0;i<n;++i) if (i<rv[i]) std:: swap(a[i],a[rv[i]]);
    for (int i=1;i<n;i<<=1) {
        LL wn=((f==1)?wn1[i]:wn2[i]);
        for (int j=0;j<n;j+=(i<<1)) {
            LL w=1;
            for (int k=0;k<i;++k,w=w*wn%MOD) {
                LL u=a[j+k],v=a[j+k+i]*w%MOD;
                a[j+k]=u+v,(a[j+k]≥MOD)?(a[j+k]-=MOD):0;
                a[j+k+i]=u-v,(a[j+k+i]<0)?(a[j+k+i]+=MOD):0;
            }
        }
    }
    if (f==-1) {
        LL ny=ksm(n,MOD-2);
        for (int i=0;i<n;++i) a[i]=a[i]*ny%MOD;
    }
}
```

```
40
41 void solve(int l,int r) {
42     if (l==r) {
43         if (l) f[l]=(g[l]+MOD-f[l])*fac[l]%MOD;
44         return ;
45     }
46     int mid=(l+r)>>1;
47     solve(l,mid);
48     int len=1,lg=0; for (;len≤(r-l+1)*2;) len<<=1,lg++;
49     for (int i=0;i<len;++i) {
50         rv[i]=(rv[i>>1]>>1)|((i&1)<<(lg-1));
51         A[i]=0,B[i]=g[i];
52     }
53     rep(i,l,mid) A[i-l]=f[i]*inv[i]%MOD;
54     NTT(A,len,1),NTT(B,len,1);
55     for (int i=0;i<len;++i) A[i]=A[i]*B[i]%MOD;
56     NTT(A,len,-1);
57     rep(i,mid+1,r) upd(f[i],A[i-l]);
58     solve(mid+1,r);
59 }
60
61 int main(void) {
62     fac[0]=fac[1]=inv[0]=inv[1]=1;
63     for (int i=1;i<N;i<<=1) {
64         wn1[i]=ksm(3,(MOD-1)/i/2);
65         wn2[i]=ksm(3,MOD-1-(MOD-1)/i/2);
66     }
67     rep(i,2,1e5) {
68         fac[i]=fac[i-1]*i%MOD;
69         inv[i]=inv[MOD%i]*(MOD-MOD/i)%MOD;
70     }
71     rep(i,2,1e5) inv[i]=inv[i-1]*inv[i]%MOD;
72     scanf("%d",&n);
73     rep(i,1,n) g[i]=ksm(2,(LL)i*(i-1)/2)*inv[i]%MOD;
74     solve(0,n);
75     puts("1\n-1");
76     rep(i,3,n) {
77         LL ans=fac[i-1]*ksm(2,(LL)i*(i-3)/2)%MOD;
78         ans=ans*ksm(f[i],MOD-2)%MOD;
79         printf("%lld\n", ans);
80     }
81     return 0;
82 }
```

24. 哈密顿问题

24.1 基本概念

1. 哈密尔顿通路：经过图中每个结点且仅经过一次的通路。
2. 哈密尔顿回路：经过图中每个结点且仅经过一次的回路。
3. 哈密尔顿图：存在哈密尔顿回路的图。
4. 竞赛图：每对顶点之间都有一条边相连的有向图，n 个顶点的竞赛图称为 n 阶竞赛图。
5. 与欧拉回路的对比：欧拉回路是指不重复地走过所有路径的回路；哈密尔顿回路是指不重复地走过所有点并且最后回到起点的回路。

   1.哈密尔顿通路的判定

   设一无向图有 n 个顶点，u、v 为图中任意不相邻的两点，deg(x) 代表 x 的度数

   若 $deg(u)+deg(v)\geqslant n-1$ 成立，则图中存在哈密尔顿通路

   2.哈密尔顿回路的判定：Dirac 定理

设一无向图有 n 个顶点，u、v 为图中任意不相邻的两点，deg(x) 代表 x 的度数

若 $deg(u)+deg(v)\geqslant n$，则图中存在哈密尔顿回路

推论：对于 $n\geqslant 3$ 的无向图，若其任一一点 u 的度数 $deg(u)\geqslant \frac{n}{2}$，则图中存在哈密尔顿回路

3.竞赛图

对于 $n\geqslant 2$ （点数）的竞赛图，一定存在哈密尔顿通路

24.2 状压DP求最短Hamilton

给定一张 n 个点的带权无向图，点从 0~n-1 标号，求起点 0 到终点 n-1 的最短Hamilton路径。 Hamilton路径的定义是从 0 到 n-1 不重不漏地经过每个点恰好一次。

```
int f[1 << 20][20];
int w[20][20];
int n, m;
int main(){
    scanf("%d", &n);
    for(int i = 0; i < n; ++ i)
    for(int j = 0; j < n; ++ j)
    scanf("%d", &w[i][j]);

    memset(f, 0x3f, sizeof f);
    f[1][0] = 0;

    for(int i = 0; i ≤ (1 << n) - 1; ++ i)
    for(int j = 0; j < n; ++ j){
        if((i >> j) & 1){
            for(int k = 0; k < n; ++ k){
                if(i ^ (1 << j) >> k & 1){
                    f[i][j] = min(f[i][j], f[i ^ (1 << j)][k] + w[k][j]);
                }
            }
        }
    }
    printf("%d\n", f[(1 << n) - 1][n - 1]);
    return 0;
}

```

24.3 dfs 搜索求哈密尔顿回路

以每个点为起点进行搜索，直到形成回路

```
#define N 101
int n,m;
int u,v;
int g[N][N];
int vis[N],appear[N];
int ans[N],num[N];
int length;
void dfs(int last,int i)//last表示上次访问的点
{
    vis[i]=1;//标记为已经访问过
    appear[i]=1;//标记为已在一张图中出现过

    ans[length++]=i;//记录答案
```

```
    for(int j=1;j≤num[i];j++)
    {
        if(g[i][j]==x&&g[i][j]≠last)//回到起点构成哈密顿环
        {
            ans[++length]=g[i][j];//存储答案

            for(int i=1;i≤length-1;i++) //找到了一个环，输出ans
                cout<<ans[i]<<' ';
            cout<<ans[length]<<endl;

            length--;//长度-1
            break;
        }
        if(!vis[g[i][j]])//遍历与i相关联的所有未访问的点。
            dfs(i,g[i][j]);
    }
    length--;
    vis[i]=0;//回溯
}
int main()
{
    memset(vis,0,sizeof(vis));
    memset(appear,0,sizeof(appear));

    cin>>n>>m; //读入点数与边数
    for(int i=1;i≤m;i++)
    {
        cin>>u>>v; //读入两点
        g[u][++num[v]]=v;//记录u-v的边
        g[v][++num[v]]=u;//记录v-u的边
    }

    for(x=1;x≤n;x++) //枚举每一个点，将其作为起点来尝试访问
    {
        if(!appear[x])//如果点x不在之前曾经被访问过的图里
        {
            length=0;//记录答案的长度
            dfs(0,x);
        }
    }
    return 0;
}
```

24.4 Dirac 定理下构造无向图的哈密顿回路

$O(n^2)$空间上由于边数非常多，所以采用邻接矩阵来存储比较适合

```
bool G[N][N];
bool vis[N];
int ans[N];
void Reverse(int arv[N],int s,int t){//将数组anv从下标s到t的部分的顺序反向
    int temp;
    while(s<t){
        swap(arv[s],arv[t]);
        s++;
        t--;
    }
}
void Hamilton(int n){

```

```
    int t;
    int s=1;//初始化取s为1号点
    for(int i=1;i≤n;i++)
        if(G[s][i]){
            t=i;//取任意邻接与s的点为t
            break;
        }

    memset(vis,false,sizeof(vis));
    vis[s]=true;
    vis[t]=true;
    ans[0]=s;
    ans[1]=t;


    int ansi=2;
    while(true){

        //从t向外扩展
        while(true){
            int i;
            for(i=1;i≤n;i++){
                if(G[t][i] && !vis[i]){
                    ans[ansi++]=i;
                    vis[i]=true;
                    t=i;
                    break;
                }
            }
            if(i>n)
                break;
        }

        //将当前得到的序列倒置
        Reverse(ans,0,ansi-1);

        //s和t互换
        swap(s,t);

        while(true){//从t继续扩展,相当于在原来的序列上从s向外扩展
            int i;
            for(i=1;i≤n;i++){
                if(G[t][i] && !vis[i]){
                    ans[ansi++]=i;
                    vis[i]=true;
                    t=i;
                    break;
                }
            }
            if(i>n)
                break;
        }


        //如果s和t不相邻,进行调整
        if(!G[s][t]){
            //取序列中的一点i,使得ans[i]与t相连,并且ans[i+1]与s相连
            int i;
            for(i=1;i<ansi-2;i++)
                if(G[ans[i]][t]&&G[s][ans[i+1]])
```

```
                    break;
            i++;
            t=ans[i];
            Reverse(ans,i,ansi-1);//将从ans[i+1]到 t 部分的ans[]倒置
        }//此时s和t相连


        //如果当前序列包含n个元素,算法结束
        if(ansi==n)
            return;

        //当前序列中元素的个数小于n,寻找点ans[i],使得ans[i]与ans[]外的一个点相连
        int i,j;
        for(j=1;j≤n;j++){
            if(vis[j])
                continue;
            for(i=1;i<ansi-2;i++)
                if(G[ans[i]][j])
                    break;
            if(G[ans[i]][j])
                break;
        }
        s=ans[i-1];
        t=j;//将新找到的点j赋给t
        Reverse(ans,0,i-1);//将ans[]中s到ans[i-1]的部分倒置
        Reverse(ans,i,ansi-1);//将ans[]中ans[i]到t的部分倒置
        ans[ansi++]=j;//将点j加入到ans[]尾部
        vis[j]=true;
    }
}

int main(){
    int n,m;
    while(scanf("%d%d",&n,&m)≠EOF&&(n||m)){

        n*=2;
        for(int i=0;i≤n;i++){
            for(int j=0;j≤n;j++){
                if(i==j){
                    G[i][j]=false;
                    G[j][i]=false;
                }
                else{
                    G[i][j]=true;
                    G[j][i]=true;
                }
            }
        }

        int ansi=0;
        memset(ans, 0, sizeof(ans));
        for(int i=1;i≤m;i++){
            int x,y;
            scanf("%d%d",&x,&y);
            G[y][x]=false;
            G[x][y]=false;
        }

        Hamilton(n);
        for(int i=0;i<n;i++)
```

```
            printf("%d ", ans[i]);
        printf("\n");
    }
    return 0;
}
```

24.5 N 阶竞赛图下构造有向图的哈密顿通路

含有N个顶点的有向图，且每对顶点之间都有一条边的图，一定存在哈密顿通路

```
int ans[105];
int map[105][105];
void Insert(int arv[], int &len, int index, int key){
    if(index>len)
        index=len;
    len++;
    for(int i=len-1; i≥0; i--){
        if(i≠index && i)
            arv[i]=arv[i-1];
        else{
            arv[i]=key;
            return;
        }
    }
}
void Hamilton(int n){
    int ansi = 1;
    ans[ansi++] = 1;
    for(int i=2; i≤n; i++){//第一种情况,直接把当前点添加到序列末尾
        if(map[i][ans[ansi-1]]==1)
            ans[ansi++]=i;
        else{
            int flag=0;
            //当前序列从后往前找到第一个满足条件的点j,使得存在<Vj,Vi>且<Vi,Vj+1>.
            for(int j=ansi-2; j>0; j--){
                if(map[i][ans[j]]==1){//找到后把该点插入到序列的第j+1个点前.
                    flag=1;
                    Insert(ans,ansi,j+1,i);
                    break;
                }
            }
            if(!flag)//否则说明所有点都邻接自点i,则把该点直接插入到序列首端.
                Insert(ans,ansi,1,i);
        }
    }
}
int main(){
    int n,m;
    scanf("%d", &n);
    m=n*(n-1)/2;
    for(int i=0;i<m;i++){
        int u,v;
        scanf("%d%d",&u,&v);
        if(u<v)
            map[v][u]=1;
    }
    Hamilton(n);
    for(int i=1;i≤n;i++)
```

```
49          printf(i==1? "%d":" %d",ans[i]);
50      printf("\n");
51      return 0;
52  }
```

25. 树的计数

25.1 prufer 序列

Prufer数列是无根树的一种数列。在组合数学中，Prufer数列由有一个对于顶点标过号的树转化来的数列，点数为n的树转化来的Prufer数列长度为n-2。

25.2 prufer 序列的性质

性质1： prufer 序列与无根树一一对应。

显然

性质2： 度数为 d_i 的节点会在 prufer 序列中出现 d_i-1 次。

当某个节点度数为 1 时，会直接被删掉，否则每少掉一个相邻的节点，它就会在序列中出现 1 次。

因此共出现 d_i-1 次。

性质3： 一个 n 个节点的完全图的生成树个数为 n^{n-2} **（Cayley 公式）**。

对于一个 n 个点的无根树，它的 prufer 序列长为 $n-2$ ，而每个位置有 n 种可能性，因此可能的 prufer 序列有 n^{n-2} 种。

又由于 prufer 序列与无根树一一对应，因此生成树个数应与 prufer 序列种树相同，即 n^{n-2} 。

性质4： 对于给定度数为 $d_{1\sim n}$ 的一棵无根树共有 $\frac{(n-2)!}{\prod_{i=1}^{n}(d_i-1)!}$ 种情况。

由上面的性质可以知道，度数为 d_i 的节点会在 prufer 序列中出现 d_i-1 次。

则就是要求出 d_i-1 个 $i(1\le i\le n)$的全排列个数。

而上面那个式子就是可重全排列公式。（即全排列个数除以重复元素内部的全排列个数）

25.3 Cayley 公式的推论

1. n 个无序点的有根树个数为 n^{n-1}
2. n 个无序点的无根树个数为 n^{n-2}
3. n 个有序点的有根树个数为 $n^{n-1}\times(n-1)!$
4. n 个有序点的无根树个数为 $n^{n-2}\times(n-1)!$

25.4 【模板】 Prufer 序列

请实现 Prufer 序列和无根树的相互转化。

为方便你实现代码，尽管是无根树，我们在读入时仍将 n 设为其根。

对于一棵无根树，设 $f_{1\ldots n-1}$ 为其父亲序列（f_i 表示 i 在 n 为根时的父亲），设 $p_{1\ldots n-2}$ 为其 Prufer 序列。

另外，对于一个长度为 m 的序列 $a_{1\ldots m}$ ，我们设其权值为 $\operatorname{xor}_{i=1}^{m} i\times a_i$ 。

```
1  const int N = 6e6 + 7, mod = 1e9 + 7;
2
3  int n, m, t;
4  int p[N], f[N], d[N];
5  ll ans = 0;
6
7  void TtoP()
8  {
```

```
    for(int i = 1; i ≤ n - 1; ++ i) {
        scanf("%d", &f[i]);
        d[f[i]] ++ ;
    }
    for(int i = 1, j = 1; i ≤ n - 2; ++ i, ++ j) {
        while(d[j]) ++ j;//从小到大找到第一个度数为 1 (d[j] == 0) 的结点
        p[i] = f[j];//将父结点放入 prufer 序列里
        while(i ≤ n - 2 && -- d[p[i]] == 0 && p[i] < j)
            p[i + 1] = f[p[i]], ++ i;
    }
    ans = 1ll * p[1];
    for(int i = 2; i ≤ n - 2; ++ i) {
        ans ^= 1ll * i * p[i];
    }
}

void PtoT()
{
    for(int i = 1; i ≤ n - 2; ++ i) {
        scanf("%d", &p[i]);
        d[p[i]] ++ ;
    }
    p[n - 1] = n;
    for(int i = 1, j = 1; i ≤ n - 1; ++ i, ++ j) {
        while(d[j]) ++ j;
        f[j] = p[i];
        while(i ≤ n - 1 &&  -- d[p[i]] == 0 && p[i] < j)
            f[p[i]] = p[i + 1], ++ i;
    }
    ans = 1ll * f[1];
    for(int i = 2; i ≤ n - 1; ++ i)
        ans ^= 1ll * i * f[i];
}

int main()
{
    scanf("%d%d", &n, &m);
    if(m == 1) TtoP();
    else PtoT();
    printf("%lld\n", ans);
    return 0;
}
```

六、网络流

$\oint$ 1. 最大流

1.1 *Dinic*

算法流程：

1、根据残量网络计算层次图。
2、在层次图中使用DFS进行增广直到不存在增广路。
3、重复以上步骤直到无法增广。

$Dinic$ 的时间复杂度为 $O(n^2m)$。

在稀疏图上效率和 EK 算法相当，但在稠密图上效率要比 EK 算法高很多。

特别地，**在求解二分图最大匹配问题时**，可以证明 Dinic 算法的时间复杂度是 $O(m\sqrt{n})$。

一般效率较高，可以处理10^4~10^5的数据。

```cpp
/*luogu P3376 【模板】网络最大流*/
typedef long long ll;
typedef pair<int,int> PII;
const ll INF = 1e18;
const int N = 5e2+7;
const int M = 2e5+7;

int head[N],nex[M],ver[M],tot = 1;
ll edge[M];
int n,m,s,t;
ll maxflow;
ll deep[N];//层级数，其实应该是level
int now[M];//当前弧优化
queue<int>q;

inline void read(int &x){
    int f=0;x=0;char c=getchar();
    while(c<'0'||c>'9')f|=c=='-',c=getchar();
    while(c≥'0'&&c≤'9')x=(x<<1)+(x<<3)+(c^48),c=getchar();
    x=f?-x:x;
}

inline void add(int x,int y,int z){//建正边和反向边
    ver[++tot] = y;edge[tot] = z;nex[tot] = head[x];head[x] = tot;
    ver[++tot] = x;edge[tot] = 0;nex[tot] = head[y];head[y] = tot;
}

inline bool bfs(){//在残量网络中构造分层图
    over(i,1,n)deep[i] = INF;
    while(!q.empty())q.pop();
    q.push(s);deep[s] = 0;now[s] = head[s];//一些初始化
    while(!q.empty()){
        int x = q.front();q.pop();
        for(int i = head[x];i;i = nex[i]){
            int y = ver[i];
            if(edge[i] > 0 && deep[y] == INF){//没走过且剩余容量大于0
                q.push(y);
                now[y] = head[y];//先初始化，暂时都一样
                deep[y] = deep[x] + 1;
                if(y == t)return 1;//找到了
            }
        }
    }
    return 0;
}

//flow是整条增广路对最大流的贡献，rest是当前最小剩余容量，用rest去更新flow
ll dfs(int x,ll flow){//在当前分层图上增广
    if(x == t)return flow;
    ll ans = 0,k,i;
    for(i = now[x];i && flow;i = nex[i]){
        now[x] = i;//当前弧优化（避免重复遍历从x出发的不可拓展的边）
        int y = ver[i];
        if(edge[i] > 0 && (deep[y] == deep[x] + 1)){//必须是下一层并且剩余容量大于0
            k = dfs(y,min(flow,edge[i]));//取最小
            if(!k)deep[y] = INF;//剪枝，去掉增广完毕的点
            edge[i] -= k;//回溯时更新
```

```
            edge[i ^ 1] += k;//成对变换
            ans += k;
            flow -= k;
        }
        //if(!flow)return rest;
    }

    return ans;
}

void dinic(){
    while(bfs())
        maxflow += dfs(s,INF);
}

int main()
{
    read(n);read(m);read(s);read(t);
    tot = 1;//成对变换
    over(i,1,m){
        int x,y,z;
        read(x);read(y);read(z);
        add(x,y,z);
    }
    dinic();
    printf("%lld\n",maxflow);
    return 0;
}
```

1.2 $ISAP$

$ISAP$ 是 SAP 算法的优化版本，的理论时间复杂度为$O(n^2m)$，经过 GAP 优化以及弧优化，可以极大地提高 $ISAP$ 的效率，大大超过了$Dinic$算法，而且代码短，只需要30多行，强推!

```
int read() {
    int x=0,f=1;
    char c=getchar();
    for (;!isdigit(c);c=getchar()) if (c=='-') f=-1;
    for (;isdigit(c);c=getchar()) x=x*10+c-'0';
    return x*f;
}

typedef long long ll;
const int N = 5007, M = 5e3 + 7, INF = 2e9 + 7;

int head[N], ver[M], nex[M], edge[M], cur[N], tot;
int n, m;
int gap[N], deep[N];
int q[M];
int cnt;
void add(int x, int y, int z){
    ver[++ tot] = y;
    edge[tot] = z;
    nex[tot] = head[x];
    head[x] = tot;
    ver[++ tot] = x;
    edge[tot] = 0;
    nex[tot] = head[y];
    head[y] = tot;
}
```

```cpp

void init(int s, int t){//!只需要O(n+m)倒序从t到s,bfs求一次增广路即可
    memset(gap, 0, sizeof gap);
    memset(deep, 0, sizeof deep);

    ++gap[deep[t] = 1];//! ! !

    for(int i = 1;i ≤ n; ++ i)
        cur[i] = head[i];
    //memcpy(cur, head, sizeof head);
    int hh = 0, tt = -1;
    q[hh = tt = 1] = t;

    while(hh ≤ tt){
        int x = q[hh ++ ];//if(hh == M)hh = 0;
        for(int i = head[x]; ~i; i = nex[i]){//这里y要写两次,因为i变了
            int y = ver[i];
            if(!deep[y])
                ++gap[deep[y] = deep[x] + 1], q[++ tt] = y;//if(tt == M)tt = 0;
        }
    }
}

int dfs(int x, int s, int t, int mi){
    if(x == t)return mi;
    int flow = 0;
    for(int &i = cur[x]; ~i; i = nex[i]){//&i = cur[x] -- 弧优化
        int y = ver[i];
        if(deep[x] == deep[y] + 1){//因为倒着走的嘛
            int k = dfs(y, s, t, min(mi, edge[i]));
            flow += k, mi -= k, edge[i] -= k, edge[i ^ 1] += k;
            if(!mi)return flow;
        }
    }
    if(!(--gap[deep[x]]))deep[s] = n + 1;//如果deep[x]这一层走完了就标记一下， 可以结束了
    ++gap[++deep[x]], cur[x] = head[x];
    return flow;
}
// augment v. 增加
int ISAP(int s, int t){
    init(s, t);
    //cout << "ok" << cnt ++ << endl;
    int res = dfs(s, s, t, INF);//必须要先跑一次，因为已经跑过一次增广路了。
    while(deep[s] ≤ n)
        res += dfs(s, s, t, INF);
    return res;
}

int main(){
    while(~scanf("%d%d", &m, &n)){
        tot = 0, memset(head, -1, sizeof head);
        for(int i= 1;i ≤ m; ++ i){
            int x = read(), y = read(), z = read();
            add(x, y, z);
        }
        int ans = ISAP(1, n);
        printf("%d\n", ans);
    }
    return 0;
}
```

1.3 *Push – Relabel* 预流推进算法

该方法在求解过程中忽略流守恒性，并每次对一个结点更新信息，以求解最大流。

1.4 通用的预流推进算法

1.4.1 HLPP 算法

最高标号预流推进算法（High Level Preflow Push）

时间复杂度：$O(n^2\sqrt{m})$

```cpp
#include<cstdio>
#include<cstring>
#include<algorithm>
#include<queue>
using std::min;
using std::vector;
using std::queue;
using std::priority_queue;
const int N=2e4+5,M=2e5+5,inf=0x3f3f3f3f;
int n,s,t,tot;
int v[M<<1],w[M<<1],first[N],next[M<<1];
int h[N],e[N],gap[N<<1],inq[N];//节点高度是可以到达2n-1的
struct cmp
{
    inline bool operator()(int a,int b) const
    {
        return h[a]<h[b];//因为在优先队列中的节点高度不会改变，所以可以直接比较
    }
};
queue<int> Q;
priority_queue<int,vector<int>,cmp> pQ;
inline void add_edge(int from,int to,int flow)
{
    tot+=2;
    v[tot+1]=from;v[tot]=to;w[tot]=flow;w[tot+1]=0;
    next[tot]=first[from];first[from]=tot;
    next[tot+1]=first[to];first[to]=tot+1;
    return;
}
inline bool bfs()
{
    int now;
    register int go;
    memset(h+1,0x3f,sizeof(int)*n);
    h[t]=0;Q.push(t);
    while(!Q.empty())
    {
        now=Q.front();Q.pop();
        for(go=first[now];go;go=next[go])
            if(w[go^1]&&h[v[go]]>h[now]+1)
                h[v[go]]=h[now]+1,Q.push(v[go]);
    }
    return h[s]≠inf;
}
inline void push(int now)//推送
{
```

```
    int d;
    register int go;
    for(go=first[now];go;go=next[go])
        if(w[go]&&h[v[go]]+1==h[now])
        {
            d=min(e[now],w[go]);
            w[go]-=d;w[go^1]+=d;e[now]-=d;e[v[go]]+=d;
            if(v[go]≠s&&v[go]≠t&&!inq[v[go]])
                pQ.push(v[go]),inq[v[go]]=1;
            if(!e[now])//已经推送完毕可以直接退出
                break;
        }
    return;
}
inline void relabel(int now)//重贴标签
{
    register int go;
    h[now]=inf;
    for(go=first[now];go;go=next[go])
        if(w[go]&&h[v[go]]+1<h[now])
            h[now]=h[v[go]]+1;
    return;
}
inline int hlpp()
{
    int now,d;
    register int i,go;
    if(!bfs())//s和t不连通
        return 0;
    h[s]=n;
    memset(gap,0,sizeof(int)*(n<<1));
    for(i=1;i≤n;i++)
        if(h[i]<inf)
            ++gap[h[i]];
    for(go=first[s];go;go=next[go])
        if(d=w[go])
        {
            w[go]-=d;w[go^1]+=d;e[s]-=d;e[v[go]]+=d;
            if(v[go]≠s&&v[go]≠t&&!inq[v[go]])
                pQ.push(v[go]),inq[v[go]]=1;
        }
    while(!pQ.empty())
    {
        inq[now=pQ.top()]=0;pQ.pop();push(now);
        if(e[now])
        {
            if(!--gap[h[now]])//gap优化，因为当前节点是最高的所以修改的节点一定不在优先队列中，不必担心修改对优先队列会造成影响
                for(i=1;i≤n;i++)
                    if(i≠s&&i≠t&&h[i]>h[now]&&h[i]<n+1)
                        h[i]=n+1;
            relabel(now);++gap[h[now]];
            pQ.push(now);inq[now]=1;
        }
    }
    return e[t];
}
int m;
signed main()
{
```

```
    int u,v,w;
    scanf("%d%d%d%d",&n,&m,&s,&t);
    while(m--)
    {
        scanf("%d%d%d",&u,&v,&w);
        add_edge(u,v,w);
    }
    printf("%d\n",hlpp());
    return 0;
}
```

$\oint$ 2. 最小割

最小割，我们建立一个超级源点和超级汇点，做一下最小割，即可得到通过割边使得整张图变成两个完全不相连的集合的最小花费。

同样的，我们也可以指定两个点为源点和汇点，求得通过割边，使得两个点不再连通的最小花费

最大流最小割定理

最小割 = 最大流

2.1 集合划分模型

> 有 n 个物品和两个集合 S,T。将一个物品放入 S 集合会花费 ai，放入 T 集合会花费 bi。还有若干个形如 u,v,w 限制条件，表示如果 u 和 v 同时不在一个集合会花费 w。每个物品必须且只能属于一个集合，求最小的代价。

我们对于每个集合设置源点 S 和汇点 T，第 i 个点由 S 连一条容量为 bi的边、向 T 连一条容量为 ai的边。对于限制条件 u,v,w，我们在 u,v 之间连容量为 w 的双向边。注意到当 S 和 T 不相连时，S 能到达 i 代表物品 i 放入 S，i 能到达 T 代表物品 i 放入 T。当割开 S→i 的边，意味着 i 放入 T；当割开 i→T 的边，意味着 i 放入 S；当割开 u,v 之间的边，意味着 u,v 不放入同一个集合。因此最小割就是最小花费。

```
//SHOI 2007 善意的投票
/*n个人有两种不同的意见并且有许多朋友，需要让朋友间尽可能的统一意见（少发生冲突），如果一个人违反自己的本意也算冲突，求最少的冲突*/
typedef long long ll;
typedef pair<int,int> PII;
const ll INF = 1e18;
const int N = 5e2+7;
const int M = 2e5+7;

int head[N],nex[M],ver[M],tot;
ll edge[M];
int n,m,s,t;
ll maxflow;
ll deep[N];//层级数，其实应该是level
int now[M];//当前弧优化
queue<int>q;

inline void read(int &x){
    int f=0;x=0;char c=getchar();
    while(c<'0'||c>'9')f|=c=='-',c=getchar();
    while(c≥'0'&&c≤'9')x=(x<<1)+(x<<3)+(c^48),c=getchar();
    x=f?-x:x;
}

inline void add(int x,int y,int z){//建正边和反向边
    ver[tot] = y;
    edge[tot] = z;
    nex[tot] = head[x];
```

```
    head[x] = tot ++ ;
}

inline bool bfs(){//在残量网络中构造分层图
    for(int i = 0; i ≤ n + 1; ++ i)deep[i] = INF;
    while(!q.empty())q.pop();
    q.push(s);deep[s] = 0;now[s] = head[s];//一些初始化
    while(!q.empty()){
        int x = q.front();q.pop();
        for(int i = head[x];~i;i = nex[i]){
            int y = ver[i];
            if(edge[i] > 0 && deep[y] == INF){//没走过且剩余容量大于0
                q.push(y);
                now[y] = head[y];//先初始化，暂时都一样
                deep[y] = deep[x] + 1;
                if(y == t)return 1;//找到了
            }
        }
    }
    return 0;
}

//flow是整条增广路对最大流的贡献，rest是当前最小剩余容量，用rest去更新flow
ll dfs(int x, ll flow){//在当前分层图上增广
    if(x == t)return flow;
    ll ans = 0,k,i;
    for(i = now[x];~i && flow;i = nex[i]){
        now[x] = i;//当前弧优化（避免重复遍历从x出发的不可拓展的边）
        int y = ver[i];
        if(edge[i] > 0 && (deep[y] == deep[x] + 1)){//必须是下一层并且剩余容量大于0
            k = dfs(y,min(flow,edge[i]));//取最小
            if(!k)deep[y] = INF;//剪枝，去掉增广完毕的点
            edge[i] -= k;//回溯时更新
            edge[i ^ 1] += k;//成对变换
            ans += k;
            flow -= k;
        }
        //if(!flow)return rest;
    }

    return ans;
}

void dinic(){
    while(bfs())
        maxflow += dfs(s,INF);
}

int main(){
    scanf("%d%d", &n, &m);
    s = 0, t = n + 1;

    memset(head, -1, sizeof head);
    for(int i = 1;i ≤ n; ++ i){
        int x;
        scanf("%d", &x);
        if(x)add(s, i, 1),add(i, s, 0);

        else add(i, t, 1), add(t, i, 0);
    }
```

```
    while(m --){
        int x, y;
        scanf("%d%d", &x, &y);
        add(x, y, 1);
        add(y, x, 1);
    }
    dinic();
    printf("%lld\n", maxflow);
    return 0;
}

```

2.2 点边转化

UVA1660 电视网络 Cable TV Network

> 给定一张 n 个点的无向图。求最少删除多少个点，使得图不连通。
>
> 点边转化，把原来无向图中的每个点 ，拆成入点 和出点 ，则在无向图中删去一个点，等价于在网络中断开 。点边转化，把原来无向图中的每个点 x，拆成入点 x 和出点 x′，则在无向图中删去一个点，等价于在网络中断开 (x,x′)。对任意 x≠S 且 x≠T 的点 x 连有向边 (x,x′)，容量为 1。对原无向图的每条边 (x,y)，连有向边 (x′,y) 和 (y′,x)，容量为 +∞，即防止割断。求最小割即可。

当然，在点边转化中也可以将一条边截成两半，中间插入一个点，把边的各种信息反映在这个点上。

```
const int N = 100, M = 5e4+7, INF = 0x3f3f3f3f;
int s1,t1,n,m;
int head[N<<1],ver[M],nex[M],edge[M],tot;
int a[N * N],b[N * N],deep[N<<1];

inline void add(int x,int y,int z){//正边反边
    ver[++tot] = y;edge[tot] = z;
    nex[tot] = head[x];head[x] = tot;
    ver[++tot] = x;edge[tot] = 0;
    nex[tot] = head[y];head[y] = tot;
}

inline bool bfs(){
    memset(deep,0,sizeof deep);
    queue<int>q;
    q.push(s1);
    deep[s1] = 1;//分层
    while(q.size()){
        int x = q.front();
        q.pop();
        for(int i = head[x];i;i = nex[i]){
            int y = ver[i],z = edge[i];//剩余容量>0才属于残量网络
            if(z > 0 && !deep[y]){//不只是更新deep数组，是在残量网络上更新deep数组
                q.push(y);
                deep[y] = deep[x] + 1;
                if(y == t1)return true;
            }
        }
    }
    return false;
}

inline int dinic(int x,int flow){
    if(x == t1)return flow;
    int res = flow;
    for(int i = head[x];i && res;i = nex[i]){
        int y = ver[i],z = edge[i];
```

```
        if(z > 0 && (deep[y] == deep[x] + 1)){
            int k = dinic(y,min(res,z));
            if(!k)deep[y] = 0;
            edge[i] -= k;
            edge[i ^ 1] += k;
            res -= k;
        }
    }
    return flow - res;
}

int main(){
    while(cin>>n>>m){
        for(int i = 0;i < m;++i){
            char str[20];
            scanf("%s",str);
            a[i] = b[i] = 0;
            int j;
            for(j = 1;str[j] ≠ ',';j++)
                a[i] = a[i] * 10 + (str[j] - '0');
            for(j++;str[j] ≠ ')';j++)
                b[i] = b[i] * 10 + (str[j] - '0');
        }
        int ans = INF;
        for (s1 = 0; s1 < n; s1++)
        for (t1 = 0; t1 < n; t1++)
        if(s1 ≠ t1){
            memset(head,0,sizeof head);
            tot = 1;
            int maxflow = 0;
            for(int i = 0;i < n;++i){
                if(i == s1 || i == t1)//i是入点,i+n是出点
                    add(i,i + n,INF);//防止被割断
                else add(i,i + n,1);
            }
            for(int i = 0;i < m;++i){
                add(a[i] + n,b[i],INF);//不能割
                add(b[i] + n,a[i],INF);
            }
            while(bfs()){
                int num;
                while((num = dinic(s1,INF)))
                    maxflow += num;
            }
            ans = min(ans,maxflow);
        }
        if(n ≤ 1 || ans == INF)ans = n;
        cout<<ans<<endl;
    }
    return 0;
}

```

2.3 最小割的可行边与必须边

首先求最大流，那么最小割的可行边与必须边都必须是满流。

- 可行边 (x,y)：在残量网络中不存在 x 到 y 的路径。

- 必须边 (x,y)：在残量网络中 S 能到 x 且 y能到 T。

最小割的可行边：被某一种最小割方案包含的边。

> 充要条件：
>
> 1. 满流。
> 2. 在残余网络中找不到入点到出点的路径。

求法：

> 在残余网络中tarjan求SCC，入出两点在同一SCC中说明残余网络中存在入点到出点的路径。
>
> 由于该边满流，它的反向边一定存在于残余网络中，反向边与其他入点到出点的路径会构成SCC。

最小割的必须边（不考虑容量为0的边）：一定在最小割中的边。

> 充要条件：
>
> 1. 满流。
> 2. 残余网络中源点能到入点，出点能到汇点。
>
> 必须边的另一种理解是扩大容量后能增大最大流的边。

求法：

> 在残余网络上从源点开始dfs，从汇点开始反向dfs，标记到达的点，然后枚举满流边判断即可。
>
> 若求过可行边，已经有了SCC，直接判断入点是否与源点在同一SCC中且出点是否与汇点在同一SCC中即可。
>
> 考虑为什么两种算法是等价的，由于源点到入点一定有流，所以一定存在入点到源点的有流路径，等价得证。

2.4 二分图的可行边和必须边

求法

- **如果存在一组完备匹配：**
  求出一组完备匹配。
  建一张新图：
  对于图上的边，若为匹配边出点向入点连边，否者入点向出点连边。
  在新图上tarjan求SCC。

可行边：此边为匹配边或入出点在同一SCC中。

必须边：此边为匹配边且入出点不在同一SCC中。

- **如果不一定存在完备匹配：**
  求出一组最大匹配。
  建一张新图：
  对于图上的边，若为匹配边出点向入点连边，否者入点向出点连边。
  加入两个点S,T。
  对于匹配的左部点u，u到S连边；右部点v，T到v连边。
  对于未匹配点的左部点u，S到u连边；右部点v，v到T连边。
  在新图上tarjan求SCC。

可行边： 此边为匹配边或入出点在同一SCC中。
必须边： 此边为匹配边且入出点不在同一SCC中。

二分图与最小割的做法本质是一样的，上述做法其实是利用二分图的性质较为简便地建出了残余网络而已。

【模板】P4126 [AHOI2009]最小割

2.5 平面图最小割

边与边只在顶点相交的图被称为平面图。

对于一个平面图，都有其对应的对偶图：

- 平面图被划分出的每一个区域当作对偶图的一个点；
- 平面图中的每一条边两边的区域对应的点用边相连，特别地，若两边为同区域则加一条回边（自环）。

这样构成的图即为原平面图的对偶图。

平面图最小割等于对偶图最短路。

【模板】P4001 [ICPC-Beijing 2006]狼抓兔子

2.6 最小割的一些小技巧

2.6.1 记录划分方案

我们可以通过从源点 S 开始 DFS，每次走残量大于 0 的边，找到所有 S 点集内的点。

```
void dfs(int u) {
  vis[u] = 1;
  for (int i = lnk[u]; i; i = nxt[i]) {
    int v = ter[i];
    if (!vis[v] && val[i]) dfs(v);
  }
}
```

T点集合同理。

2.6.2 求割边数量

只需要将每条边的容量变为 1 ，然后重新跑 $Dinic$ 即可。

2.6.3 求最小边的最小割

设总边数为E，跑最大流之前所有的边权都乘E+1然后再+1

得到的结果应该是mincut∗(E+1)+割边数量（这个比较显然吧）

由于割边数越小，跑出来结果越小，所以就自动选了割边数量小的边（但相同不能保证字典序）

结果就是最小边数

E+1也可以替换成大于边数的数。

2.6.4 输出任意一种最小割的方案

跑过一次最大流之后，在残量网络上，s和t之间不连通了
进行一次dfs/bfs，求出从s出发能到达的点集S，和不能到达的点集T
所有从S跨越到T的满流边（残留网络为0）构成了一组最小割

2.6.5 判断一条边是否满流

运行一次最大流算法，得到一个残量网络
取残量网络上的一条满流边(u, v)，判断这条边是否一定满流
对残量网络运行Tarjan算法，求出所有SCC
当u和v不属于同一个SCC的时候，这条边一定满流
否则，我们可以在SCC中找到一个包含这条边的反向边的环，沿着环增广一次，仍然不破坏流量平衡，但是这条边已经不满流了

2.6.6 判断某一条边是否可能为最小割中的一条

所有一定满流的边都可能为最小割

2.6.7 判断某一条边是否为最小割中的一条

我们考虑一条边是否一定在最小割中。明显，我们跑完一遍网络流之后残量网络里满流的边都是可能在最小割里的，然后又到了跑Tarjan的时候了，如果这条边在最小割里，那么跑完Tarjan之后它所连接的两个端点一定是在不同的强连通分量里。

2.6.8 判断某条边是否一定出现在最小割中

首先还是对残量网络求SCC
考虑一条满流边(u, v)，判断她是否一定出现在最小割中
当u和s属于同一个SCC，并且v和t属于同一个SCC的时候，这条边一定出现在最小割中

$\oint$ 3.费用流

3.1 最小费用最大流

3.1.1类dinic模板

时间复杂度为$O(nmf)$， f 为最大流量， 效率较高，一般不会被卡

```cpp
#include<iostream>
#include<cstdio>
#include<algorithm>
#include<cstring>
#include<queue>

using namespace std;
typedef long long ll;

namespace dinic{//MCMF

    const int N = 1e5 + 7, M = 2e6 + 7;
    const ll INF = 0x3f3f3f3f3f3f;//!因为是long long 所以是五个3f
    int n, S, T;
    int head[N], ver[M], nex[M], tot, cur[N];
    ll dist[N], edge[M], cost[M], maxflow, mincost;
    bool vis[N];

    inline void add(int x, int y, ll z, ll c, bool o = 1){
        ver[tot] = y;
        edge[tot] = z;
        cost[tot] = c;
        nex[tot] = head[x];
        head[x] = tot ++ ;
        if(o)add(y, x, 0, -c, 0);
    }

    inline bool spfa(){//对费用 cost 求最短路
        //memset(dist, 0x3f, sizeof dist);//!0x3f3f3f3f3f → 3f x 5
        for(int i = 1;i ≤ n; ++ i)dist[i] = INF;
        memset(vis, 0, sizeof vis);
        queue<int>q;
        q.push(S);
        dist[S] = 0;
        vis[S] = 1;
        while(q.size()){
            int x = q.front();
            q.pop();
```

```
            vis[x] = 0;
            for(int i = head[x]; ~i; i = nex[i]){
                int y = ver[i];
                ll z = edge[i], c = cost[i];
                if(dist[y] ≤ dist[x] + c || !z)continue;
                dist[y] = dist[x] + c;
                if(!vis[y])
                    q.push(y), vis[y] = 1;
            }
        }
        return dist[T] ≠ INF;
    }


    ll dfs(int x, ll flow = INF){
        if(x == T)return flow;
        ll ans = 0, k, i;
        vis[x] = 1;
        for(i = cur[x]; ~i && flow; i = nex[i]){
            cur[x] = i;
            int y = ver[i];
            ll z = edge[i], c = cost[i];
            if(!z || (dist[y] ≠ dist[x] + c) || vis[y])continue;
            k = dfs(y, min(flow, z));
            if(!k)dist[y] = INF;
            edge[i] -= k;
            edge[i ^ 1] += k;
            ans += k, mincost += k * c, flow -= k;
        }
        vis[x] = 0;
        return ans;
    }

    inline void main(){
        while(spfa()){
            for(int i = 1; i ≤ n; ++ i)
                cur[i] = head[i];
            //memcpy(cur, head, sizeof head);
            ll now;
            while((now = dfs(S)))maxflow += now;//!
        }
    }

    inline void init(int _n, int _S, int _T){
        n = _n, S = _S, T = _T, tot = 0, maxflow = 0, mincost = 0;
        memset(head, -1, sizeof head);
    }
}


int n, m, S, T;
int main(){
    scanf("%d%d%d%d", &n, &m, &S, &T);
    dinic::init(n, S, T);//注意这个n，如果源点汇点要自己设置的话，这个n要和上面的T相等或者大于T，不然会T

    for(int i = 1; i ≤ m; ++ i){
        int x, y;
        ll z, c;
        scanf("%d%d%lld%lld", &x, &y, &z, &c);
        dinic::add(x, y, z, c, 1);
```

```
    }
    dinic::main();
    printf("%lld %lld\n", dinic::maxflow, dinic::mincost);
    return 0;
}
```

3.1.2 ZKW费用流

zkw费用流更快一点

> 例题：给定一些黑点白点，要求一个黑点链接一个白点并且线段不相交（转成二分图最大权匹配使用费用流解决）《训练指南》P351
> 输出方案：满流即为答案（满流是指这条路的流量跑满了，也就是说edge[i] == 0剩余流量为0）

```
const int N = 507, M = 200007;
const double DINF = 1010580540, eps = 1e-5;
int n, m;
int head[N], ver[M], nex[M], tot, edge[M];
double cost[M];
double dist[N];
int maxflow, maxcost;
int S, T;
bool vis[N];
int xa[N], xb[N], ya[N], yb[N];
int ax[N], bx[N], ay[N], by[N];
deque<int>q;
bool ins[N];
int cur[N];

void add(int x, int y, int z, double c, bool o = 1)
{
    ver[tot] = y;
    edge[tot] = z;
    cost[tot] = c;
    nex[tot] = head[x];
    head[x] = tot ++ ;
    if(o)add(y, x, 0, -c, 0);
}

double get_dist(int x, int y){
    return sqrt((double)(xa[x] - xb[y]) * (xa[x] - xb[y]) + (double)(ya[x] - yb[y]) * (ya[x] - yb[y]));
}

bool spfa()
{
    memset(ins, 0, sizeof ins);
    for(int i = S; i ≤ T; ++ i)
        dist[i] = DINF;
    q.push_front(T);
    dist[T] = 0;
    ins[T] = 1;
    while(!q.empty()){
        int x = q.front();
        q.pop_front();
        ins[x] = 0;
        for(int i= head[x]; ~i; i = nex[i]){
            int y = ver[i];
            if(edge[i ^ 1] && dist[y] - dist[x] + cost[i] > eps){
                dist[y] = dist[x] - cost[i];
                if(!ins[y]){
```

```
                    if(!q.empty() && dist[y] < dist[q.front()])q.push_front(y);
                    else q.push_back(y);
                    ins[y] = 1;
                }
            }
        }
    }
    return dist[S] != DINF;
}

int dfs(int x, int flow)
{
    vis[x] = 1;
    if(x == T || !flow)return flow;
    int used = 0;
    for(int i = cur[x]; ~i; i = nex[i])
    {
        cur[x] = i;
        int y = ver[i];
        if(dist[y] - dist[x] + cost[i] <= eps && !vis[y] && edge[i]){
            int fl = dfs(y, min(flow, edge[i]));
            if(fl){
                used += fl;
                flow -= fl;
                edge[i] -= fl;
                edge[i ^ 1] += fl;
                maxcost += fl *cost[i];
                if(!flow)break;
            }
        }
    }
    return used;
}

void zkwmcmf(){
    while(spfa()){
        vis[T] = 1;
        while(vis[T]){
            memcpy(cur, head, sizeof head);
            memset(vis, 0, sizeof vis);
            int fl = dfs(S, DINF);
            maxflow += fl;
        }
    }
}

double Calc(int x,int y)
{
    return sqrt((double)(ax[x]-bx[y])*(ax[x]-bx[y])+(double)(ay[x]-by[y])*(ay[x]-by[y]));
}

int main()
{
    while(scanf("%d", &n) != EOF){
        memset(head, -1, sizeof head);
        maxflow = maxcost = tot = 0;
        S = 0, T = 2 * n + 1;
        for(int i = 1; i <= n; ++ i)
            scanf("%d %d", &ax[i], &ay[i]);
        for(int i = 1; i <= n; ++i)
```

```
            scanf("%d %d", &bx[i], &by[i]);
        for(int i = 1; i ≤ n; ++i){
            for(int j = 1; j ≤ n; ++j){
                add(i, j + n, 1, Calc(i, j));
            }
        }

        for(int i = 1; i ≤ n; ++ i)
            add(S, i, 1, 0);
        for(int i = 1; i ≤ n; ++ i)
            add(i + n, T, 1, 0);

        zkwmcmf();
        for(int i = 1; i ≤ n; ++ i)
        for(int j = head[i]; ~j; j = nex[j]){
            int y = ver[j];
            if(!edge[j]){
                printf("%d\n", y - n);
                break;
            }
        }
    }
    return 0;
}

```

3.2 最大费用最大流

最大费用最大流，实现上有一个小技巧，即把费用设为负数 然后跑最小费用最大流，答案取相反数即可。

也可以如下例求最长路。

3.3 解决二分图带权最大匹配

类似最大流解决二分图多重匹配，每条边的权值就是他的单位费用。

> 给定一个 n×n 的矩阵，每一格有一个非负整数 ai,j。现在从 (1,1) 出发，可以往右或者往下走，最后到达 (n,n)。每到达一格，把该格子的数取出来，该格子的数就变成 0。这样一共走 k 次，现在要求 k 次所达到的方格的数的和最大。

- 点边转化：把每个格子 (i,j) 拆成一个入点一个出点。
- 从每个入点向对应的出点连两条有向边：一条容量为 1，费用为格子 ai,j；另一条容量为 k−1，费用为 0。
- 从 (i,j) 的出点到 (i,j+1) 和 (i+1,j) 的入点连有向边，容量为 k，费用为 0。
- 以 (1,1) 的入点为源点，(n,n) 的出点为汇点，求最大费用最大流。

```
const ll INF = 1e18;
const int N = 5e3+7, M = 5e5+7;
int maxflow,s,t,k;
int n,m,ans,e;
int head[N],ver[M],nex[M],edge[M],cost[M],tot;
bool vis[N];
int dis[N],incf[N],pre[N];

void add(int x,int y,int z,int c){//正边反边
    ver[++tot] = y;edge[tot] = z;cost[tot] = c;
    nex[tot] = head[x];head[x] = tot;
    ver[++tot] = x;edge[tot] = 0;cost[tot] = -c;
    nex[tot] = head[y];head[y] = tot;
}

int num(int i,int j,int k){
    return (i - 1) * n + j + k * n * n;
```

```
}

bool spfa(){//spfa求最长路
    queue<int>q;
    memset(vis,0,sizeof vis);
    memset(dis,0xcf,sizeof dis);//-INF
    q.push(s);
    dis[s] = 0;vis[s] = 1;
    incf[s] = 1<<30;//增广路各边的最小剩余容量
    while(q.size()){
        int x = q.front();q.pop();
        vis[x] = 0;//spfa的操作
        for(int i = head[x];i;i = nex[i]){
            if(edge[i]){//剩余容量要>0,才在残余网络中
                int y = ver[i];
                if(dis[y] < dis[x] + cost[i]){
                    dis[y] = dis[x] + cost[i];
                    incf[y] = min(incf[x],edge[i]);//最小剩余容量
                    pre[y] = i;//记录前驱（前向星编号），方便找到最长路的实际方案
                    if(!vis[y])
                        vis[y] = 1,q.push(y);
                }
            }
        }
    }
    if(dis[t] == 0xcfcfcfcf)
        return false;//汇点不可达，已求出最大流
    return true;
}

//EK的老操作了,更新最长增广路及其反向边的剩余容量
void update(){
    int x = t;
    while(x != s){
        int i = pre[x];
        edge[i] -= incf[t];
        edge[i ^ 1] += incf[t];//成对变换,反边加
        x = ver[i ^ 1];//反边回去的地方就是上一个结点
    }
    maxflow += incf[t];//顺便求最大流
    ans += dis[t] * incf[t];//题目要求
}

void EK(){
    while(spfa())//疯狂找增广路
        update();
}

int main(){
    cin>>n>>k;
    s = 1;t = 2 * n * n;
    tot = 1;
    over(i,1,n)
    over(j,1,n){
        int c;
        scanf("%d",&c);
        add(num(i,j,0),num(i,j,1),1,c);//自己（入点0）与自己（出点1）
        add(num(i,j,0),num(i,j,1),k-1,0);//两条边（取k次嘛，第一次有值，以后就没值了，用作下次选取）
        if(i < n)add(num(i,j,1),num(i+1,j,0),k,0);//自己（出点1）与下一行（入点0）或者下一列（入点0）
        if(j < n)add(num(i,j,1),num(i,j+1,0),k,0);
```

```
    }
    EK();
    printf("%d\n",ans);
    return 0;
}

```

3.4 费用提前计算+动态开点

$\oint$ 4. 上下界网络流

4.1 无源汇上下界可行流

```
/*
      首先建立一个源S和一个汇T，一般称为附加源和附加汇。
    对于图中的每条弧<u,v>，假设它容量上界为c，下界b，那么把这条边拆为三条只有上界的弧。
    一条为<S,v>，容量为b；
    一条为<u,T>，容量为b；
    一条为<u,v>，容量为c-b。
     其中前两条弧一般称为附加弧。
     然后对这张图跑最大流，以S为源，以T为汇，如果所有的附加弧都满流，则原图有可行流;否则就是无解。
     这时，每条非附加弧的流量加上它的容量下界，就是原图中这条弧应该有的流量。

     对于原图中的每条弧，我们把c-b称为它的自由流量，意思就是只要它流满了下界，这些流多少都没问题。
    既然如此，对于每条弧<u,v>，我们强制给v提供b单位的流量，并且强制从u那里拿走b单位的流量，这一步对应着两条附加弧。
    如果这一系列强制操作能完成的话，也就是有一组可行流了。
    注意：这张图的最大流只是对应着原图的一组可行流，而不是原图的最大或最小流。
*/
#include <bits/stdc++.h>
using namespace std;
const int oo = (1LL << 31) - 1;
const int LEN = 1e5 + 5;
struct node {
    int x, y, l, r;
} a[LEN];
namespace ISAP {
    int flow, tot, n, m, src, tar, qh, qt, cnt, ans;
    struct edge {
        int vet, next, len;
    } E[LEN * 2];
    int dis[LEN], gap[LEN], head[LEN], cur[LEN], q[LEN], vis[LEN], IN[LEN];
    void add(int u, int v, int c) {
        E[++tot] = (edge){v, head[u], c};
        head[u] = tot;
    }
    void join(int u, int v, int c) {
        add(u, v, c);
        add(v, u, 0);
    }
    void bfs(int s) {
        qh = qt = 0;
        q[++qt] = s;
        dis[s] = 0;
        vis[s] = 1;
        while (qh < qt) {
            int u = q[++qh];
            gap[dis[u]]++;
```

```
            for (int e = head[u]; e ≠ -1; e = E[e].next) {
                int v = E[e].vet;
                if (E[e ^ 1].len && !vis[v]) {
                    dis[v] = dis[u] + 1;
                    vis[v] = 1;
                    q[++qt] = v;
                }
            }
        }
    }
    int isap(int u, int aug) {
        if (u == tar) return aug;
        int flow = 0;
        for (int e = head[u]; e ≠ -1; e = E[e].next) {
            int v = E[e].vet;
            if (E[e].len && dis[v] == dis[u] - 1) {
                int tmp = isap(v, min(aug - flow, E[e].len));
                E[e].len -= tmp;
                E[e ^ 1].len += tmp;
                flow += tmp;
                head[u] = e;
                if (flow == aug || dis[src] == cnt) return flow;
            }
        }
        if (!--gap[dis[u]++]) dis[src] = cnt;
        ++gap[dis[u]];
        head[u] = cur[u];
        return flow;
    }
    void init() {
        tot = -1, gap[0] = 0;
        for (int i = 1; i ≤ cnt; i++) {
            dis[i] = gap[i] = vis[i] = IN[i] = 0;
            head[i] = -1;
        }
    }
    int maxflow(int s, int t) {
        src = s, tar = t;
        int res = 0;
        for (int i = 1; i ≤ cnt; i++) cur[i] = head[i];
        bfs(tar);
        while (dis[src] < cnt) res += isap(src, oo);
        return res;
    }
}
using namespace ISAP;
int main() {
    scanf("%d %d", &n, &m);
    cnt = n;
    src = ++cnt, tar = ++cnt;
    init();
    for (int i = 1; i ≤ m; i++) {
        int x, y, l, r;
        scanf("%d %d %d %d", &x, &y, &l, &r);
        a[i] = (node){x, y, l, r};
        join(x, y, r - l);
        IN[y] += l, IN[x] -= l;
    }
    for (int i = 1; i ≤ n; i++) {
        if (IN[i] < 0) join(i, tar, -IN[i]);
```

```
        else {
            join(src, i, IN[i]);
            flow += IN[i];
        }
    }
    int ans = maxflow(src, tar);
    if (flow == ans) {
        puts("YES");
        for (int i = 1; i ≤ m; i++) printf("%d\n", a[i].l + E[i * 2 - 1].len);
    } else puts("NO");
    return 0;
}

```

4.2 有源汇有上下界可行流

> 模型:现在的网络有一个**源点s和汇点t**,求出一个流使得源点的总流出量等于汇点的总流入量,其他的点满足流量守恒,而且每条边的流量满足上界和下界限制.
> 源点和汇点不满足流量守恒,这让我们很难办,因此我们想办法把问题转化成容易处理的每个点都满足流量守恒的无源汇情况.
> 为了使源汇点满足流量守恒,我们需要有边流入源点s,有边流出汇点t.注意到源点s的流出量等于汇点t的流入量,我们就可以从汇点t向源点s连一条下界为0上界为无穷大的边,相当于把从源点s流出的流量再流回来.在这样的图中套用上面的算法求出一个可行的循环流,拆掉从汇点t到源点s的边就得到一个可行的有源汇流.
> 这里有一个小问题:最后得到的可行的有源汇流的流量是多少?
> 可以发现,循环流中一定满足s流出的总流量=流入s的总流量,假定原图中没有边流入s,那么s流出的流量就是t到s的无穷边的流量,也就是s-t可行流的流量.因此我们最后看一下t到s的无穷边的流量(即dinic跑完之后反向边的权值)即可知道原图中有源汇可行流的流量.
> 代码:这个可行流算法在有源汇有上下界最大流/最小流中都会用到,可以看下面两个算法的代码

4.3 有源汇上下界最大流

```
/*
    先来看有源汇可行流
    建模方法：
    建立弧<t,s>，容量下界为0，上界为oo。
    然后对这个新图（实际上只是比原图多了一条边）按照无源汇可行流的方法建模，
    如果所有附加弧满流，则存在可行流。
    求原图中每条边对应的实际流量的方法，同无源汇可行流，只是忽略掉弧<t,s>就好。
    而且这时候弧<t,s>的流量就是原图的总流量。
    理解方法：
    有源汇相比无源汇的不同就在于，源和汇是不满足流量平衡的，那么连接<t,s>之后，
    源和汇也满足了流量平衡，就可以直接按照无源汇的方式建模。
    注意：这张图的最大流只是对应着原图的一组可行流，而不是原图的最大或最小流。

    有源汇最大流
    建模方法：
    首先按照有源汇可行流的方法建模，如果不存在可行流，更别提什么最大流了。
    如果存在可行流，那么在运行过有源汇可行流的图上（就是已经存在流量的那张图，流量不要清零），
    跑一遍从s到t的最大流（这里的s和t是原图的源和汇，不是附加源和附加汇），就是原图的最大流。
    理解方法：
    为什么要在那个已经有了流量的图上跑最大流？因为那张图保证了每条弧的容量下界，在这张图上跑最大流，
    实际上就是在容量下界全部满足的前提下尽量多得获得“自由流量”。
    注意，在这张已经存在流量的图上，弧<t,s>也是存在流量的，千万不要忽略这条弧。
    因为它的相反弧<s,t>的流量为<t,s>的流量的相反数，且<s,t>的容量为0，所以这部分的流量也是会被算上的。
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
```

```cpp
const int LEN = 1e5 + 5;
const int oo = (1LL << 31) - 1;
namespace DINIC {
    int tot, n, m, src, tar, qh, qt, cnt, s, t, S, T;
    int ans, flow;
    struct edge {
        int vet, next, len;
    } E[LEN * 2];
    int dis[LEN], gap[LEN], head[LEN], cur[LEN], q[LEN], vis[LEN], IN[LEN];
    void add(int u, int v, int c) {
        E[++tot] = (edge){v, head[u], c};
        head[u] = tot;
    }
    void join(int u, int v, int c) {
        add(u, v, c);
        add(v, u, 0);
    }
    void init() {
        tot = -1;
        for (int i = 1; i ≤ cnt; i++) head[i] = -1;
    }
    bool bfs() {
        for (int i = 1; i ≤ cnt; i++) dis[i] = 0;
        qh = qt = 0;
        q[++qt] = src;
        dis[src] = 1;
        while (qh < qt) {
            int u = q[++qh];
            for (int e = head[u]; e ≠ -1; e = E[e].next) {
                int v = E[e].vet;
                if (E[e].len && !dis[v]) {
                    dis[v] = dis[u] + 1;
                    if (v == tar) return 1;
                    q[++qt] = v;
                }
            }
        }
        return dis[tar];
    }
    int dfs(int u, int aug) {
        if (u == tar || !aug) return aug;
        int tmp = 0;
        for (int &e = cur[u]; e ≠ -1; e = E[e].next) {
            int v = E[e].vet;
            if (dis[v] == dis[u] + 1) {
                if (tmp = dfs(v, min(aug, E[e].len))) {
                    E[e].len -= tmp;
                    E[e ^ 1].len += tmp;
                    return tmp;
                }
            }
        }
        return 0;
    }
    int maxflow(int s, int t) {
        src = s, tar = t;
        int res = 0, flow = 0;
        while (bfs()) {
            for (int i = 1; i ≤ cnt; i++) cur[i] = head[i];
            while (flow = dfs(src, oo)) res += flow;
```

```
        }
        return res;
    }
}
using namespace DINIC;
int main() {
    scanf("%d %d %d %d", &n, &m, &s, &t);
    cnt = n;
    S = ++cnt, T = ++cnt;
    init();
    for (int i = 1; i ≤ m; i++) {
        int x, y, l, r;
        scanf("%d %d %d %d", &x, &y, &l, &r);
        join(x, y, r - l);
        IN[y] += l, IN[x] -= l;
    }
    for (int i = 1; i ≤ n; i++) {
        if (IN[i] < 0) join(i, T, -IN[i]);
        else if (IN[i] > 0) {
            flow += IN[i];
            join(S, i, IN[i]);
        }
    }
    join(t, s, oo);
    ans = maxflow(S, T);
    if (ans ≠ flow) puts("please go home to sleep");
    else {
        ans = maxflow(s, t);
        printf("%lld\n", ans);
    }
    return 0;
}
```

4.4 有源汇上下界最小流

```
/*
    先来看有源汇可行流
    建模方法：
    建立弧<t,s>，容量下界为0，上界为oo。
    然后对这个新图（实际上只是比原图多了一条边）按照无源汇可行流的方法建模，
    如果所有附加弧满流，则存在可行流。
    求原图中每条边对应的实际流量的方法，同无源汇可行流，只是忽略掉弧<t,s>就好。
    而且这时候弧<t,s>的流量就是原图的总流量。
    理解方法：
    有源汇相比无源汇的不同就在于，源和汇是不满足流量平衡的，那么连接<t,s>之后，
    源和汇也满足了流量平衡，就可以直接按照无源汇的方式建模。
    注意：这张图的最大流只是对应着原图的一组可行流，而不是原图的最大或最小流。

    有源汇最小流
    有源汇最小流的常见建模方法比较多，我就只说我常用的一种。
    建模方法：
    首先按照有源汇可行流的方法建模，但是不要建立<t,s>这条弧。
    然后在这个图上，跑从附加源ss到附加汇tt的最大流。
    这时候再添加弧<t,s>，下界为0，上界oo。
    在现在的这张图上，从ss到tt的最大流，就是原图的最小流。
    理解方法：
    我们前面提到过，有源汇可行流的流量只是对应一组可行流，并不是最大或者最小流。
    并且在跑完有源汇可行流之后，弧<t,s>的流量就是原图的流量。
    从这个角度入手，我们想让弧<t,s>的流量尽量小，就要尽量多的消耗掉那些“本来不需要经过<t,s>”的流量。
```

```
    于是我们在添加<t,s>之前，跑一遍从ss到tt的最大流，就能尽量多的消耗那些流量啦QwQ。
*/
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const int LEN = 2e5 + 5;
const int oo = (1LL << 31) - 1;
namespace DINIC {
    int tot, n, m, src, tar, qh, qt, cnt, s, t, S, T, ans, flow;
    struct edge {
        int vet, next, len;
    } E[LEN * 2];
    int dis[LEN], gap[LEN], head[LEN], cur[LEN], q[LEN], vis[LEN], IN[LEN];
    void add(int u, int v, int c) {
        E[++tot] = (edge){v, head[u], c};
        head[u] = tot;
    }
    void join(int u, int v, int c) {
        add(u, v, c);
        add(v, u, 0);
    }
    void init() {
        tot = -1;
        for (int i = 1; i ≤ cnt; i++) head[i] = -1;
    }
    bool bfs() {
        for (int i = 1; i ≤ cnt; i++) dis[i] = 0;
        qh = qt = 0;
        q[++qt] = src;
        dis[src] = 1;
        while (qh < qt) {
            int u = q[++qh];
            for (int e = head[u]; e ≠ -1; e = E[e].next) {
                int v = E[e].vet;
                if (E[e].len && !dis[v]) {
                    dis[v] = dis[u] + 1;
                    if (v == tar) return 1;
                    q[++qt] = v;
                }
            }
        }
        return dis[tar];
    }
    int dfs(int u, int aug) {
        if (u == tar || !aug) return aug;
        int tmp = 0;
        for (int &e = cur[u]; e ≠ -1; e = E[e].next) {
            int v = E[e].vet;
            if (dis[v] == dis[u] + 1) {
                if (tmp = dfs(v, min(aug, E[e].len))) {
                    E[e].len -= tmp;
                    E[e ^ 1].len += tmp;
                    return tmp;
                }
            }
        }
        return 0;
    }
    int maxflow(int s, int t) {
        src = s, tar = t;
```

```
        int res = 0, flow = 0;
        while (bfs()) {
            for (int i = 1; i ≤ cnt; i++) cur[i] = head[i];
            while (flow = dfs(src, oo)) res += flow;
        }
        return res;
    }
}
using namespace DINIC;
int main() {
    scanf("%d %d %d %d", &n, &m, &s, &t);
    cnt = n;
    S = ++cnt, T = ++cnt;
    init();
    for (int i = 1; i ≤ m; i++) {
        int x, y, l, r;
        scanf("%d %d %d %d", &x, &y, &l, &r);
        join(x, y, r - l);
        IN[y] += l, IN[x] -= l;
    }
    for (int i = 1; i ≤ n; i++) {
        if (IN[i] < 0) join(i, T, -IN[i]);
        else if (IN[i] > 0) {
            flow += IN[i];
            join(S, i, IN[i]);
        }
    }
    ans = maxflow(S, T);
    flow -= ans;
    join(t, s, oo);
    ans = maxflow(S, T);
    if (ans ≠ flow) puts("please go home to sleep");
    else printf("%d\n", ans);
    return 0;
}
```

七、数论

$\oint$ 1. 基础数论

试除法判定质数

```
bool is_prime(int x)
{
    if (x < 2) return false;
    for (int i = 2; i ≤ x / i; i ++ )
        if (x % i == 0)
            return false;
    return true;
}
```

试除法分解质因数

```cpp
void divide(int x)
{
    for (int i = 2; i ≤ x / i; i ++ )
        if (x % i == 0)
        {
            int s = 0;
            while (x % i == 0) x /= i, s ++ ;
            cout << i << ' ' << s << endl;
        }
    if (x > 1) cout << x << ' ' << 1 << endl;
    cout << endl;
}
```

朴素筛法求素数

```cpp
int primes[N], cnt;     // primes[]存储所有素数
bool st[N];         // st[x]存储x是否被筛掉

void get_primes(int n)
{
    for (int i = 2; i ≤ n; i ++ )
    {
        if (st[i]) continue;
        primes[cnt ++ ] = i;
        for (int j = i + i; j ≤ n; j += i)
            st[j] = true;
    }
}
```

线性筛法求素数

```cpp
int primes[N], cnt;     // primes[]存储所有素数
bool st[N];         // st[x]存储x是否被筛掉

void get_primes(int n)
{
    for (int i = 2; i ≤ n; i ++ )
    {
        if (!st[i]) primes[cnt ++ ] = i;
        for (int j = 0; primes[j] ≤ n / i; j ++ )
        {
            st[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}
```

试除法求所有约数

```cpp
vector<int> get_divisors(int x)
{
    vector<int> res;
    for (int i = 1; i ≤ x / i; i ++ )
        if (x % i == 0)
        {
            res.push_back(i);
            if (i ≠ x / i) res.push_back(x / i);
        }
    sort(res.begin(), res.end());
    return res;
}
```

约数个数和约数之和

```
如果 N = p1^c1 * p2^c2 * ... *pk^ck
约数个数: (c1 + 1) * (c2 + 1) * ... * (ck + 1)
约数之和: (p1^0 + p1^1 + ... + p1^c1) * ... * (pk^0 + pk^1 + ... + pk^ck)
```

欧几里得算法

```cpp
/*求最大公约数*/
int gcd(int a, int b)
{
    return b ? gcd(b, a % b) : a;
}
```

求欧拉函数

```cpp
int phi(int x)
{
    int res = x;
    for (int i = 2; i ≤ x / i; i ++ )
        if (x % i == 0)
        {
            res = res / i * (i - 1);
            while (x % i == 0) x /= i;
        }
    if (x > 1) res = res / x * (x - 1);

    return res;
}
```

筛法求欧拉函数

```cpp
int primes[N], cnt;     // primes[]存储所有素数
int euler[N];           // 存储每个数的欧拉函数
bool st[N];         // st[x]存储x是否被筛掉


void get_eulers(int n)
{
    euler[1] = 1;
    for (int i = 2; i ≤ n; i ++ )
    {
```

```
        if (!st[i])
        {
            primes[cnt ++ ] = i;
            euler[i] = i - 1;
        }
        for (int j = 0; primes[j] ≤ n / i; j ++ )
        {
            int t = primes[j] * i;
            st[t] = true;
            if (i % primes[j] == 0)
            {
                euler[t] = euler[i] * primes[j];
                break;
            }
            euler[t] = euler[i] * (primes[j] - 1);
        }
    }
}
```

快速幂

```
求 m^k mod p，时间复杂度 O(logk)。

int qmi(int m, int k, int p)
{
    int res = 1 % p, t = m;
    while (k)
    {
        if (k&1) res = res * t % p;
        t = t * t % p;
        k >>= 1;
    }
    return res;
}
```

扩展欧几里得算法

```
// 求x, y，使得ax + by = gcd(a, b)
int exgcd(int a, int b, int &x, int &y)
{
    if (!b)
    {
        x = 1; y = 0;
        return a;
    }
    int d = exgcd(b, a % b, y, x);
    y -= (a/b) * x;
    return d;
}
```

高斯消元

```
/*高斯消元解线性方程组*/
// a[N][N]是增广矩阵
int gauss()
{
    int c, r;
    for (c = 0, r = 0; c < n; c ++ )
```

```
    {
        int t = r;
        for (int i = r; i < n; i ++ )   // 找到绝对值最大的行
            if (fabs(a[i][c]) > fabs(a[t][c]))
                t = i;

        if (fabs(a[t][c]) < eps) continue;

        for (int i = c; i ≤ n; i ++ ) swap(a[t][i], a[r][i]);       // 将绝对值最大的行换到最顶端
        for (int i = n; i ≥ c; i -- ) a[r][i] /= a[r][c];       // 将当前上的首位变成1
        for (int i = r + 1; i < n; i ++ )       // 用当前行将下面所有的列消成0
            if (fabs(a[i][c]) > eps)
                for (int j = n; j ≥ c; j -- )
                    a[i][j] -= a[r][j] * a[i][c];

        r ++ ;
    }

    if (r < n)
    {
        for (int i = r; i < n; i ++ )
            if (fabs(a[i][n]) > eps)
                return 2; // 无解
        return 1; // 有无穷多组解
    }

    for (int i = n - 1; i ≥ 0; i -- )
        for (int j = i + 1; j < n; j ++ )
            a[i][n] -= a[i][j] * a[j][n];

    return 0; // 有唯一解
}
```

递归法求组合数

```
// c[a][b] 表示从a个苹果中选b个的方案数
for (int i = 0; i < N; i ++ )
    for (int j = 0; j ≤ i; j ++ )
        if (!j) c[i][j] = 1;
        else c[i][j] = (c[i - 1][j] + c[i - 1][j - 1]) % mod;
```

通过预处理逆元的方式求组合数

首先预处理出所有阶乘取模的余数fact[N]，以及所有阶乘取模的逆元infact[N]
如果取模的数是质数，可以用费马小定理求逆元

```
int qmi(int a, int k, int p)    // 快速幂模板
{
    int res = 1;
    while (k)
    {
        if (k & 1) res = (LL)res * a % p;
        a = (LL)a * a % p;
        k >>= 1;
    }
    return res;
}

```

```
// 预处理阶乘的余数和阶乘逆元的余数
fact[0] = infact[0] = 1;
for (int i = 1; i < N; i ++ )
{
    fact[i] = (LL)fact[i - 1] * i % mod;
    infact[i] = (LL)infact[i - 1] * qmi(i, mod - 2, mod) % mod;
}
```

Lucas定理

```
若p是质数，则对于任意整数 1 ≤ m ≤ n，有：
    C(n, m) = C(n % p, m % p) * C(n / p, m / p) (mod p)

int qmi(int a, int k)        // 快速幂模板
{
    int res = 1;
    while (k)
    {
        if (k & 1) res = (LL)res * a % p;
        a = (LL)a * a % p;
        k >>= 1;
    }
    return res;
}


int C(int a, int b)      // 通过定理求组合数C(a, b)
{
    int res = 1;
    for (int i = 1, j = a; i ≤ b; i ++, j -- )
    {
        res = (LL)res * j % p;
        res = (LL)res * qmi(i, p - 2) % p;
    }
    return res;
}


int lucas(LL a, LL b)
{
    if (a < p && b < p) return C(a, b);
    return (LL)C(a % p, b % p) * lucas(a / p, b / p) % p;
}
```

分解质因数法求组合数

```
当我们需要求出组合数的真实值，而非对某个数的余数时，分解质因数的方式比较好用：
    1. 筛法求出范围内的所有质数
    2. 通过 C(a, b) = a! / b! / (a - b)! 这个公式求出每个质因子的次数。 n! 中p的次数是 n / p + n / p^2 + n / p^3 + ...
    3. 用高精度乘法将所有质因子相乘

int primes[N], cnt;      // 存储所有质数
int sum[N];      // 存储每个质数的次数
bool st[N];      // 存储每个数是否已被筛掉


void get_primes(int n)      // 线性筛法求素数
```

```
{
    for (int i = 2; i ≤ n; i ++ )
    {
        if (!st[i]) primes[cnt ++ ] = i;
        for (int j = 0; primes[j] ≤ n / i; j ++ )
        {
            st[primes[j] * i] = true;
            if (i % primes[j] == 0) break;
        }
    }
}


int get(int n, int p)         // 求n! 中的次数
{
    int res = 0;
    while (n)
    {
        res += n / p;
        n /= p;
    }
    return res;
}


vector<int> mul(vector<int> a, int b)         // 高精度乘低精度模板
{
    vector<int> c;
    int t = 0;
    for (int i = 0; i < a.size(); i ++ )
    {
        t += a[i] * b;
        c.push_back(t % 10);
        t /= 10;
    }

    while (t)
    {
        c.push_back(t % 10);
        t /= 10;
    }

    return c;
}

get_primes(a);  // 预处理范围内的所有质数

for (int i = 0; i < cnt; i ++ )      // 求每个质因数的次数
{
    int p = primes[i];
    sum[i] = get(a, p) - get(b, p) - get(a - b, p);
}

vector<int> res;
res.push_back(1);

for (int i = 0; i < cnt; i ++ )      // 用高精度乘法将所有质因子相乘
    for (int j = 0; j < sum[i]; j ++ )
        res = mul(res, primes[i]);
```

卡特兰数

给定n个0和n个1，它们按照某种顺序排成长度为2n的序列，满足任意前缀中0的个数都不少于1的个数的序列的数量为： Cat(n) = C(2n, n) / (n + 1)

NIM游戏

给定N堆物品，第i堆物品有Ai个。两名玩家轮流行动，每次可以任选一堆，取走任意多个物品，可把一堆取光，但不能不取。取走最后一件物品者获胜。两人都采取最优策略，问先手是否必胜。
我们把这种游戏称为NIM博弈。把游戏过程中面临的状态称为局面。整局游戏第一个行动的称为先手，第二个行动的称为后手。若在某一局面下无论采取何种行动，都会输掉游戏，则称该局面必败。
所谓采取最优策略是指，若在某一局面下存在某种行动，使得行动后对面面临必败局面，则优先采取该行动。同时，这样的局面被称为必胜。我们讨论的博弈问题一般都只考虑理想情况，即两人均无失误，都采取最优策略行动时游戏的结果。

NIM博弈不存在平局，只有先手必胜和先手必败两种情况。
定理： NIM博弈先手必胜，当且仅当 A1 ^ A2 ^ … ^ An != 0

公平组合游戏ICG
若一个游戏满足：

由两名玩家交替行动；
在游戏进程的任意时刻，可以执行的合法行动与轮到哪名玩家无关；
不能行动的玩家判负；

则称该游戏为一个公平组合游戏。
NIM博弈属于公平组合游戏，但城建的棋类游戏，比如围棋，就不是公平组合游戏。因为围棋交战双方分别只能落黑子和白子，胜负判定也比较复杂，不满足条件2和条件3。

有向图游戏
给定一个有向无环图，图中有一个唯一的起点，在起点上放有一枚棋子。两名玩家交替地把这枚棋子沿有向边进行移动，每次可以移动一步，无法移动者判负。该游戏被称为有向图游戏。
任何一个公平组合游戏都可以转化为有向图游戏。具体方法是，把每个局面看成图中的一个节点，并且从每个局面向沿着合法行动能够到达的下一个局面连有向边。

Mex运算
设S表示一个非负整数集合。定义mex(S)为求出不属于集合S的最小非负整数的运算，即：
mex(S) = min{x}, x属于自然数，且x不属于S

SG函数

在有向图游戏中，对于每个节点x，设从x出发共有k条有向边，分别到达节点y1, y2, …, yk，定义SG(x)为x的后继节点y1, y2, …, yk的SG函数值构成的集合再执行mex(S)运算的结果，即：

```
SG(x) = mex({SG(y1), SG(y2), …, SG(yk)})
```

特别地，整个有向图游戏G的SG函数值被定义为有向图游戏起点s的SG函数值，即SG(G) = SG(s)。

有向图游戏的和 —— 模板题 AcWing 893. 集合-Nim游戏
设G1, G2, …, Gm 是m个有向图游戏。定义有向图游戏G，它的行动规则是任选某个有向图游戏Gi，并在Gi上行动一步。G被称为有向图游戏G1, G2, …, Gm的和。
有向图游戏的和的SG函数值等于它包含的各个子游戏SG函数值的异或和，即：

```
  SG(G) = SG(G1) ^ SG(G2) ^ … ^ SG(Gm)
```

定理
有向图游戏的某个局面必胜，当且仅当该局面对应节点的SG函数值大于0。
有向图游戏的某个局面必败，当且仅当该局面对应节点的SG函数值等于0。

$\oint$ 2. 质数筛法

线性筛法

```cpp
int primes[N], cnt;
bool vis[N];

void init(int x){
    vis[1] = vis[0] = 1;
    for(int i = 2; i ≤ x; ++ i){
        if(!vis[i])primes[cnt ++ ] = i;
        for(int j = 0; primes[j] * i ≤ x; ++ j){
            vis[primes[j] * i] = true;
            if(i % primes[j] == 0)break;
        }
    }
}
```

二次筛法

```cpp
/*AcWing 196. 质数距离
整数数列1~n,给出区间[L,R] (L,R≤2^31,R-L≤1e6) 求区间内相邻质数距离的最大值和最小值*/
#include <cstring>
#include <iostream>
#include <algorithm>

using namespace std;

typedef long long LL;

const int N = 1000010;

int primes[N], cnt;
bool st[N];

void init(int n)
{
    memset(st, 0, sizeof st);
    cnt = 0;
    for (int i = 2; i ≤ n; i ++ )
    {
        if (!st[i]) primes[cnt ++ ] = i;
        for (int j = 0; primes[j] * i ≤ n; j ++ )
        {
            st[i * primes[j]] = true;
            if (i % primes[j] == 0) break;
        }
    }
}

int main()
{
    int l, r;
    while (cin >> l >> r)
    {
        init(50000);

```

```
        memset(st, 0, sizeof st);
        for (int i = 0; i < cnt; i ++ )
        {
            LL p = primes[i];
            for (LL j = max(p * 2, (l + p - 1) / p * p); j ≤ r; j += p)
                st[j - l] = true;
        }

        cnt = 0;
        for (int i = 0; i ≤ r - l; i ++ )
            if (!st[i] && i + l ≥ 2)
                primes[cnt ++ ] = i + l;

        if (cnt < 2) puts("There are no adjacent primes.");
        else
        {
            int minp = 0, maxp = 0;
            for (int i = 0; i + 1 < cnt; i ++ )
            {
                int d = primes[i + 1] - primes[i];
                if (d < primes[minp + 1] - primes[minp]) minp = i;
                if (d > primes[maxp + 1] - primes[maxp]) maxp = i;
            }

            printf("%d,%d are closest, %d,%d are most distant.\n",
                primes[minp], primes[minp + 1],
                primes[maxp], primes[maxp + 1]);
        }
    }

    return 0;
}
```

$\oint$ 3. 分解质因数

试除法分解质因数

```
void divide(int x)
{
    for (int i = 2; i ≤ x / i; i ++ )
        if (x % i == 0)
        {
            int s = 0;
            while (x % i == 0) x /= i, s ++ ;
            cout << i << ' ' << s << endl;
        }
    if (x > 1) cout << x << ' ' << 1 << endl;
    cout << endl;
}
```

Pollard Rho因数分解

时间复杂度： $O(n^{\frac{1}{4}})$

```c
#include "stdio.h"
#include "conio.h"
main()
{
    int n,i;
    scanf("%d",&n);
    for(i=2;i≤n;i++)
        while(n≠i)
            if(n%i==0)
                printf("%d*",i),n=n/i;
            else break;
   printf("%d",n);
}
```

快速质因数分解阶乘

$n! = 1 * 2 * 3 * 4 * ... * n$,利用这一性质快速分解质因数
我们发现N!中质数因子p的个数,就是 $1\ N$ 中每个数含有的质因数p个数.既然如此的话,那么我们发现,至少有一个质因子 p 的显然有 $[\frac{n}{p}]$个,而至少有两个质因子p数的显然是有$[\frac{n}{p^2}]$。

时间复杂度：$O(nloglogn)$

```c
/*AcWing 197. 阶乘分解
给定整数 N ，试把阶乘 N! 分解质因数，按照算术基本定理的形式输出分解结果中的 pi 和 ci 即可。*/
#include <cstdio>
#include <cstring>
#include <iostream>
#include <algorithm>

using namespace std;

const int N = 1000010;

int primes[N], cnt;
bool st[N];

void init(int n)
{
    for (int i = 2; i ≤ n; i ++ )
    {
        if (!st[i]) primes[cnt ++ ] = i;
        for (int j = 0; primes[j] * i ≤ n; j ++ )
        {
            st[i * primes[j]] = true;
            if (i % primes[j] == 0) break;
        }
    }
}

int main()
{
    int n;
    cin >> n;
    init(n);

    for (int i = 0; i < cnt; i ++ )
    {
        int p = primes[i];
        int s = 0;
        for (int j = n; j; j /= p) s += j / p;
```

```
39          printf("%d %d\n", p, s);
40      }
41
42      return 0;
43  }
```

$\oint$ 4. 线性基

线性基是一种擅长处理异或问题的数据结构.设值域为$[1,N]$，就可以用一个长度为$\lceil \log_2 N \rceil$的数组来描述一个线性基。特别地，线性基第i位上的数二进制下最高位也为第i位。

一个线性基满足，对于它所表示的所有数的集合S，S中任意多个数异或所得的结果均能表示为线性基中的元素互相异或的结果，即意，线性基能使用异或运算来表示原数集使用异或运算能表示的所有数。运用这个性质，我们可以极大地缩小异或操作所需的查询次数。

本模板支持以下操作：

- 插入和判断
- 查询异或最值
- 查询k小值

```
1  typedef long long ll;
2  const int N = 60;
3  bool flag;
4  ll a[N + 1],tmp[N + 1];
5
6  void Insert(ll x){
7      for(int i = N;~i;i--)
8          if(x & (1ll << i))//强转成 long long（因为x是ll）
9              if(!a[i])
10                 {a[i] = x;return ;}
11             else x ^= a[i];
12     flag = true;//表示线性基里至少有一个
13 }
14
15 bool check(ll x){
16
17     for(int i = N;~i;i--)
18         if(x & (1ll << i))
19             if(!a[i])
20                 return false;
21             else x ^= a[i];
22     return true;
23 }
24
25 ll qmax(ll res = 0){
26
27     for(int i = N;~i;i--)
28         res = max(res,res ^ a[i]);
29     return res;
30 }
31
32 ll qmin(){
33     if(flag)return 0;
34     for(int i = 0;i ≤ N;++i)//找最小值从低位到高位
35         if(a[i])return a[i];
36 }
37
38 ll query(ll k){
39     ll res = 0;
```

```
    int cnt = 0;
    k -= flag;//因为是从第0位开始的，有元素的话就-1
    if(!k)return 0;
    for(int i = 0;i ≤ N;++i){
        for(int j = i - 1;~j;j--)
            if(a[i] & (1ll << j))
                a[i] ^= a[j];
        if(a[i])tmp[cnt++] = a[i];
    }
    if(k ≥ (1ll << cnt))return -1;
    for(int i = 0;i < cnt;++i)
        if(k & (1ll << i))res ^= tmp[i];
    return res;
}
int n;
ll x;
int main(){

    scanf("%d",&n);
    for(int i = 1;i ≤ n;++i)
        scanf("%lld",&x),Insert(x);
    printf("%lld\n",qmax());
    return 0;
}

```

§ 5. 质数

1.暴力判

【模板】素数、コンテスト、素数 [AT807]

```
#include<cstdio>
#include<cmath>
int x;
inline bool judge(int n){
    if(n<4)return 1;
    if(n%2==0)return 0;
    int half=sqrt(n);
    for(int i=3;i≤half;i+=2)
        if(n%i==0)return 0;
    return 1;
}
int main(){
    scanf("%d",&x);
    puts(judge(x)?"YES":"NO");
}
```

2.Miller Rabin

【模板】质数判定 [Loj143]

```
#include<cstdio>
#define LL long long
#define Re register LL
LL n,prime[10]={2,3,5,7,11,13,17,19,23,29};
inline LL cf(Re x,Re k,Re P){
    Re s=0;
```

```
    while(k){
        if(k&1)(s+=x)%=P;
        (x<<=1)%=P,k>>=1;
    }
    return s%P;
}
inline LL mi(Re x,Re k,Re P){
    Re s=1;
    while(k){
        if(k&1)s=cf(s,x,P)%P;
        x=cf(x,x,P)%P,k>>=1;
    }
    return s%P;
}
inline bool Miller_Rabin(Re x){
    Re s=0,t=x-1;
    if(x==2)return 1;
    if(x<2||!(x&1))return 0;
    while(!(t&1))s++,t>>=1;
    for(Re i=0;i<10&&prime[i]<x;++i){
        Re a=prime[i],b=mi(a,t,x);
        for(Re j=1,k;j≤s;++j){
            k=cf(b,b,x);
            if(k==1&&b≠1&&b≠x-1)return 0;
            b=k;
        }
        if(b≠1)return 0;
    }
    return 1;
}
int main(){
    while(scanf("%lld",&n)≠EOF)puts(Miller_Rabin(n)?"Y":"N");
}
```

【模板】PON - Prime or Not [SP288]

3.埃氏筛

【模板】线性筛素数 [P3383]

```
#include<algorithm>
#include<cstdio>
#define Re register int
using namespace std;
const int N=1e7+3;
int n,x,T,cnt,pri[N/3];bool pan[N];
inline void in(Re &x){
    int f=0;x=0;char c=getchar();
    while(c<'0'||c>'9')f|=c=='-',c=getchar();
    while(c≥'0'&&c≤'9')x=(x<<1)+(x<<3)+(c^48),c=getchar();
    x=f?-x:x;
}
inline void get_pri(Re N){
    pan[0]=pan[1]=1;
    for(Re i=2;i≤N;i++)
        if(!pan[i]){
            pri[++cnt]=i;
            for(Re j=i;j≤N/i;j++)pan[i*j]=1;
        }
}
```

```
int main(){
    in(n),in(T),get_pri(n);
    while(T--)in(x),puts(pan[x]?"No":"Yes");
}
```

4.欧拉筛

【模板】线性筛素数

```
#include<cstdio>
#include<ctime>
const int N=1e6;
int cnt,i,j,pri[N/3];bool pan[N+1];
inline void get_pri(){
    pan[0]=pan[1]=1;
    for(i=2;i≤N;++i){
        if(!pan[i])pri[++cnt]=i;
        for(j=1;j≤cnt&&i*pri[j]≤N;++j){
            pan[i*pri[j]]=1;
            if(i%pri[j]==0)break;
        }
    }
}
int main(){
    get_pri();
    for(j=0,i=1;i≤cnt;++i)printf("%d ",pri[i]);
}
```

5. Min_25 筛法快速求素数和

> 不能算是Min_25筛，该模板只是在较短时间内求1e11~1e12内的素数和

```
#include <bits/stdc++.h>
#define ll long long
using namespace std;
const int N = 1000010;
int prime[N], id1[N], id2[N], flag[N], ncnt, m;
ll g[N], sum[N], a[N], T;
ll n;
int ID(ll x) {
    return x ≤ T ? id1[x] : id2[n / x];
}
ll calc(ll x) {
    return x * (x + 1) / 2 - 1;
}
ll f(ll x) {
    return x;
}
ll init(ll n){
    T = sqrt(n + 0.5);
    for (int i = 2; i ≤ T; i++) {
        if (!flag[i]) prime[++ncnt] = i, sum[ncnt] = sum[ncnt - 1] + i;
        for (int j = 1; j ≤ ncnt && i * prime[j] ≤ T; j++) {
            flag[i * prime[j]] = 1;
            if (i % prime[j] == 0) break;
        }
    }
    for (ll l = 1; l ≤ n; l = n / (n / l) + 1) {
        a[++m] = n / l;
```

```
        if (a[m] ≤ T) id1[a[m]] = m; else id2[n / a[m]] = m;
        g[m] = calc(a[m]);
    }
    for (int i = 1; i ≤ ncnt; i++)
        for (int j = 1; j ≤ m && (ll)prime[i] * prime[i] ≤ a[j]; j++)
            g[j] = g[j] - (ll)prime[i] * (g[ID(a[j] / prime[i])] - sum[i - 1]);
}
ll solve(ll x){
    if(x≤1){return x;}
    return n=x,init(n),g[ID(n)];
}
int main() {
    while(1)
    {
        memset(g,0,sizeof(g));
        memset(a,0,sizeof(a));
        memset(sum,0,sizeof(sum));
        memset(prime,0,sizeof(prime));
        memset(id1,0,sizeof(id1));
        memset(id2,0,sizeof(id2));
        memset(flag,0,sizeof(flag));
        ncnt=m=0;
        scanf("%lld", &n);
        printf("%lld\n", solve(n));
    }
}
```

$\oint$ 6. 约数

1.欧几里得 (gcd)

```
#include<algorithm>
#include<cstdio>
int a,b;
inline int gcd(Re a,Re b){return (!b)?a:(a%b==0?b:gcd(b,a%b));}
//来自某大佬无比魔性的诡异写法（至今未看懂）：
inline int GCD(int a,int b){while(b^=a^=b^=a%=b);return a;}
int main(){
    while(scanf("%d%d",&a,&b))
    printf("%d\n",gcd(a,b));
}
```

2.欧拉函数（埃氏筛）

```
#include<cstdio>
const int N=1e6;
int cnt,n=N,phi[N/3],pan[N],pri[N];
inline void get_phi(){
    pan[0]=pan[1]=1,phi[1]=1;
    for(int i=2;i≤n;i++)if(!phi[i]){
        for(int j=i;j≤n;j+=i){
            if(!phi[j])phi[j]=j;
            phi[j]=phi[j]/i*(i-1);
        }
    }
}
int main(){
```

```
    get_phi();
    for(int i=1;i≤n;++i)printf("%d: %d\n",i,phi[i]);
}
```

3.欧拉函数（欧拉筛）

```
#include<cstdio>
const int N=1e6;
int cnt,i,j,pan[N+1],pri[N/3],phi[N+1];
inline void get_phi(){
    pan[0]=pan[1]=1,phi[1]=1;
    for(i=2;i≤N;++i){
        if(!pan[i])pri[++cnt]=i,phi[i]=i-1;
        for(j=1;j≤cnt&&i*pri[j]≤N;++j){
            pan[i*pri[j]]=1;
            if(i%pri[j])phi[i*pri[j]]=phi[i]*(pri[j]-1);
            else{phi[i*pri[j]]=phi[i]*pri[j];break;}
        }
    }
}
int main(){
    get_phi();
    for(i=1;i≤N;++i)printf("%d: %d\n",i,phi[i]);
}
```

$\oint$ 7. 同余

1.逆元

【模板】乘法逆元

给定 n, p 求 $1 \sim n$ 中所有整数在模 p 意义下的乘法逆元。

(1).递推

```
#include<cstdio>
#define LL long long
#define Re register int
const int N=3e6+3;
int n,P,inv[N];
int main(){
    // freopen("123.txt","r",stdin);
    scanf("%d%d",&n,&P);
    inv[1]=1;
    for(Re i=2;i≤n;++i)inv[i]=(LL)(P-P/i)*inv[P%i]%P;
    for(Re i=1;i≤n;++i)printf("%d\n",inv[i]);
}
```

(2).快速幂

```
#include<cstdio>
#define LL long long
#define Re register int
int n,P;
inline int inv(Re x,Re P){
```

```
    Re s=1,k=P-2;
    while(k){
        if(k&1)s=(LL)s*x%P;
        x=(LL)x*x%P,k>>=1;
    }
    return s%P;
}
int main(){
    scanf("%d%d",&n,&P);
    for(Re i=1;i≤n;++i)printf("%d\n",inv(i,P));
}
```

3.扩展欧拉定理

【模板】扩展欧拉定理（abmodm）

给你三个正整数，a, m, b，你需要求：$a^b \bmod m$

```
#include<cstdio>
#include<cctype>
#define LL long long
#define Re register LL
LL i,a,b,m,x,s,phi,flag;char c;
inline LL cf(Re x,Re k){
    Re s=0;
    while(k){
        if(k&1)(s+=x)%=m;
        (x<<=1)%=m,k>>=1;
    }
    return s;
}
int main(){
    scanf("%lld%lld",&a,&m);phi=x=m;
    for(i=2;i*i≤m;++i)
        if(!(x%i)){phi-=phi/i;while(!(x%i))x/=i;}
    if(x>1)phi-=phi/x;
    while(!isdigit(c=getchar()));
    while(isdigit(c))flag|=((b=(b<<1)+(b<<3)+(c^48))≥phi),b%=phi,c=getchar();
    b+=flag?phi:0;x=1;
    while(b){if(b&1)x=cf(x,a)%m;a=cf(a,a)%m,b>>=1;}
    printf("%lld",x);
}
```

4.扩展欧几里得 (Exgcd)（同余方程）

【模板】同余方程

求关于x的同余方程 $ax \equiv 1 \pmod b$ 的最小正整数解。

```cpp
#include<cstdio>
#define Re register int
int a,b,x,y;
inline void exgcd(Re a,Re b,Re &x,Re &y){
    if(!b){x=1,y=0;return;}
    exgcd(b,a%b,x,y);Re x0=x,y0=y;
    x=y0,y=x0-a/b*y0;return;
}
int main(){
    scanf("%d%d",&a,&b),exgcd(a,b,x,y);
    printf("%d\n",(x%b+b)%b);
}
```

5.中国剩余定理 (CRT)

【模板】猜数字

现有两组数字，每组 k 个。

第一组中的数字分别用 $a_1, a_2, \cdots, a_k$ 表示，第二组中的数字分别用 $b_1, b_2, \cdots, b_k$ 表示。

其中第二组中的数字是两两互素的。求最小的 $n \in \mathbb{N}$，满足对于 $\forall i \in [1, k]$，有 $b_i | (n - a_i)$。

```cpp
#include<cstdio>
#define LL long long
#define Re register LL
const int N=13;
LL n,ans,lcm=1,a[N],m[N];
inline void in(Re &x){
    int f=0;x=0;char c=getchar();
    while(c<'0'||c>'9')f|=c=='-',c=getchar();
    while(c≥'0'&&c≤'9')x=(x<<1)+(x<<3)+(c^48),c=getchar();
    x=f?-x:x;
}
inline void exgcd(Re a,Re b,Re &x,Re &y){
    if(!b){x=1,y=0;return;}
    exgcd(b,a%b,x,y);Re x0=x,y0=y;
    x=y0,y=x0-a/b*y0;return;
}
inline LL cf(Re x,Re k,Re P){
    Re s=0;
    while(k){
        if(k&1)(s+=x)%=P;
        (x+=x)%=P,k>>=1;
    }
    return s;
}
int main(){
//    freopen("123.txt","r",stdin);
    in(n);
    for(Re i=1;i≤n;++i)in(m[i]),in(a[i]),lcm*=m[i];
    for(Re i=1;i≤n;++i){
        Re x,y,M=lcm/m[i];exgcd(M,m[i],x,y);//M[i]*inv+m[i]*y=1
        x=(x%m[i]+m[i])%m[i],(ans+=cf(cf(a[i],M,lcm),x,lcm))%=lcm;
    }
    printf("%lld\n",ans);
}
```

6.扩展中国剩余定理 (EXCRT)

【模板】扩展中国剩余定理（EXCRT）

给定 n 组非负整数 a_i, b_i ，求解关于 x 的方程组的最小非负整数解。

$$\begin{cases} x \equiv b_1 \pmod{a_1} \\ x \equiv b_2 \pmod{a_2} \\ \dots \\ x \equiv b_n \pmod{a_n} \end{cases}$$

```cpp
#include<cstdio>
#define LL long long
#define Re register LL
const int N=1e5+3;
LL n,ans,lcm=1,a[N],m[N];
inline void in(Re &x){
    int f=0;x=0;char c=getchar();
    while(c<'0'||c>'9')f|=c=='-',c=getchar();
    while(c≥'0'&&c≤'9')x=(x<<1)+(x<<3)+(c^48),c=getchar();
    x=f?-x:x;
}
inline LL exgcd(Re a,Re b,Re &x,Re &y){
    if(!b){x=1,y=0;return a;}
    Re gcd=exgcd(b,a%b,x,y);Re x0=x,y0=y;
    x=y0,y=x0-a/b*y0;return gcd;
}
inline LL cf(Re x,Re k,Re P){
    Re s=0;
    while(k){
        if(k&1)(s+=x)%=P;
        (x+=x)%=P,k>>=1;
    }
    return s;
}
int main(){
//    freopen("123.txt","r",stdin);
    in(n);
    for(Re i=1;i≤n;++i)in(m[i]),in(a[i]),lcm*=m[i];
    ans=a[1]%m[1],lcm=m[1];
    for(Re i=2;i≤n;++i){//ans+lcm*k=a[i](mod m[i]) → lcm*k=a[i]-ans(mod m[i]) → lcm*k+m[i]*y=a[i]-ans
        Re c=((a[i]-ans)%m[i]+m[i])%m[i],x,y;
        Re gcd=exgcd(lcm,m[i],x,y);
        Re k=cf(x,c/gcd,m[i]/gcd);
        ans+=lcm*k,lcm*=m[i]/gcd,ans=(ans%lcm+lcm)%lcm;
    }
    printf("%lld\n",ans);
}
```

7.拔山盖世 / 大步小步 / BSGS (Baby Steps Giant Steps)

给定一个质数 p，以及一个整数 b，一个整数 n，现在要求你计算一个最小的 l，满足 $b^l \equiv n \pmod{p}$ 。

```cpp
#include<algorithm>
#include<cstdio>
#include<cmath>
#include<map>
#define LL long long
#define Re register int
using namespace std;
const int N=1e5+3;
int x,y,m,P;map<int,int>B;
inline void in(Re &x){
    int f=0;x=0;char c=getchar();
    while(c<'0'||c>'9')f|=c=='-',c=getchar();
    while(c≥'0'&&c≤'9')x=(x<<1)+(x<<3)+(c^48),c=getchar();
    x=f?-x:x;
}
//x^k = y (mod P)
//k = am-b (m=sqrt(P)+1, a\in[1,m], b\in[0,m-1])
//(x^m)^a = y x^b (mod P)
inline int mi(Re x,Re k){
    Re s=1;
    while(k){
        if(k&1)s=(LL)s*x%P;
        x=(LL)x*x%P,k>>=1;
    }
    return s;
}
int main(){
//    freopen("123.txt","r",stdin);
    in(P),in(x),in(y),m=sqrt(P)+1;
    for(Re b=0,s=y;b<m;++b)B[s]=b,s=(LL)s*x%P;
    Re tmp=mi(x,m);
    for(Re a=1,s=1;a≤m;++a){
        s=(LL)s*tmp%P;
        if(B.find(s)≠B.end()){printf("%d\n",a*m-B[s]);return 0;}
    }
    puts("no solution");
}
```

8.扩展 BSGS (EXBSGS)

【模板】扩展 BSGS [P4195]

给定 a,p,b，求满足 $a^x \equiv b \pmod p$ 的最小自然数 x 。

```cpp
int a,p,b;
int power(int a,ll b,int modd){
    int c=1;
    for(;b;b>>=1){
    if(b&1) c=(ll) c*a%modd;
    a=(ll)a*a%modd;
    }
    return c;
}
int BSGS(int a,int b,int p)//p 是素数
{
    map<int ,int>hash;
    hash.clear();
    b%=p;
    int t=(int)sqrt(p)+1;
```

```
    for(int j=0;j<t;j++){
        int val =(ll)b*power(a,j,p)%p;
        hash[val]=j;
        }
        a=power(a,t,p);
        if(a==0) return b==0 ?1:-1;
        for(int i=0;i≤t;i++)
        {
            int val=power(a,i,p);
            int j=hash.find(val)==hash.end()?-1:hash[val];
            if(j≥0&&i*t-j≥0) return i*t-j;
        }
        return -1;
}

int ex_BSGS(int A,int B,int C)//C 不是素数
{
    map<int ,int>mp;
    if(B==1) return 0;
    int k=0,tmp=1,d;
    while(1)
    {
        d=__gcd(A,C);
        if(d==1) break;
        if(B%d) return -1;
        B/=d; C/=d;
        tmp=1LL*tmp*(A/d)%C;
        k++;
        if(tmp==B) return k;
    }
    mp.clear();
    int mul=B;
    mp[B]=0;
    int m=ceil(sqrt(1.0*C));
    for(int j=1;j≤m;++j)
    {
        mul=1LL*mul*A%C;
        mp[mul]=j;
    }
    int am=power(A,m,C);
    mul=tmp;
    for(int j=1;j≤m;++j)
    {
        mul=1LL*mul*am%C;
        if(mp.count(mul)) return j*m-mp[mul]+k;
    }
    return -1;
}
int main()
{
    while(scanf("%d %d %d",&a,&p,&b)≠EOF)
    {
        if(a==0&&b==0&&a==p) return 0;
        int flag=ex_BSGS(a,b,p);
        if(flag==-1) cout<<"No Solution"<<endl;
        else cout<<flag<<endl;
    }
    return 0;
}

```

9.二次剩余

求解方程 $x^2 \equiv N (\mathrm{mod}\ p)$

【模板】二次剩余 多组数据。

保证p是奇素数

[P5491]

```
#include<algorithm>
#include<cstring>
#include<cstdio>
#define LL long long
#define Re register int
using namespace std;
int n,T,P,W,det;
inline void in(Re &x){
    int f=0;x=0;char c=getchar();
    while(c<'0'||c>'9')f|=c=='-',c=getchar();
    while(c≥'0'&&c≤'9')x=(x<<1)+(x<<3)+(c^48),c=getchar();
    x=f?-x:x;
}
inline int mi(Re x,Re k){
    Re s=1;
    while(k){
        if(k&1)s=(LL)s*x%P;
        x=(LL)x*x%P,k>>=1;
    }
    return s;
}
struct CP{
    int x,y;CP(Re X=0,Re Y=0){x=X,y=Y;}
    inline CP operator*(const CP &O)const{return CP(((LL)x*O.x%P+(LL)y*O.y%P*W%P)%P,((LL)x*O.y%P+
(LL)y*O.x%P)%P);}
    inline CP operator*=(const CP &O){return *this=*this*O;}
};
inline CP mi_(CP x,Re k){
    CP s=CP(1,0);
    while(k){
        if(k&1)s*=x;
        x*=x,k>>=1;
    }
    return s;
}
inline LL Sqrt(Re n){
    if(!n)return 0;Re x;
    while(1){
        x=rand()%P,W=((LL)x*x%P-n+P)%P;
        if(mi(W,(P-1)/2)==P-1)break;
    }
    return mi_(CP(x,1),(P+1)/2).x;
}
int main(){
//    freopen("123.txt","r",stdin);
    in(T);
    while(T--){
        in(n),in(P);
        if(mi(n,(P-1)/2)==P-1)puts("Hola!");//无解
```

```
49          else{
50              det=Sqrt(n);
51              if(det)printf("%d %d\n",min(det,P-det),max(det,P-det));//两解
52              else printf("%d\n",det);//单解
53          }
54      }
55  }
```

10.N 次剩余

【模板】N 次剩余 [P5668]

你需要解方程 $x^n \equiv k \pmod m$，其中 $x \in [0, m-1]$。

```cpp
#include <bits/stdc++.h>

typedef long long LL;

int A, B, mod;
int pow(int x, int y, int mod = 0, int ans = 1) {
    if (mod) {
        for (; y; y >>= 1, x = (LL) x * x % mod)
            if (y & 1) ans = (LL) ans * x % mod;
    } else {
        for (; y; y >>= 1, x = x * x)
            if (y & 1) ans = ans * x;
    }
    return ans;
}

struct factor {
    int prime[20], expo[20], pk[20], tot;
    void factor_integer(int n) {
        tot = 0;
        for (int i = 2; i * i ≤ n; ++i) if (n % i == 0) {
            prime[tot] = i, expo[tot] = 0, pk[tot] = 1;
            do ++expo[tot], pk[tot] *= i; while ((n /= i) % i == 0);
            ++tot;
        }
        if (n > 1) prime[tot] = n, expo[tot] = 1, pk[tot++] = n;
    }
    int phi(int id) const {
        return pk[id] / prime[id] * (prime[id] - 1);
    }
} mods, _p;

int p_inverse(int x, int id) {
    assert(x % mods.prime[id] ≠ 0);
    return pow(x, mods.phi(id) - 1, mods.pk[id]);
}

void exgcd(int a, int b, int &x, int &y) {
    if (!b) x = 1, y = 0;
    else exgcd(b, a % b, y, x), y -= a / b * x;
}
int inverse(int x, int mod) {
    assert(std::__gcd(x, mod) == 1);
    int ret, tmp;
    exgcd(x, mod, ret, tmp), ret %= mod;
```

```
    return ret + (ret >> 31 & mod);
}

std::vector<int> sol[20];

void solve_2(int id, int k) {
    int mod = 1 << k;
    if (k == 0) { sol[id].emplace_back(0); return; }
    else {
        solve_2(id, k - 1); std::vector<int> t;
        for (int s : sol[id]) {
            if (!((pow(s, A) ^ B) & mod - 1))
                t.emplace_back(s);
            if (!((pow(s | 1 << k - 1, A) ^ B) & mod - 1))
                t.emplace_back(s | 1 << k - 1);
        }
        std::swap(sol[id], t);
    }
}

int BSGS(int B, int g, int mod) { // g^x = B (mod M) ⟹ g^iL = B*g^j (mod M) : iL - j
    std::unordered_map<int, int> map;
    int L = std::ceil(std::sqrt(mod)), t = 1;
    for (int i = 1; i ≤ L; ++i) {
        t = (LL) t * g % mod;
        map[(LL) B * t % mod] = i;
    }
    int now = 1;
    for (int i = 1; i ≤ L; ++i) {
        now = (LL) now * t % mod;
        if (map.count(now)) return i * L - map[now];
    }
    assert(0);
}

int find_primitive_root(int id) {
    int phi = mods.phi(id); _p.factor_integer(phi);
    auto check = [&] (int g) {
        for (int i = 0; i < _p.tot; ++i)
            if (pow(g, phi / _p.prime[i], mods.pk[id]) == 1)
                return 0;
        return 1;
    };
    for (int g = 2; g < mods.pk[id]; ++g) if (check(g)) return g;
    assert(0);
}

void division(int id, int a, int b, int mod) { //  ax = b (mod M)
    int M = mod, g = std::__gcd(std::__gcd(a, b), mod);
    a ⊨ g, b ⊨ g, mod ⊨ g;
    if (std::__gcd(a, mod) > 1) return;
    int t = (LL) b * inverse(a, mod) % mod;
    for (; t < M; t += mod) sol[id].emplace_back(t);
}

void solve_p(int id, int B = ::B) {
    int p = mods.prime[id], e = mods.expo[id], mod = mods.pk[id];
    if (B % mod == 0) {
        int q = pow(p, (e + A - 1) / A);
        for (int t = 0; t < mods.pk[id]; t += q)
```

```
            sol[id].emplace_back(t);
    } else if (B % p != 0) {
        int phi = mods.phi(id);
        int g = find_primitive_root(id), z = BSGS(B, g, mod);
        division(id, A, z, phi);
        for (int &x : sol[id]) x = pow(g, x, mod);
    } else {
        int q = 0; while (B % p == 0) B /= p, ++q;
        int pq = pow(p, q);
        if (q % A != 0) return;
        mods.expo[id] -= q, mods.pk[id] /= pq;
        solve_p(id, B);
        mods.expo[id] += q, mods.pk[id] *= pq;
        if (!sol[id].size()) return;

        int s = pow(p, q - q / A);
        int t = pow(p, q / A);
        int u = pow(p, e - q);

        std::vector<int> res;
        for (int y : sol[id]) {
            for (int i = 0; i < s; ++i)
                res.emplace_back((i * u + y) * t);
        }
        std::swap(sol[id], res);
    }
}

std::vector<int> allans;
void dfs(int dep, int ans, int mod) {
    if (dep == mods.tot) {allans.emplace_back(ans); return;}
    int p = mods.pk[dep], k = p_inverse(mod % p, dep);
    for (int a : sol[dep]) {
        int nxt = (LL) (a - ans % p + p) * k % p * mod + ans;
        dfs(dep + 1, nxt, mod * p);
    }
}

void solve() {
    std::cin >> A >> mod >> B, mods.factor_integer(mod);
    allans.clear();
    for (int i = mods.tot - 1; ~i; --i) {
        sol[i].clear();
        mods.prime[i] == 2 ? solve_2(i, mods.expo[i]) : solve_p(i);
        if (!sol[i].size()) {return std::cout << 0 << '\n', void(0);}
    }
    dfs(0, 0, 1), std::sort(allans.begin(), allans.end());
    std::cout << allans.size() << '\n';
    for (int i : allans) std::cout << i << ' '; std::cout << '\n';
}

int main() {
    std::ios::sync_with_stdio(0), std::cin.tie(0);
    int tc; std::cin >> tc; while (tc--) solve();
    return 0;
}
```

$\oint$ 8. 类欧几里得算法

【模板】类欧几里得算法 [P5170]

给定 n, a, b, c，分别求 $\sum_{i=0}^{n}\lfloor\frac{ai+b}{c}\rfloor$，$\sum_{i=0}^{n}\lfloor\frac{ai+b}{c}\rfloor^2$，$\sum_{i=0}^{n}i\lfloor\frac{ai+b}{c}\rfloor$，答案对 998244353 取模。多组数据。

```cpp
#include <bits/stdc++.h>
using namespace std;

#define Int register int
#define mod 998244353ll
#define int long long

int inv2 = 499122177ll,inv6 = 166374059ll;

struct Ans{int f,g,h;};

Ans Solve (int a,int b,int c,int n)
{
    if (!a)
    {
        int f = (n + 1) * (b / c) % mod;
        int g = (n + 1) * (b / c) % mod * (b / c) % mod;
        int h = n * (n + 1) % mod * inv2 % mod * (b / c) % mod;
        return Ans {f % mod,g % mod,h % mod};
    }
    else if (a ≥ c || b ≥ c)
    {
        Ans fucker = Solve (a % c,b % c,c,n);
        int F = fucker.f + n * (n + 1) / 2 % mod * (a / c) % mod + (n + 1) * (b / c) % mod;
        int G = fucker.g + 2 * (a / c) % mod * fucker.h % mod + 2 * (b / c) % mod * fucker.f +
        n % mod * (n + 1) % mod * (2 * n % mod + 1) % mod * inv6 % mod * (a / c) % mod * (a / c) % mod +
        n % mod * (n + 1) % mod * (a / c) % mod * (b / c) % mod + (n + 1) * (b / c) % mod * (b / c) %
mod;
        int H = fucker.h + n % mod * (n + 1) % mod * (2 * n + 1) % mod * inv6 % mod * (a / c) % mod + n
% mod * (n + 1) % mod * inv2 % mod * (b / c) % mod;
        return Ans {F % mod,G % mod,H % mod};
    }
    else
    {
        int M = (a * n + b) / c;
        Ans fucker = Solve (c,c - b - 1,a,M - 1);
        int F = n * M % mod - fucker.f;
        int G = n * M % mod * (M + 1) % mod - 2 * fucker.h % mod + mod - 2 * fucker.f % mod + mod - F %
mod;
        int H = (M * n % mod * (n + 1) % mod - fucker.g + mod - fucker.f) % mod * inv2 % mod;
        return Ans {F % mod,G % mod,H % mod};
    }
}

int read ()
{
    int x = 0;char c = getchar();int f = 1;
    while (c < '0' || c > '9'){if (c == '-') f = -f;c = getchar();}
    while (c ≥ '0' && c ≤ '9'){x = (x << 3) + (x << 1) + c - '0';c = getchar();}
    return x * f;
}

void write (int x)
{
```

```
    if (x < 0){x = -x;putchar ('-');}
    if (x > 9) write (x / 10);
    putchar (x % 10 + '0');
}

signed main()
{
    int times = read ();
    while (times --)
    {
        int n = read (),a = read (),b = read (),c = read ();
        Ans Putout = Solve (a,b,c,n);
        write ((Putout.f + mod) % mod),putchar (' '),write ((Putout.g + mod) % mod),putchar (' '),write ((Putout.h + mod) % mod),putchar ('\n');
    }
    return 0;
}
```

$\oint$ 9. 各类反演及数论筛法

1.莫比乌斯函数(Mobius)

(1).【埃氏筛】

```
#include<cstdio>
const int N=1e7+3;
int i,j,n,cnt,pan[N],miu[N];
inline void get_miu(){
    for(i=1;i≤n;++i)miu[i]=1;
    for(i=2;i≤n;++i)
        if(!pan[i]){
            miu[i]=-1;
            for(j=i<<1;j≤n;j+=i)pan[j]=1,miu[j]*=(j/i%i==0)?0:-1;
        }
}
int main(){
    n=N;
    get_miu();
    for(i=1;i≤n;++i)printf("%d: %d\n",i,miu[i]);
}
```

(2).【线性筛】

```
#include<cstdio>
const int N=1e7+3;
int i,j,n,cnt,pri[N>>1],pan[N],miu[N];
inline void get_miu(){
    pan[1]=1,miu[1]=1;
    for(i=2;i≤n;++i){
        if(!pan[i])pri[++cnt]=i,miu[i]=-1;
        for(j=1;j≤cnt&&i*pri[j]≤n;++j){
            pan[i*pri[j]]=1;
            if(i%pri[j])miu[i*pri[j]]=-miu[i];
            else{miu[i*pri[j]]=0;break;}
        }
    }
}
int main(){
    n=N;
```

```
    get_miu();
    for(i=1;i≤n;++i)printf("%d: %d\n",i,miu[i]);
}
```

2.杜教筛

(1).map 记忆化

【模板】 杜教筛（Sum）

给定一个正整数，求
$ans_1 = \sum_{i=1}^{n} \varphi(i)$
$ans_2 = \sum_{i=1}^{n} \mu(i)$

```
#include<algorithm>
#include<cstring>
#include<cstdio>
#include<cmath>
#include<map>
#define Re register int
#define LL long long
using namespace std;
const int N23=1664511+3,N=2147483647;
int x,T,cnt,pan[N23],pri[N23],phi[N23],miu[N23];
int TO,Smiu[N23];LL Sphi[N23];
map<int,int>Smiu_;map<int,LL>Sphi_;
inline void in(Re &x){
    Re fu=0;x=0;char ch=getchar();
    while(ch<'0'||ch>'9')fu|=ch=='-',ch=getchar();
    while(ch≥'0'&&ch≤'9')x=(x<<1)+(x<<3)+(ch^48),ch=getchar();
    x=fu?-x:x;
}
inline void get_(Re N){
    miu[1]=phi[1]=pan[1]=1;
    for(Re i=2;i≤N;++i){
        if(!pan[i])pri[++cnt]=i,miu[i]=-1,phi[i]=i-1;
        for(Re j=1;j≤cnt&&pri[j]≤N/i;++j){
            pan[i*pri[j]]=1;
            if(i%pri[j])miu[i*pri[j]]=-miu[i],phi[i*pri[j]]=phi[i]*phi[pri[j]];
            else{miu[i*pri[j]]=0,phi[i*pri[j]]=phi[i]*pri[j];break;}
        }
    }
    for(Re i=1;i≤N;++i)Smiu[i]=Smiu[i-1]+miu[i],Sphi[i]=Sphi[i-1]+phi[i];
}
inline LL get_Sphi(Re n){
    if(n≤TO)return Sphi[n];
    if(Sphi_[n])return Sphi_[n];
    LL ans=(LL)n*(n+1)>>1;
    for(Re l=2,r;l≤n;l=r+1){
        r=n/(n/l);
        ans-=(LL)(r-l+1)*get_Sphi(n/l);
    }
    return Sphi_[n]=ans;
}
inline int get_Smiu(Re n){
    if(n≤TO)return Smiu[n];
    if(Smiu_[n])return Smiu_[n];
    Re ans=1;
    for(Re l=2,r;l≤n;l=r+1){
```

```
        r=n/(n/l);
        ans-=(r-l+1)*get_Smiu(n/l);
    }
    return Smiu_[n]=ans;
}
int main(){
//    freopen("123.txt","r",stdin);
    in(T),get_(TO=N23-3);
    while(T--)in(x),printf("%lld %d\n",get_Sphi(x),get_Smiu(x));
}
```

(2).数组记忆化

【模板】 杜教筛（Sum）[P4213]

```
#include<algorithm>
#include<cstring>
#include<cstdio>
#include<cmath>
#define Re register int
#define LL long long
using namespace std;
const int N23=1664511+3,N13=1303,N=2147483647;
int x,T,cnt,pan[N23],pri[N23],phi[N23],miu[N23];
int TO,vis1[N13],vis2[N13],Smiu[N23],Smiu_[N13];
LL Sphi[N23],Sphi_[N13];
inline void in(Re &x){
    Re fu=0;x=0;char ch=getchar();
    while(ch<'0'||ch>'9')fu|=ch=='-',ch=getchar();
    while(ch≥'0'&&ch≤'9')x=(x<<1)+(x<<3)+(ch^48),ch=getchar();
    x=fu?-x:x;
}
inline void get_(Re N){
    miu[1]=phi[1]=pan[1]=1;
    for(Re i=2;i≤N;++i){
        if(!pan[i])pri[++cnt]=i,miu[i]=-1,phi[i]=i-1;
        for(Re j=1;j≤cnt&&pri[j]≤N/i;++j){
            pan[i*pri[j]]=1;
            if(i%pri[j])miu[i*pri[j]]=-miu[i],phi[i*pri[j]]=phi[i]*phi[pri[j]];
            else{miu[i*pri[j]]=0,phi[i*pri[j]]=phi[i]*pri[j];break;}
        }
    }
    for(Re i=1;i≤N;++i)Smiu[i]=Smiu[i-1]+miu[i],Sphi[i]=Sphi[i-1]+phi[i];
}
inline LL get_Sphi(Re n){
    if(n≤TO)return Sphi[n];
    Re nn=N/n;
    if(vis1[nn])return Sphi_[nn];
    LL ans=(LL)n*(n+1)>>1;
    for(Re l=2,r;l≤n;l=r+1){
        r=n/(n/l);
        ans-=(LL)(r-l+1)*get_Sphi(n/l);
    }
    vis1[nn]=1;
    return Sphi_[nn]=ans;
}
inline int get_Smiu(Re n){
    if(n≤TO)return Smiu[n];
    Re nn=N/n;
    if(vis2[nn])return Smiu_[nn];
```

```
    Re ans=1;
    for(Re l=2,r;l≤n;l=r+1){
        r=n/(n/l);
        ans-=(r-l+1)*get_Smiu(n/l);
    }
    vis2[nn]=1;
    return Smiu_[nn]=ans;
}
int main(){
//    freopen("123.txt","r",stdin);
    in(T),get_(TO=N23-3);
    while(T--){
        in(x),printf("%lld %d\n",get_Sphi(x),get_Smiu(x));
        memset(vis1,0,sizeof(vis1));//注意对于每次询问的n，n/i都不同
        memset(vis2,0,sizeof(vis2));//所以要清空
    }
}
```

3.最值反演 (Min-Max容斥)

【模板】 Card Collector [Hdu4336]

> 题意：有n种卡片，可以通过买干脆面收集卡片，每包干脆面最多一张卡片，问收集完n种卡片时买的干脆面包数的期望。

```
#include<algorithm>
#include<cstring>
#include<cstdio>
#define LD double
#define LL long long
#define Re register int
using namespace std;
const int N=23,M=1048576+3;
int n,V,cnt[M];LD ans,p[N],Min[M];
inline void in(Re &x){
    int f=0;x=0;char ch=getchar();
    while(ch<'0'||ch>'9')f|=ch=='-',ch=getchar();
    while(ch≥'0'&&ch≤'9')x=(x<<1)+(x<<3)+(ch^48),ch=getchar();
    x=f?-x:x;
}
int main(){
//    freopen("123.txt","r",stdin);
    while(~scanf("%d",&n)){
        V=(1<<n)-1,ans=0;
        for(Re i=1;i≤n;++i)scanf("%lf",&p[i]);
        for(Re s=1;s≤V;++s){
            Min[s]=0,cnt[s]=cnt[s>>1]+(s&1);
            for(Re i=1;i≤n;++i)if(s&(1<<i-1))Min[s]+=p[i];
            Min[s]=1.0/Min[s];
        }
        for(Re t=1;t≤V;++t)ans+=(cnt[t]&1)?Min[t]:-Min[t];
        printf("%lf\n",ans);
    }
}
```

八、计算几何

$\oint$ 1. 计算几何注意事项

在写计算几何的题目的时候一般习惯数组从0开始（至少我的大多数模板都是从0开始的，所以注意一下）

1.1 计算误差

- 尽量少用三角函数、除法、开方、求幂、取对数运算
  如：$1.0/2.0*3.0/4.0*5.0=(1.0*3.0*5.0)/(2.0*4.0)$
- 在不溢出的情况下将除式比较转化为乘式比较
  $ab>c\Leftrightarrow a>bc$

在使用除法、开根号、和三角函数的时候，我们要考虑由浮点误差运算产生的代价

1.2 解决方案：误差判别法

```
const double eps = 1e-9;
int dcmp(double x, double y){
    if(fabs(x - y) < eps)
        return 0;
    if(x > y)
        return 1;
    return -1;
}
```

1.3 解决方案：化浮为整

在不溢出整数范围的情况下，可以通过乘上10^k转化为整数运算，最后再将结果转化为浮点数输出

1.4 注意负零

输出时一定要小心不要输出 -0，比如

```
double a = -0.000001;
printf("%.4f", a);
```

1.5 注意反三角函数的值域

使用反三角函数时，要注意定义域的范围，比如，经过计算 x = 1.000001

```
double x = 1.000001;
double acx = acos(x);
//可能会返回runtime error（cos值域是-1到1）

//此时我们可以加一句判断
double x = 1.000001;
if(fabs(x - 1.0) < eps || fabs(x + 1.0) < eps)//如果x等于+1/-1就四舍五入
    x = round(x);
double acx = acos(x);
```

1.6 计算几何常用开头模板

```
#include <cstdio>
#include <cmath>

using namespace std;

const double pi = acos(-1.0);
const double inf = 1e100;
```

```cpp
const double eps = 1e-8;//一般出题人比较喜欢用这个数
int sgn(double d){
    if(fabs(d) < eps)
        return 0;
    if(d > 0)
        return 1;
    return -1;
}
int dcmp(double x, double y){
    if(fabs(x - y) < eps)
        return 0;
    if(x > y)
        return 1;
    return -1;
}
int main() {
    double x = 1.49999;
    int fx = floor(x);//向下取整函数
    int cx = ceil(x);//向上取整函数
    int rx = round(x);//四舍五入函数
    printf("%f %d %d %d\n", x, fx, cx, rx);
    //输出结果 1.499990 1 2 1
    return  0 ;
}
```

$\oint$ 2. 注意事项

1. 注意舍入方式(0.5的舍入方向);防止输出-0.
2. 几何题注意多测试不对称数据.
3. 整数几何注意xmult和dmult是否会出界; 符点几何注意eps的使用.
4. 避免使用斜率;注意除数是否会为0.
5. 公式一定要化简后再代入.
6. 判断同一个$2 * PI$域内

两角度差应该是

$$abs(a1 - a2) < beta||abs(a1 - a2) > pi + pi - beta;$$

相等应该是

$$abs(a1 - a2) < eps||abs(a1 - a2) > pi + pi - eps;$$

7. 需要的话尽量使用atan2,注意:
8. $atan2(0, 0) = 0$
   $atan2(1, 0) = pi/2$
   $atan2(-1, 0) = -pi/2$
   $atan2(0, 1) = 0$
   $atan2(0, -1) = pi.$
9. $crossproduct = |u| * |v| * sin(a)$
   $dotproduct = |u| * |v| * cos(a)$
10. $(P1 - P0)x(P2 - P0)$结果的意义:
    正: $< P0, P1 >$在$< P0, P2 >$顺时针$(0, pi)$内
    负: $< P0, P1 >$在$< P0, P2 >$逆时针$(0, pi)$内
    0 : $< P0, P1 >$,$< P0, P2 >$共线,夹角为0或pi
11. 误差限缺省使用$1e - 8$!

$\oint$ 3. 几何公式

余弦定理

$cosA = \frac{(b^2+c^2-a^2)}{2bc}$

1.三角形

13. 半周长 $P = (a+b+c)/2$
14. 面积 $S = aHa/2 = absin(C)/2 = sqrt(P(P-a)(P-b)(P-c))$
15. 中线 $Ma = sqrt(2(b^2+c^2)-a^2)/2 = sqrt(b^2+c^2+2bccos(A))/2$
16. 角平分线 $Ta = sqrt(bc((b+c)^2-a^2))/(b+c) = 2bccos(A/2)/(b+c)$
17. 高线 $Ha = bsin(C) = csin(B) = sqrt(b^2-((a^2+b^2-c^2)/(2a))^2)$
18. 内切圆半径 $r = S/P = asin(B/2)sin(C/2)/sin((B+C)/2)$

$= 4Rsin(A/2)sin(B/2)sin(C/2)$
$= sqrt((P-a)(P-b)(P-c)/P)$

$= Ptan(A/2)tan(B/2)tan(C/2)$

19. 外接圆半径 $R = abc/(4S) = a/(2sin(A)) = b/(2sin(B)) = c/(2sin(C))$

20. 海伦公式

假设在平面内，有一个三角形，边长分别为a、b、c，三角形的面积S可由以下公式求得：
$S = sqrt(s1*(s1-a)*(s1-b)*(s1-c))$，其中s1表示的是三角形的周长的一半。

2.四边形

D1,D2为对角线,M对角线中点连线,A为对角线夹角

20. $a^2+b^2+c^2+d^2 = D1^2+D2^2+4M^2$
21. $S = D1D2sin(A)/2$
    (以下对圆的内接四边形)
22. $ac+bd = D1D2$
23. $S = sqrt((P-a)(P-b)(P-c)(P-d))$,P为半周长

3.正n边形

R为外接圆半径,r为内切圆半径

24. 中心角 $A = 2PI/n$
25. 内角 $C = (n-2)PI/n$
26. 边长 $a = 2sqrt(R^2-r^2) = 2Rsin(A/2) = 2rtan(A/2)$
27. 面积 $S = nar/2 = nr^2tan(A/2) = nR^2sin(A)/2 = na^2/(4tan(A/2))$

4.圆

28. 弧长 $l = rA$
29. 弦长 $a = 2sqrt(2hr-h^2) = 2rsin(A/2)$
30. 弓形高 $h = r-sqrt(r^2-a^2/4) = r(1-cos(A/2)) = atan(A/4)/2$
31. 扇形面积 $S1 = rl/2 = r^2A/2$
32. 弓形面积 $S2 = (rl-a(r-h))/2 = r^2(A-sin(A))/2$
33. 已知三边和内切圆或外切圆求面积

$S = abc/(4R)$ 其中R是三角形外接圆半径

$S = (a+b+c)*r/2$ (r是三角形内切圆半径)

34. 已知四面体顶点坐标，求内切圆圆心。

ansx=(Sabca[4].x+Sabda[3].x + Sacda[2].x+Sbcda[1].x)/S;
ansy=(Sabca[4].y+Sabda[3].y + Sacda[2].y+Sbcda[1].y)/S;
ansz=(Sabca[4].z+Sabda[3].z + Sacda[2].z+Sbcda[1].z)/S;
其中S为表面积。

5.棱柱

35. 体积 $V = Ah$,A为底面积,h为高
36. 侧面积 $S = lp$,l为棱长,p为直截面周长
37. 全面积 $T = S + 2A$

6.棱锥

38. 体积 $V = Ah/3$,A为底面积,h为高
    (以下对正棱锥)
39. 侧面积 $S = lp/2$,l为斜高,p为底面周长
40. 全面积 $T = S + A$

7.棱台

39. 体积 $V = (A1 + A2 + sqrt(A1A2))h/3, A1.A2$为上下底面积,h为高(以下为正棱台)
40. 侧面积 $S = (p1 + p2)l/2, p1.p2$为上下底面周长,l为斜高
41. 全面积 $T = S + A1 + A2$

8.圆柱

42. 侧面积 $S = 2PIrh$
43. 全面积 $T = 2PIr(h + r)$
44. 体积 $V = PIr^2h$

9.圆锥

45. 母线 $l = sqrt(h^2 + r^2)$
46. 侧面积 $S = PIrl$
47. 全面积 $T = PIr(l + r)$
48. 体积 $V = PIr^2h/3$
49. 已知圆锥表面积S求最大体积V

$V = S * sqrt(S/(72 * Pi))$

10.圆台

50. 母线 $l = sqrt(h^2 + (r1 - r2)^2)$
51. 侧面积 $S = PI(r1 + r2)l$
52. 全面积 $T = PIr1(l + r1) + PIr2(l + r2)$
53. 体积 $V = PI(r1^2 + r2^2 + r1r2)h/3$

11.球

54. 全面积 $T = 4PIr^2$
55. 体积 $V = 4PIr^3/3$

12.球台

56. 侧面积 $S = 2PIrh$
57. 全面积 $T = PI(2rh + r1^2 + r2^2)$
58. 体积 $V = PIh(3(r1^2 + r2^2) + h^2)/6$

13.球扇形

59. 全面积 $T = PIr(2h + r0)$,h为球冠高,r0为球冠底面半径
60. 体积 $V = 2PIr^2h/3$

```cpp
inline double R_to_D(double rad)//弧度转角度
{ return 180/Pi*rad; }
inline double D_to_R(double D)//角度转弧度
{ return Pi/180*D; }
```

需要注意的是:点加减向量为点、点减点为向量

点加减向量为点:
实例:
我们根据一长方形个中心坐标求得长方形四个端点的坐标。
每行5个实数 x，y，w，h，j，其中（x，y）是木板（长方形）中心的坐标，w是宽，h是高，j 是顺时针旋转的角度（j = 0表示不旋转，此时长度为w的那条边应该是水平方向）。

枚举四个方向，分别将对应的向量（$\pm\frac{s1}{2}$，$\pm\frac{s2}{2}$进行旋转，然后加上原先点的坐标即可（$s1, s2$分别为长和宽）。

```cpp
    Point o(x, y);//!
//要注意看题，这里给的角度j是顺时针的角度，然而我们的Rotate函数是取的逆时针的角度，所以一定要取相反数
    deg = -D_to_R(j);//角度转换成弧度，还要记得取相反数
    p[tot ++ ] = o + Rotate(Vector(-w / 2, -h / 2), deg);
    p[tot ++ ] = o + Rotate(Vector(w / 2, -h / 2), deg);
    p[tot ++ ] = o + Rotate(Vector(-w / 2, h / 2), deg);
    p[tot ++ ] = o + Rotate(Vector(w / 2, h / 2), deg);
```

$\oint$ 二维几何基础

$\oint$ 1. 基本运算

旋转θ角时坐标变换
$x' = xcos\theta - ysin\theta$
$y' = xsin\theta + ycos\theta$

```cpp
#include<bits/stdc++.h>
using namespace std;

const int N = 5007, M = 50007, INF = 0x3f3f3f3f;
const double DINF = 12345678910, eps = 1e-10;
struct Point{
    double x, y;
    Point(double x = 0, double y = 0):x(x), y(y){ }//构造函数
};

//!注意区分点和向量
typedef Point Vector;
//!向量 + 向量 = 向量，点 + 向量 = 向量
Vector operator + (Vector A, Vector B){return Vector(A.x + B.x, A.y + B.y);}
//!点 - 点 = 向量(向量BC = C - B)
Vector operator - (Point A, Point B){return Vector(A.x - B.x, A.y - B.y);}
//!向量 * 数 = 向量
Vector operator * (Vector A, double p){return Vector(A.x * p, A.y * p);}
//!向量 / 数= 向量
Vector operator / (Vector A, double p){return Vector(A.x / p, A.y / p);}

//!点/向量的比较函数
bool operator < (const Point& a, const Point& b) {return a.x < b.x || (a.x == b.x && a.y < b. y);}
//!三态函数dcmp用于判断相等，减少精度问题

int dcmp(double x){
    if(fabs(x) < eps) return 0;
```

```
    else return x < 0 ? -1 : 1;
}
//重载等于运算符
bool operator == (const Point& a, const Point& b){return !dcmp(a.x - b.x) && !dcmp(a.y - b.y);}

//!求极角
//单位弧度rad
double Polar_angle(Vector A){return atan2(A.y, A.x);}
```

1.1 判断正负函数(sgn)

```
int sgn(double x){
    if(fabs(x) < eps)
        return 0;
    if(x < 0)
        return -1;
    return 1;
}
```

传统意义	修正写法1	修正写法2
a==b	sgn(a-b) ==0	fabs(a -b) <eps
a!=b	sgn(a-b) !=0	fabs(a-b)>eps
a<b	sgn(a-b)<0	a-b<-eps
a<=b	sgn(a-b)<=0	a-b<eps
a>b	sgn(a-b)>0	a-b>eps
a>=b	sgn(a-b)>=0	a-b>-eps

1.2 点积（数量积、内积）(Dot)

α·β=|α||β|cosθ
对加法满足分配律(满足交换律)

```
//!点积(满足交换律)
double Dot(Vector A, Vector B){return A.x * B.x + A.y * B.y;}
```

1.3 向量积，叉积(Cross)

α×β=|α||β|sinθ
θ表示向量α旋转到向量β所经过的夹角

对加法满足分配律(不满足交换律)

几何意义

向量α与β所张成的平行四边形的有向面积
判断外积的符号

右手定则

α×β
若β在α的逆时针方向，则为正值、顺时针则为负值、两向量共线则为0

```
//!向量的叉积(不满足交换律)
//等于两向量有向面积的二倍(从v的方向看,w左,叉积>0,w右,叉积<0,共线,叉积=0)
//cross(x, y) = -cross(y, x)
//cross(x, y) : xAyB - xByA
double Cross(Vector A, Vector B){return A.x *  B.y - B.x * A.y;}
```

1.4 取模（求长度）(Length)

```
double Length(Vector A){return sqrt(Dot(A, A));}
```

1.5 计算两向量夹角(Angle)

返回值为弧度制下的夹角

```
double Angle(Vector A, Vector B){return acos(Dot(A, B) / Length(A) / Length(B));}
```

1.6 计算两向量构成的平行四边形有向面积(Area2)

```
//!三个点确定两个向量,(交点为A的两个向量AB和AC)然后求这两个向量的叉积(叉乘)
double Area2(Point A, Point B, Point C){return Cross(B - A, C - A);}
```

1.7 计算向量逆时针旋转后的向量(Rotate)

```
//!一个向量旋转rad弧度之后的新向量
//!x' = xcosa - ysina, y' = xsina + ycosa
//rad:弧度 且为逆时针旋转的角
Vector Rotate(Vector A, double rad){
    return Vector(A.x * cos(rad) - A.y * sin(rad), A.x * sin(rad) + A.y * cos(rad));
}
```

1.8 计算向量逆时针旋转九十度的单位法线(Normal)

```
//!向量的单位法线实际上就是将该向量向左旋转90°
//因为是单位法线所以长度归一化调用前请确保A不是零向量
Vector Normal(Vector A){
    double L = Length(A);
    return Vector(-A.y / L, A.x / L);
}
```

```
inline Vector Format(const Vector &A)
{
    double L=Length(A);
    return Vector(A.x/L,A.y/L);
}
```

1.9 判断折线（向量）bc是不是向（向量）ab的逆时针方向（左边）转向(ToLeftTest)

凸包构造时将会频繁用到此公式

```
bool ToLeftTest(Point a, Point b, Point c){
    return Cross(b - a, c - b) > 0;
}
```

1.10 点绕着 p 点逆时针旋转 angle(rotate)

```cpp
//绕着 p 点逆时针旋转 angle
Point rotate(Point p,double angle){
    Point v = (*this) - p;
    double c = cos(angle), s = sin(angle);
    return Point(p.x + v.x*c - v.y*s,p.y + v.x*s + v.y*c);
}
```

1.11 判断点和直线关系(relation)

```cpp
//点和直线关系
//1 在左侧
//2 在右侧
//3 在直线上 //直线上两点s和e
int relation(Point p){
    int c = sgn((p-s)^(e-s));
    if(c < 0)return 1;
    else if(c > 0)return 2;
    else return 3;
}
```

1.12 复数表示

```cpp
#include <complex>
using namespace std;
typedef complex<double> Point;
typedef Point Vector;//复数定义向量后，自动拥有构造函数、加减法和数量积
const double eps = 1e-9;
int sgn(double x){
    if(fabs(x) < eps)
        return 0;
    if(x < 0)
        return -1;
    return 1;
}
double Length(Vector A){
    return abs(A);
}
double Dot(Vector A, Vector B){//conj(a+bi)返回共轭复数a-bi
    return real(conj(A)*B);
}
double Cross(Vector A, Vector B){
    return imag(conj(A)*B);
}
Vector Rotate(Vector A, double rad){
    return A*exp(Point(0, rad));//exp(p)返回以e为底复数的指数
}
```

$\oint$ 2. 点与线

- 一般式$ax + by + c = 0$
- 点向式$x0 + y0 + vxt + vyt = 0$
- 斜截式$y = kx + b$
  计算机中常用点向式表示直线，即参数方程形式表示

直线可以用直线上的一个点P0和方向向量v表示

$P = P0 + vt$其中t为参（可以通过限制参数来表示线段和射线）

2.1 直线的实现(Line)

```
struct Line{//直线定义
    Point v, p;
    Line(Point v, Point p):v(v), p(p) {}
    Point point(double t){//返回点P = v + (p - v)*t
        return v + (p - v)*t;
    }
};
```

2.2 判断点在直线上

- 利用三点共线的等价条件α×β==0
- 直线上取两不同点与待测点构成向量求叉积是否为零来判断点是否在直线上（叉积为0互相平行）
  判断点与直线的关系（见1.11）

2.3 计算两直线交点(Get_line_intersection)

```
//调用前要确保两直线p + tv 和 Q + tw之间有唯一交点，当且仅当Corss(v, w) ≠ 0;（t是参数）
Point Get_line_intersection(Point P,Vector v,Point Q,Vector w)
{
    Vector u = P - Q;
    double t = Cross(w, u) / Cross(v, w);
    return P + v * t;
}

//!直线的参数式 直线 AB:A + tv(v为向量AB, t为参数)
```

2.4 计算点到直线的距离(Distance_point_to_line)

```
double Distance_point_to_line(Point P, Point A, Point B)
{
    Vector v1 = B - A, v2 = P - A;
    return fabs(Cross(v1, v2) / Length(v1));//如果不取绝对值，那么得到的是有向距离
}
```

2.5 计算点到线段的距离(Distance_point_to_segment)

```
//垂线距离或者PA或者PB距离
double Distance_point_to_segment(Point P, Point A, Point B)
{
    if(A == B) return Length(P - A);//（如果重合那么就是两个点之间的距离，直接转成向量求距离即可）
    Vector v1 = B - A, v2 = P - A, v3 = P - B;
    if(dcmp(Dot(v1, v2)) < 0) return Length(v2);//A点左边
    if(dcmp(Dot(v1, v3)) > 0)return Length(v3);//B点右边
    return fabs(Cross(v1, v2) / Length(v1));//垂线的距离
}
```

2.6 求点在直线上的投影点(Get_line_projection)

```cpp
//!点在直线上的投影
//点积满足分配率:两直线垂直，点积为0，向量v,Q = A + t0v,
//点P,v与直线PQ垂直:
//Dot(v, p - (A + t0v)) = 0 -> Dot(v, P - A) - t0 * Dot(v, v) = 0;
Point Get_line_projection(Point P, Point A, Point B)
{
    Vector v = B - A;
    return A + v * (Dot(v, P - A) / Dot(v, v));
}
```

2.7 判断点是否在线段上(OnSegment)

```cpp
bool OnSegment(Point p, Point a1, Point a2){
    return dcmp(Cross(a1-p, a2-p)) == 0 && dcmp(Dot(a1-p, a2-p)) ≤ 0;//紫书上写的是小于，但是真正写题的时候写成≤才A
}
```

```cpp
另一种写法，尽量先用这个
bool onsegment(point pi,point pj,point Q)
{
    if((Q.x-pi.x)*(pj.y-pi.y)==(pj.x-pi.x)*(Q.y-pi.y)&&min(pi.x,pj.x)≤Q.x&&Q.x≤max(pi.x,pj.x)&&min(pi.y,pj.y)≤Q.y&&Q.y≤max(pi.y,pj.y)){
        return true;
    }else{
        return false;
    }
}
```

2.8 判断两线段是否相交（不算在端点处相交）(segment_proper_intersection)

```cpp
bool segment_proper_intersection(Point a1, Point a2, Point b1, Point b2)
{
    double c1 = Cross(a2 - a1, b1 - a1), c2 = Cross(a2 - a1, b2 - a1);
    double c3 = Cross(b2 - b1, a1 - b1), c4 = Cross(b2 - b1, a2 - b1);
    return dcmp(c1) * dcmp(c2) < 0 && dcmp(c3) * dcmp(c4) < 0;
}
```

2.9 判断两线段是否相交（包含端点处相交）(Segment_proper_intersection)

```cpp
bool Segment_proper_intersection(Point a1, Point a2, Point b1, Point b2){
    double c1 = Cross(a2-a1, b1-a1), c2 = Cross(a2-a1, b2-a1);
    double c3 = Cross(b2-b1, a1-b1), c4 = Cross(b2-b1, a2-b1);
    //if判断控制是否允许线段在端点处相交，根据需要添加
    if(!sgn(c1) || !sgn(c2) || !sgn(c3) || !sgn(c4)){
        bool f1 = OnSegment(b1, a1, a2);
        bool f2 = OnSegment(b2, a1, a2);
        bool f3 = OnSegment(a1, b1, b2);
        bool f4 = OnSegment(a2, b1, b2);
        bool f = (f1|f2|f3|f4);
        return f;
    }
    return (sgn(c1)*sgn(c2) < 0 && sgn(c3)*sgn(c4) < 0);
}
```

2.10 判断点是否在一条线段上(不含端点)(on_segment)

```
bool on_segment(Point P, Point a1, Point a2)
{
    return dcmp(Cross(a1 - P, a2 - P)) == 0 && dcmp(Dot(a1 - P, a2 - P)) < 0;
}

```

2.11 两向量的关系(parallel)

```
//直线v和直线Line(s, e);
bool parallel(Line v){
    return sgn((e-s)^(v.e-v.s)) == 0;
}
```

2.12 两直线关系(linecrossline)

```
//两直线关系
//0 平行
//1 重合
//2 相交
int linecrossline(Line v){
    if((*this).parallel(v))
    return v.relation(s)==3;
    return 2;
}
```

$\oint$ 3. 多边形

普通多边形

通常按照逆时针储存所有顶点

定义

多边形

由在同一平面且不在同一直线上的多条线段首位顺次连接且不相交所组成的图形叫做多边形

简单多边形

简单多边形是除相邻边外其它边不相交的多边形

凸多边形

过多边形的任意一边做一条直线，如果其他各个顶点都在这条直线的同侧，则把这个多边形叫做凸多边形
（所有的正多边形都是凸多边形，所有的三角形都是凸多边形）

任意凸多边形外角和均为360°
任意凸多边形内角和为(n−2) * 180°

3.0.三角形

设三角形的三条边为a, b, c, 且不妨假设a <= b <= c.

三角形面积

- 利用两条边叉积除以二取绝对值
- 海伦公式

$$S=\sqrt{p(p-a)(p-b)(p-c)}, p=\frac{(a+b+c)}{2}$$

- $S=\frac{absinC}{2}$

3.0.1 三角形四心

- **外心**：三边中垂线交点，到三角形三个顶点距离相同
  三点求圆心坐标。

```
Point waixin(Point a, Point b, Point c)
{
    double a1 = b.x - a.x, b1 = b.y - a.y, c1 = (a1 * a1 + b1 * b1) / 2;
    double a2 = c.x - a.x, b2 = c.y - a.y, c2 = (a2 * a2 + b2 * b2) / 2;
    double d = a1 * b2 - a2 * b1;
    return Point(a.x + (c1 * b2 - c2 * b1) / d, a.y + (a1 * c2 -a2 * c1) / d);
}
```

- **内心**：角平分线的交点，到三角形三边的距离相同

角平分线的交点。
令`x = (a + b - c) / 2, y = (a - b + c) / 2, z = (-a + b + c) / 2, h = s / p`.
计算公式为`sqrt(x * x + h * h) + sqrt(y * y + h * h) + sqrt(z * z + h * h)`。

- **垂心**：三条高线的交点

垂线的交点。
计算公式如下:
`3 * (c / 2 / sqrt(1 - cosC * cosC))`。

- **重心**：三条中线的交点，到三角形三顶点距离的平方和最小的点，三角形内到三边距离之积最大的点

中线的交点。
计算公式如下:
`2.0 / 3 * (sqrt((2 * (a * a + b * b) - c * c) / 4)+ sqrt((2 * (a * a + c * c) - b * b) / 4) + sqrt((2 * (b * b + c * c) - a * a) / 4))`。

三角形重心

三点各坐标的平均值$(\frac{x1+x2+x3}{3}, \frac{y1+y2+y3}{3})$

3.0.2 关键点与A, B, C三顶点距离之和

3.0.2.1 费马点

该点到三角形三个顶点的距离之和最小。
有个有趣的结论:
若三角形的三个内角均小于120度,那么该点连接三个顶点形成的三个角均为120度;若三角形存在一个内角大于120度,则该顶点就是费马点。
计算公式如下:
若有一个内角大于120度(这里假设为角C),则距离为`a + b`;若三个内角均小于120度,则距离为`sqrt((a * a + b * b + c * c + 4 * sqrt(3.0) * s) / 2)`。

3.0.3向量求三角形垂心(getcircle)

```cpp
inline Circle getcircle(Point A,Point B,Point C){//【三点确定一圆】向量垂心法
    Point P1=(A+B)*0.5,P2=(A+C)*0.5;
    Point O=cross_LL(P1,P1+Normal(B-A),P2,P2+Normal(C-A));
    return Circle(O,Len(A-O));
}
```

3.1.0 正多边形的一些性质和概念

外接圆
把圆分为n(n≥3)等份，依次连接各分点所得的多边形就是这个圆的内接正n边形，也就是正n边形的外接圆。

内切圆
把圆分为m(m≥3)等份，经过各分点作圆的切线，以相邻切线的交点为顶点的多边形就是这个圆的外切正m边形，也就是正m边形的内切圆。

内角
正n边形的内角和度数为：（n - 2)×180°;
正n边形的一个内角是(n-2)×180°÷n.
正n边形内固定一个边，按角度从小到大遍历所有点，每个角的角度为(180.0-deg)/2*(i-2)，deg是一个内角，i是所选点的标号，从1~n顺时针编号可由内角公式推出

外角
正n边形外角和等于n·180° - (n - 2)·180°=360°
所以正n边形的一个外角为：360°÷n.
所以正n边形的一个内角也可以用这个公式：180°-360°÷n.

中心角
任何一个正多边形，都可作一个外接圆，多边形的中心就是所作外接圆的圆心，所以每条边的中心角，实际上就是这条边所对的弧的圆心角，因此这个角就是360度÷边数。
正多边形中心角：360°÷n
因此可证明，正n边形中，外角=中心角=360°÷n对角线
在一个正多边形中，所有的顶点可以与除了他相邻的两个顶点的其他顶点连线，就成了顶点数减2（2是那两个相邻的点）个三角形。三角形内角和：180度，所以把边数减2乘上180度，就是这个正多边形的内角和。对角线数量的计算公式：n(n-3)÷2。

面积
设正n边形的半径为R，边长为an，中心角为αn，边心距为rn，则αn=360°÷n，an=2Rsin(180°÷n)，rn=Rcos(180°÷n)，R^2=rn2+(an÷2)2，周长pn=n×an，面积Sn=pn×rn÷2。

对称轴
正多边形的对称轴——
奇数边：连接一个顶点和顶点所对的边的中点的线段所在的直线，即为对称轴；
偶数边：连接相对的两个边的中点，或者连接相对称的两个顶点的线段所在的直线，都是对称轴。
正N边形边数、角数、对称轴数都为N。

3.1 求多边形面积(convex_polygon_area)

我们可以从第一个顶点除法把凸多边形分成n-2个三角形，然后把面积加起来，最后返回值说为有向面积更贴近本质

```cpp
//!求凸多边形的有向面积
//叉积的几何意义就是三角形有向面积的二倍，所以这里要除以二
double convex_polygon_area(Point* p, int n)
{
    double area = 0;
    for(int i = 1; i ≤ n - 2; ++ i)
        area += Cross(p[i] - p[0], p[i + 1] - p[0]);
    return area / 2;
}
//!求非凸多边形的有向面积
//我们叉积求得的三角形面积是有向的，在外面的面积可以正负抵消掉，
//因此非凸多边形也适用，可以从任意点出发划分
//可以取原点为起点，减少叉乘次数
double polyg_on_area(Point* p, int n)
```

```
{
    double area = 0;
    for(int i = 1; i ≤ n - 2; ++ i)
        area += Cross(p[i] - p[0], p[i + 1] - p[0]);
    return area / 2;
}
```

3.2 判断点在多边形内(is_point_in_polygon)

有射线法与转角法。
转角法的基本思想是看多边形相对于这个点转了多少度
如果是三百六十度，说明点在多边形内
如果是零度，说明点在多边形外
如果是一百八十度，说明点在多边形边界上
如果直接按照定义来算，则需要计算大量反三角函数，不仅速度慢，而且容易产生精度问题
因此我们采用winding number绕数来计算

```
//判断点是否在多边形内，若点在多边形内返回1，在多边形外部返回0，在多边形上返回-1
int is_point_in_polygon(Point p, vector<Point> poly){//待判断的点和该多边形的所有点的合集
    int wn = 0;
    int n = poly.size();
    for(int i = 0; i < n; ++i){
        if(OnSegment(p, poly[i], poly[(i+1)%n])) return -1;
        int k = sgn(Cross(poly[(i+1)%n] - poly[i], p - poly[i]));
        int d1 = sgn(poly[i].y - p.y);
        int d2 = sgn(poly[(i+1)%n].y - p.y);
        if(k > 0 && d1 ≤ 0 && d2 > 0) wn++;
        if(k < 0 && d2 ≤ 0 && d1 > 0) wn--;
    }
    if(wn ≠ 0)
        return 1;
    return 0;
}
```

3.3 判断点在凸多边形内

只需要判断点是否在所有边的左边（按逆时针顺序排列的顶点集）可以使用ToLeftTest，$O(n)$

3.4 求多边形重心(barycenter)

```
point barycenter(int n,point* p){
    point ret,t;
    double t1=0,t2;
    int i;
    ret.x=ret.y=0;
    for (i=1;i<n-1;i++)
        if (fabs(t2=xmult(p[0],p[i],p[i+1]))>eps){
            t=barycenter(p[0],p[i],p[i+1]);
            ret.x+=t.x*t2;
            ret.y+=t.y*t2;
            t1+=t2;
        }
    if (fabs(t1)>eps)
        ret.x/=t1,ret.y/=t1;
    return ret;
}
```

3.5 判定凸多边形(is_convex)

3.6 判点在凸多边形内或多边形边上(inside_convex)

3.7 判点在任意多边形内(inside_polygon)

3.8 判线段在任意多边形内(inside_polygon)

```
#include <stdlib.h>
#include <math.h>
#define MAXN 1000
#define offset 10000
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
#define _sign(x) ((x)>eps?1:((x)<-eps?2:0))
struct point{double x,y;};
struct line{point a,b;};


double xmult(point p1,point p2,point p0){
    return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}


//判定凸多边形,顶点按顺时针或逆时针给出,允许相邻边共线
int is_convex(int n,point* p){
    int i,s[3]={1,1,1};
    for (i=0;i<n&&s[1]|s[2];i++)
        s[_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;
    return s[1]|s[2];
}


//判定凸多边形,顶点按顺时针或逆时针给出,不允许相邻边共线
int is_convex_v2(int n,point* p){
    int i,s[3]={1,1,1};
    for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
        s[_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;
    return s[0]&&s[1]|s[2];
}


//判点在凸多边形内或多边形边上,顶点按顺时针或逆时针给出
int inside_convex(point q,int n,point* p){
    int i,s[3]={1,1,1};
    for (i=0;i<n&&s[1]|s[2];i++)
        s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
    return s[1]|s[2];
}


//判点在凸多边形内,顶点按顺时针或逆时针给出,在多边形边上返回0
int inside_convex_v2(point q,int n,point* p){
    int i,s[3]={1,1,1};
    for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
        s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
    return s[0]&&s[1]|s[2];
```

```
}


//判点在任意多边形内,顶点按顺时针或逆时针给出
//on_edge表示点在多边形边上时的返回值,offset为多边形坐标上限
int inside_polygon(point q,int n,point* p,int on_edge=1){
    point q2;
    int i=0,count;
    while (i<n)
        for (count=i=0,q2.x=rand()+offset,q2.y=rand()+offset;i<n;i++)
            if (zero(xmult(q,p[i],p[(i+1)%n]))&&(p[i].x-q.x)*(p[(i+1)%n].x-q.x)<eps&&(p[i].y-q.y)*(p[(i+1)%n].y-q.y)<eps)
                return on_edge;
            else if (zero(xmult(q,q2,p[i])))
                break;
            else if (xmult(q,p[i],q2)*xmult(q,p[(i+1)%n],q2)←eps&&xmult(p[i],q,p[(i+1)%n])*xmult(p[i],q2,p[(i+1)%n])←eps)
                count++;
    return count&1;
}


inline int opposite_side(point p1,point p2,point l1,point l2){
    return xmult(l1,p1,l2)*xmult(l1,p2,l2)←eps;
}


inline int dot_online_in(point p,point l1,point l2){
    return zero(xmult(p,l1,l2))&&(l1.x-p.x)*(l2.x-p.x)<eps&&(l1.y-p.y)*(l2.y-p.y)<eps;
}


//判线段在任意多边形内,顶点按顺时针或逆时针给出,与边界相交返回1
int inside_polygon(point l1,point l2,int n,point* p){
    point t[MAXN],tt;
    int i,j,k=0;
    if (!inside_polygon(l1,n,p)||!inside_polygon(l2,n,p))
        return 0;
    for (i=0;i<n;i++)
        if (opposite_side(l1,l2,p[i],p[(i+1)%n])&&opposite_side(p[i],p[(i+1)%n],l1,l2))
            return 0;
        else if (dot_online_in(l1,p[i],p[(i+1)%n]))
            t[k++]=l1;
        else if (dot_online_in(l2,p[i],p[(i+1)%n]))
            t[k++]=l2;
        else if (dot_online_in(p[i],l1,l2))
            t[k++]=p[i];
    for (i=0;i<k;i++)
        for (j=i+1;j<k;j++){
            tt.x=(t[i].x+t[j].x)/2;
            tt.y=(t[i].y+t[j].y)/2;
            if (!inside_polygon(tt,n,p))
                return 0;
        }
    return 1;
}


point intersection(line u,line v){
    point ret=u.a;
```

```
    double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))
            /((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
    ret.x+=(u.b.x-u.a.x)*t;
    ret.y+=(u.b.y-u.a.y)*t;
    return ret;
}


point barycenter(point a,point b,point c){
    line u,v;
    u.a.x=(a.x+b.x)/2;
    u.a.y=(a.y+b.y)/2;
    u.b=c;
    v.a.x=(a.x+c.x)/2;
    v.a.y=(a.y+c.y)/2;
    v.b=b;
    return intersection(u,v);
}

//多边形重心
point barycenter(int n,point* p){
    point ret,t;
    double t1=0,t2;
    int i;
    ret.x=ret.y=0;
    for (i=1;i<n-1;i++)
        if (fabs(t2=xmult(p[0],p[i],p[i+1]))>eps){
            t=barycenter(p[0],p[i],p[i+1]);
            ret.x+=t.x*t2;
            ret.y+=t.y*t2;
            t1+=t2;
        }
    if (fabs(t1)>eps)
        ret.x/=t1,ret.y/=t1;
    return ret;
}
```

3.9 多边形切割(常用于半平面交)

```
//多边形切割
//可用于半平面交
#define MAXN 100
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
struct point{double x,y;};


double xmult(point p1,point p2,point p0){
    return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}


int same_side(point p1,point p2,point l1,point l2){
    return xmult(l1,p1,l2)*xmult(l1,p2,l2)>eps;
}


point intersection(point u1,point u2,point v1,point v2){
```

```
    point ret=u1;
    double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))
            /((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
    ret.x+=(u2.x-u1.x)*t;
    ret.y+=(u2.y-u1.y)*t;
    return ret;
}


//将多边形沿l1,l2确定的直线切割在side侧切割,保证l1,l2,side不共线
void polygon_cut(int& n,point* p,point l1,point l2,point side){
    point pp[MAXN];
    int m=0,i;
    for (i=0;i<n;i++){
        if (same_side(p[i],side,l1,l2))
            pp[m++]=p[i];
        if (!same_side(p[i],p[(i+1)%n],l1,l2)&&!
(zero(xmult(p[i],l1,l2))&&zero(xmult(p[(i+1)%n],l1,l2))))
            pp[m++]=intersection(p[i],p[(i+1)%n],l1,l2);
    }
    for (n=i=0;i<m;i++)
        if (!i||!zero(pp[i].x-pp[i-1].x)||!zero(pp[i].y-pp[i-1].y))
            p[n++]=pp[i];
    if (zero(p[n-1].x-p[0].x)&&zero(p[n-1].y-p[0].y))
        n--;
    if (n<3)
        n=0;
}
```

$\oint$ 4.圆

计算机中储存圆通常记录圆心坐标与半径即可

定义

```
struct Circle
{
    Point c;
    double r;
    Circle(Point c=Point(),double r=0):c(c),r(r){}
    inline Point point(double a)//通过圆心角求坐标
    { return Point(c.x+cos(a)*r,c.y+sin(a)*r); }
};
inline Circle read_circle()
{
    Circle C;
    scanf("%lf%lf%lf",&C.c.x,&C.c.y,&C.r);
    return C;
}
```

4.1 圆与直线交点(getLineCircleIntersection)

```
//求圆与直线交点
int getLineCircleIntersection(Line L, Circle C, double& t1, double& t2, vector<Point>& sol){
    double a = L.v.x, b = L.p.x - C.c.x, c = L.v.y, d = L.p.y - C.c.y;
    double e = a*a + c*c, f = 2*(a*b + c*d), g = b*b + d*d - C.r*C.r;
    double delta = f*f - 4*e*g;//判别式
    if(sgn(delta) < 0)//相离
```

```
        return 0;
    if(sgn(delta) == 0){//相切
        t1 = -f /(2*e);
        t2 = -f /(2*e);
        sol.push_back(L.point(t1));//sol存放交点本身
        return 1;
    }
    //相交
    t1 = (-f - sqrt(delta))/(2*e);
    sol.push_back(L.point(t1));
    t2 = (-f + sqrt(delta))/(2*e);
    sol.push_back(L.point(t2));
    return 2;
}
```

4.2 求两圆交点(get_circle_circle_intersection)

```
//两圆相交保存所有交点返回交点个数（至多两个）
int get_circle_circle_intersection(Circle c1, Circle c2, vector<Point>& sol)
{
    double d = Length(c1.c - c2.c);
    if(dcmp(d) == 0){
        if(dcmp(c1.r - c2.r) == 0)return -1;//两圆重合
        return 0;
    }
    if(dcmp(c1.r + c2.r - d) < 0)return 0;//相离
    if(dcmp(fabs(c1.r - c2.r) - d) > 0)return 0;//在另一个圆的内部

    double a = angle(c2.c - c1.c);//向量c1c2的极角
    double da = acos((c1.r * c1.r + d * d - c2.r * c2.r) / (2 * c1.r * d));
    //c1c2到c1p1的角
    Point p1 = c1.point(a - da), p2 = c1.point(a + da);
    sol.push_back(p1);
    if(p1 == p2)return 1;
    sol.push_back(p2);
    return 2;
}

```

4.3 点到圆的切线(get_tangents)

```

//过点p到圆c的切线，v[i]是第i条切线，  返回切线的条数
int get_tangents(Point p, Circle C, Vector* v){
    Vector u = C.c - p;
    double dist = Length(u);
    if(dist < C.r)return 0;//点在内部，没有切线
    else if(dcmp(dist - C.r) == 0){//p在圆上，只有一条切线
        v[0] = Rotate(u, PI / 2);//切线就是垂直嘛
        return 1;
    }
    else {//否则是两条切线
        double ang = asin(C.r / dist);
        v[0] = Rotate(u, - ang);
        v[1] = Rotate(u, +ang);
        return 2;
    }
}
```

4.4 两圆的公切线(get_tangents)

两圆的公切线。
根据两圆的圆心距从小到大排列，一共有6种情况。
情况一：两圆完全重合。有无数条公切线。
情况二：两圆内含，没有公共点。没有公切线。
情况三：两圆内切。有1条外公切线。
情况四：两圆相交。有2条外公切线。
情况五：两圆外切。有3条公切线，其中一条内公切线，两条外公切线。
情况六：两圆相离。有4条公切线，其中内公切线两条，外公切线两条。
可以根据圆心距和半径的关系辨别出这6种情况，然后逐一求解。
情况一和情况二没什么需要求的，情况三和情况五中的内公切线都对应于“过圆上一点求圆的切线”，只需连接圆心和切点，旋转90°后即可知道切线的方向向量。这样，问题的关键是求出情况四、五中的外公切线和情况六中的内公切线。

```
//返回切线的条数，-1表示无穷条切线
//a[i]和b[i] 分别是第i条切线在圆A和圆B上的切点
int get_tangents(Circle A, Circle B, Point* a, Point* b)
{
    int cnt = 0;
    if(A.r < B.r)swap(A, B), swap(a, b);
    int d2 = (A.x - B.x) * (A.x - B.x) + (A.y - B.y) * (A.y - B.y);
    int rdiff = A.r - B.r;
    int rsum = A.r + B.r;
    if(d2 < rdiff * rdiff)return 0;//内含
    double base = atan2(B.y - A.y, B.x - A.x);
    if(d2 == 0 && A.r == B.r)return -1;//无限多条切线
    if(d2 == rdiff * rdiff){//内切，1条切线
        a[cnt] = A.point(base);
        b[cnt] = B.point(base); cnt ++ ;
        return 1;
    }
    //有外共切线
    double ang = acos((A.r - B.r) / sqrt(d2));
    a[cnt] = A.point(base + ang);b[cnt] = B.point(base + ang);cnt ++;
    a[cnt] = A.point(base - ang);b[cnt] = B.point(base - ang);cnt ++;
    if(d2 == rsum * rsum){//一条内公切线
        a[cnt] = A.point(base);b[cnt] = B.point(PI + base); cnt ++ ;
    }
    else if(d2 > rsum * rsum){//两条内公切线
        double ang = acos((A.r + B.r) / sqrt(d2));
        a[cnt] = A.point(base + ang); b[cnt] = B.point(PI + base + ang); cnt ++ ;
        a[cnt] = A.point(base - ang); b[cnt] = B.point(PI + base - ang); cnt ++ ;
    }
    return cnt;
}
```

4.5 两圆相交面积(AreaOfOverlap)

通过计算两个圆相交所构成的两个扇形面积和减去其构成的箏形的面积

```
double AreaOfOverlap(Point c1, double r1, Point c2, double r2){
    double d = Length(c1 - c2);
    if(r1 + r2 < d + eps)
        return 0.0;
    if(d < fabs(r1 - r2) + eps){
        double r = min(r1, r2);
        return pi*r*r;
    }
```

```
    double x = (d*d + r1*r1 - r2*r2)/(2.0*d);
    double p = (r1 + r2 + d)/2.0;
    double t1 = acos(x/r1);
    double t2 = acos((d - x)/r2);
    double s1 = r1*r1*t1;
    double s2 = r2*r2*t2;
    double s3 = 2*sqrt(p*(p - r1)*(p - r2)*(p - d));
    return s1 + s2 - s3;
}
```

4.6 模板合集

```
#include <math.h>
#define eps 1e-8
struct point{double x,y;};


double xmult(point p1,point p2,point p0){
    return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}


double distance(point p1,point p2){
    return sqrt((p1.x-p2.x)*(p1.x-p2.x)+(p1.y-p2.y)*(p1.y-p2.y));
}


double disptoline(point p,point l1,point l2){
    return fabs(xmult(p,l1,l2))/distance(l1,l2);
}


point intersection(point u1,point u2,point v1,point v2){
    point ret=u1;
    double t=((u1.x-v1.x)*(v1.y-v2.y)-(u1.y-v1.y)*(v1.x-v2.x))
            /((u1.x-u2.x)*(v1.y-v2.y)-(u1.y-u2.y)*(v1.x-v2.x));
    ret.x+=(u2.x-u1.x)*t;
    ret.y+=(u2.y-u1.y)*t;
    return ret;
}

```

4.7 判直线和圆相交(intersect_line_circle)

```
//判直线和圆相交,包括相切
int intersect_line_circle(point c,double r,point l1,point l2){
    return disptoline(c,l1,l2)<r+eps;
}
```

4.8 判线段和圆相交(intersect_seg_circle)

```cpp
//判线段和圆相交,包括端点和相切
int intersect_seg_circle(point c,double r,point l1,point l2){
    double t1=distance(c,l1)-r,t2=distance(c,l2)-r;
    point t=c;
    if (t1<eps||t2<eps)
        return t1>-eps||t2>-eps;
    t.x+=l1.y-l2.y;
    t.y+=l2.x-l1.x;
    return xmult(l1,c,t)*xmult(l2,c,t)<eps&&disptoline(c,l1,l2)-r<eps;
}
```

4.9 判圆和圆相交(intersect_circle_circle)

```cpp
//判圆和圆相交,包括相切
int intersect_circle_circle(point c1,double r1,point c2,double r2){
    return distance(c1,c2)<r1+r2+eps&&distance(c1,c2)>fabs(r1-r2)-eps;
}

```

4.10 计算圆上到点p最近点(dot_to_circle)

```cpp
//计算圆上到点p最近点,如p与圆心重合,返回p本身
point dot_to_circle(point c,double r,point p){
    point u,v;
    if (distance(p,c)<eps)
        return p;
    u.x=c.x+r*fabs(c.x-p.x)/distance(c,p);
    u.y=c.y+r*fabs(c.y-p.y)/distance(c,p)*((c.x-p.x)*(c.y-p.y)<0?-1:1);
    v.x=c.x-r*fabs(c.x-p.x)/distance(c,p);
    v.y=c.y-r*fabs(c.y-p.y)/distance(c,p)*((c.x-p.x)*(c.y-p.y)<0?-1:1);
    return distance(u,p)<distance(v,p)?u:v;
}

```

4.11 计算直线/线段与圆的交点（intersection_line_circle）

计算线段与圆的交点可用这个函数求得交点之后再判点是否在线段上

```cpp
//计算直线与圆的交点,保证直线与圆有交点
void intersection_line_circle(point c,double r,point l1,point l2,point& p1,point& p2){
    point p=c;
    double t;
    p.x+=l1.y-l2.y;
    p.y+=l2.x-l1.x;
    p=intersection(p,c,l1,l2);
    t=sqrt(r*r-distance(p,c)*distance(p,c))/distance(l1,l2);
    p1.x=p.x+(l2.x-l1.x)*t;
    p1.y=p.y+(l2.y-l1.y)*t;
    p2.x=p.x-(l2.x-l1.x)*t;
    p2.y=p.y-(l2.y-l1.y)*t;
}

```

4.12 计算圆与圆的交点（intersection_circle_circle）

```cpp
//计算圆与圆的交点,保证圆与圆有交点,圆心不重合
void intersection_circle_circle(point c1,double r1,point c2,double r2,point& p1,point& p2){
```

```
    point u,v;
    double t;
    t=(1+(r1*r1-r2*r2)/distance(c1,c2)/distance(c1,c2))/2;
    u.x=c1.x+(c2.x-c1.x)*t;
    u.y=c1.y+(c2.y-c1.y)*t;
    v.x=u.x+c1.y-c2.y;
    v.y=u.y-c1.x+c2.x;
    intersection_line_circle(c1,r1,u,v,p1,p2);
}


//将向量p逆时针旋转angle角度
Point Rotate(Point p,double angle) {
    Point res;
    res.x=p.x*cos(angle)-p.y*sin(angle);
    res.y=p.x*sin(angle)+p.y*cos(angle);
    return res;
}
```

4.13 求圆外一点对圆的两个切点(TangentPoint_PC)

```
//求圆外一点对圆(o,r)的两个切点result1和result2
void TangentPoint_PC(Point poi,Point o,double r,Point &result1,Point &result2) {
    double line=sqrt((poi.x-o.x)*(poi.x-o.x)+(poi.y-o.y)*(poi.y-o.y));
    double angle=acos(r/line);
    Point unitvector,lin;
    lin.x=poi.x-o.x;
    lin.y=poi.y-o.y;
    unitvector.x=lin.x/sqrt(lin.x*lin.x+lin.y*lin.y)*r;
    unitvector.y=lin.y/sqrt(lin.x*lin.x+lin.y*lin.y)*r;
    result1=Rotate(unitvector,-angle);
    result2=Rotate(unitvector,angle);
    result1.x+=o.x;
    result1.y+=o.y;
    result2.x+=o.x;
    result2.y+=o.y;
    return;
}
```

4.14 求三角形的外接圆(get_circumcircle)

```
//get_circumcircle
Circle WaiJieYuan(Point p1,Point p2,Point p3)
{
    double Bx=p2.x-p1.x,By=p2.y-p1.y;
    double Cx=p3.x-p1.x,Cy=p3.y-p1.y;
    double D=2*(Bx*Cy-By*Cx);
    double ansx=(Cy*(Bx*Bx+By*By)-By*(Cx*Cx+Cy*Cy))/D+p1.x;
    double ansy=(Bx*(Cx*Cx+Cy*Cy)-Cx*(Bx*Bx+By*By))/D+p1.y;
    Point p(ansx,ansy);
    return Circle(p,Length(p1-p));
}
```

4.15 求三角形的内接圆(get_incircle)

```cpp
//get_incircle
Circle NeiJieYuan(Point p1,Point p2,Point p3)
{
    double a=Length(p2-p3);
    double b=Length(p3-p1);
    double c=Length(p1-p2);
    Point p=(p1*a+p2*b+p3*c)/(a+b+c);
    return Circle(p,DistanceToLine(p,p1,p2));
}
```

4.16 二维几何110_2合一！

具体问题见蓝书（《算法竞赛入门经典训练指南》P267）

```cpp
const char* q1="CircumscribedCircle";
const char* q2="InscribedCircle";
const char* q3="TangentLineThroughPoint";
const char* q4="CircleThroughAPointAndTangentToALineWithRadius";
const char* q5="CircleTangentToTwoLinesWithRadius";
const char* q6="CircleTangentToTwoDisjointCirclesWithRadius";
char cmd[100];
void output(vector<double> res)
{
    sort(res.begin(),res.end());
    printf("[");
    for(int i=0;i<res.size();i++)
    {
        if(i) printf(",");
        printf("%.6lf",res[i]);
    }
    printf("]\n");
}
void output(vector<Point> res)
{
    sort(res.begin(),res.end());
    printf("[");
    for(int i=0;i<res.size();i++)
    {
        if(i) printf(",");
        printf("(%.6lf,%.6lf)",res[i].x,res[i].y);
    }
    printf("]\n");
}
int main()
{
    while(scanf("%s",cmd)==1){
        if(strcmp(cmd,q1)==0){
            Point p1,p2,p3;
            p1=read_point();
            p2=read_point();
            p3=read_point();
            Circle C=WaiJieYuan(p1,p2,p3);
            printf("(%.6lf,%.6lf,%.6lf)\n",C.c.x,C.c.y,C.r);
        }
        if(strcmp(cmd,q2)==0){
            Point p1,p2,p3;
            p1=read_point();
            p2=read_point();
            p3=read_point();
```

```
            Circle C=NeiJieYuan(p1,p2,p3);
            printf("(%.6lf,%.6lf,%.6lf)\n",C.c.x,C.c.y,C.r);
        }
        if(strcmp(cmd,q3)==0){
            Circle C=read_circle();
            Point p=read_point();
            vector<Point> v;
            vector<double> res;
            GetTangents(p,C,v);
            for(int i=0;i<v.size();i++)
            {
                double tmp=R_to_D(Angle(v[i]-p));
                if(tmp<0) tmp+=180;
                if(tmp≥180) tmp-=180;
                res.push_back(tmp);
            }
            output(res);
        }
        if(strcmp(cmd,q4)==0){
            Point p=read_point();
            Point A=read_point();
            Point B=read_point();
            double r;
            scanf("%lf",&r);
            Circle C(p,r);
            Vector v=Normal(B-A)*r;//向两侧平移r
            Point A1=A+v,B1=B+v;
            Point A2=A-v,B2=B-v;
            vector<Point> res;
            GetLineCircleIntersection(A1,B1,C,res);
            GetLineCircleIntersection(A2,B2,C,res);
            output(res);
        }
        if(strcmp(cmd,q5)==0){
            Point p1=read_point();
            Point p2=read_point();
            Point p3=read_point();
            Point p4=read_point();
            double r;
            scanf("%lf",&r);
            Vector v=Normal(p1-p2)*r;
            Point A1=p1+v,B1=p2+v;
            Point A2=p1-v,B2=p2-v;
            v=Normal(p3-p4)*r;
            Point A3=p3+v,B3=p4+v;
            Point A4=p3-v,B4=p4-v;//向两侧平移r
            vector<Point> res;
            res.push_back(GetLineIntersection(A1,B1,A3,B3));
            res.push_back(GetLineIntersection(A1,B1,A4,B4));
            res.push_back(GetLineIntersection(A2,B2,A3,B3));
            res.push_back(GetLineIntersection(A2,B2,A4,B4));
            output(res);
        }
        if(strcmp(cmd,q6)==0){
            Circle c1=read_circle();
            Circle c2=read_circle();
            double r;
            scanf("%lf",&r);
            c1.r+=r;c2.r+=r;//向外膨胀r
            vector<Point> res;
```

```
106             GetCircleCircleIntersection(c1,c2,res);
107             output(res);
108         }
109     }
110     return 0;
111 }
112
```

4.17 经纬度转换为空间坐标

蓝书P269

4.18 球面距离

蓝书P270

4.19 三点确定一圆

设 $x^2+y^2+Dx+Ey+F=0$，圆心为 O，半径为 r，带入三点 $A(x_1,y_1),B(x_2,y_2),C(x_3,y_3)$，解得：

$$\begin{cases} D=\frac{(x_2^2+y_2^2-x_3^2-y_3^2)(y_1-y_2)-(x_1^2+y_1^2-x_2^2-y_2^2)(y_2-y_3)}{(x_1-x_2)(y_2-y_3)-(x_2-x_3)(y_1-y_2)} \\ E=\frac{x_1^2+y_1^2-x_2^2-y_2^2+D(x_1-x_2)}{y_2-y_1} \\ F=-(x_1^2+y_1^2+Dx_1+Ey_1) \\ O=(-\frac{D}{2},-\frac{E}{2}) \\ r=\frac{D^2+E^2-4F}{4} \end{cases}$$

> 给你不共线的三个点的座标，你的任务是算出通过这三个点的唯一圆的方程式并以下列2种方式输出：
> $(x-h)2+(y-k)2=r2x2+y2+cx+dy-e=0$

```
#include<bits/stdc++.h>
#define pi  3.141592653589793
using namespace std;
double pf(double a) {//平方
    return a*a;
};
//两点距离
double func1(double a1,double a2,double b1,double b2){
    double delta_x = a1-b1;
    double delta_y = a2-b2;
    double t1=pf(delta_x);
    double t2=pf(delta_y);
    return sqrt(t1+t2);
};
//三阶行列式求解//a1 a2 a3 竖着写
double func2(double a1,double a2,double a3,double b1,double b2,double b3,double c1,double c2,double c3){
    return  a1*b2*c3+b1*c2*a3+c1*a2*b3-a3*b2*c1-b3*c2*a1-c3*a2*b1;
}
int main()
{
    double a1,a2,b1,b2,c1,c2;
    while (cin>>a1>>a2>>b1>>b2>>c1>>c2) {
        //先求出三边的边长
```

```
        double a,b,c;
        a=func1(a1, a2, b1, b2);
        b=func1(c1, c2, b1, b2);
        c=func1(a1, a2, c1, c2);
        //用海伦公式求面积
        //先求半周长
        double p = (a+b+c)/2;
        //求S
        double s = sqrt(p*(p-a)*(p-b)*(p-c));
        //外接圆半径 R= abc/(4S)
        double r = a*b*c / (4*s);
        //cout<<r<<endl;
        //由公式求圆心坐标
        double x = 0.5 * func2(1,1,1,pf(a1)+pf(a2),pf(b1)+pf(b2),pf(c1)+pf(c2),a2,b2,c2) /
func2(1,1,1,a1,b1,c1,a2,b2,c2);
        //cout<<x<<endl;
        double y = 0.5 * func2(1,1,1,a1,b1,c1,pf(a1)+pf(a2),pf(b1)+pf(b2),pf(c1)+pf(c2)) /
func2(1,1,1,a1,b1,c1,a2,b2,c2);
       // cout<<y<<endl;
        //输出//加fabs，浮点数的绝对值函数，防止-0和0出现
        if(x≥0) printf("(x - %.3lf)^2 +",fabs(x));
        else printf("(x + %.3lf)^2 +",fabs(x));
        if(y≥0) printf(" (y - %.3lf)^2",fabs(y));
        else printf(" (y + %.3lf)^2",fabs(y));
        printf(" = %.3lf^2\n",r);
        printf("x^2 + y^2");
        if(x≥0) printf(" - %.3lfx",2*fabs(x));
        else printf(" + %.3lfx",2*fabs(x));
        if(y≥0) printf(" - %.3lfy",2*fabs(y));
        else printf(" + %.3lfy",2*fabs(y));
        double temp = pf(x) + pf(y) - pf(r);
        if(temp≥0) printf(" + %.3lf = 0",fabs(temp));
        else printf(" - %.3lf = 0",fabs(temp));
        cout<<endl<<endl;
    }
    return 0;
}
```

4.20 两个圆的关系(relationcircle)

```
//两圆的关系
//5 相离
//4 外切
//3 相交
//2 内切
//1 内含
//需要 Point 的 distance //测试：UVA12304
int relationcircle(circle v){
    double d = p.distance(v.p);
    if(sgn(d-r-v.r) > 0)return 5;
    if(sgn(d-r-v.r) == 0)return 4;
    double l = fabs(r-v.r);
    if(sgn(d-r-v.r)<0 && sgn(d-l)>0)return 3;
    if(sgn(d-l)==0)return 2;
    if(sgn(d-l)<0)return 1;
}
```

$\oint$ 5. 网格

5.1 多边形上的网格点个数

5.2 多边形内的网格点个数

```cpp
#define abs(x) ((x)>0?(x):-(x))
struct point{int x,y;};
int gcd(int a,int b){return b?gcd(b,a%b):a;}
//多边形上的网格点个数
int grid_onedge(int n,point* p){
    int i,ret=0;
    for (i=0;i<n;i++)
        ret+=gcd(abs(p[i].x-p[(i+1)%n].x),abs(p[i].y-p[(i+1)%n].y));
    return ret;
}
//多边形内的网格点个数
int grid_inside(int n,point* p){
    int i,ret=0;
    for (i=0;i<n;i++)
        ret+=p[(i+1)%n].y*(p[i].x-p[(i+2)%n].x);
    return (abs(ret)-grid_onedge(n,p))/2+1;
}
```

$\oint$ 6.一些函数/定理/应用

6.1 欧拉定理

> 题目大意：给出一个平面上n-1个点的回路，第n个顶点与第1个顶点相同，求它把整个平面分成了几个部分（包括内部围起来的部分和外面的无限大的区域）。

欧拉定理：设平面图的顶点数、边数和面数分别为V，E，F，则V+F-E=2。

> 那么我们先求所有的交点(不算n个端点,端点另外再加,所以这些交点一定是在线段内部的,即刘汝佳所说的规范相交): 由于一共n条线段,且任意线段不会部分重叠,所以任意两条线段要不平行(不相交)要不就规范相交. 所以我们只需要枚举所有的线段,然后求出他们规范相交的节点保存在数组中即可.
>
> 然后我们再把原本的n-1个顶点(起点与终点相同不用都加)加进去,然后去重.
>
> 下面我们要知道该平面一共有多少条线段,那么我们只要知道原来的每条线段到底被分成了多少段即可. 对于原来的每条线段,我们用上面求得的所有点去判断,当前这个点是不是在该线段内(不包含端点),如果该点在线段内,那么就说明该线段被该点截断,多出来1段.所以总的线段数目在n的基础上要+1.

```cpp
//上述均为模板全部省略
Point p[N], v[N * N];
int n;
int kcase;
int main()
{
    while(~scanf("%d", &n) && n){
        for(int i = 0; i < n; ++ i)
            scanf("%lf %lf", &p[i].x, &p[i].y), v[i] = p[i];
        n -- ;
        int c = n, e = n;
        for(int i = 0; i < n; ++ i)
            for(int j = i + 1; j < n; ++ j)
            if(segment_proper_intersection(p[i], p[i + 1], p[j], p[j + 1]))
            //方向向量和一个点确定一个直线P =  A + tv
```

```
            v[c ++ ] = Get_line_intersection(p[i], p[i + 1] - p[i], p[j], p[j + 1] - p[j]);
        sort(v, v + c);
        c = unique(v, v + c) - v;//去重防止三点共线
        //原来有n个点，有n-1个线段，新增点以后，
        //我们看有多少个新增的点在原来的线段上，
        //每有一个在原来线段的上面就会把这个线段割开成两个线段，答案+1
        for(int i = 0; i < c; ++ i)
            for(int j = 0; j < n; ++ j)
            if(on_segment(v[i], p[j], p[j + 1]))e ++ ;
        printf("Case %d: There are %d pieces.\n", ++ kcase, e + 2 - c);
    }
    return 0;
}

```

6.2 Pick定理（根据点在多边形内和边界的个数求面积）

皮克定理是指一个计算点阵中顶点在格点上的多边形面积公式，该公式可以表示为

$$2S = 2a + b - 2$$

其中a表示多边形内部的点数，b表示多边形边界上的点数，S表示多边形的面积。

常用形式

$$S = a + \frac{b}{2} - 1$$

常用计算

给你多边形的顶点，问多边形内部有多少点

$$a = S - \frac{b}{2} + 1$$

```
//A = b / 2 + i -1  其中 b 与 i 分别表示在边界上及内部的格子点之个數
typedef struct TPoint
{
    int x;
    int y;
}TPoint;
typedef struct TLine
{
    int a, b, c;
}TLine;
int triangleArea(TPoint p1, TPoint p2, TPoint p3)
{
    //已知三角形三个顶点的坐标，求三角形的面积
    int k = p1.x * p2.y + p2.x * p3.y + p3.x * p1.y
      - p2.x * p1.y - p3.x * p2.y - p1.x * p3.y;
    if(k < 0) return -k;
    else return k;
}
TLine lineFromSegment(TPoint p1, TPoint p2)
{
    //线段所在直线,返回直线方程的三个系统
    TLine tmp;
    tmp.a = p2.y - p1.y;
    tmp.b = p1.x - p2.x;
    tmp.c = p2.x * p1.y - p1.x * p2.y;
    return tmp;
}
void swap(int &a, int &b)
{
    int t;
```

```
    t = a;
    a = b;
    b = t;
}
int Count(TPoint p1, TPoint p2)
{
    int i, sum = 0, y;
    TLine l1 =  lineFromSegment(p1, p2);
    if(l1.b == 0) return abs(p2.y - p1.y) + 1;
    if(p1.x > p2.x) swap(p1.x, p2.x); //这里没有交换WA两次
    for(i = p1.x;i ≤ p2.x;i++){
        y = -l1.c - l1.a * i;
        if(y % l1.b == 0) sum++;
    }
    return sum;
}
int main()
{
    //freopen("in.in", "r", stdin);
    //freopen("OUT.out", "w", stdout);
    TPoint p1, p2, p3;
    while(scanf("%d%d%d%d%d%d", &p1.x, &p1.y, &p2.x, &p2.y, &p3.x, &p3.y) ≠ EOF){
        if(p1.x == 0 && p1.y == 0 && p2.x == 0 && p2.y == 0 && p3.x == 0 && p3.y == 0) break;
        int  A = triangleArea(p1, p2, p3);//A为面积的两倍
        int b = 0;
        int i;
        b = Count(p1, p2) + Count(p1, p3) + Count(p3, p2) - 3;//3个顶点多各多加了一次
        //i = A / 2- b / 2 + 1;
        i = (A - b) / 2 + 1;
        printf("%d\n", i);
    }
    return 0;
}
```

6.3 判断四点共面（混合积）

```
struct  point
{
    double x, y, z;
    point  operator - (point &o)
    {
        point  ans;
        ans.x = this→x - o.x;
        ans.y = this→y - o.y;
        ans.z = this→z - o.z;
        return ans;
    }
};

double  dot_product(const point &a, const point &b)
{
    return a.x * b.x + a.y * b.y + a.z * b.z;
}

point cross_product(const point &a, const point &b)
{
    point  ans;
    ans.x = a.y * b.z - a.z * b.y;
```

```
    ans.y = a.z * b.x - a.x * b.z;
    ans.z = a.x * b.y - a.y * b.x;
    return ans;
}

int main()
{
    point p[4];
    int T;
    for (scanf("%d", &T); T--;)
    {
        for (int i = 0; i < 4; ++i)
        {
            scanf("%lf%lf%lf", &p[i].x, &p[i].y, &p[i].z);
        }
        puts(dot_product(p[3] - p[0], cross_product(p[2] - p[0], p[1] - p[0])) == 0.0 ? "Yes\n" :
"No\n");
    }
    return 0;
}
```

$\oint$ 7. 平面扫描

平面扫描是用来降低算法的复杂度的方法，通过扫描线在平面上按照给定的轨迹移动的同时，不断根据扫描线扫过的部分更新信息，从而得到整体所要求的结果，既可以从左向右平移与y轴平行的直线，也可以固定射线的端点逆时针转动。

> 平面上有N NN个两两没有公共点的圆，i ii号圆的圆心在(x_i, y_i),半径为r_i。求所有最外层的，即不包含与其他圆内部的圆。

> 利用垂直于x轴的线去从左往右扫描，观察点为圆的左右端点。因为圆是没有公共点的，这使得包含的判断变得简单。如果某个圆被包含那么一定是被最近的两个外圈的其中一个圆包含（y轴上），假设这个圆已经被包含，如果远的圆如果想包含这个圆必须把两个圆都包住，那么第一层包裹已经不再是外层。所以如果存在被包括在某个外层里，一定是被包裹在最近的两个外层的其中一个。

```
typedef long long ll;
typedef pair<double, int>PDL;
const int N = 50007, M = 5000007, INF = 0x3f3f3f3f;
set<PDL>out;
vector<int>ans;
int n, m;
int cnt, num;
PDL p[N * 2];
double x[N], y[N], r[N];
bool inside(int i, int j)
{
    double dx = x[i] - x[j];
    double dy = y[i] - y[j];
    return dx * dx + dy * dy ≤ r[j] * r[j];
}

int main()
{
    scanf("%d", &n);

    for(int i = 1; i ≤ n; ++ i){
        scanf("%lf%lf%lf", &r[i], &x[i], &y[i]);
        p[++num].first = x[i] - r[i];//左端点
        p[num].second = i;
        p[++num].first = x[i] + r[i];//右端点
        p[num].second = i + n;
```

```
    }
    sort(p + 1, p + 1 + num);

    for(int i = 1; i ≤ num; ++ i){
        int id = p[i].second;
        if(id ≤ n){//左端点
            set<PDL> :: iterator it = out.lower_bound(make_pair(y[id], id));
            if(it ≠ out.end() && inside(id, it→second))continue;//被包含，不是答案，跳过
            if(it ≠ out.begin() && inside(id, (--it)→second))continue;
            ans.push_back(id);
            out.insert(make_pair(y[id], id));//外圈
        }
        else {//到了这个圆的右端点该删除了，已经没用了，留着会影响答案
            id -= n;
            out.erase(make_pair(y[id], id));
        }
    }
    printf("%d\n", ans.size());
    sort(ans.begin(),ans.end());
    for(int i = 0; i < ans.size(); ++i)
        printf("%d ", ans[i]);

    return 0;
}
```

$\oint$ 8. 凸包

过多边形的任意一边做一条直线，如果其他各个顶点都在这条直线的同侧，则把这个多边形叫做凸多边形。

凸包求解算法的基础便是凸多边形的定义与性质。

假设平面上n个点，过某些点作一个多边形，使这个多边形能把所有点都“包”起来。当这个多边形是凸多边形的时候，我们就叫它“凸包”。

假设每种颜料都拥有(R,G,B)三种属性，表示该种颜料红色，绿色，与蓝色的化学成分所占的比重

给你若干种已有的不限量的颜料，问是否能够勾兑出目标颜料(R0,G0,B0)

将每一种颜料映射为二维欧氏空间中的一个点，我们可以将已经给定的颜料与目的颜料在空间中标定出来

经过观察与思考，我们可以发现，一个颜料能够被勾兑出来当且仅当该颜料对应的点，位于以给定颜料所构成的凸包之中

8.1 Andrew算法

判断这样连是否为凸多边形的一部分：
如果p2-p3的斜率小于p1-p3，那么我们就没必要选择p2这个点，而是直接走p1~p3即可，也就是说我们把已经放进去的p2踢掉

```
//计算凸包，输入点数组p，个数为p输出点数组ch，函数返回凸包顶点个数。
//输入不能有重复的点，函数执行完后的输入点的顺序将被破坏（因为要排序，可以加一个数组存原来的id）
//如果不希望在凸包边上有输入点，把两个≤改成<即可
int ConvexHull(Point* p, int n, Point* ch)
{
    sort(p, p + n);
    int m = 0;
    for(int i = 0; i < n; ++ i){//下凸包
        //如果叉积≤0说明新边斜率小说明已经不是凸包边了，赶紧踢走
        while(m > 1 && Cross(ch[m - 1] - ch[m - 2], p[i] - ch[m - 2]) ≤ 0)m -- ;
        ch[m ++ ] = p[i];
    }
    int k = m;
    for(int i = n - 2; i ≥ 0; -- i){//上凸包
```

```
        while(m > k && Cross(ch[m - 1] - ch[m - 2], p[i] - ch[m - 2]) ≤ 0)m -- ;
        ch[m ++ ] = p[i];
    }
    if(n > 1) m -- ;
    return m;
}
```

8.2 直线两点式转为一般式使用公式O(1)计算点到直线距离

> 平面上有n个点，求一条直线，使得这n个点都在这条直线上或同侧，且每个点到该直线的距离之和尽量小。

首先，一条直线不分割这n个点当且仅当不分割这n个点的凸包。并且为了使这n个点到该直线的距离最小，这条直线应是凸包上某一条边所在直线。
由几何知识可得，对于点$P(x_0,y_0)$和直线$Ax+By+C=0$之间的距离

$$dis=\frac{|Ax_0+By_0+C|}{\sqrt{A^2+B^2}}$$

又因为这n个点在这条直线的同侧，所以对于所有的点Ax_0+By_0+C符号相同，所以预处理出所有点x坐标和y坐标的和就**可以$\Theta(1)$算出每个点到该直线的距离之和。**

还有一个问题，如何将直线的两点式
$l_{PQ},P(x_P,y_P),Q(x_Q,y_Q)$转化为一般式$Ax+By+C=0$

简单计算可得一个解为
$A=y_Q-y_P$
$B=x_P-x_Q$
$C=-x_P*A-y_P*B$

注意千万不能用除法，否则会计算结果会特别大或NaN导致WA.

```
//Ax + By + C = 0;
double get(double A, double B, double C) {
    double k = fabs(A*sumx + B*sumy + n*C);
    double v = sqrt(A*A + B*B);//v ≠ 0;
    return k/v;
}

double getDist(const point &a, const point &b) {
    double A = a.y-b.y;
    double B = b.x-a.x;
    double C = a.x*b.y - a.y*b.x;
    return get(A, B, C);
}

int main()
{
    int counter = 0;
    int T;
    point t;
    scanf("%d", &T);
    while(T--) {
        init();
        scanf("%d", &n);
        for(int i = 0; i < n; i++) {
            scanf("%lf%lf", &t.x, &t.y);
            sumx += t.x; sumy += t.y;
            tmp.push_back(t);
        }
        polygon_convex tres = convex_hull(tmp);
        int Size = (int)tres.P.size();
        printf("Case #%d: ", ++counter);
        if(Size == 2 || Size == 1) { //刚开始wa一次，看了看题目n>0.又把n=1考虑一下。
```

```
            printf("0.000\n");
            continue;
        }
        for(int i = 0; i < Size; i++) {
           double temp = getDist(tres.P[i], tres.P[(i+1)%Size]);
           ans = min(ans, temp);
        }
        printf("%.3lf\n", ans/n);
    }
    return 0;
}
```

8.3 判断点是否在凸包内

先判定和边界的关系
然后找到与其极角相邻的两点，凭此判断
须保证A[1]=(0,0)

```
ll in(Node a)
{
    if(a*A[1]>0||A[tot]*a>0) return 0;
    ll ps=lower_bound(A+1,A+tot+1,a,cmp2)-A-1;
    return (a-A[ps])*(A[ps%tot+1]-A[ps])≤0;
}
```

$\oint$ 9. 旋转卡壳

利用凸包上一些奇妙的单调性，求解

- 多边形直径
- 多边形宽度
- 最小面积矩形覆盖

复杂度$O(n)$

```
for(int i=1,p=1;i≤n;i++)
{
    //这份代码只卡了一个点，还可以同时卡多个点
    while(H(A[i],A[i+1],A[p%n+1])≥H(A[i],A[i+1],A[p])) p=p%n+1;
    Ans=max(Ans,max((A[p]-A[i]).dis(),(A[p]-A[i+1]).dis()));
}
```

9.1 求凸包的直径

> 给定平面上 n 个点，求凸包直径。
> 第一行一个正整数 n。接下来 n 行，每行两个整数 x,y，表示一个点的坐标。输出一行一个整数，表示答案的平方。

```
#include<bits/stdc++.h>
using namespace std;
const int N = 50007, M = 50007, INF = 0x3f3f3f3f;
const double DINF = 12345678910, eps = 1e-10, PI = acos(-1);
struct Point{
    int x, y;
    Point(double x = 0, double y = 0):x(x), y(y){ }//构造函数
};
typedef Point Vector;
Vector operator + (Vector A, Vector B){return Vector(A.x + B.x, A.y + B.y);}
Vector operator - (Point A, Point B){return Vector(A.x - B.x, A.y - B.y);}
```

```cpp
Vector operator * (Vector A, double p){return Vector(A.x * p, A.y * p);}
Vector operator / (Vector A, double p){return Vector(A.x / p, A.y / p);}
bool operator < (const Point& a, Point& b) {return a.x < b.x || (a.x == b.x && a.y < b. y);}
int dcmp(double x){
    if(fabs(x) < eps) return 0;
    else return x < 0 ? -1 : 1;
}
bool operator == (const Point& a, const Point& b){return !dcmp(a.x - b.x) && !dcmp(a.y - b.y);}
double Polar_angle(Vector A){return atan2(A.y, A.x);}
inline double D_to_R(double D)//角度转弧度
{ return PI/180*D; }
double Cross(Vector A, Vector B){return A.x * B.y - B.x * A.y;}
Vector Rotate(Vector A, double rad){
    return Vector(A.x * cos(rad) - A.y * sin(rad), A.x * sin(rad) + A.y * cos(rad));
}
Point Get_line_intersection(Point P,Vector v,Point Q,Vector w)
{
    Vector u = P - Q;
    double t = Cross(w, u) / Cross(v, w);
    return P + v * t;
}
double convex_polygon_area(Point* p, int n)
{
    double area = 0;
    for(int i = 1; i ≤ n - 2; ++ i)
        area += Cross(p[i] - p[0], p[i + 1] - p[0]);
    return area / 2;
}
int ConvexHull(Point* p, int n, Point* ch)
{
    sort(p, p + n);
    int m = 0;
    for(int i = 0; i < n; ++ i){//下凸包
        while(m > 1 && Cross(ch[m - 1] - ch[m - 2], p[i] - ch[m - 2]) ≤ 0)m -- ;
        ch[m ++ ] = p[i];
    }
    int k = m;
    for(int i = n - 2; i ≥ 0; -- i){//上凸包
        while(m > k && Cross(ch[m - 1] - ch[m - 2], p[i] - ch[m - 2]) ≤ 0)m -- ;
        ch[m ++ ] = p[i];
    }
    if(n > 1) m -- ;
    return m;
}
int get_dist (const Point &x){return x.x * x.x + x.y * x.y;}
Point p[N], con[N];
int con_num;
double Rotating_calipers()
{
    int op = 1, ans = 0;
    for(int i = 0; i < con_num; ++ i){
        while(Cross((con[i] - con[op]), (con[i + 1] - con[i])) < Cross((con[i] - con[op + 1]), (con[i + 1] - con[i])))
        //（写成≤会被两个点的数据卡掉，所以必须写成<）
            op = (op + 1) % con_num;
            ans = max(ans, max(get_dist(con[i] - con[op]), get_dist(con[i + 1] - con[op])));
    }
    printf("%d\n", ans);
    return ans;
}
```

```
int n ;
int main()
{
    scanf("%d", &n);
    for(int i = 0; i < n; ++ i){
        scanf("%d%d", &p[i].x, &p[i].y);
    }
    con_num = ConvexHull(p, n, con);
    double res = Rotating_calipers();
    return 0;
}

```

9.2 求两个凸多边形(凸包)间最小距离

给你两个凸多边形，求这两个凸多边形间最小距离。
先分别选出两凸包最上点和最下点，从这两点开始向逆时针方向旋转卡壳。用叉乘判断是否旋转旋转，具体操作跟求凸包直径差不多。

```
#include <bits/stdc++.h>
using namespace std;
const double eps = 1e-8;
const int inf = 0x3f3f3f3f;
const int maxn = 10010;

struct Point{
    double x, y;
    Point(double x = 0, double y = 0): x(x), y(y){}
};
typedef Point Vector;

Vector operator - (Point a, Point b) {
    return Vector(a.x - b.x, a.y - b.y);
}
Vector operator + (Vector a, Vector b) {
    return Vector(a.x + b.x, a.y + b.y);
}
Vector operator * (Vector a, double p) {
    return Vector(a.x * p, a.y * p);
}
Vector operator / (Vector a, double p) {
    return Vector(a.x / p, a.y / p);
}
bool operator < (Point a, Point b) {
    return a.x < b.x || a.x == b.x && a.y < b.y;
}
int dcmp(double x) {
    if(fabs(x) < eps) return 0;
    return x < 0 ? -1 : 1;
}

bool operator == (Point a, Point b) {
    return dcmp(a.x - b.x) == 0 && dcmp(a.y - b.y) == 0;
}
double Dot(Vector a, Vector b) {
    return a.x * b.x + a.y * b.y;
}
double Cross(Vector a, Vector b) {
    return a.x * b.y - a.y * b.x;
}
```

```cpp
double Length(Vector a) {
    return sqrt(Dot(a, a));
}
double Angle(Vector a) {
    return atan2(a.y, a.x);
}
double DistanceToSegment(Point p, Point a, Point b) {
    if(a == b) return Length(p-a);
    Vector v1 = b - a, v2 = p - a, v3 = p - b;
    if(dcmp(Dot(v1, v2)) < 0) return Length(v2);
    if(dcmp(Dot(v1, v3)) > 0) return Length(v3);
    return fabs(Cross(v1, v2)) / Length(v1);
}
double SegmentDistance(Point a1, Point b1, Point a2, Point b2) {
    return min(min(DistanceToSegment(a1, a2, b2), DistanceToSegment(b1, a2, b2)),
               min(DistanceToSegment(a2, a1, b1), DistanceToSegment(b2, a1, b1)));
}
//两凸包的最短距离
double RotateCalipers(Point *p1, int n, Point *p2, int m) {
    int minp1 = 0, maxp2 = 0;
    for(int i = 0; i < n; i++){
        if(p1[i].y < p1[minp1].y) minp1 = i;
    }
    for(int i = 0; i < m; i++){
        if(p2[i].y > p2[maxp2].y) maxp2 = i;
    }
    p1[n] = p1[0];
    p2[m] = p2[0];
    double dis = inf, temp;
    for(int i = 0; i < n; i++) {
        while(dcmp(Cross(p1[minp1] - p1[minp1+1], p2[maxp2+1] - p1[minp1+1]) - Cross(p1[minp1] - p1[minp1+1], p2[maxp2] - p1[minp1+1])) < 0) maxp2 = (maxp2+1) % m;
        dis = min(dis, SegmentDistance(p1[minp1], p1[minp1+1], p2[maxp2], p2[maxp2+1]));
        minp1 = (minp1+1) % n;
    }
    return dis;
}
Point p1[maxn], p2[maxn];
int main(){
    int n, m;
    //freopen("in.txt", "r", stdin);
    while(scanf("%d %d", &n, &m) && (n+m)) {
        for(int i = 0; i < n; i++){
            scanf("%lf %lf", &p1[i].x, &p1[i].y);
        }
        for(int i = 0; i < m; i++){
            scanf("%lf %lf", &p2[i].x, &p2[i].y);
        }
        printf("%lf\n", RotateCalipers(p1, n, p2, m));
    }
    return 0;
}
```

∮ 10. 半平面交

定义一个半平面为一个向量的左侧，半平面交即为若干个半平面的交集。

10.1 有向直线

```

//有向直线。它的左边就是对应的半平面
struct Line
{
    Point P;//直线上任意一点
    Vector v;//方向向量。左边就是对应的半平面
    double deg;//极角
    Line(){}
    Line(Point P, Vector v):P(P), v(v){deg = atan2(v.y, v.x);}
    bool operator < (const Line& L)const {//排序时使用的比较运算符
        return deg < L.deg;
    }
};
```

10.2 $O(nlogn)$的半平面交

```
//半平面交一般是一个凸多边形，但是有时候会是一个无界多边形
//甚至会是一个直线、线段、点，但是结果一定是凸的

//点p在有向直线L的左边（线上的不算）（叉积大于0a在b的左侧，小于0在右侧[sin夹角]）
bool on_left(Line L, Point P){return Cross(L.v, P - L.P) > 0;}
//两个有向直线的交点/假定交点唯一存在
Point get_intersection(Line a, Line b)
{
    Vector u = a.P - b.P;
    double t = Cross(b.v, u) / Cross(a.v, b.v);
    return a.P + a.v * t;
}
//半平面交的主过程
//L数组存所有有向直线，n为有向直线个数，半平面交的交点存在poly数组中
int half_plane_intersection(Line* L, int n, Point* poly)
{
    sort(L, L + n);//按照极角排序

    int first, last;//双端队列
    Point *p = new Point[n];//p[i]为q[i]和q[i + 1]的交点
    Line *q = new Line[n];//手写的Line类型的双端队列（数组）
    q[first = last = 0] = L[0];//双端队列初始化的时候只有一个半平面L[0]
    for(int i = 1; i < n; ++ i){
        while(first < last && !on_left(L[i], p[last - 1]))last -- ;
        while(first < last && !on_left(L[i], p[first]))first ++ ;
        q[++ last] = L[i];//新的点是一定要放进去的
        if(fabs(Cross(q[last].v, q[last - 1].v)) < eps){
        //相邻的两个向量平行且同向，取内侧的那一个
            last -- ;//如果新的向量上的一个点在老的向量的左侧就取新的
            if(on_left(q[last], L[i].P))q[last] = L[i];
        }
        if(first < last)p[last - 1] = get_intersection(q[last - 1], q[last]);
    }
    while(first < last && !on_left(q[first], p[last - 1]))last -- ;
    //删除无用的平面

    if(last - first ≤ 1)return 0;//空集
    p[last] = get_intersection(q[last], q[first]);//计算首尾两个半平面交（环状）
    //从手写deque中复制答案到输出数组中
    int m = 0;
    for(int i = first ; i ≤ last; ++ i)poly[m ++ ] = p[i];
```

```
    return m;
}
```

> 逆时针给出n个凸多边形的顶点坐标，求它们交的面积。

10.3 二分 + 半平面交

> 在大海的中央，有一个凸n边形的小岛，求出岛上离海边最远的一个点到海边的距离，保留6位小数。

```
//有向直线。它的左边就是对应的半平面
struct Line
{
    Point P;//直线上任意一点
    Vector v;//方向向量。左边就是对应的半平面
    double deg;//极角
    Line(){}
    Line(Point P, Vector v):P(P), v(v){deg = atan2(v.y, v.x);}
    bool operator < (const Line& L)const {//排序时使用的比较运算符
        return deg < L.deg;
    }
};
bool on_left(Line L, Point P){return Cross(L.v, P - L.P) > 0;}
//两个有向直线的交点/假定交点唯一存在
Point get_intersection(Line a, Line b)
{
    Vector u = a.P - b.P;
    double t = Cross(b.v, u) / Cross(a.v, b.v);
    return a.P + a.v * t;
}
//半平面交的主过程
int half_plane_intersection(Line* L, int n, Point* poly)
{
    sort(L, L + n);//按照极角排序

    int first, last;//双端队列
    Point *p = new Point[n];//p[i]为q[i]和q[i + 1]的交点
    Line *q = new Line[n];//手写的Line类型的双端队列（数组）
    q[first = last = 0] = L[0];//双端队列初始化的时候只有一个半平面L[0]
    for(int i = 1; i < n; ++ i){
        while(first < last && !on_left(L[i], p[last - 1]))last -- ;
        while(first < last && !on_left(L[i], p[first]))first ++ ;
        q[++ last] = L[i];//新的点是一定要放进去的
        if(fabs(Cross(q[last].v, q[last - 1].v)) < eps){
        //相邻的两个向量平行且同向，取内侧的那一个
            last -- ;//如果新的向量上的一个点在老的向量的左侧就取新的
            if(on_left(q[last], L[i].P))q[last] = L[i];
        }
        if(first < last)p[last - 1] = get_intersection(q[last - 1], q[last]);
    }
    while(first < last && !on_left(q[first], p[last - 1]))last -- ;
    //删除无用的平面

    if(last - first ≤ 1)return 0;//空集
    p[last] = get_intersection(q[last], q[first]);//计算首尾两个半平面交（环状）
    //从手写deque中复制答案到输出数组中
    int m = 0;
    for(int i = first ; i ≤ last; ++ i)poly[m ++ ] = p[i];
    return m;
}
Point p[N], poly[N];
```

```
Line L[N];
Vector v[M], v2[M];
int n;

int main()
{
    while(scanf("%d", &n) ≠ EOF && n)
    {
        int m, x, y;
        for(int i = 0; i < n; ++i){
            scanf("%d%d", &x, &y);
            p[i] = Point(x, y);
        }
        for(int i = 0; i < n; ++ i){
            v[i] = p[(i + 1) % n] - p[i];
            v2[i] = Normal(v[i]);//单位法向量
        }
        double l = 0, r = 20000;
        while(r - l > eps){
            double mid = l + (r - l) / 2;
             for(int i = 0; i < n; ++ i)
                L[i] = Line(p[i] + v2[i] * mid, v[i]);
            //收缩/放大整个凸多边形
            //点+单位向量*长度=平移后的点
            m = half_plane_intersection(L, n, poly);
            if(!m)r = mid;
            else l = mid;
        }
        printf("%.6f\n", l);
    }
    return 0;
}

```

∮ 11. 闵可夫斯基和

> 定义p+q=(p.x+q.x,p.y+q.y)，给定两个点集，求{pi+qj}的凸包（凸壳）的问题

```
Vector V1[N],V2[N];
inline int Mincowski(Point *P1,Re n,Point *P2,Re m,Vector *V){//【闵可夫斯基和】求两个凸包{P1},{P2}的向量集合{V}={P1+P2}构成的凸包
    for(Re i=1;i≤n;++i)V1[i]=P1[i<n?i+1:1]-P1[i];
    for(Re i=1;i≤m;++i)V2[i]=P2[i<m?i+1:1]-P2[i];
    Re t=0,i=1,j=1;V[++t]=P1[1]+P2[1];
    while(i≤n&&j≤m)++t,V[t]=V[t-1]+(dcmp(Cro(V1[i],V2[j]))>0?V1[i++]:V2[j++]);
    while(i≤n)++t,V[t]=V[t-1]+V1[i++];
    while(j≤m)++t,V[t]=V[t-1]+V2[j++];
    return t;
}
```

∮ 12.平面区域

当平面上有很多线段时，组成的图形往往不止一个多边形，而是一个平面直线图（PSLG），它代表一个平面区域划分，其中一个区域是一个多边形。

如果只有点和边的信息，如何找出所有区域呢？为方便起见，我们把每条边u-v拆成两条半边 u-v 和 v-u.并且每条半边只与它左边的面相邻。接下来，我们从一条半边出发遍历，每次像卷包裹算法那样找一个逆时针转的尽量多的边作为下一条边，直到回到出发的那条半边。

程序实现上可以把边表扩大一倍，让编号为i的半边的反向边的编号为i^1.

首先看一下边的存储结构定义：

```
//边
struct Edge
{
    int from, to; //起点，终点
    double ang;    //极角
};
```

平面直线图：

```
//平面直线图。
struct PSLG
{
    int n, m;               //顶点数 边数
    int face_cnt;           //面数
    double x[maxn];         //顶点
    double y[maxn];
    vector<Edge> edges;
    vector<int> G[maxn];
    bool vis[maxn*2];       // 每条边是否被访问过
    int left[maxn*2];       //左面的编号
    int prev[maxn*2];       //相同起点的上一条边，即顺时针旋转碰到的下一条边的编号
    vector<Polygon> faces;  //面
    double areas[maxn];     //每个polygon的面积

    void init(int n)
    {
        this->n = n;
        edges.clear();
        faces.clear();
        for(int i = 0; i < n; i++)
            G[i].clear();
    }

    //有向线段from-to的极角
    double getAngle(int from, int to)
    {
        return atan2(y[to]-y[from], x[to]-x[from]);
    }

    void AddEdge(int from, int to)
    {
        edges.push_back((Edge){from, to, getAngle(from, to)});
        edges.push_back((Edge){to, from, getAngle(to, from)});
        m = edges.size();
        G[from].push_back(m-2);
        G[to].push_back(m-1);
    }

    //找出faces并计算面积
    void Build()
    {
        int u, i, j, d, from, e;
        //给从u开始个各条边按照极角排序
        for(u = 0; u < n; u++)
```

```
        {
            d = G[u].size();
            for(i = 0; i < d; i++)
                for(j = i+1; j < d; j++)    //这里假设从u出发的线段不会太多
                    if(edges[G[u][i]].ang > edges[G[u][j]].ang)
                        swap(G[u][i], G[u][j]);
            for(i = 0; i < d; i++)
                prev[G[u][(i+1)%d]] = G[u][i];
        }

        face_cnt = 0;
        memset(vis, false, sizeof(vis));
        for(u = 0; u < n; u++)
            for(i = 0; i < G[u].size(); i++)
            {
                e = G[u][i];
                if(!vis[e])                                 //逆时针找圈
                {
                    face_cnt++;                             //找到一个新的面
                    Polygon poly;
                    for(;;)
                    {
                        vis[e] = true;
                        left[e] = face_cnt;
                        from = edges[e].from;
                        poly.push_back(Point(x[from], y[from]));  //把新的点加入面中
                        e = prev[e ^ 1];                        //e^1为反向边，然后prev就是需要走的下一条边
                        if(e == G[u][i])                        //回到了起始边
                            break;
                    }
                    faces.push_back(poly);
                }
            }
        //对于连通图，惟一一个面积小于零的面是无限面
        //对于内部区域来说，无限面多边形的各个顶点是顺时针的
        for(i = 0; i < faces.size(); i++)                   //计算各个面的面积
            areas[i] = PolygonArea(faces[i]);
    }
};
```

∮ 13.平面最近点对

> 给定平面上n个点，找出其中的一对点的距离，使得在这n个点的所有点对中，该距离为所有点对中最小的

```
struct Point
{
    double x, y;
    Point(double x = 0, double y = 0):x(x), y(y){ }
};
Point p[N];
int tmp[N], n;
bool cmp(const Point &a, const Point &b){
        if(a.x == b.x)return a.y < b.y;
        return a.x < b.x;
}
bool cmps(const int &a, const int &b){
    return p[a].y < p[b].y;
}
```

```cpp
double get_dist(int i, int j)
{
    return sqrt((p[i].x - p[j].x) * (p[i].x - p[j].x) + (p[i].y - p[j].y) * (p[i].y - p[j].y));
}
double merge(int l, int r)
{
    double d = DINF;
    if(l == r)return d;
    if(l + 1 == r)return get_dist(l, r);
    int mid = l + r >> 1;
    double d1 = merge(l, mid), d2 = merge(mid + 1, r);
    d = min(d1, d2);
    int tot = 0;
    for(int i = l; i ≤ r; ++ i)
        if(fabs(p[mid].x - p[i].x) ≤ d)
            tmp[++ tot] = i;
    sort(tmp + 1, tmp + 1 + tot, cmps);

    for(int i = 1; i ≤ tot; ++ i)
        for(int j = i + 1; j ≤ tot; ++ j){
            if(fabs(p[tmp[j]].y - p[tmp[i]].y) > d)break;
            d = min(d, get_dist(tmp[i], tmp[j]));
        }
    return d;
}
int main()
{
    scanf("%d", &n);
    for(int i = 1; i ≤ n; ++ i)
        scanf("%lf%lf", &p[i].x, &p[i].y);
    sort(p + 1, p + 1 + n, cmp);
    double ans = merge(1, n);
    printf("%.4f\n", ans);
    return 0;
}

```

> 题意实际上就是给定长度为 n 的一串序列$a_1, a_2, \ldots, a_n$，找到两个正整数$i, j \in [1, n]$，求$(i-j)^2 + (\sum_{k=i+1}^{j} a_k)^2$的最小值
>
> 分析
>
> 将原式中的$\sum_{k=i+1}^{j} a_k$用前缀和$S_j - S_i$替代，则原式变换为$(i-j)^2 + (S_j - S_i)^2$
>
> 不难看出变换后的式子为两点之间欧几里德距离的平方，于是原题转化为求平面上的最近点对距离的平方。
>
> 关于最近点对距离的求解可使用二分法

注意本题的数据会卡普通的$O(nlogn)$的分治求平面最近点对算法，我们需要加一些玄学优化，比如在筛y轴的时候一旦大于d就直接break，因为是排过序的，当前的y都大了，说明接下来的所有点都不能用了。

```cpp
struct Point {
    ll x, y;
    bool operator < (const Point& t)const {
        return y < t.y;
    }
};

Point v[N], tmp[N];
int n;

ll get_dist(Point A, Point B)
{
```

```
    return (ll)(A.x - B.x) * (A.x - B.x) + (A.y - B.y) * (A.y - B.y);
}

ll solve(const int l, const int r){//求平面最近点对的分治算法
    if(l == r)return 0x3f3f3f3f3f;
    if(l + 1 == r)return get_dist(v[l], v[r]);//分治到了一个点对，直接返回答案
    int mid = (l + r) >> 1;
    ll d1 = solve(l, mid), d2 = solve(mid + 1, r);
    ll d = min(d1, d2);
    int tot = 0;
    //先筛选x
    for(int i = l; i ≤ r; ++ i){
        if((v[mid].x - v[i].x) * (v[mid].x - v[i].x) ≤ d)tmp[ ++ tot] = v[i];
    }
    sort(tmp + 1, tmp + tot + 1);//按y排序
    //再筛选y
    for(int i = 1; i ≤ tot; ++ i){
        for(int j = i + 1; j ≤ tot; ++ j){
            if((tmp[i].y - tmp[j].y) * (tmp[i].y - tmp[j].y) > d)break;//剪枝优化，不加过不了本题
            d = min(d, get_dist(tmp[i], tmp[j]));
        }
    }
    return d;
}

int main()
{
    scanf("%d", &n);
    for(int i = 1; i ≤ n; ++ i){
        int x;
        scanf("%d", &x);
        v[i].y = v[i - 1].y + x;//y就是前缀和，我们通过公式推出来的
        v[i].x = i;
    }
    ll minv = solve(1, n);
    printf("%lld\n", minv);
    return 0;
}

```

更快的神仙操作

```
@3A17K巨佬的神仙操作!
我们充分发扬人类智慧:
将所有点全部绕原点旋转同一个角度，然后按 x 坐标排序
根据数学直觉，在随机旋转后，答案中的两个点在数组中肯定不会离得太远
所以我们只取每个点向后的5个点来计算答案
这样速度快得飞起，在 n=1000000 时都可以在1s内卡过
注意旋转θ角时坐标变换
x'=xcosθ-ysinθ
y'=xsinθ+ycosθ
代码如下:
#include<cstdio>
#include<cmath>
#include<iostream>
#include<algorithm>
using namespace std;
const int N=2e5+50;
#define D double
struct spot{
```

```
    D a[4];
}p[N];
D x,y,x_,y_,z,w,ans;
int n;
bool mmp(const spot &u,const spot &v){
    return u.a[0]<v.a[0];
}
int main(){
    scanf("%d",&n);
    z=sin(1),w=cos(1);  //旋转1弧度≈57°
    for(int i=1;i≤n;i++){
        scanf("%lf%lf",&x,&y);
        x_=x*w-y*z;
        y_=x*z+y*w;   //计算旋转后的坐标
        p[i].a[0]=x_;
        p[i].a[1]=y_;
        p[i].a[2]=x;
        p[i].a[3]=y;   //存下来
    }
    sort(p+1,p+n+1,mmp);   //排序
    for(int i=n+1;i≤n+10;i++)
    p[i].a[0]=p[i].a[1]=-N-0.01;  //边界处理
    ans=2e9+0.01;  //初始化答案
    for(int i=1;i≤n;i++)
    for(int j=1;j≤5;j++){  //枚举
        x=p[i].a[2];y=p[i].a[3];
        x_=p[i+j].a[2];y_=p[i+j].a[3];
        z=sqrt((x-x_)*(x-x_)+(y-y_)*(y-y_));  //计算距离
        if(ans>z)ans=z;   //更新答案
    }
    printf("%.4lf\n",ans);  //输出
}
```

∮ 14. 三维基础操作

```
const double EPS=0.000001;

typedef struct Point_3D {
    double x, y, z;
    Point_3D(double xx = 0, double yy = 0, double zz = 0): x(xx), y(yy), z(zz) {}

    bool operator == (const Point_3D& A) const {
        return x==A.x && y==A.y && z==A.z;
    }
};
typedef Point_3D Vector_3D;

struct Line_3D   //空间直线
{
    Point_3D a, b;
};

struct Plane_3D   //空间平面
{
    Point_3D a, b, c;
    Plane_3D(){}
    Plane_3D( Point_3D a, Point_3D b, Point_3D c ):
    a(a), b(b), c(c) { }
```

```
};

Point_3D read_Point_3D() {
    double x,y,z;
    scanf("%lf%lf%lf",&x,&y,&z);
    return Point_3D(x,y,z);
}

Vector_3D operator + (const Vector_3D & A, const Vector_3D & B) {
    return Vector_3D(A.x + B.x, A.y + B.y, A.z + B.z);
}

Vector_3D operator - (const Point_3D & A, const Point_3D & B) {
    return Vector_3D(A.x - B.x, A.y - B.y, A.z - B.z);
}

Vector_3D operator * (const Vector_3D & A, double p) {
    return Vector_3D(A.x * p, A.y * p, A.z * p);
}

Vector_3D operator / (const Vector_3D & A, double p) {
    return Vector_3D(A.x / p, A.y / p, A.z / p);
}

//取平面法向量
Point_3D NormalVector( plane3 s )
{
    return Cross_3D( Subt( s.a, s.b ), Subt( s.b, s.c ) );
}
Point_3D NormalVector( Point3 a, Point3 b, Point3 c )
{
    return Cross_3D( Subt( a, b ), Subt( b, c ) );
}

//两点距离
double TwoPointDistance( Point3 p1, Point3 p2 )
{
    return sqrt( (p1.x - p2.x)*(p1.x - p2.x) + (p1.y - p2.y)*(p1.y - p2.y) + (p1.z - p2.z)*(p1.z - p2.z)
);
}

```

1.1 三维点积(Dot3)

```
double Dot_3D(const Vector_3D & A, const Vector_3D & B) {
    return A.x * B.x + A.y * B.y + A.z * B.z;
}

double Length(const Vector_3D & A) {
    return sqrt(Dot(A, A));
}

double Angle(const Vector_3D & A, const Vector_3D & B) {
    return acos(Dot(A, B) / Length(A) / Length(B));
}
```

1.2 三维叉积(Cross3)

可以认为叉积同时垂直于v1和v2，当且仅当v1和v2平行时，叉积为0。

```cpp
Vector_3D Cross(const Vector_3D & A, const Vector_3D & B) {
    return Vector_3D(A.y * B.z - A.z * B.y, A.z * B.x - A.x * B.z, A.x * B.y - A.y * B.x);
}
// 三角形abc面积的两倍
double Area2(const Point_3D & A, const Point_3D & B, const Point_3D & C) {
    return Length(Cross(B - A, C - A));
}
```

过不共线三点的平面，法向量为 `Cross(p2 - p0, p1 - p0)`，我们在任取一个点即可得到平面的点法式。

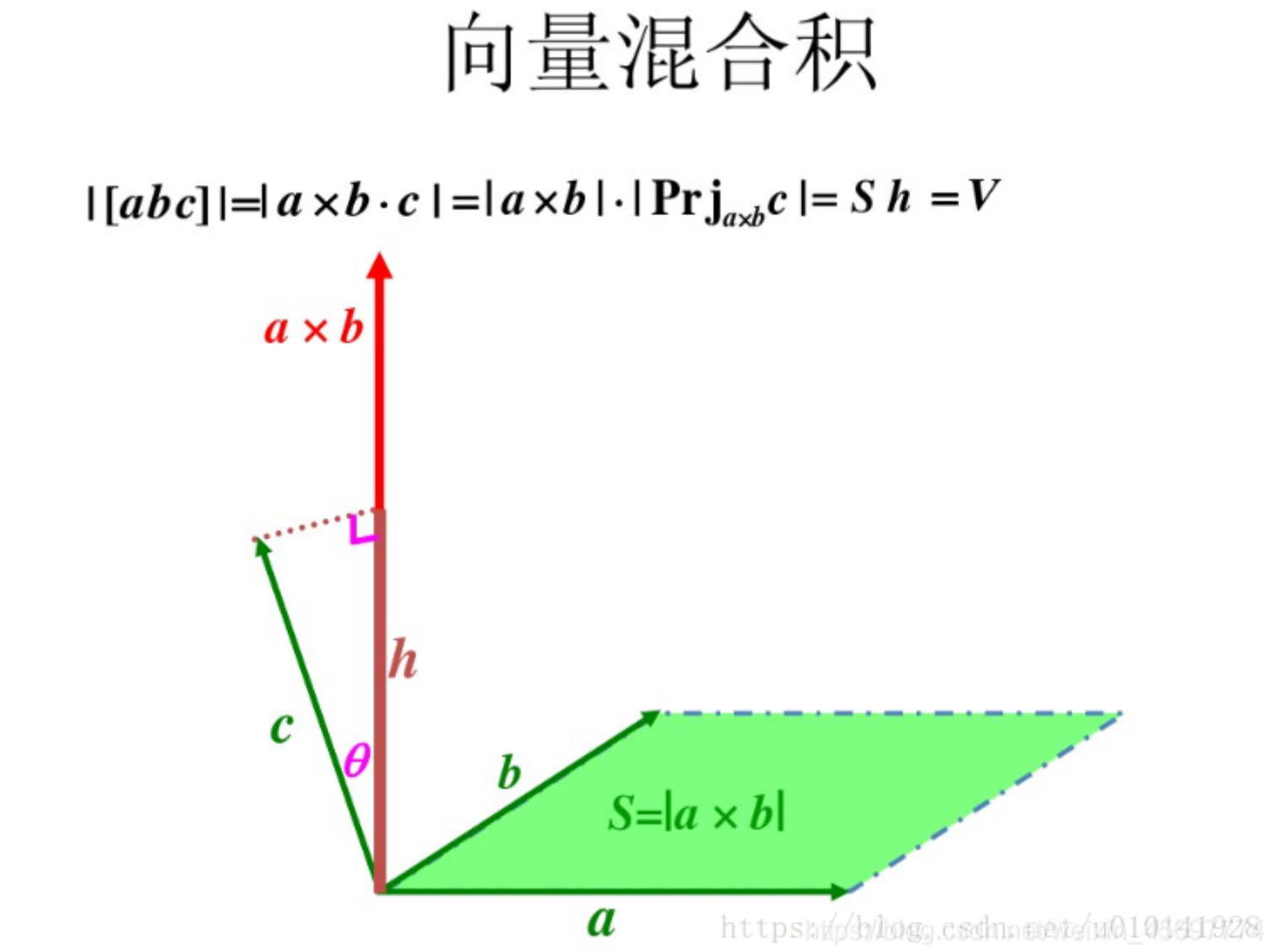
1.3 矢量差（Subt）

```cpp
Point_3D Subt( Point3 u, Point3 v )
{
    Point_3D ret;
    ret.x = u.x - v.x;
    ret.y = u.y - v.y;
    ret.z = u.z - v.z;
    return ret;
}
```

1.4.1 返回ab，ac，ad的混合积(Volume6)

对于三个三维向量a, b, c，定义它们的混合积为$(a \times b) \cdot c$，其中$\times$ 表示叉乘，$\cdot$表示点乘，记为$[a\ b\ c]$

几何意义

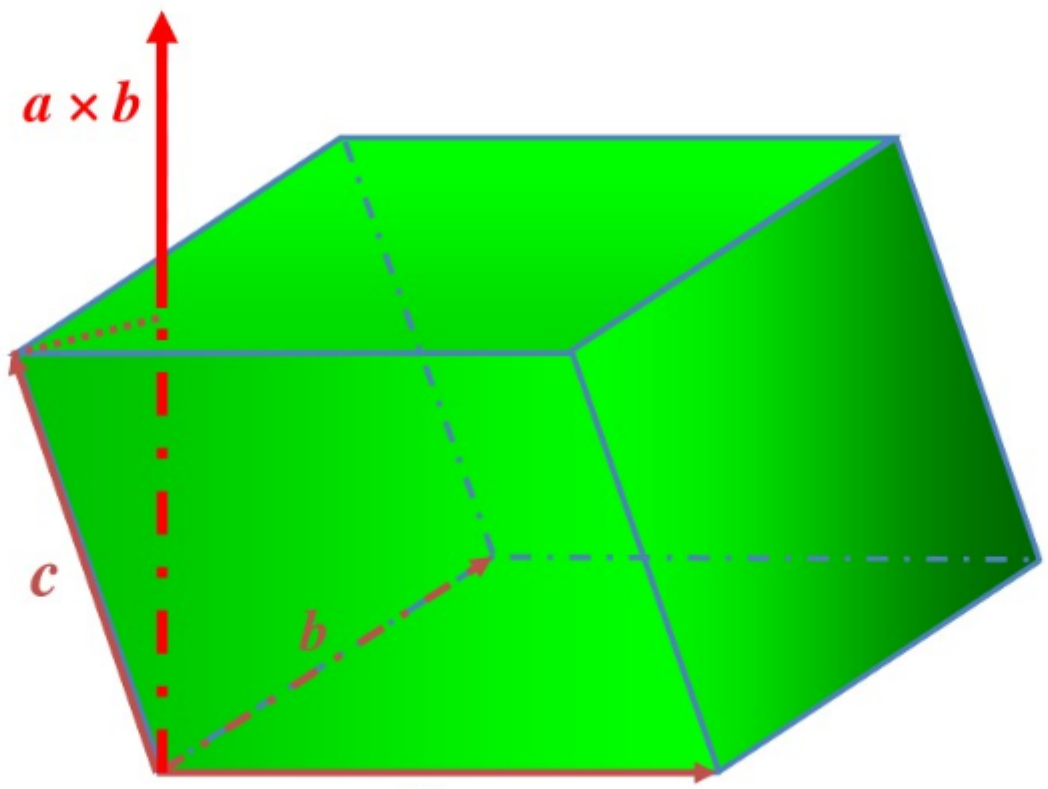


其中 $Prj_{a\times bc}$ 代表的是c这个向量在a×b这个向量上的投影

那么显然我们最后得到的是以这三个向量为三条临边的一个**六面体的体积**

几何意义 $|[abc]|=|a\times b\cdot c|=|a\times b|\cdot|\mathrm{Prj}_{a\times b}c|=Sh=V$

三矢 a, b, c共面 $\Leftrightarrow$ 其混合积 $[abc]=0$




```cpp
// 返回ab，ac，ad的混合积。它等于四面体(三角形体)abcd的有
//向体积的6倍(六面体(正方形体)的体积)
double Volume6(const Point_3D & A, const Point_3D & B, const Point_3D & C, const Point_3D & D) {
    return Dot(D - A, Cross(B - A, C - A));
}
```

1.4.2 四面体体积(Volume6)

假设四面体的四个顶点分别为A,B,C,D，并设三个向量$a=B-A, b=C-A, c=D-A$，那么这个正四面体的体积就是$\frac{1}{6}[abc]$(混合积)

```cpp
// 返回ab，ac，ad的混合积。它等于四面体(三角形体)abcd的有
//向体积的6倍(六面体(正方形体)的体积)
double Volume6(const Point_3D & A, const Point_3D & B, const Point_3D & C, const Point_3D & D) {
    return Dot(D - A, Cross(B - A, C - A));
}
```

1.5 求四面体的重心(Centroid)

```cpp
// 四面体的重心
Point_3D Centroid(const Point_3D & A, const Point_3D & B, const Point_3D & C, const Point_3D & D) {
    return (A + B + C + D) / 4.0;
}
```

1.6 凸多面体的重心

我们首先考虑凸多边形的重心

对于三角形，它的重心就是它所有坐标的平均值

那么对于凸多边形，我们把它三角剖分了，记第i块的重心为 a_i，面积为 m_i，那么凸多边形的重心就是

$\frac{\sum_{i=1}^{n} a_i \times m_i}{\sum_{i=1}^{n} m_i}$

那么凸多面体也差不多了，我们把它给四面体剖分了，然后也差不多按上面的算就好了

关于四面体剖分，具体的说我们在多面体中随便选一个点，比方说是p1，然后把每一个面给三角剖分，那么三角形就和选定的点构成了一个四面体。设vi表示第i个四面体的体积，ai表示重心，则最终多面体的重心为

$$\frac{\sum_{i=1}^{n} a_i \times v_i}{\sum_{i=1}^{n} v_i}$$

1.7 二面角

简单来说就是两个平面的夹角

我们假设现在有a,b,c三个向量，要求ab这个平面和ac这个平面的二面角

那么求出ab和ac的法向量（法向量可以直接用叉积算），两个法向量之间的夹角就是二面角了，法向量之间的夹角直接用点积除以长度计算(转换成单位法向量)

∮ 15. 三维点线面

三维的直线仍然可以用参数方程（点+向量表示），同时射线和线段为“参数有取值范围”的直线

```
struct Line_3D    //空间直线
{
    Point_3D a, b;
};
```

三维平面通常使用点法式(p_0, n)表示，其中p_0是平面上的一个点，向量n是平面的法向量。（我们的平面把空间分成了两个部分，一般取法向量所背离的半空间）

法向量垂直于平面上所有直线，意味着平面上的任意点p满足$Dot(n, p - p0) = 0$。

我们设$p(x, y), p_0(x_0, y_0)$，法向量$n(A, B, C)$

化简得到平面的一般式：$Ax + By + Cz - D = 0 (D = -(Ax_0 + By_0 + Cz_0))$

当$Ax + By + Cz - D > 0$时，上述点积大于0，说明点$p(x, y, z)$在版空间(p_0, n)之外。

```
struct Plane_3D    //空间平面
{
    Point_3D a, b, c;
    Plane_3D(){}
    Plane_3D( Point_3D a, Point_3D b, Point_3D c ):
    a(a), b(b), c(c) { }
};
```

2.1 取平面法向量(NormalVector)

```
Point3 NormalVector( plane_3D s )
{
    return Cross3( Subt( s.a, s.b ), Subt( s.b, s.c ) );
}
Point3 NormalVector( Point_3D a, Point_3D b, Point_3D c )
{
    return Cross3( Subt( a, b ), Subt( b, c ) );
}
```

2.2 求两点距离(TwoPointDistance)

```cpp
double TwoPointDistance( Point_3D p1, Point_3D p2 )
{
    return sqrt( (p1.x - p2.x)*(p1.x - p2.x) + (p1.y - p2.y)*(p1.y - p2.y) + (p1.z - p2.z)*(p1.z - p2.z)
);
}
```

2.3 点p到平面的距离(DistanceToPlane)

```cpp
// 点p到平面p0-n的距离。n必须为单位向量
double DistanceToPlane(const Point_3D & p, const Point_3D & p0, const Vector_3D & n)
{
    return fabs(Dot(p - p0, n)); // 如果不取绝对值，得到的是有向距离
}
```

2.4 点p在平面上的投影(GetPlaneProjection)

```cpp
// 点p在平面p0-n上的投影。n必须为单位向量(如果不是单位向量就/Length(n)嘛)
Point_3D GetPlaneProjection(const Point_3D & p, const Point_3D & p0, const Vector_3D & n)
{
    return p - n * Dot(p - p0, n);
}
```

2.5 直线与平面的交点(LinePlaneIntersection)

```cpp
//直线p1-p2 与平面p0-n的交点
Point_3D LinePlaneIntersection(Point_3D p1, Point_3D p2, Point_3D p0, Vector_3D n)
{
    Vector_3D v= p2 - p1;
    double t = (Dot(n, p0 - p1) / Dot(n, p2 - p1)); //分母为0，直线与平面平行或在平面上
    return p1 + v * t; //如果是线段 判断t是否在0~1之间
}
```

2.6 空间直线距离(LineToLine)

```cpp
//空间直线距离,tmp为两直线的公共法向量
double LineToLine( Line_3D u, Line_3D v, Point_3D& tmp )
{
    tmp = Cross3( Subt( u.a, u.b ), Subt( v.a, v.b ) );
    return fabs( Dot3( Subt(u.a, v.a), tmp ) ) / VectorLenth(tmp);
}
```

2.7 点到直线的距离(DistanceToLine)

```cpp
// 点P到直线AB的距离
double DistanceToLine(const Point_3D & P, const Point_3D & A, const Point_3D & B)
{
    Vector_3D v1 = B - A, v2 = P - A;
    return Length(Cross(v1, v2)) / Length(v1);
}
```

2.8 点到线段的距离(DistanceToSeg)

```cpp
double DistanceToSeg(Point_3D p, Point_3D a, Point_3D b)
{
    if(a == b)
```

```
        return Length(p - a);
    Vector_3D v1 = b - a, v2 = p - a, v3 = p - b;
    if(Dot(v1, v2) + EPS < 0)
        return Length(v2);
    else{
        if(Dot(v1, v3) - EPS > 0)
            return Length(v3);
        else
            return Length(Cross(v1, v2)) / Length(v1);
    }
}
```

2.9 求异面直线与的公垂线对应的s(LineDistance3D)

```
//求异面直线 p1+s*u与p2+t*v的公垂线对应的s 如果平行|重合，返回false
bool LineDistance3D(Point_3D p1, Vector_3D u, Point_3D p2, Vector_3D v, double & s)
{
    double b = Dot(u, u) * Dot(v, v) - Dot(u, v) * Dot(u, v);
    if(abs(b) ≤ EPS)
        return false;
    double a = Dot(u, v) * Dot(v, p1 - p2) - Dot(v, v) * Dot(u, p1 - p2);
    s = a / b;
    return true;
}
```

2.10 p1和p2是否在线段a-b的同侧(SameSide)

```
bool SameSide(const Point_3D & p1, const Point_3D & p2, const Point_3D & a, const Point_3D & b){
    return Dot(Cross(b - a, p1 - a), Cross(b - a, p2 - a)) - EPS ≥ 0;
}
```

2.11 判断点P是否在三角形中(PointInTri)

另法详见《训练指南》P288

```
//点P在三角形P0, P1, P2中
bool PointInTri(const Point_3D & P, const Point_3D & P0, const Point_3D & P1, const Point_3D & P2){
    return SameSide(P, P0, P1, P2) && SameSide(P, P1, P0, P2) && SameSide(P, P2, P0, P1);
}
```

2.12 三角形P0、P1、P2是否和线段AB相交(TriSegIntersection)

```
bool TriSegIntersection(const Point_3D & P0, const Point_3D & P1, const Point_3D & P2, const Point_3D &
A, const Point_3D & B, Point_3D & P)
{
    Vector_3D n = Cross(P1 - P0, P2 - P0);

    if(abs(Dot(n, B - A)) ≤ EPS)
        return false;     // 线段A-B和平面P0P1P2平行或共面
    else    // 平面A和直线P1-P2有惟一交点
    {
        double t = Dot(n, P0 - A) / Dot(n, B - A);

        if(t + EPS < 0 || t - 1 - EPS > 0)
            return false;     // 不在线段AB上
        P = A + (B - A) * t; // 交点
        return PointInTri(P, P0, P1, P2);
```

```
    }
}
```

2.13 空间两三角形是否相交(TriTriIntersection)

```
bool TriTriIntersection(Point_3D * T1, Point_3D * T2)
{
    Point_3D P;

    for(int i = 0; i < 3; i++)
    {
        if(TriSegIntersection(T1[0], T1[1], T1[2], T2[i], T2[(i + 1) % 3], P))
            return true;
        if(TriSegIntersection(T2[0], T2[1], T2[2], T1[i], T1[(i + 1) % 3], P))
            return true;
    }

    return false;
}
```

2.14 空间两直线上最近点对 返回最近距离(SegSegDistance)

```
//空间两直线上最近点对 返回最近距离 点对保存在ans1 ans2中
double SegSegDistance(Point_3D a1, Point_3D b1, Point_3D a2, Point_3D b2, Point_3D& ans1, Point_3D&
ans2)
{
    Vector_3D v1 = (a1 - b1), v2 = (a2 - b2);
    Vector_3D N = Cross(v1, v2);
    Vector_3D ab = (a1 - a2);
    double ans = Dot(N, ab) / Length(N);
    Point_3D p1 = a1, p2 = a2;
    Vector_3D d1 = b1 - a1, d2 = b2 - a2;
    double t1, t2;
    t1 = Dot((Cross(p2 - p1, d2)), Cross(d1, d2));
    t2 = Dot((Cross(p2 - p1, d1)), Cross(d1, d2));
    double dd = Length((Cross(d1, d2)));
    t1 /= dd * dd;
    t2 /= dd * dd;
    ans1 = (a1 + (b1 - a1) * t1);
    ans2 = (a2 + (b2 - a2) * t2);
    return fabs(ans);
}
```

2.15判断P是否在三角形A, B, C中，并且到三条边的距离都至少为mindist(InsideWithMinDistance)

```cpp
// 判断P是否在三角形A, B, C中，并且到三条边的距离都至少为mindist。保证P, A, B, C共面
bool InsideWithMinDistance(const Point_3D & P, const Point_3D & A, const Point_3D & B, const Point_3D & C, double mindist)
{
    if(!PointInTri(P, A, B, C))
        return false;
    if(DistanceToLine(P, A, B) < mindist)
        return false;
    if(DistanceToLine(P, B, C) < mindist)
        return false;
    if(DistanceToLine(P, C, A) < mindist)
        return false;
    return true;
}
```

2.16判断P是否在凸四边形中，并且到四条边的距离都至少为mindist(InsideWithMinDistance)

```cpp
// 判断P是否在凸四边形ABCD（顺时针或逆时针）中，并且到四条边的距离都至少为mindist。保证P, A, B, C, D共面
bool InsideWithMinDistance(const Point_3D & P, const Point_3D & A, const Point_3D & B, const Point_3D & C, const Point_3D & D, double mindist)
{
    if(!PointInTri(P, A, B, C))
        return false;
    if(!PointInTri(P, C, D, A))
        return false;
    if(DistanceToLine(P, A, B) < mindist)
        return false;
    if(DistanceToLine(P, B, C) < mindist)
        return false;
    if(DistanceToLine(P, C, D) < mindist)
        return false;
    if(DistanceToLine(P, D, A) < mindist)
        return false;
    return true;
}

```

$\oint$ 16. 三维凸包

3.1 加干扰防止多点共面(add_noise)

```cpp
//加干扰防止多点共面
double rand01()
{
    return rand() / (double)RAND_MAX;
}
double randeps()
{
    return (rand01() - 0.5) * EPS;
}
Point_3D add_noise(const Point_3D & p)
{
    return Point_3D(p.x + randeps(), p.y + randeps(), p.z + randeps());
}
```

3.2 凸包的定义(Face)

```cpp
struct Face
{
    int v[3];
    Face(int a, int b, int c)
    {
        v[0] = a;
        v[1] = b;
        v[2] = c;
    }//逆时针旋转
    Vector_3D Normal(const vector<Point_3D> & P) const
    {
        return Cross(P[v[1]] - P[v[0]], P[v[2]] - P[v[0]]);
    }
    // f是否能看见P[i]
    int CanSee(const vector<Point_3D> & P, int i) const
    {
        return Dot(P[i] - P[v[0]], Normal(P)) > 0;
    }
};
```

3.3 增量法求三维凸包(CH3D)

```cpp
// 增量法求三维凸包
// 注意：没有考虑各种特殊情况（如四点共面）。实践中，请在调用前对输入点进行微小扰动
//vector<Face>CH3D(Point_3D* p, int n)//所有面的点集和点数
vector<Face> CH3D(const vector<Point_3D> & P)
{
    int n = P.size();
    vector<vector<int> > vis(n);
    for(int i = 0; i < n; i++){
        vis[i].resize(n);
    }
    vector<Face> cur;
    cur.push_back(Face(0, 1, 2)); // 由于已经进行扰动，前三个点不共线
    cur.push_back(Face(2, 1, 0));

    for(int i = 3; i < n; i++)
    {
        vector<Face> next;

        // 计算每条边的“左面”的可见性
        for(int j = 0; j < cur.size(); j++)
        {
            Face & f = cur[j];
            int res = f.CanSee(P, i);

            if(!res)
            {
                next.push_back(f);
            }

            for(int k = 0; k < 3; k++)
            {
                vis[f.v[k]][f.v[(k + 1) % 3]] = res;
            }
        }

```

```
        for(int j = 0; j < cur.size(); j++)
            for(int k = 0; k < 3; k++)
            {
                int a = cur[j].v[k], b = cur[j].v[(k + 1) % 3];

                if(vis[a][b] ≠ vis[b][a] && vis[a][b]) // (a,b)是分界线，左边对P[i]可见
                {
                    next.push_back(Face(a, b, i));
                }
            }

        cur = next;
    }

    return cur;
}

```

3.4 凸多面体(ConvexPolyhedron)

```
struct ConvexPolyhedron
{
    int n;
    vector<Point_3D> P, P2;
    vector<Face> faces;

    bool read()
    {
        if(scanf("%d", &n) ≠ 1)
        {
            return false;
        }

        P.resize(n);
        P2.resize(n);

        for(int i = 0; i < n; i++)
        {
            P[i] = read_Point_3D();
            P2[i] = add_noise(P[i]);
        }

        faces = CH3D(P2);
        return true;
    }

    //三维凸包重心
    Point_3D centroid()
    {
        Point_3D C = P[0];
        double totv = 0;
        Point_3D tot(0, 0, 0);

        for(int i = 0; i < faces.size(); i++)
        {
            Point_3D p1 = P[faces[i].v[0]], p2 = P[faces[i].v[1]], p3 = P[faces[i].v[2]];
            double v = -Volume6(p1, p2, p3, C);
            totv += v;
            tot = tot + Centroid(p1, p2, p3, C) * v;
```

```
        }

        return tot / totv;
    }
    //凸包重心到表面最近距离
    double mindist(Point_3D C)
    {
        double ans = 1e30;

        for(int i = 0; i < faces.size(); i++)
        {
            Point_3D p1 = P[faces[i].v[0]], p2 = P[faces[i].v[1]], p3 = P[faces[i].v[2]];
            ans = min(ans, fabs(-Volume6(p1, p2, p3, C) / Area2(p1, p2, p3)));
        }

        return ans;
    }
};
```

3.5 给三维凸包求出重心到各面的最小距离。

给我们一个三维凸包，让我们求出重心到各个面的最小距离。

```
const int MAXN=550;
const double eps=1e-8;
struct Point
{
    double x,y,z;
    Point(){}

    Point(double xx,double yy,double zz):x(xx),y(yy),z(zz){}

    //两向量之差
    Point operator -(const Point p1)
    {
        return Point(x-p1.x,y-p1.y,z-p1.z);
    }

    //两向量之和
    Point operator +(const Point p1)
    {
        return Point(x+p1.x,y+p1.y,z+p1.z);
    }

    //叉乘
    Point operator *(const Point p)
    {
        return Point(y*p.z-z*p.y,z*p.x-x*p.z,x*p.y-y*p.x);
    }

    Point operator *(double d)
    {
        return Point(x*d,y*d,z*d);
    }

    Point operator / (double d)
    {
        return Point(x/d,y/d,z/d);
    }

```

```
    //点乘
    double  operator ^(Point p)
    {
        return (x*p.x+y*p.y+z*p.z);
    }
};

struct CH3D
{
    struct face
    {
        //表示凸包一个面上的三个点的编号
        int a,b,c;
        //表示该面是否属于最终凸包上的面
        bool ok;
    };
    //初始顶点数
    int n;
    //初始顶点
    Point P[MAXN];
    //凸包表面的三角形数
    int num;
    //凸包表面的三角形
    face F[8*MAXN];
    //凸包表面的三角形
    int g[MAXN][MAXN];
    //向量长度
    double vlen(Point a)
    {
        return sqrt(a.x*a.x+a.y*a.y+a.z*a.z);
    }
    //叉乘
    Point cross(const Point &a,const Point &b,const Point &c)
    {
        return Point((b.y-a.y)*(c.z-a.z)-(b.z-a.z)*(c.y-a.y),
                     (b.z-a.z)*(c.x-a.x)-(b.x-a.x)*(c.z-a.z),
                     (b.x-a.x)*(c.y-a.y)-(b.y-a.y)*(c.x-a.x)
                     );
    }
    //三角形面积*2
    double area(Point a,Point b,Point c)
    {
        return vlen((b-a)*(c-a));
    }
    //四面体有向体积*6
    double volume(Point a,Point b,Point c,Point d)
    {
        return (b-a)*(c-a)^(d-a);
    }
    //正：点在面同向
    double dblcmp(Point &p,face &f)
    {
        Point m=P[f.b]-P[f.a];
        Point n=P[f.c]-P[f.a];
        Point t=p-P[f.a];
        return (m*n)^t;
    }
    void deal(int p,int a,int b)
    {
        int f=g[a][b];//搜索与该边相邻的另一个平面
```

```
        face add;
        if(F[f].ok)
        {
            if(dblcmp(P[p],F[f])>eps)
              dfs(p,f);
            else
            {
                add.a=b;
                add.b=a;
                add.c=p;//这里注意顺序，要成右手系
                add.ok=true;
                g[p][b]=g[a][p]=g[b][a]=num;
                F[num++]=add;
            }
        }
    }
    void dfs(int p,int now)//递归搜索所有应该从凸包内删除的面
    {
         F[now].ok=0;
         deal(p,F[now].b,F[now].a);
         deal(p,F[now].c,F[now].b);
         deal(p,F[now].a,F[now].c);
    }
    bool same(int s,int t)
    {
        Point &a=P[F[s].a];
        Point &b=P[F[s].b];
        Point &c=P[F[s].c];
        return fabs(volume(a,b,c,P[F[t].a]))<eps &&
               fabs(volume(a,b,c,P[F[t].b]))<eps &&
               fabs(volume(a,b,c,P[F[t].c]))<eps;
    }
    //构建三维凸包
    void create()
    {
        int i,j,tmp;
        face add;

        num=0;
        if(n<4)return;
    //********************************************
        //此段是为了保证前四个点不共面
        bool flag=true;
        for(i=1;i<n;i++)
        {
            if(vlen(P[0]-P[i])>eps)
            {
                swap(P[1],P[i]);
                flag=false;
                break;
            }
        }
        if(flag)return;
        flag=true;
        //使前三个点不共线
        for(i=2;i<n;i++)
        {
            if(vlen((P[0]-P[1])*(P[1]-P[i]))>eps)
            {
                swap(P[2],P[i]);
```

```
158                 flag=false;
159                 break;
160             }
161         }
162         if(flag)return;
163         flag=true;
164         //使前四个点不共面
165         for(int i=3;i<n;i++)
166         {
167             if(fabs((P[0]-P[1])*(P[1]-P[2])^(P[0]-P[i]))>eps)
168             {
169                 swap(P[3],P[i]);
170                 flag=false;
171                 break;
172             }
173         }
174         if(flag)return;
175     //*****************************************
176         for(i=0;i<4;i++)
177         {
178             add.a=(i+1)%4;
179             add.b=(i+2)%4;
180             add.c=(i+3)%4;
181             add.ok=true;
182             if(dblcmp(P[i],add)>0)swap(add.b,add.c);
183             g[add.a][add.b]=g[add.b][add.c]=g[add.c][add.a]=num;
184             F[num++]=add;
185         }
186         for(i=4;i<n;i++)
187         {
188             for(j=0;j<num;j++)
189             {
190                 if(F[j].ok&&dblcmp(P[i],F[j])>eps)
191                 {
192                     dfs(i,j);
193                     break;
194                 }
195             }
196         }
197         tmp=num;
198         for(i=num=0;i<tmp;i++)
199           if(F[i].ok)
200             F[num++]=F[i];
201
202     }
203     //表面积
204     double area()
205     {
206         double res=0;
207         if(n==3)
208         {
209             Point p=cross(P[0],P[1],P[2]);
210             res=vlen(p)/2.0;
211             return res;
212         }
213         for(int i=0;i<num;i++)
214           res+=area(P[F[i].a],P[F[i].b],P[F[i].c]);
215         return res/2.0;
216     }
217     double volume()
```

```
    {
        double res=0;
        Point tmp(0,0,0);
        for(int i=0;i<num;i++)
           res+=volume(tmp,P[F[i].a],P[F[i].b],P[F[i].c]);
        return fabs(res/6.0);
    }
    //表面三角形个数
    int triangle()
    {
        return num;
    }
    //表面多边形个数
    int polygon()
    {
        int i,j,res,flag;
        for(i=res=0;i<num;i++)
        {
            flag=1;
            for(j=0;j<i;j++)
              if(same(i,j))
              {
                  flag=0;
                  break;
              }
            res+=flag;
        }
        return res;
    }
    //三维凸包重心
    Point barycenter()
    {
        Point ans(0,0,0),o(0,0,0);
        double all=0;
        for(int i=0;i<num;i++)
        {
            double vol=volume(o,P[F[i].a],P[F[i].b],P[F[i].c]);
            ans=ans+(o+P[F[i].a]+P[F[i].b]+P[F[i].c])/4.0*vol;
            all+=vol;
        }
        ans=ans/all;
        return ans;
    }
    //点到面的距离
    double ptoface(Point p,int i)
    {
        return fabs(volume(P[F[i].a],P[F[i].b],P[F[i].c],p)/vlen((P[F[i].b]-P[F[i].a])*(P[F[i].c]-
P[F[i].a])));
    }
};
CH3D hull;
int main()
{
    while(scanf("%d",&hull.n)==1)
    {
        for(int i=0;i<hull.n;i++)
        {
            scanf("%lf%lf%lf",&hull.P[i].x,&hull.P[i].y,&hull.P[i].z);
        }
        hull.create();
```

```
277         Point p=hull.barycenter();//求重心
278         double ans1=1e20;
279         for(int i=0;i<hull.num;i++)
280         {
281             ans1=min(ans1,hull.ptoface(p,i));
282         }
283         printf("%.3f\n",ans1);
284     }
285     return 0;
286 }
```

$\oint$ 17. 最小圆覆盖

【定理】 如果点 pp 不在集合 {S}{S} 的最小覆盖圆内，则 pp 一定在 {S}∪p{S}∪p 的最小覆盖圆上。

```
1  inline int PIC(Circle C,Point a){return dcmp(Len(a-C.O)-C.r)≤0;}//判断点A是否在圆C内
2  inline void Random(Point *P,Re n){for(Re i=1;i≤n;++i)swap(P[i],P[rand()%n+1]);}//随机一个排列
3  inline Circle Min_Circle(Point *P,Re n){//【求点集P的最小覆盖圆】
4  //  random_shuffle(P+1,P+n+1);
5      Random(P,n);Circle C=Circle(P[1],0);
6      for(Re i=2;i≤n;++i)if(!PIC(C,P[i])){
7          C=Circle(P[i],0);
8          for(Re j=1;j<i;++j)if(!PIC(C,P[j])){
9              C.O=(P[i]+P[j])*0.5,C.r=Len(P[j]-C.O);
10             for(Re k=1;k<j;++k)if(!PIC(C,P[k]))C=getcircle(P[i],P[j],P[k]);
11         }
12     }
13     return C;
14 }
```

> 题目描述
>
> 给出N个点，让你画一个最小的包含所有点的圆。

```
1  struct vec
2  {
3      double x, y;
4      vec (const double& x0 = 0, const double& y0 = 0) : x(x0), y(y0) {}
5      vec operator + (const vec& t) const {return vec(x+t.x, y+t.y);}
6      vec operator - (const vec& t) const {return vec(x-t.x, y-t.y);}
7      vec operator * (const double& t) const {return vec(x*t, y*t);}
8      vec operator / (const double& t) const {return vec(x/t, y/t);}
9      const double len2 () const {return x*x + y*y;}
10     const double len () const {return sqrt(len2());}
11     vec norm() const {return *this/len();}
12     vec rotate_90_c () {return vec(y, -x);}
13 };
14
15 double dot(const vec& a, const vec& b) {return a.x*b.x + a.y*b.y;}
16 double crs(const vec& a, const vec& b) {return a.x*b.y - a.y*b.x;}
17
18 vec lin_lin_int(const vec& p0, const vec& v0, const vec& p1, const vec& v1)
19 {
20     double t = crs(p1-p0, v1) / crs(v0, v1);
21     return p0 + v0 * t;
22 }
```

```
23
24 vec circle(const vec& a, const vec& b, const vec& c)
25 {
26     return lin_lin_int((a+b)/2, (b-a).rotate_90_c(), (a+c)/2, (c-a).rotate_90_c());
27 }
28
29 int n;
30 vec pot[100005];
31
32 int main()
33 {
34     scanf("%d", &n);
35     for(int i=1; i≤n; i++) scanf("%lf%lf", &pot[i].x, &pot[i].y);
36     random_shuffle(pot+1, pot+n+1);
37     vec o;
38     double r2 = 0;
39     for(int i=1; i≤n; i++)
40     {
41         if((pot[i]-o).len2() > r2)
42         {
43             o = pot[i], r2 = 0;
44             for(int j=1; j<i; j++)
45             {
46                 if((pot[j]-o).len2() > r2)
47                 {
48                     o = (pot[i]+pot[j])/2, r2 = (pot[j]-o).len2();
49                     for(int k=1; k<j; k++)
50                     {
51                         if((pot[k]-o).len2() > r2)
52                         {
53                             o = circle(pot[i], pot[j], pot[k]), r2 = (pot[k]-o).len2();
54                         }
55                     }
56                 }
57             }
58         }
59     }
60     printf("%.10lf\n%.10lf %.10lf\n", sqrt(r2), o.x, o.y);
61     return 0;
62 }
63
```

$\oint$ 18. 三角剖分(求圆与三角形的公共面积)

> 已知平面上有一个三角形和一个圆。您的任务是计算这两个图形的公共面积。每个测试用例由9个浮点数x1, y1, x2, y2, x3, y3, x4, y4和r组成，其中(x1, y1)， (x2, y2)和(x3, y3)是三角形的三个顶点，(x4, y4)是圆心，r是半径。我们保证三角形和圆不会退化。

```
1 const double eps = 1e-8;
2 const double pi = acos(-1.0);
3 int dcmp(double x)
4 {
5     if(x > eps) return 1;
6     return x < -eps ? -1 : 0;
7 }
```

```cpp

struct Point
{
    double x, y;
    Point(){
        x = y = 0;
    }
    Point(double a, double b){
        x = a, y = b;
    }
    inline void read(){
        scanf("%lf%lf", &x, &y);
    }
    inline Point operator-(const Point &b)const{
        return Point(x - b.x, y - b.y);
    }
    inline Point operator+(const Point &b)const{
        return Point(x + b.x, y + b.y);
    }
    inline Point operator*(const double &b)const{
        return Point(x * b, y * b);
    }
    inline double dot(const Point &b)const{
        return x * b.x + y * b.y;
    }
    inline double cross(const Point &b, const Point &c)const{
        return (b.x - x) * (c.y - y) - (c.x - x) * (b.y - y);
    }
    inline double Dis(const Point &b)const{
        return sqrt((*this - b).dot(*this - b));
    }
    inline bool InLine(const Point &b, const Point &c)const{//三点共线
        return !dcmp(cross(b, c));
    }
    inline bool OnSeg(const Point &b, const Point &c)const    {//点在线段上，包括端点

        return InLine(b, c) && (*this - c).dot(*this - b) < eps;
    }
};

inline double min(double a, double b){
    return a < b ? a : b;
}
inline double max(double a, double b){
    return a > b ? a : b;
}
inline double Sqr(double x){
    return x * x;
}
inline double Sqr(const Point &p){
    return p.dot(p);
}

Point LineCross(const Point &a, const Point &b, const Point &c, const Point &d){
    double u = a.cross(b, c), v = b.cross(a, d);
    return Point((c.x * v + d.x * u) / (u + v), (c.y * v + d.y * u) / (u + v));
}

double LineCrossCircle(const Point &a, const Point &b, const Point &r,
                       double R, Point &p1, Point &p2){
```

```
    Point fp = LineCross(r, Point(r.x + a.y - b.y, r.y + b.x - a.x), a, b);
    double rtol = r.Dis(fp);
    double rtos = fp.OnSeg(a, b) ? rtol : min(r.Dis(a), r.Dis(b));
    double atob = a.Dis(b);
    double fptoe = sqrt(R * R - rtol * rtol) / atob;
    if(rtos > R - eps) return rtos;
    p1 = fp + (a - b) * fptoe;
    p2 = fp + (b - a) * fptoe;
    return rtos;
}

double SectorArea(const Point &r, const Point &a, const Point &b, double R){
//不大于180度扇形面积，r→a→b逆时针
    double A2 = Sqr(r - a), B2 = Sqr(r - b), C2 = Sqr(a - b);
    return R * R * acos((A2 + B2 - C2) * 0.5 / sqrt(A2) / sqrt(B2)) * 0.5;
}

double TACIA(const Point &r, const Point &a, const Point &b, double R){
//TriangleAndCircleIntersectArea，逆时针，r为圆心
    double adis = r.Dis(a), bdis = r.Dis(b);
    if(adis < R + eps && bdis < R + eps) return r.cross(a, b) * 0.5;
    Point ta, tb;
    if(r.InLine(a, b)) return 0.0;
    double rtos = LineCrossCircle(a, b, r, R, ta, tb);
    if(rtos > R - eps) return SectorArea(r, a, b, R);
    if(adis < R + eps) return r.cross(a, tb) * 0.5 + SectorArea(r, tb, b, R);
    if(bdis < R + eps) return r.cross(ta, b) * 0.5 + SectorArea(r, a, ta, R);
    return r.cross(ta, tb) * 0.5 +
           SectorArea(r, a, ta, R) + SectorArea(r, tb, b, R);
}
const int N = 505;
Point p[N];
double SPICA(int n, Point r, double R)//SimplePolygonIntersectCircleArea{
    int i;
    double res = 0, if_clock_t;
    for(i = 0; i < n; ++ i){
        if_clock_t = dcmp(r.cross(p[i], p[(i + 1) % n]));
        if(if_clock_t < 0) res -= TACIA(r, p[(i + 1) % n], p[i], R);
        else res += TACIA(r, p[i], p[(i + 1) % n], R);
    }
    return fabs(res);
}
double r;
int main()
{
    while (~scanf("%lf%lf", &p[0].x, &p[0].y)){
        for (int i = 1; i < 4; i++)
            p[i].read();
        scanf("%lf", &r);
        printf("%.2f\n", SPICA(3, p[3], r));
    }
    return 0;
}
```

∮ 19. K次圆覆盖

题意： 给出n个圆，问共被i个圆覆盖的面积

```cpp
#include<bits/stdc++.h>
using namespace std;
const int maxn=1009;
const double eps=1e-8;
const double pi=acos(-1);
int dcmp(double x) {
    return fabs(x)<eps?0:(x<0?-1:1);
}
struct circle {
    double x,y,r,angle;
    int d;
    circle() {}
    circle(double x,double y,double angle=0,int d=0)
        :x(x),y(y),angle(angle),d(d) {}
};
typedef circle Circle;
circle cir[maxn],tp[maxn<<1];
double area[maxn];

double sqr(double x){return x*x;}
double dis(circle a,circle b) {
    return sqrt(sqr(a.x-b.x)+sqr(a.y-b.y));
}
double cross(circle a,circle b,circle c) {
    return (b.x-a.x)*(c.y-a.y)-(b.y-a.y)*(c.x-a.x);
}
int CirCrossCir(Circle p1, double r1,Circle p2, double r2,Circle &cp1,Circle &cp2) {
    double mx = p2.x - p1.x, sx = p2.x + p1.x, mx2 = mx * mx;
    double my = p2.y - p1.y, sy = p2.y + p1.y, my2 = my * my;
    double sq = mx2 + my2, d = -(sq - sqr(r1 - r2)) * (sq - sqr(r1 + r2));
    if (d + eps < 0)
        return 0;
    if (d < eps)
        d = 0;
    else
        d = sqrt(d);
    double x = mx * ((r1 + r2) * (r1 - r2) + mx * sx) + sx * my2;
    double y = my * ((r1 + r2) * (r1 - r2) + my * sy) + sy * mx2;
    double dx = mx * d, dy = my * d;
    sq *= 2;
    cp1.x = (x - dy) / sq;
    cp1.y = (y + dx) / sq;
    cp2.x = (x + dy) / sq;
    cp2.y = (y - dx) / sq;
    if (d > eps)
        return 2;
    else
        return 1;
}

bool circmp(const Circle& u, const Circle& v) {
    return dcmp(u.r - v.r) < 0;
}
bool cmp(const Circle& u, const Circle& v) {
    if (dcmp(u.angle - v.angle))
        return u.angle < v.angle;
    return u.d > v.d;
}

double calc(Circle cir,Circle cp1,Circle cp2) {
```

```
    double ans = (cp2.angle - cp1.angle) * sqr(cir.r)
                 - cross(cir, cp1, cp2) + cross(Circle(0, 0), cp1, cp2);
    return ans / 2;
}

void CirUnion(Circle cir[], int n) {
    Circle cp1, cp2;
    sort(cir, cir + n, circmp);
    for (int i = 0; i < n; ++i)
        for (int j = i + 1; j < n; ++j)
            if (dcmp(dis(cir[i], cir[j]) + cir[i].r - cir[j].r) ≤ 0)
                cir[i].d++;
    for (int i = 0; i < n; ++i) {
        int tn = 0, cnt = 0;
        for (int j = 0; j < n; ++j) {
            if (i == j)
                continue;
            if (CirCrossCir(cir[i], cir[i].r, cir[j], cir[j].r,
                            cp2, cp1) < 2)
                continue;
            cp1.angle = atan2(cp1.y - cir[i].y, cp1.x - cir[i].x);
            cp2.angle = atan2(cp2.y - cir[i].y, cp2.x - cir[i].x);
            cp1.d = 1;
            tp[tn++] = cp1;
            cp2.d = -1;
            tp[tn++] = cp2;
            if (dcmp(cp1.angle - cp2.angle) > 0)
                cnt++;
        }
        tp[tn++] = Circle(cir[i].x - cir[i].r, cir[i].y, pi, -cnt);
        tp[tn++] = Circle(cir[i].x - cir[i].r, cir[i].y, -pi, cnt);
        sort(tp, tp + tn, cmp);
        int p, s = cir[i].d + tp[0].d;
        for (int j = 1; j < tn; ++j) {
            p = s;
            s += tp[j].d;
            area[p] += calc(cir[i], tp[j - 1], tp[j]);
        }
    }
}
int n;
void solve() {
    for(int i=0; i<n; i++){
        scanf("%lf%lf%lf", &cir[i].x, &cir[i].y, &cir[i].r);
        cir[i].d = 1;
    }
    memset(area,0,sizeof area);
    CirUnion(cir,n);
    for(int i=1; i≤n; i++) {
        area[i]-=area[i+1];
        printf("[%d] = %.3f\n", i, area[i]);
    }
}
int main() {
    while(scanf("%d",&n)≠EOF)
        solve();

    return 0;
}
```

九、其他 Other

$\oint$ 1. 常用函数

1.1 全排列：next_permutation

`next_permutation`

包含在头文件 `<algorithm>` 中

```
int a[];
do
{}
while(next_permutation(a,a+n));
```

例题：输出自然数1到n所有不重复的排列，即n的全排列，要求所产生的任一数字序列中不允许出现重复的数字。

```
const int maxn=1e5+10;
const int mod=1e9+7;
int n,a[maxn];
int main()
{
    cin>>n;
    for(int i=1;i≤n;i++)a[i]=i;
    do{
        for(int i=1;i≤n;i++)printf("%5d",a[i]);
        cout<<endl;
    }while(next_permutation(a+1,a+1+n));
    return 0;
}
```

P1088 火星人

输入

`5`

`3`

`1 2 3 4 5`

输出

`1 2 4 5 3`

```
const int maxn=1e5+10;
const int mod=1e9+7;
int n,a[maxn],m;
int main()
{
    cin>>n>>m;
    for(int i=1;i≤n;i++)cin>>a[i];
    while(m--)//全排列m次就是题意中的第m大的数
        next_permutation(a+1,a+1+n);
    for(int i=1;i≤n;i++)printf("%d ",a[i]);//输出即可
    cout<<endl;
    return 0;
}
```

1.2 读写优化

```cpp
inline int read()//快读
{
    int x=0,f=0;
    char ch=getchar();
    while(ch>'9'||ch<'0'){f|=(ch=='-');ch=getchar();}
    while(ch≤'9'&&ch≥'0'){x=(x<<1)+(x<<3)+(ch^48);ch=getchar();}
    return f?-x:x;
}
```

```cpp
ios::sync_with_stdio(false);//cin加速
```

读出优化

```cpp
inline void write(int x)
{
    char f[200];
    int cnt=0,tmp=x>0?x:-x;
    if(x<0)putchar('-');
    while(tmp>0)f[cnt++]=tmp%10+'0',tmp/=10;
    while(cnt>0)putchar(f[--cnt]);
}
```

推荐数据较多时使用

1.3 返回容器内最大最小值

1. `min_element(first,last)` 寻找范围内最小值，返回**迭代器**

```cpp
int main()
{
    vector<int> v;
    vector<int> ::iterator pos;
    v.push_back(1);
    v.push_back(3);
    v.push_back(-8);
    v.push_back(8);
    pos=min_element(v.begin(),v.end());
    cout<<*pos<<endl;
    return 0;
}
```

输出：-8

2. `max_element(first,last)` 寻找范围内最大值，返回**迭代器**

```cpp
int main()
{
    vector<int> v;
    vector<int> ::iterator pos;
    v.push_back(1);
    v.push_back(3);
    v.push_back(-8);
    v.push_back(8);
    pos=max_element(v.begin(),v.end());
    cout<<*pos<<endl;
    return 0;
}
```

输出：8

1.4 复制函数

1. `copy(first,last,result)` 将first到last区间的元素复制到result中：

```
int main()
{
    int myints[]= {10,20,30,40,50,60,70};
    vector<int> myvector;
    vector<int>::iterator it;
    myvector.resize(7);
    copy ( myints, myints+7, myvector.begin() );
    for(int i=0; i<myvector.size(); i++)
        cout<<myvector[i]<<" ";
    cout<<endl;
    return 0;
}
```

2.亦可用于字符串中

```
int main()
{
    char s[24]="123456789";
    char s1[24]="";
    copy(s,s+5,s1);
    cout<<s1<<endl;
    return 0;
}
```

1.5 容器删除函数

`remove(first,last,value)` 将区间first到last区间中值为value的元素删除

```
int main()
{
    int num[]= {1,2,3,4,5,1,0};
    int *pbegin=num;
    int *pend=num+sizeof(num)/sizeof(int);
    pend=remove(pbegin,pend,1);
    for(int *p=pbegin; p≠pend; p++)
        cout<<*p<<" ";
    cout<<endl;
    return 0;
}
```

1.6 容器填充函数

`fill(first,last,value)` 将区间first到last区间全部初始为value

```
int main()
{
    vector<int> v(8);
    fill(v.begin(),v.end(),1);
    for(int i=0; i<v.size(); i++)
        cout<<v[i]<<" ";
    cout<<endl;
    return 0;
}
```

1.7 查找函数

`find(first,last,value)` 查找first到last区间第一个值为value元素的位置，返回一个迭代器

```cpp
int main()
{
    int num[]= {1,2,3,4,5};
    int *p;
    p=find(num,num+5,3);
    if(p==num+5)
        cout<<"Not found!"<<endl;
    else
        cout<<*p<<endl;
    return 0;
}
```

1.8 字符串转换整数 / 数字转字符串

`atoi(char);`把字符串（字符char）转换成整型数的一个函数

```cpp
int main ()
{
    int data;
    data=atoi("123");
    printf("%d\n",data);
    return 0;
}
```

`stoi;`把字符串（string）转换成整数

```cpp
typedef unsigned long long ll;
ll stoi(char *ss,ll l,ll r)//自己写一个字符串转整数。想知道为什么cstring里包含的stoi()用不了
{
    ll res=0;
    if(l>r)return 0;
    for (ll i=l;i≤r;i++)
    {
        res*=10;
        res+=ss[i]-'0';
    }
    return res;
}
```

```cpp
/*数字转字符串*/
int main()
{
    scanf("%d", &n);
    for(int i = 1; i ≤ n; ++i)
        s += to_string(i);
    cout << s[n - 1] <<endl;
    return 0;
}

```

1.9 取部分字符串函数

```cpp
string s="0123456789";
string sub1=s.substr(5);  //表示从下标为5开始一直到字符串结束
sub1="56789";
string sub2=s.substr(3,5);//表示从下标为3开始往后取五位
sub2="34567";
```

∮ 2. 高精度计算

```
struct bign
{
    int len,s[N];
    bign()  {  memset(s,0,sizeof(s));  len=1;  }
    bign(int num)  {  *this=num; }
    bign(char *num) { *this=num; }
    bign operator =(int num)
    {
        char c[N];
        sprintf(c,"%d",num);
        *this=c;
        return *this;
    }
    bign operator =(const char *num)
    {
        len=strlen(num);
        for (int i=0;i<len;i++) s[i]=num[len-1-i]-'0';
        return *this;
    }
    string str()
    {
        string res="";
        for (int i=0;i<len;i++) res=(char)(s[i]+'0')+res;
        return res;
    }
    void clean()
    {
        while (len>1&&!s[len-1]) len--;
    }
    bign operator +(const bign &b)
    {
        bign c;
        c.len=0;
        for (int i=0,g=0;g||i<len||i<b.len;i++)
        {
            int x=g;
            if (i<len) x+=s[i];
            if (i<b.len) x+=b.s[i];
            c.s[c.len++]=x%10;
            g=x/10;
        }
        return c;
    }
    bign operator -(const bign &b)
    {
        bign c;
        c.len=0;
        int x;
        for (int i=0,g=0;i<len;i++)
        {
            x=s[i]-g;
            if (i<b.len) x-=b.s[i];
            if (x≥0) g=0;
            else{
                x+=10;
                g=1;
```

```
            };
            c.s[c.len++]=x;
        }
        c.clean();
        return c;
    }
    bign operator *(const bign &b)
    {
        bign c;
        c.len=len+b.len;
        for (int i=0;i<len;i++) for (int j=0;j<b.len;j++) c.s[i+j]+=s[i]*b.s[j];
        for (int i=0;i<c.len-1;i++) { c.s[i+1]+=c.s[i]/10; c.s[i]%=10; }
        c.clean();
        return c;
    }
    bool operator <(const bign &b)
    {
        if (len≠b.len) return len<b.len;
        for (int i=len-1;i≥0;i--)
             if (s[i]≠b.s[i]) return s[i]<b.s[i];
        return false;
    }
    bign operator +=(const bign &b)
    {
        *this=*this+b;
        return *this;
    }
    bign operator -=(const bign &b)
    {
        *this=*this-b;
        return *this;
    }
};
istream& operator >>(istream &in,bign &x)
{
  string s;
  in>>s;
  x=s.c_str();
  return in;
}
ostream& operator <<(ostream &out,bign &x)
{
    out<<x.str();
    return out;
}
int main(){
    bign a,b,c;
    ios::sync_with_stdio(false);
    cin>>a>>b;
//    cout<<a<<endl;
//    cout<<b<<endl;
    c=a+b;
    cout<<c<<endl;
    return 0;
}

```

∮ 3. 离散化

一个对闭合区间离散化的小技巧

- 有若干个闭合区间[Li,Ri]，把它们离散化成若干个区间
- 新的区间排序后也有一个序列
- 做法是把所有Li和Ri + 1拿出来排序，对于相邻的两个元素可以 得到一个区间[Vi,Vi+1 – 1]
- 这样每个原来的区间都包含了新的区间序列的一个区间

```
/*去重离散化*/
int tot = 0;
        scanf("%d",&n);
        over(i,1,n){
            scanf("%d",&a[i]);
            b[tot++] = a[i];
        }
        sort(b,b+tot);
        int res = unique(b,b+tot)-b;
        over(i,1,n)
            a[i] = lower_bound(b,b+res,a[i]) - b;
/*主要注意unique的用法，返回的是一个迭代器，而且unique并没有真正删除元素*/
```

```
/*不去重离散化，相同也会排开*/
struct node
{
    ll data,id;
    bool operator<(const node &t)const{return data<t.data;}
}a[N];
ll n,m,ranks[N];
int main()
{
    scanf("%lld",&n);
    over(i,1,n)scanf("%lld",&a[i].data),a[i].id=i;
    sort(a+1,a+1+n);
    over(i,1,n)
    ranks[a[i].id]=i;
    over(i,1,n)
    printf("%lld   ",ranks[i]);
    return 0;
}
```

∮ 4. 基础算法

4.1 快速排序

```
void quick_sort(int q[], int l, int r)
{
    if (l ≥ r) return;

    int i = l - 1, j = r + 1, x = q[l + r >> 1];
    while (i < j)
    {
        do i ++ ; while (q[i] < x);
        do j -- ; while (q[j] > x);
        if (i < j) swap(q[i], q[j]);
    }
    quick_sort(q, l, j), quick_sort(q, j + 1, r);
}
```

4.2 归并排序

```cpp
void merge_sort(int q[], int l, int r)
{
    if (l ≥ r) return;

    int mid = l + r >> 1;
    merge_sort(q, l, mid);
    merge_sort(q, mid + 1, r);

    int k = 0, i = l, j = mid + 1;
    while (i ≤ mid && j ≤ r)
        if (q[i] ≤ q[j]) tmp[k ++ ] = q[i ++ ];
        else tmp[k ++ ] = q[j ++ ];

    while (i ≤ mid) tmp[k ++ ] = q[i ++ ];
    while (j ≤ r) tmp[k ++ ] = q[j ++ ];

    for (i = l, j = 0; i ≤ r; i ++, j ++ ) q[i] = tmp[j];
}
```

4.3 整数二分

```cpp
bool check(int x) {/*  ...  */} // 检查x是否满足某种性质

// 区间[l, r]被划分成[l, mid]和[mid + 1, r]时使用:
int bsearch_1(int l, int r)
{
    while (l < r)
    {
        int mid = l + r >> 1;
        if (check(mid)) r = mid;    // check()判断mid是否满足性质
        else l = mid + 1;
    }
    return l;
}
// 区间[l, r]被划分成[l, mid - 1]和[mid, r]时使用:
int bsearch_2(int l, int r)
{
    while (l < r)
    {
        int mid = l + r + 1 >> 1;
        if (check(mid)) l = mid;
        else r = mid - 1;
    }
    return l;
}
```

4.4 浮点数二分

```cpp
bool check(double x) {/* ... */} // 检查x是否满足某种性质

double bsearch_3(double l, double r)
{
    const double eps = 1e-6;   // eps 表示精度，取决于题目对精度的要求
    while (r - l > eps)
    {
        double mid = (l + r) / 2;
        if (check(mid)) r = mid;
        else l = mid;
    }
    return l;
}
```

4.5 高精度加法

```cpp
// C = A + B, A ≥ 0, B ≥ 0
vector<int> add(vector<int> &A, vector<int> &B)
{
    if (A.size() < B.size()) return add(B, A);

    vector<int> C;
    int t = 0;
    for (int i = 0; i < A.size(); i ++ )
    {
        t += A[i];
        if (i < B.size()) t += B[i];
        C.push_back(t % 10);
        t /= 10;
    }

    if (t) C.push_back(t);
    return C;
}
```

4.6 高精度减法

```cpp
// C = A - B, 满足A ≥ B, A ≥ 0, B ≥ 0
vector<int> sub(vector<int> &A, vector<int> &B)
{
    vector<int> C;
    for (int i = 0, t = 0; i < A.size(); i ++ )
    {
        t = A[i] - t;
        if (i < B.size()) t -= B[i];
        C.push_back((t + 10) % 10);
        if (t < 0) t = 1;
        else t = 0;
    }

    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
```

4.7 高精度乘低精度

```cpp
// C = A * b, A ≥ 0, b > 0
```

```cpp
vector<int> mul(vector<int> &A, int b)
{
    vector<int> C;

    int t = 0;
    for (int i = 0; i < A.size() || t; i ++ )
    {
        if (i < A.size()) t += A[i] * b;
        C.push_back(t % 10);
        t /= 10;
    }

    while (C.size() > 1 && C.back() == 0) C.pop_back();

    return C;
}
```

4.8 高精度除以低精度

```cpp
// A / b = C ... r, A ≥ 0, b > 0
vector<int> div(vector<int> &A, int b, int &r)
{
    vector<int> C;
    r = 0;
    for (int i = A.size() - 1; i ≥ 0; i -- )
    {
        r = r * 10 + A[i];
        C.push_back(r / b);
        r %= b;
    }
    reverse(C.begin(), C.end());
    while (C.size() > 1 && C.back() == 0) C.pop_back();
    return C;
}
```

4.9 一维前缀和

```
S[i] = a[1] + a[2] + ... a[i]
a[l] + ... + a[r] = S[r] - S[l - 1]
```

4.10 二维前缀和

```
S[i, j] = 第i行j列格子左上部分所有元素的和
以(x1, y1)为左上角，(x2, y2)为右下角的子矩阵的和为：
S[x2, y2] - S[x1 - 1, y2] - S[x2, y1 - 1] + S[x1 - 1, y1 - 1]
```

4.10.1 二维前缀和求矩阵和

> $n*m$的棋盘输入k和q，k行输入x1y1x2y2使得矩阵x1≤x≤x2,y1≤y≤y2的所有点权值+1
> q行x1y1x2y2，输出矩阵x1≤x≤x2,y1≤y≤y2权值和

注意运用前缀和修改矩阵以后要先求一遍前缀和还原原矩阵，然后再求一遍前缀和才是真正的前缀和

```cpp
ll n,m,a[N][N];
void add(ll x_1,ll y_1,ll x_2,ll y_2)
{
```

```
    a[x_1][y_1]++;
    a[x_2+1][y_2+1]++;
    a[x_1][y_2+1]--;
    a[x_2+1][y_1]--;
}
ll k,q;
ll x_1,x_2,y_1,y_2;
int main()
{
    scanf("%lld%lld%lld%lld",&n,&m,&k,&q);
    while(k--){
        scanf("%lld%lld%lld%lld",&x_1,&y_1,&x_2,&y_2);
        add(x_1,y_1,x_2,y_2);
    }
    over(i,1,n)over(j,1,m)//求两次前缀和
        a[i][j]=a[i-1][j]+a[i][j-1]-a[i-1][j-1]+a[i][j];
    over(i,1,n)over(j,1,m)
        a[i][j]=a[i-1][j]+a[i][j-1]-a[i-1][j-1]+a[i][j];
    while(q--)
    {
        scanf("%lld%lld%lld%lld",&x_1,&y_1,&x_2,&y_2);//求区间的和
        ll ans=a[x_2][y_2]+a[x_1-1][y_1-1]-a[x_1-1][y_2]-a[x_2][y_1-1];
        printf("%lld\n",ans);
    }
    return 0;
}

```

4.11 一维差分

修改区间之后要求一遍前缀和才能差分数组转换成修改过后的原数组，再进行各种操作

```
给区间[l, r]中的每个数加上c：B[l] += c, B[r + 1] -= c
```

4.12 二维差分

```
给以(x1, y1)为左上角，(x2, y2)为右下角的子矩阵中的所有元素加上c：
S[x1, y1] += c, S[x2 + 1, y1] -= c, S[x1, y2 + 1] -= c, S[x2 + 1, y2 + 1] += c
```

4.12.1 区间修改使得数列全部相等的最小次数

> 给定一个长度为 n 的数列 $a_1, a_2, \ldots, a_n$，每次可以选择一个区间 [l,r]，使下标在这个区间内的数都加一或者都减一。
>
> 求至少需要多少次操作才能使数列中的所有数都一样，并求出在保证最少次数的前提下，最终得到的数列可能有多少种。

要求使一个数列中的数全部相同，就可以转化成使这个数列的差分数组全部为0。

因为我们只要求A序列中所有的数相同，而不在意这些方案具体是什么，所以说我们就可以转化题目，也就是将对A序列的$+1, -1$操作，让A序列相同，改成目标把$B_2, \ldots, B_n$变成全0即可，也就是A序列全部相等。

利用差分的修改：$a[l] += 1; a[r + 1] -= 1$，每一次选取一个Bi和Bj，（2<=i,j<=n）,而且这两个数，一个为正数，一个为负数，至于为什么要是正负配对，因为我们是要这个B序列2~n都要为0，所以这样负数增加，正数减少，就可以最快地到达目标为0的状态。

至于那些无法配对的数Bk可以选B1或者Bn+1，这两个不影响的数，进行修改。

所以说我们这道题目得出的答案就是，最少操作数$min(pos, neg) + abs(pos - neg) = max(pos, neg)$，然后最终序列a可能会有$abs(pos - neg) + 1$种情况。

```
ll n,m,a[N],p,q;
int main()
{
    scanf("%lld",&n);
    over(i,1,n)
        scanf("%lld",&a[i]);
    over(i,2,n){
        ll c=a[i]-a[i-1];
        if(c>0)p+=c;
        else q-=c;
    }
    printf("%lld\n%lld\n",max(p,q),abs(p-q)+1);
    return 0;
}

```

4.13 双指针算法

```
for (int i = 0, j = 0; i < n; i ++ )
{
    while (j < i && check(i, j)) j ++ ;

    // 具体问题的逻辑
}
```

常见问题分类：
(1) 对于一个序列，用两个指针维护一段区间
(2) 对于两个序列，维护某种次序，比如归并排序中合并两个有序序列的操作

4.14 区间合并

```
// 将所有存在交集的区间合并
void merge(vector<PII> &segs)
{
    vector<PII> res;

    sort(segs.begin(), segs.end());

    int st = -2e9, ed = -2e9;
    for (auto seg : segs)
        if (ed < seg.first)
        {
            if (st != -2e9) res.push_back({st, ed});
            st = seg.first, ed = seg.second;
        }
        else ed = max(ed, seg.second);

    if (st != -2e9) res.push_back({st, ed});

    segs = res;
}
```

4.15 龟速乘

```cpp
ll quick_cheng(ll a1, ll a2 ){
    ll sum=0;
    ll q=a2;
    while(a1){
        if(a1&1){
            sum=(sum+q)%c;
        }
        q=(q+q)%c;
        a1=a1>>1;
    }
    return sum%c;
}
```

$\oint$ 5. 表达式求值

5.1 表达式求值（前中后）

中缀表达式：最常见的表达式，如$3 * (1 - 2)$
前缀表达式：又称波兰式，例如 *3-1 \ 2
后缀表达式：又称逆波兰式，例如3 4 + 5 × 6 -
后缀表达式可以在$\Theta(N)$的时间内求值。
后缀表达式求值方式：
建立一个栈，从左往右扫描表达式：

1. 遇到数字，入栈
2. 遇到运算符，弹出栈中的两个元素，计算结果后再将结果压入栈扫描完成之后，栈中只剩下一个数字，最终结果。
   中缀表达式转后缀表达式：

- 建立一个用于储存运算符的栈，逐一扫描中缀表达式中的元素

1. 扫描到数字，输出该数；
2. 遇到左括号，将左括号入栈；
3. 遇到右括号，不断取出栈顶元素并且输出，直到栈顶为左括号，弹出左括号舍弃
4. 遇到运算符，如果栈顶的运算符优先级大于当前扫描到的运算符，就不断取出栈顶的元素，最后将新符号入栈；

- 将栈中剩余的运算符输出，所有的输出结果即为转化后的后缀表达式。

5.2 递归法求中缀表达式的值，O(n^2)

```cpp
int calc(int l, int r) {
    // 寻找未被任何括号包含的最后一个加减号
    for (int i = r, j = 0; i ≥ l; i--) {
        if (s[i] == '(') j++;
        if (s[i] == ')') j--;
        if (j == 0 && s[i] == '+') return calc(l, i - 1) + calc(i + 1, r);
        if (j == 0 && s[i] == '-') return calc(l, i - 1) - calc(i + 1, r);
    }
    // 寻找未被任何括号包含的最后一个乘除号
    for (int i = r, j = 0; i ≥ l; i--) {
        if (s[i] == '(') j++;
        if (s[i] == ')') j--;
        if (j == 0 && s[i] == '*') return calc(l, i - 1) * calc(i + 1, r);
        if (j == 0 && s[i] == '/') return calc(l, i - 1) / calc(i + 1, r);
    }
    for (int i = r, j = 0; i ≥ l; i--) {//计算平方
        if (s[i] == '(') j++;
        if (s[i] == ')') j--;
        if (j == 0 && s[i] == '^') return pow(calc(l, i - 1),calc(i + 1, r));
    }
```

```
    // 首尾是括号
    if (s[l] == '('&&s[r] == ')') return calc(l + 1, r - 1);
    // 是一个数
    int ans = 0;
    for (int i = l; i ≤ r; i++) ans = ans * 10 + s[i] - '0';
    return ans;
}
```

5.3 后缀表达式转中缀表达式，同时求值，O(n)

```

// 数值栈
vector<int> nums;
// 运算符栈
vector<char> ops;

// 优先级
int grade(char op) {
    switch (op) {
    case '(':
        return 1;
    case '+':
    case '-':
        return 2;
    case '*':
    case '/':
        return 3;
    }
    return 0;
}

// 处理后缀表达式中的一个运算符
void calc(char op) {
    // 从栈顶取出两个数
    int y = *nums.rbegin();
    nums.pop_back();
    int x = *nums.rbegin();
    nums.pop_back();
    int z;
    switch (op) {
    case '+':
        z = x + y;
        break;
    case '-':
        z = x - y;
        break;
    case '*':
        z = x * y;
        break;
    case '/':
        z = x / y;
        break;
    }
    // 把运算结果放回栈中
    nums.push_back(z);
}

```

5.4 中缀表达式转后缀表达式，同时对后缀表达式求值

```cpp
int solve(string s) {
    nums.clear();
    ops.clear();
    int top = 0, val = 0;
    for (int i = 0; i < s.size(); i++) {
        // 中缀表达式的一个数字
        if (s[i] ≥ '0' && s[i] ≤ '9') {
            val = val * 10 + s[i] - '0';
            if (s[i+1] ≥ '0' && s[i+1] ≤ '9') continue;
            // 后缀表达式的一个数，直接入栈
            nums.push_back(val);
            val = 0;
        }
        // 中缀表达式的左括号
        else if (s[i] == '(') ops.push_back(s[i]);
        // 中缀表达式的右括号
        else if (s[i] == ')') {
            while (*ops.rbegin() ≠ '(') {
                // 处理后缀表达式的一个运算符
                calc(*ops.rbegin());
                ops.pop_back();
            }
            ops.pop_back();
        }
        // 中缀表达式的加减乘除号
        else {
            while (ops.size() && grade(*ops.rbegin()) ≥ grade(s[i])) {
                calc(*ops.rbegin());
                ops.pop_back();
            }
            ops.push_back(s[i]);
        }
    }
    while (ops.size()) {
        calc(*ops.rbegin());
        ops.pop_back();
    }
    // 后缀表达式栈中最后剩下的数就是答案
    return *nums.begin();
}
```

$\oint$ 6. 递归

6.1 奇怪的汉诺塔（n盘m柱）

> **n盘m柱**的汉诺塔最小移动次数

类比只有三个柱子的汉诺塔, 设 $f[i][j]$ 为有i个盘子 j 个柱子时的最少步数. 那么肯定是把一些上面盘子移动到某根不是 j 的柱子上, 然后把剩下的盘子移动到 j , 然后再把上面的盘子移动到 j . 于是就有递推式
$f[i][j] = minf[k][j] * 2 + f[i-k][j-1]$
代码如下:

```cpp
int n,m;
ll dp[N][N];

```

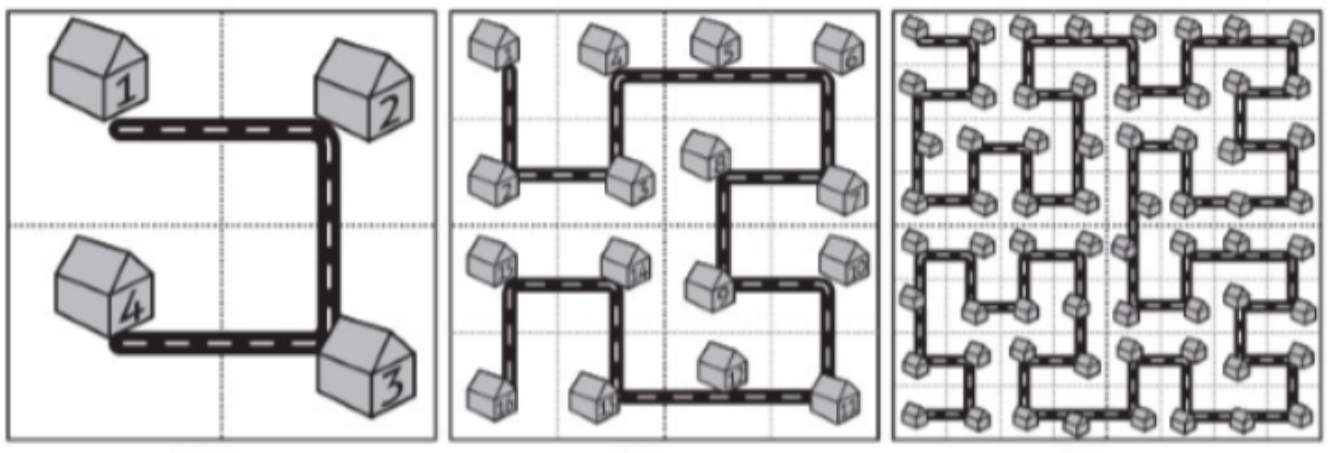
```cpp
int main()
{
    scanf("%d%d",&n,&m);
    memset(dp,0x3f,sizeof dp);
    dp[3][0]=0;
    for(int i=1;i≤n;++i) dp[3][i]=dp[3][i-1]<<1|1;//*2+1
    for(int i=4;i≤m;++i) dp[i][0]=0,dp[i][1]=1;
/*  for(int i=2;i≤n;++i)
    for(int j=1;j<i;++j)
        dp[4][i]=min(dp[4][i],2*dp[4][i-j]+mpow(2,j)-1);*/
    for(int i=4;i≤m;++i)
    for(int j=2;j≤n;++j)
    for(int k=1;k<j;++k) dp[i][j]=min(dp[i][j],dp[i][j-k]*2+dp[i-1][k]);
    printf("%lld\n",dp[m][n]);
    return 0;
}

```

6.2 分形递归

城市的规划在城市建设中是个大问题。

不幸的是，很多城市在开始建设的时候并没有很好的规划，城市规模扩大之后规划不合理的问题就开始显现。

而这座名为 Fractal 的城市设想了这样的一个规划方案，如下图所示：



当城区规模扩大之后，Fractal 的解决方案是把和原来城区结构一样的区域按照图中的方式建设在城市周围，提升城市的等级。

对于任意等级的城市，我们把正方形街区从左上角开始按照道路标号。

虽然这个方案很烂，Fractal 规划部门的人员还是想知道，如果城市发展到了等级 N，编号为 A 和 B 的两个街区的直线距离是多少。

街区的距离指的是街区的中心点之间的距离，每个街区都是边长为 10 米的正方形。

```cpp
pair<ll,ll> calc(ll n,ll m)
{
    if(n==0)
        return make_pair(0,0);
    ll len=1ll<<(n-1),cnt=1ll<<(2*n-2);
    pair<ll,ll> pos=calc(n-1,m%cnt);
    ll x=pos.first,y=pos.second;
    ll z=m/cnt;
    if(z==0)
        return make_pair(y,x);
    if(z==1)
        return make_pair(x,y+len);
```

```
    if(z==2)
        return make_pair(x+len,y+len);
    if(z==3)
        return make_pair(2*len-y-1,len-x-1);
}

ll t;
int main()
{
    scanf("%lld",&t);
    while(t--)
    {
        ll n,a,b;
        scanf("%lld%lld%lld",&n,&a,&b);
        pair<ll,ll> ma=calc(n,a-1);
        pair<ll,ll> mb=calc(n,b-1);
        ll x=ma.first-mb.first,y=ma.second-mb.second;
        double dir=sqrt(double(x*x+y*y))*10;
        printf("%0.f\n",dir);
    }
    return 0;
}
```

lowbit+hash找出二进制下所有1 的位数

lowbit配合hash可以找出整数二进制表示下的所有的是1 的位数。

```
const int N=1<<20;
int H[N+1];
for(int i=0;i<=20;i++)
    H[1<<i]=i;
while(cin>>n)
{
    while(n>0){
        cout<<H[n&-n]<<" ";
        n-=n&-n;
    }
    cout<<endl;
}
```

§ 7. 二分和三分

二分的流程：

1. 分析问题，确定左右半段哪一个是可行区间，以及 mid 归属哪一半段
2. 根据分析结果，选择 `r=mid,l=mid+1,mid=(l+r)>>1` 和 `l=mid,r=dmi-1,mid=(l+r+1)>>1`
3. 二分终止条件是 `l==r` ，该值就是答案所在的位置。

7.1 三分法求单峰函数的极值

以单峰函数$f()$为例，我们在函数的定义域$[l,r]$上人为地取两个点$lmid$和$rmid$，其中$l<limd<rmid<r$，把函数分成三段:

如果 $f(lmid)<f(rimd)$，则 $lmid$ 和 $rmid$ 要么同时处于极大值点左侧，要么分别位于极大值点两侧，无论是哪种情况，都可以确定极大值点在 $lmid$ 右侧，可令 $l=lmid$ 。

如果 $f(lmid) > f(rmid)$ ，则 $lmid$ 和 $rmid$ 要么同时处于极大值点右侧，要么分别位于极大值点两侧，无论是哪种情况，都可以确定极大值点在 $rimd$ 左侧，可令 $r = rmid$。

如果在三分中遇到了$lmid = rmid$，对于严格单调的函数，此时取$l = lmid$或者$r = rmid$ 均可，对于非严格单调的函数，此时三分法不再适用。

> 给出一个 N 次函数，保证在范围 [l,r] 内存在一点 x，使得 [l,x] 上单调增，[x,r] 上单调减。试求出 x 的值。
> 输入格式：第一行一次包含一个正整数 N 和两个实数 l,r，含义如题目描述所示。
> 第二行包含 N+1 个实数，从高到低依次表示该 N 次函数各项的系数。
> 输出格式：输出为一行，包含一个实数，即为 x 的值。四舍五入保留 5 位小数。

```cpp
ll n,m;
double a[20],l,r;

//秦九韶公式求多项式
inline double f(double x)
{
    double ans=0;
    lver(i,n,0){
        ans=ans*x+a[i];
    }
    return ans;
}
int main()
{
    cin>>n>>l>>r;
    lver(i,n,0)scanf("%lf",&a[i]);
    while(l+EPS<r){
        double x=(2*l+r)/3;//三分
        double y=(2*r+l)/3;
        if(f(x)>f(y))r=y;
        else l=x;
    }
    printf("%.5f\n",l);//输出要用f
    return 0;
}

```

7.2 二分求序列最大平均值

题目描述 展开

给定一个长度为 n 的序列 a，定义 a_i 为第 i 个元素的价值。现在需要找出序列中最有价值的“段落”。段落的定义是长度在 $[S, T]$ 之间的连续序列。最有价值段落是指平均值最大的段落。

段落的平均值 等于 **段落总价值** 除以 **段落长度**。

首先二分答案，即：二分最大平均值。
我们将a全部减去mid，问题转化为判断是否存在一个长度在s~t范围内的区间它的和为正，如果有说明还有更大的平均值。用前缀和和单调队列维护。
然后用单调队列求出`sum[i]-min(sum[i-t]~sum[i-s]),`然后判断是否大于0即可。

```cpp
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
ifstream in("input.txt");
ofstream out("output.txt");
#define debug(x) cout<<"# "<<x<<endl
const ll N=100005;
const ll base=137;
const ll mod=2147483647;
const ll INF=1<<30;
```

```
ll n,m,a[N],s,t;
double sum[N],ans;

bool check(double mid)
{
    for(int i=1;i≤n;++i)
        sum[i]=sum[i-1]+(double)a[i]-mid;
    ll head=1,tail=0,q[N];
    for(int i=s;i≤n;++i)//区间的长度最少为s所以求前缀和，至少从s开始
    {
        while(head≤tail&&sum[q[tail]]≥sum[i-s])//如果≥说明队列里出现了降序
            tail--;//把最前面的丢掉直到队列保持升序为止
        q[++tail]=i-s;
        //while(!q.empty()&&q.front.index<i-t)q.pop_front();
        while(head≤tail&&q[head]<i-t)//队列里的元素多于最大上限t就pop掉最先进入的值
            head++;
        if(head≤tail&&sum[i]-sum[q[head]]≥0)//如果队列里区间和≥0说明还有更大的平均值(前缀和嘛减后的值就是区间和)
            return 1;
    }
    return 0;
}
int main()
{
    scanf("%lld",&n);
    scanf("%lld %lld",&s,&t);
    for(int i=1;i≤n;++i)
        scanf("%lld",&a[i]);
    double l=-10000.0,r=10000.0;
    while(r - l ≥ 1e-4)
    {
        double mid=(l+r)/2.0;
        if(check(mid))ans=mid,l=mid;
        else r=mid;
    }
    printf("%.3f\n",ans);
    return 0;
}

```

$\oint$ 8. 矩阵快速幂

给定 $n \times n$的矩阵 A，求 A^k。 $(n < 100)$

```
int n;
struct ahaha{
    ll a[maxn][maxn];      //一定要用long long存矩阵，否则在过程中会爆掉
    ahaha(){
        memset(a,0,sizeof a);
    }
    inline void build(){      //建造单位矩阵
        for(int i=1;i≤n;++i)a[i][i]=1;
    }
}a;
ahaha operator *(const ahaha &x,const ahaha &y){      //重载运算符
    ahaha z;
    for(int k=1;k≤n;++k)
        for(int i=1;i≤n;++i)

```

```
            for(int j=1;j≤n;++j)
                z.a[i][j]=(z.a[i][j]+x.a[i][k]*y.a[k][j]%mo)%mo;
    return z;
}
ll k;
inline void init(){
    n=read();k=read();
    for(int i=1;i≤n;++i)
        for(int j=1;j≤n;++j)
            a.a[i][j]=read();
}
int main(){
    init();
    ahaha ans;ans.build();
    do{     //递推快速幂，与普通的递推快速幂无异，但*不能缩写为*=
        if(k&1)ans=ans*a;
        a=a*a;k>>=1;
    }while(k);
    for(int i=1;i≤n;putchar('\n'),++i)
        for(int j=1;j≤n;++j)
            printf("%d ",ans.a[i][j]);
    return 0;
}
```

∮ 9. set 和 multiset

set 和 multiset 用法一样，就是 multiset 允许重复元素。
元素放入容器时，会按照一定的排序法则自动排序，默认是按照 less<> 排序规则来排序。 不能修改容器里面的元素值，只能插入和删除。
自定义 int 排序函数：（默认的是从小到大的，下面这个从大到小）

```
struct classcomp {
bool operator() (const int& lhs, const int& rhs) const
{return lhs>rhs;}
};//这里有个逗号的，注意
multiset<int,classcomp> fifth; // class as Compare
上面这样就定义成了从大到小排列了。
结构体自定义排序函数：
（定义 set 或者 multiset 的时候定义了排序函数，定义迭代器时一样带上排序函数）
struct Node
{
int x,y;
};
struct classcomp//先按照 x 从小到大排序，x 相同则按照 y 从大到小排序 6 {
bool operator()(const Node &a,const Node &b)const
{
if(a.x≠b.x)return a.x<b.x;
else return a.y>b.y;
}
}; //注意这里有个逗号
multiset<Node,classcomp>mt;
multiset<Node,classcomp>::iterator it;
主要函数:
begin() 返回指向第一个元素的迭代器
clear() 清除所有元素
count() 返回某个值元素的个数
empty() 如果集合为空，返回 true
end() 返回指向最后一个元素的迭代器
```

```
erase() 删除集合中的元素 (参数是一个元素值，或者迭代器)
find() 返回一个指向被查找到元素的迭代器
insert() 在集合中插入元素
size() 集合中元素的数目
lower_bound() 返回指向大于（或等于）某值的第一个元素的迭代器
upper_bound() 返回大于某个值元素的迭代器
equal_range() 返回集合中与给定值相等的上下限的两个迭代器
(注意对于 multiset 删除操作之间删除值会把所以这个值的都删掉，删除一个要用迭代
器)
```

∮ 10. bitset的用法

使用 `bitset` 类型需 `#include<bitset>`

`bitset` 类型在定义时就需要指定所占的空间，例如

```
bitset<233>bit;
```

`bitset` 类型可以用string和整数初始化（整数转化成对应的二进制）

```
#include<iostream>
#include<bitset>
#include<cstring>
using namespace std;
int main()
{
    bitset<23>bit (string("11101001"));
    cout<<bit<<endl;
    bit=233;
    cout<<bit<<endl;
    return 0;
}
```

输出结果

```
00000000000000011101001
00000000000000011101001
```

支持所有的位运算

```
对于一个叫做bit的bitset:
bit.size()       返回大小（位数）
bit.count()      返回1的个数
bit.any()        返回是否有1
bit.none()       返回是否没有1
bit.set()        全都变成1
bit.set(p)       将第p + 1位变成1（bitset是从第0位开始的！）
bit.set(p, x)    将第p + 1位变成x
bit.reset()      全都变成0
bit.reset(p)     将第p + 1位变成0
bit.flip()       全都取反
bit.flip(p)      将第p + 1位取反
bit.to_ulong()   返回它转换为unsigned long的结果，如果超出范围则报错
bit.to_ullong()  返回它转换为unsigned long long的结果，如果超出范围则报错
bit.to_string()  返回它转换为string的结果
```

∮ 11. 进制转换（n进制转m进制）

```cpp
#include<cstdio>
const int N=1e5+3;
int n,m,x,tmp,ansl;char a[N],ans[N];
inline int n_to_ten(){
    int x=0;
    for(int i=0;a[i];++i){
        x*=n;
        if (a[i]≥'A'&&a[i]≤'F')x+=(a[i]-'A'+10);
        else x+=(a[i]-'0');
   }
   return x;
}
inline void ten_to_m(){
    do{
        tmp=x%m;
        if(tmp<10)ans[++ansl]='0'+tmp;
        else ans[++ansl]='A'+tmp-10;
        x/=m;
    }while(x);
}
int main(){
   scanf("%d%s%d",&n,a,&m);
   x=n_to_ten();
   ten_to_m();
   for(int i=ansl;i;--i)printf("%c",ans[i]);puts("");
}
```

∮ 12. java高精度

```java
import java.math.BigInteger;
import java.util.Scanner;

public class Main {
    public static void main(String args[]) {
        // read
        Scanner in = new Scanner(System.in);
        BigInteger a = in.nextBigInteger();

        // 3 special numbers
        a = BigInteger.ZERO;
        a = BigInteger.ONE;
        a = BigInteger.TEN;

        // convert an int to BigInteger
        BigInteger b = BigInteger.valueOf(2333);
        BigInteger p = BigInteger.valueOf(998244353);

        // convert a BigInteger to int
        int i = b.intValue();
        // convert a BigInteger to long
        long l = b.longValue();

        // operations:
```

```java
        // a + b;                                   BigInteger add(BigInteger b);
        a.add(b);
        // a - b;                                   BigInteger subtract(BigInteger b);
        a.subtract(b);
        // a * b;                                   BigInteger multiply(BigInteger b);
        a.multiply(b);
        // a / b;                                   BigInteger divide(BigInteger b);
        a.divide(b);
        // a % b;                                   BigInteger mod(BigInteger b);
        a.mod(b);
        // -a;                                      BigInteger negate();
        a.negate();
        // a < 0 ? -1 : (a > 0 ? 1 : 0);            int signum();
        a.signum();
        // signum(a - b);                           int compareTo(BigInteger b);
        a.compareTo(b);
        // a == b;                                  boolean equals(BigInteger b);
        a.equals(b);

        // abs(a);                                  BigInteger abs();
        a.abs();
        // max(a, b);                               BigInteger max(BigInteger b);
        a.max(b);
        // min(a, b);                               BigInteger min(BigInteger b);
        a.min(b);
        // gcd(a, b);                               BigInteger gcd(BigInteger b);
        a.gcd(b);
        // pow(a, i);                               BigInteger pow(int i);
        a.pow(i);

        // modPow(a, b, p);                 BigInteger modPow(BigInteger b, BigInteger p);
        a.modPow(b, p);
        // modPow(a, p - 2, p);                     BigInteger modInverse(BigInteger p);
        a.modInverse(p);
        // isPrime(a); (probability:1 - 0.5^i)  boolean isProbablePrime(int certainty);
        a.isProbablePrime(i);

        // a << i;                                  BigInteger shiftLeft(int i);
        a.shiftLeft(i);
        // a >> i;                                  BigInteger shiftRight(int i);
        a.shiftRight(i);
        // a ^ b;                                   BigInteger xor(BigInteger b);
        a.xor(b);
        // a | b;                                   BigInteger or(BigInteger b);
        a.or(b);
        // a & b;                                   BigInteger and(BigInteger b);
        a.and(b);
        // ~a;                                      BigInteger not();
        a.not();
        // a & ~b;                                  BigInteger andNot(BigInteger b);
        a.andNot(b);

        // ((a >> i) & 1) == 1;                     BigInteger testBit(int i);
        a.testBit(i);
        // a | (1 << i);                            BigInteger setBit(int i);
        a.setBit(i);
        // a & ~(1 << i);                           BigInteger clearBit(int i);
        a.clearBit(i);
        // a ^ (1 << i);                            BigInteger flipBit(int i);
        a.flipBit(i);
```

```
85          // a & -a;                                  BigInteger getLowerSetBit();
86          a.getLowestSetBit();
87          // __builtin_popcount(a);                   int bitCount();
88          a.bitCount();
89          // ceil(log2(this < 0 ? -this : this+1)) int bitLength();
90          a.bitLength();
91      }
92  }
```