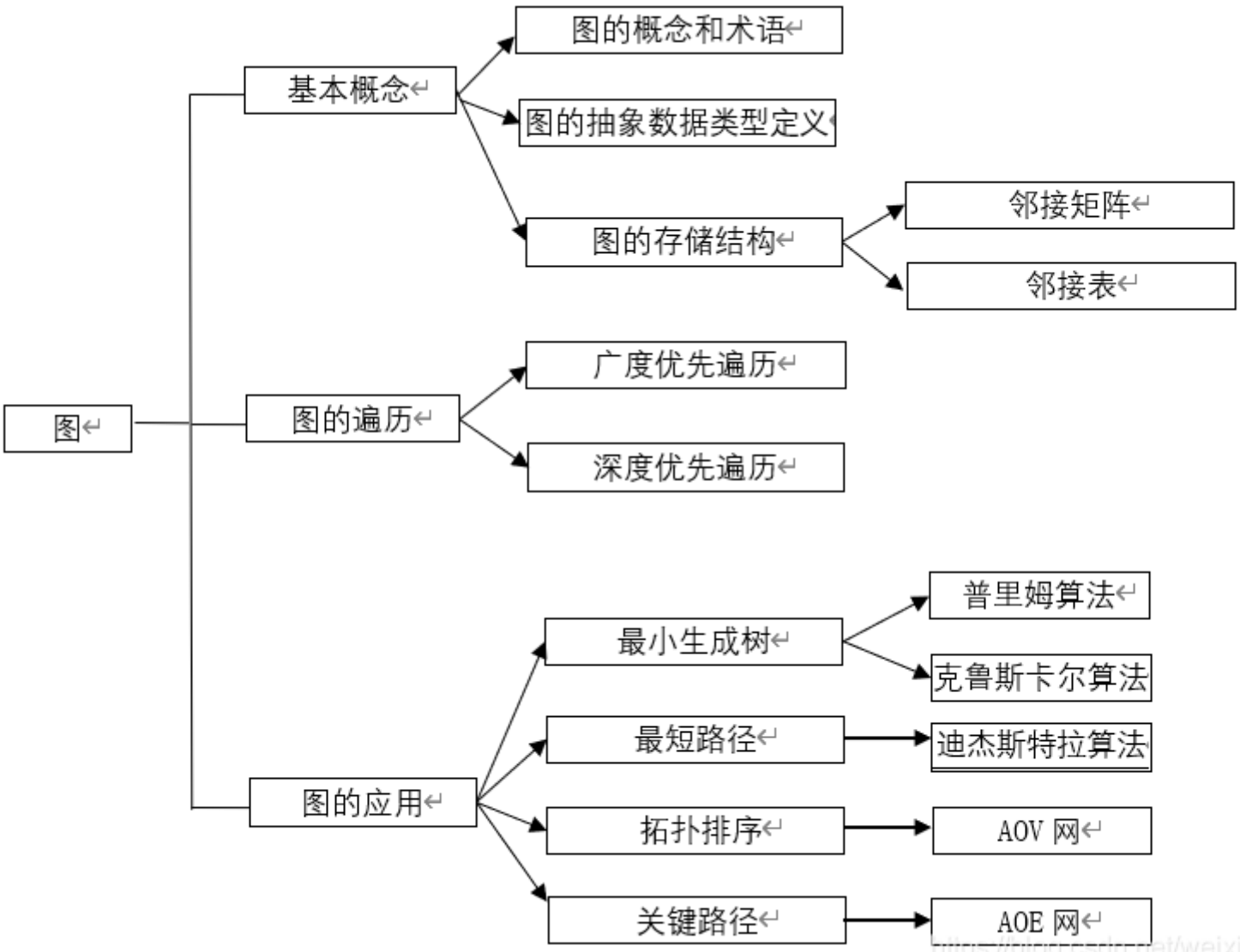
> **本系列博客为《数据结构》（C语言版）的学习笔记（上课笔记），仅用于学习交流和自我复习**

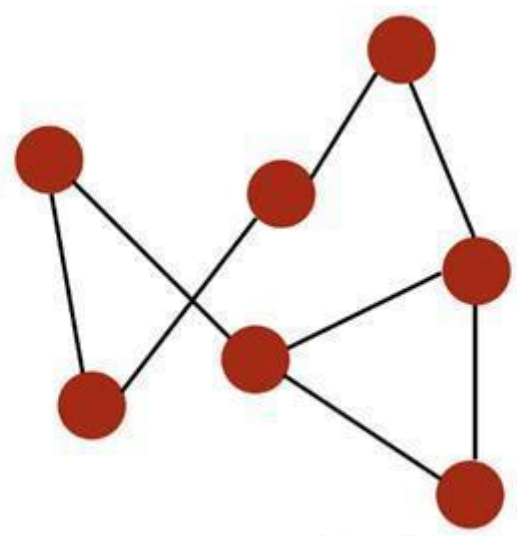
数据结构合集链接： 《数据结构》C语言版（严蔚敏版） 全书知识梳理（超详细清晰易懂）



一、 图的基本定义和术语

一.图的基本概念

图（graph）并不是指图形图像（image）或地图（map）。通常来说，我们会把图视为一种由“顶点”组成的抽象网络，网络中的各顶点可以通过“边”实现彼此的连接，表示两顶点有关联。注意上面图定义中的两个关键字，由此得到我们最基础最基本的2个概念，顶点（vertex）和边（edge）。



如上图所示，节点（vertex）用红色标出，通过黑色的边（edge）连接。

1.度

与结点关联的边数，在有向图中为入度与出度之和。

- 出度：在有向图中以这个结点为起点的有向边的数目。(可形象的理解为离开这个结点的边的数目)

- 入度：在有向图中以这个结点为终点的有向边的数目。(可形象的理解为进入/指向这个结点的边的数目)

> 任意一个图的总度数等于其边数的2倍

2.连通

如果在同一无向图中两个结点存在一条路径相连，则称这两个结点连通。

(1)连通图

如果无向图中任意两个结点都是连通的，则称之为连通图。

(2)强连通/强连通图

如果有向图中任意两个结点之间存在两条路径(即(i,j)两点中，既从i到j有一条路径，j到i也有一条路径)，则两点是强连通的。当一个图中任意两点间都是强连通的，则该图称之为强连通图。

> 在强连通图中，必定有一条回路经过所有顶点。

强连通分量：非强连通图有向图中的最大子强连通图。

3.回路

起点与相同的路径，又叫“环”。

4.完全图

任意两点间都存在边使其相连的无向图或任意两点间都存在两条不同边的有向图称作完全图

N个顶点的完全图:

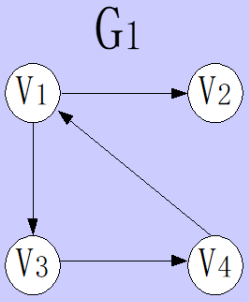
> 有向 有n(n-1)条边
> 无向 有n(n-1)/2条边

图的定义和术语

图：Graph=(V,E)

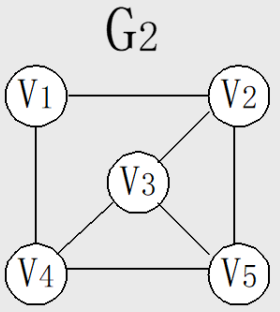
V：顶点(数据元素)的有穷非空集合；

E：边的有穷集合。



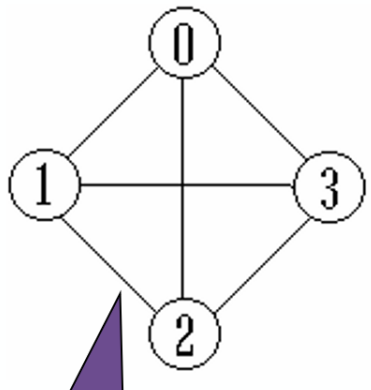
无向图：每条边都是无方向的

有向图：每条边都是有方向的



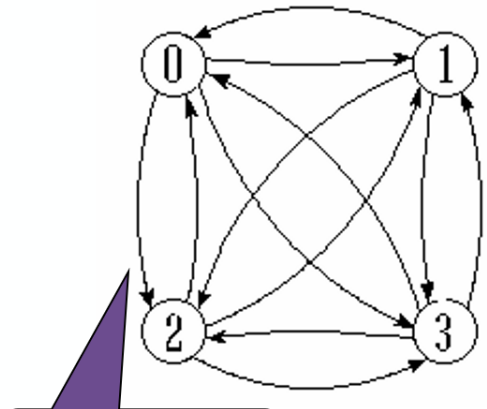
图的定义和术语

完全图：任意两个点都有一条边相连



无向完全图

$n(n-1)/2$ 条边



有向完全图

$n(n-1)$ 条边

完全图：任意两个点都有一条边相连

无向完全图

n个结点，一共有$C(n,2)$条边

有向完全图

$n(n-1)$

图的定义和术语

稀疏图

有很少边或弧的图。

网

边/弧带权的图。

稠密图

有较多边或弧的图。

邻接

有边/弧相连的两个顶点之间的关系。
存在(vi, vj)，则称vi和vj互为邻接点；
存在<vi, vj>，则称vi邻接到vj， vj邻接于vi

关联(依附)：边/弧与顶点之间的关系。
存在(vi, vj)/ <vi, vj>， 则称该边/弧关联于vi和vj

图的定义和术语

顶点的度：与该顶点相关联的边的数目，记为TD(v)

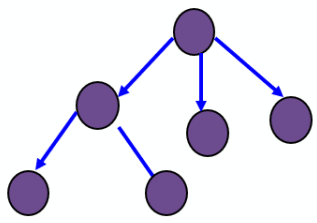
在有向图中, 顶点的度等于该顶点的入度与出度之和。

顶点 v 的入度是以 v 为终点的有向边的条数, 记作 ID(v)

顶点 v 的出度是以 v 为始点的有向边的条数, 记作OD(v)

问：当有向图中仅1个顶点的入度为0,其余顶点的入度均为1，此时是何形状?

答：是树！而且是一棵有向树！



图的定义和术语

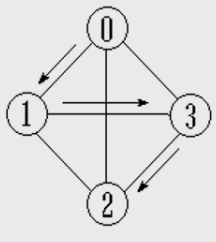
路径：接续的边构成的顶点序列。

路径长度：路径上边或弧的数目/权值之和。

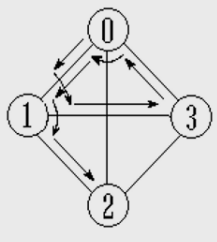
回路(环)：第一个顶点和最后一个顶点相同的路径。

简单路径：除路径起点和终点可以相同外，其余顶点均不相同的路径。

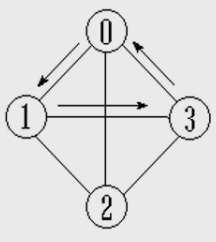
简单回路(简单环)：除路径起点和终点相同外，其余顶点均不相同的路径。



(a) 简单路径



(b) 非简单路径

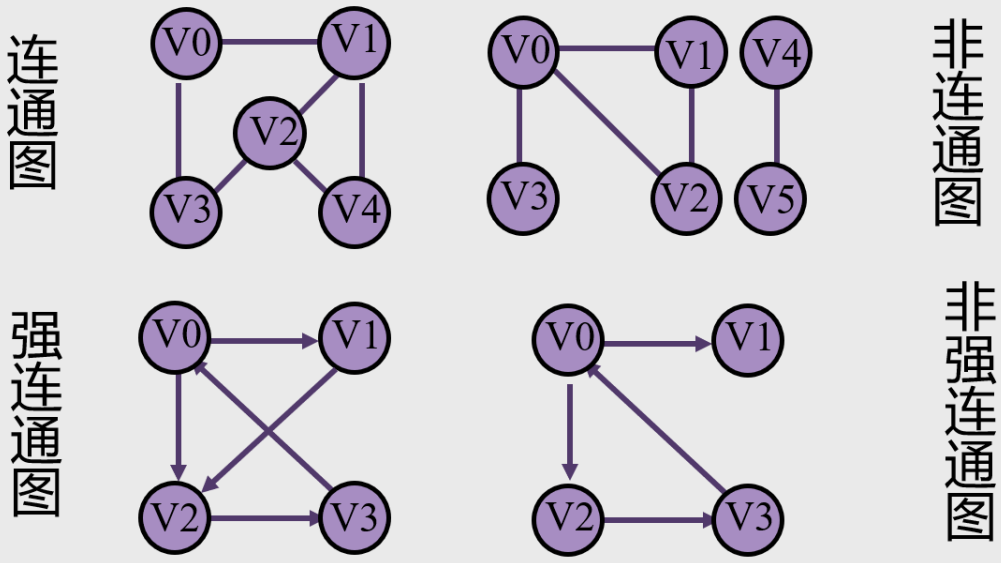


(c) 回路

图的定义和术语

连通图（强连通图）

在无（有）向图G=(V, {E})中，若对任何两个顶点 v、u 都存在从v 到 u 的路径，则称G是连通图（强连通图）。



图的定义和术语

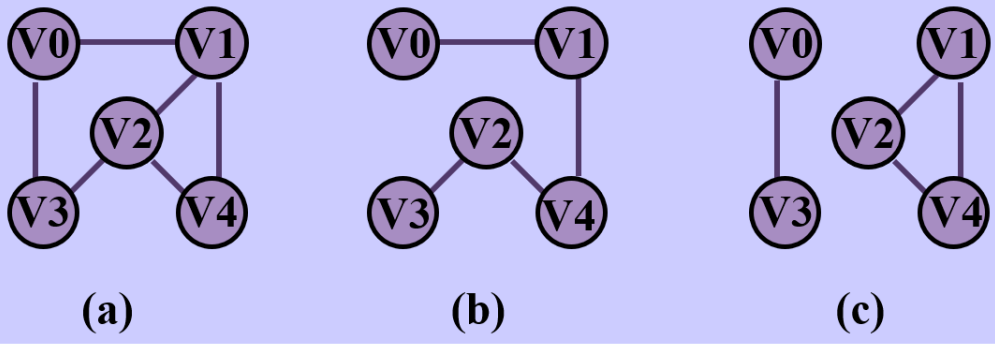
权与网

图中边或弧所具有的相关数称为权。表明从一个顶点到另一个顶点的距离或耗费。带权的图称为网。

子图

设有两个图G=（V，{E}）、G1=（V1，{E1}），若V1□V，E1 □ E，则称 G1是G的子图。

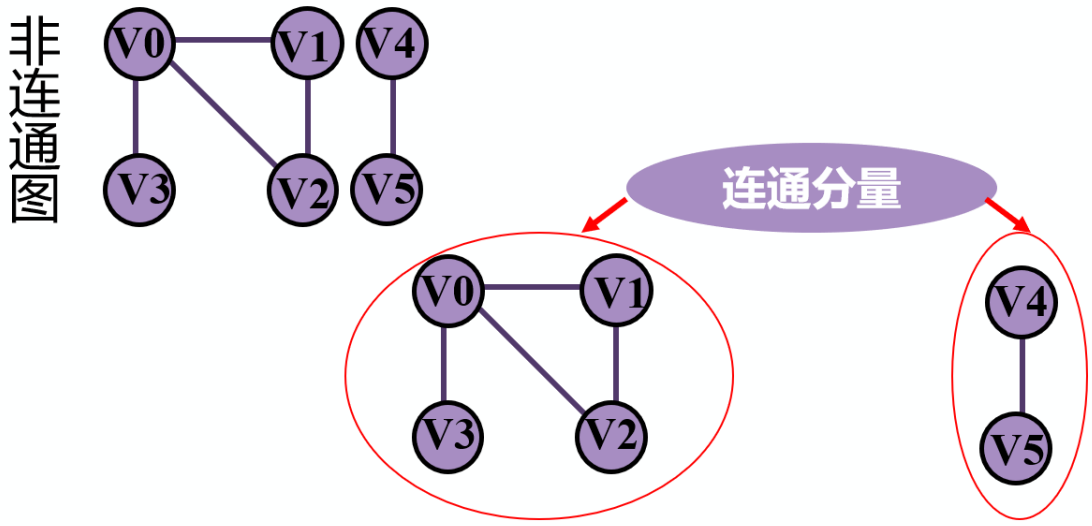
例:(b)、(c) 是 (a) 的子图



图的定义和术语

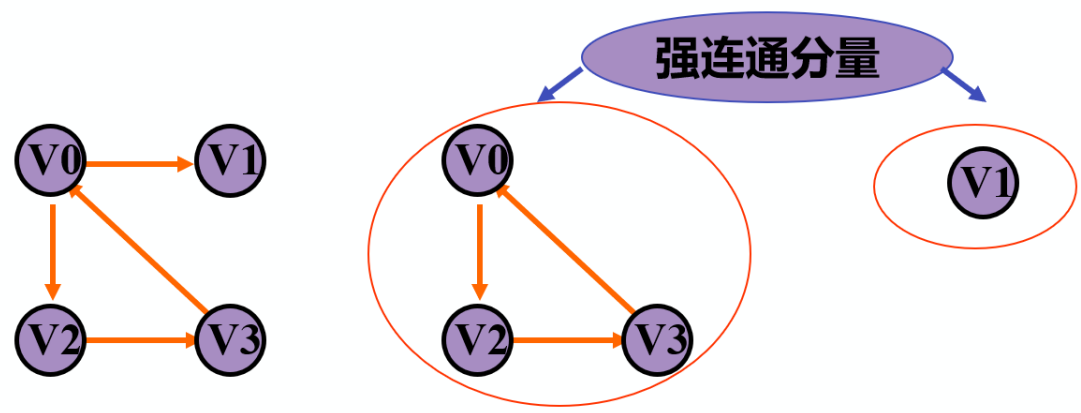
连通分量（强连通分量）

无向图G 的极大连通子图称为G的连通分量。
极大连通子图意思是：该子图是 G 连通子图，将G的任何不在该子图中的顶点加入，子图不再连通。



图的定义和术语

有向图G 的极大强连通子图称为G的强连通分量。极大强连通子图意思是：该子图是G的强连通子图，将D的任何不在该子图中的顶点加入，子图不再是强连通的。

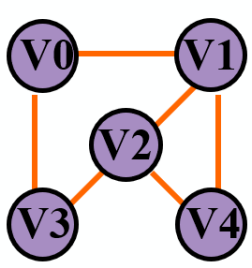


图的定义和术语

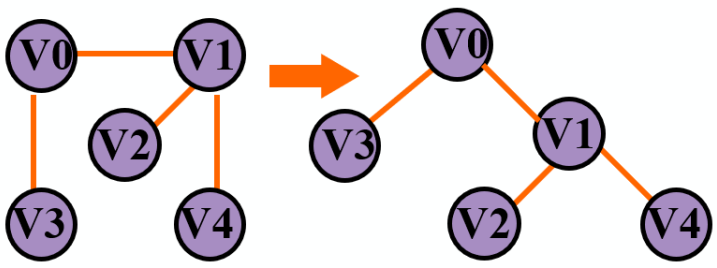
极小连通子图：该子图是G 的连通子图，在该子图中删除任何一条边，子图不再连通。

生成树：包含无向图G 所有顶点的极小连通子图。

生成森林：对非连通图，由各个连通分量的生成树的集合。



连通图 G1



G1的生成树

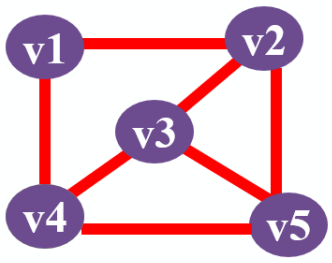
二、图的三种存储结构

1.邻接矩阵表示法

所谓邻接矩阵存储结构就每个顶点用一个一维数组存储边的信息，这样所有点合起来就是用矩阵表示图中各顶点之间的邻接关系。所谓矩阵其实就是二维数组。

```
int g[N][N];
int main() {
    int n, m; //n个点 m条边
    scanf("%d%d", &n, &m);
    int u, v; //从u到v
    for (int i = 0; i < m; ++i) {
        scanf("%d%d", &u, &v);
        g[u][v] = 1;
        //g[v][u] = 1;//无向图要建双边
        //g[u][v] = w; //带权图
    }
}
```

无向图的邻接矩阵表示法



顶点表：(v1 v2 v3 v4 v5)

邻接矩阵：

$$A.Edge = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix} \begin{matrix} v1 \\ v2 \\ v3 \\ v4 \\ v5 \end{matrix}$$

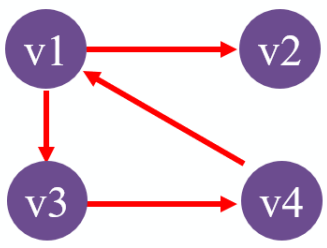
分析1：无向图的邻接矩阵是对称的；

分析2：顶点i 的度 = 第 i 行 (列) 中1 的个数；

特别：完全图的邻接矩阵中，对角元素为0，其余1。

有向图的邻接矩阵表示法

A



顶点表：(v1 v2 v3 v4)

邻接矩阵：

$$A.Edge = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{matrix} v1 \\ v2 \\ v3 \\ v4 \end{matrix}$$

注：在有向图的邻接矩阵中，

第i行含义：以结点v_i为尾的弧(即出度边）；

第i列含义：以结点v_i为头的弧(即入度边）。

分析1：有向图的邻接矩阵可能是不对称的。

分析2：顶点的出度=第i行元素之和

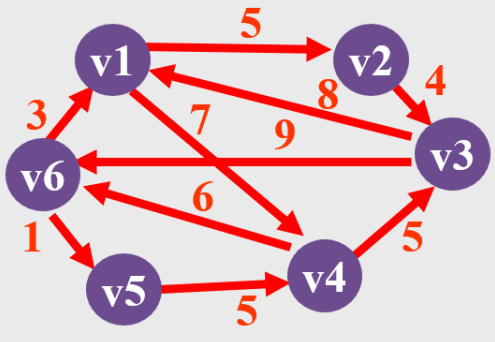
顶点的入度=第i列元素之和

顶点的度=第i行元素之和+第i列元素之和

网（即有权图）的邻接矩阵表示法

定义为：$A.Edge[i][j] = \begin{cases} W_{ij} & <vi, vj> \text{或} (vi, vj) \in VR \\ \infty & \text{无边（弧）} \end{cases}$

N



顶点表：(v1 v2 v3 v4 v5 v6)

邻接矩阵：

$$N.Edge = \begin{pmatrix} \infty & 5 & \infty & 7 & \infty & \infty \\ \infty & \infty & 4 & \infty & \infty & \infty \\ 8 & \infty & \infty & \infty & \infty & 9 \\ \infty & \infty & 5 & \infty & \infty & 6 \\ \infty & \infty & \infty & 5 & \infty & \infty \\ 3 & \infty & \infty & \infty & 1 & \infty \end{pmatrix}$$

邻接矩阵表示法的特点

优点：

容易实现图的操作，如：求某顶点的度、判断顶点之间是否有边、找顶点的邻接点等等。

缺点：

n个顶点需要$n*n$个单元存储边;空间效率为$O(n^2)$。 对稀疏图而言尤其浪费空间。

2.邻接表（链式）表示法

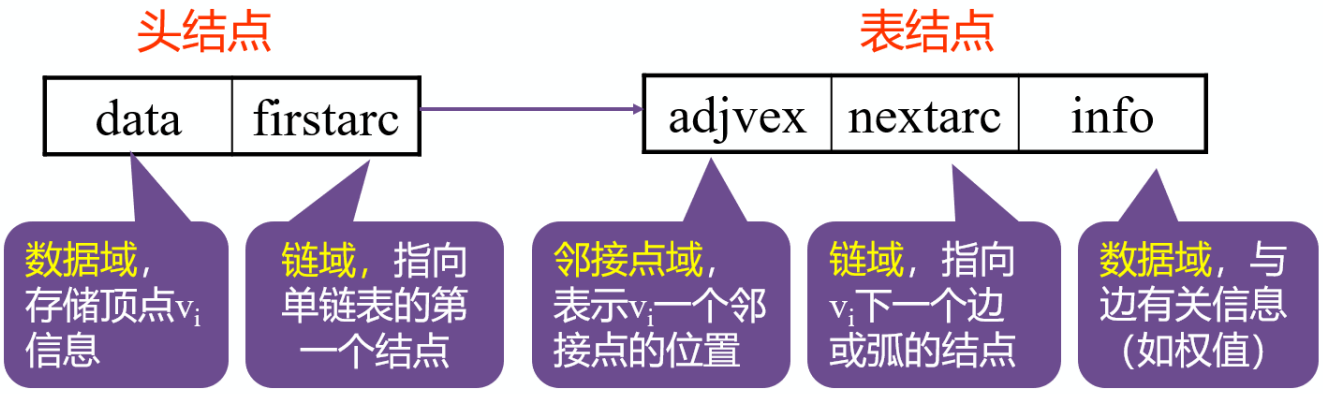
邻接表：

```cpp
#include<bits/stdc++.h>
using namespace std;
#define debug(x) cout<<"#  "<<x<<" "<<endl;
typedef long long ll;
const ll mod=2147483647;
const ll N=1e4+7;
vector<ll>G[N];//graph
/*
如果边上有属性（带权图）
struct edge
{
    ll to,cost;
};
vector<edge>G[N];
*/
int main()
{
    ll V,E;//V个顶点和E条边
    scanf("%lld %lld",&V,&E);
    for(int i=0;i<E;++i)
    {
        ll s,t;//从s到t
        scanf("%lld %lld",&s,&t);
        G[s].push_back(t);
    }
    /*
```

```
    **********各种对图的操作
    */
    return 0;
}
```

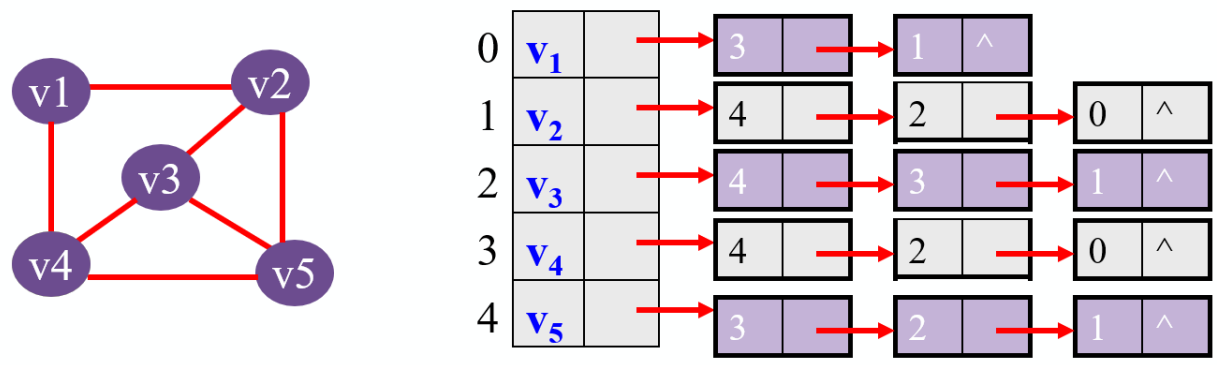
邻接表（链式）表示法

- 对每个顶点v_i建立一个单链表，把与v_i有关联的边的信息链接起来，每个结点设为3个域；



- 每个单链表有一个头结点（设为2个域），存v_i信息；
- 每个单链表的头结点另外用顺序存储结构存储。

无向图的邻接表表示



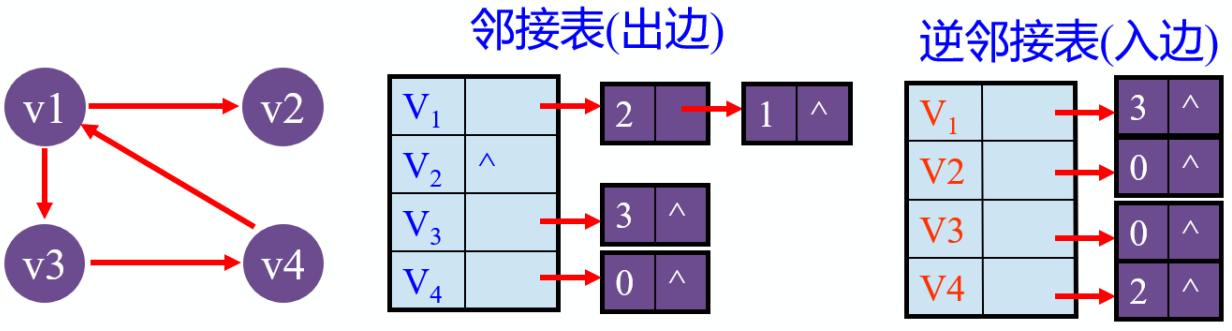
注：邻接表不唯一，因各个边结点的链入顺序是任意的

空间效率为$O(n+2e)$。

若是稀疏图($e<<n^2$)，比邻接矩阵表示法$O(n^2)$省空间。

TD(Vi)=单链表中链接的结点个数

有向图的邻接表表示



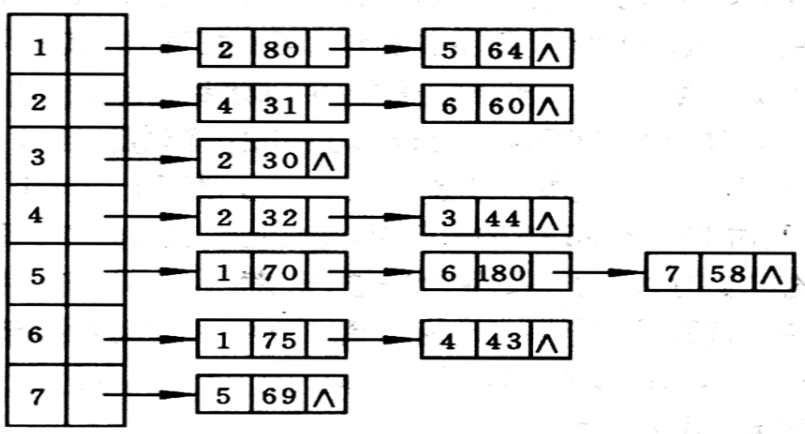
空间效率为$O(n+e)$

出度　OD(Vi) = 单链出边表中链接的结点数

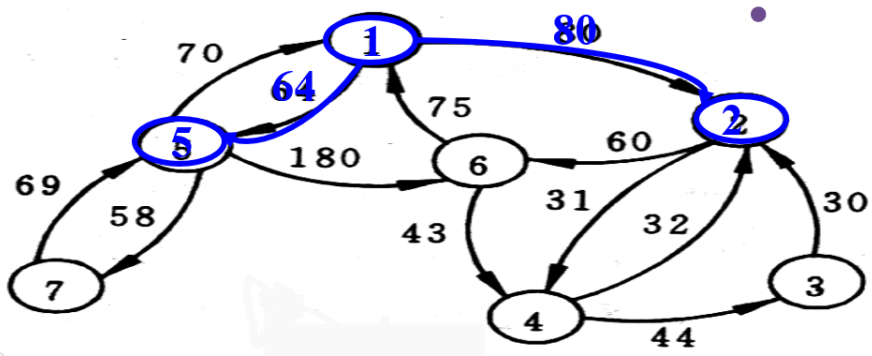
入度　ID(Vi) = 邻接点域为Vi的弧个数

度：　TD(Vi) = OD(Vi) + I D(Vi)

已知某网的邻接（出边）表，请画出该网络。



当邻接表的存储结构形成后，图便唯一确定！



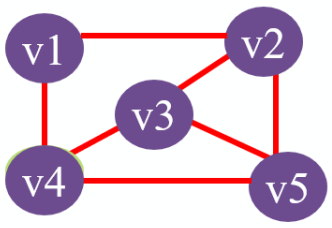

```
#define MVNum 100                          //最大顶点数
typedef struct ArcNode{                    //边结点
    int adjvex;                                 //该边所指向的顶点的位置
    struct ArcNode * nextarc;          //指向下一条边的指针
    OtherInfo info;                                 //和边相关的信息
}ArcNode;
typedef struct VNode{
    VerTexType data;                       //顶点信息
    ArcNode * firstarc;                //指向第一条依附该顶点的边的指针
}VNode, AdjList[MVNum];                //AdjList表示邻接表类型
typedef struct{
    AdjList vertices;                      //邻接表
    int vexnum, arcnum;                    //图的当前顶点数和边数
}ALGraph;

```

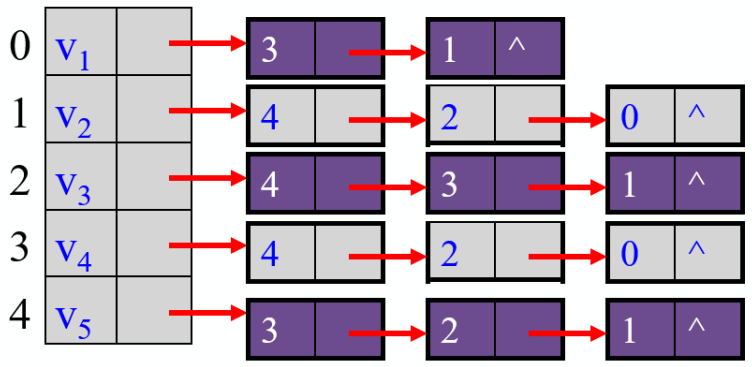
优点 ： 空间效率高，容易寻找顶点的邻接点；

缺点 ： 判断两顶点间是否有边或弧，需搜索两结点对应的单链表，没有邻接矩阵方便。

3.邻接矩阵和邻接表的区别

邻接矩阵与邻接表表示法的关系



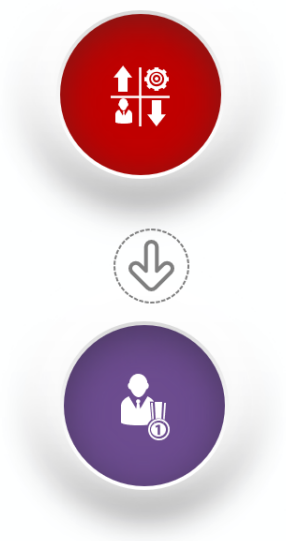


(v1 v2 v3 v4 v5)

0	1	0	1	0	v1
1	0	1	0	1	v2
0	1	0	1	1	v3
1	0	1	0	1	v4
0	1	1	1	0	v5

1. 联系：邻接表中每个链表对应于邻接矩阵中的一行，链表中结点个数等于一行中非零元素的个数。

邻接矩阵与邻接表表示法的关系



2. **区别：**

① 对于任一确定的无向图，邻接矩阵是唯一的（行列号与顶点编号一致），但邻接表不唯一（链接次序与顶点编号无关）。

② 邻接矩阵的空间复杂度为$O(n^2)$,而邻接表的空间复杂度为$O(n+e)$。

3. 用途：邻接矩阵多用于稠密图；而邻接表多用于稀疏图

https://blog.csdn.net/weixin_45697774

4.链式前向星

如果说邻接表是不好写但效率好，邻接矩阵是好写但效率低的话，前向星就是一个相对中庸的数据结构。前向星固然好些，但效率并不高。而在优化为链式前向星后，效率也得到了较大的提升（~~主要是看着舒服~~）。

```
1 struct node
2 {
3     int v,nex,val,u;
4 }e[N];
5
6 int head[N],cnt;
7
8 inline void add(int u,int v,int val)//从u到v，从父节点到子节点
```

```
{
    e[++cnt].nex=head[u];
    e[cnt].val=val;//可有可无
    e[cnt].v=v;
    e[cnt].u=u;//可有可无
    head[u]=cnt;
}
```

遍历所有结点方法：

```
for(int i=head[u];i;i=e[i].nex)
    {
        int v=e[i].v;
        ————————————
    }

//这样我们就可以遍历全部的点了！！

```

三、图的遍历

搜索引擎的两种基本抓取策略 ---深度优先/广度优先

两种策略结合=**先广后深 +权重优先**

先把这个页面所有的链接都抓取一次再根据这些URL的权重来判定URL的权重高，就采用深度优先，URL权重低，就采用宽度优先或者不抓取 。

我把我之前写的博客的内容全部直接搬过来啦 ， 下面的可能会有点难度
0x21.搜索 - 树与图的遍历、拓扑排序

注：以下图的建立都是使用链式前向星建图。

```
int head[N],ver[N],nex[N],edge[N],tot;

void add(int u,int v,int val){//链式前向星建图
    ver[++tot] = v;
    edge[tot] = val;
    nex[tot] = head[u];
    head[u] = tot;
}
```

1.）树与图的深度优先遍历及树的一些性质

1.树与图的深度优先遍历

深度优先遍历，就是在每个点x上面的的多条分支时，任意选择一条边走下去，执行递归，直到回溯到点x后再走其他的边

```
int vis[N];//标记每一个点的状态

void dfs(int u){
    vis[u] = 1;
    for(int i = head[u];i;i = nex[i]){
        int v = ver[i];
        if(vis[v])
            continue;
        dfs(v);
    }
}
```

注：下面的2，3，4，5，6小节的内容不要求掌握，我就是看着有关联就放到这里的，都是竞赛相关的内容，有兴趣可以看一下，都比较简单

2.时间戳

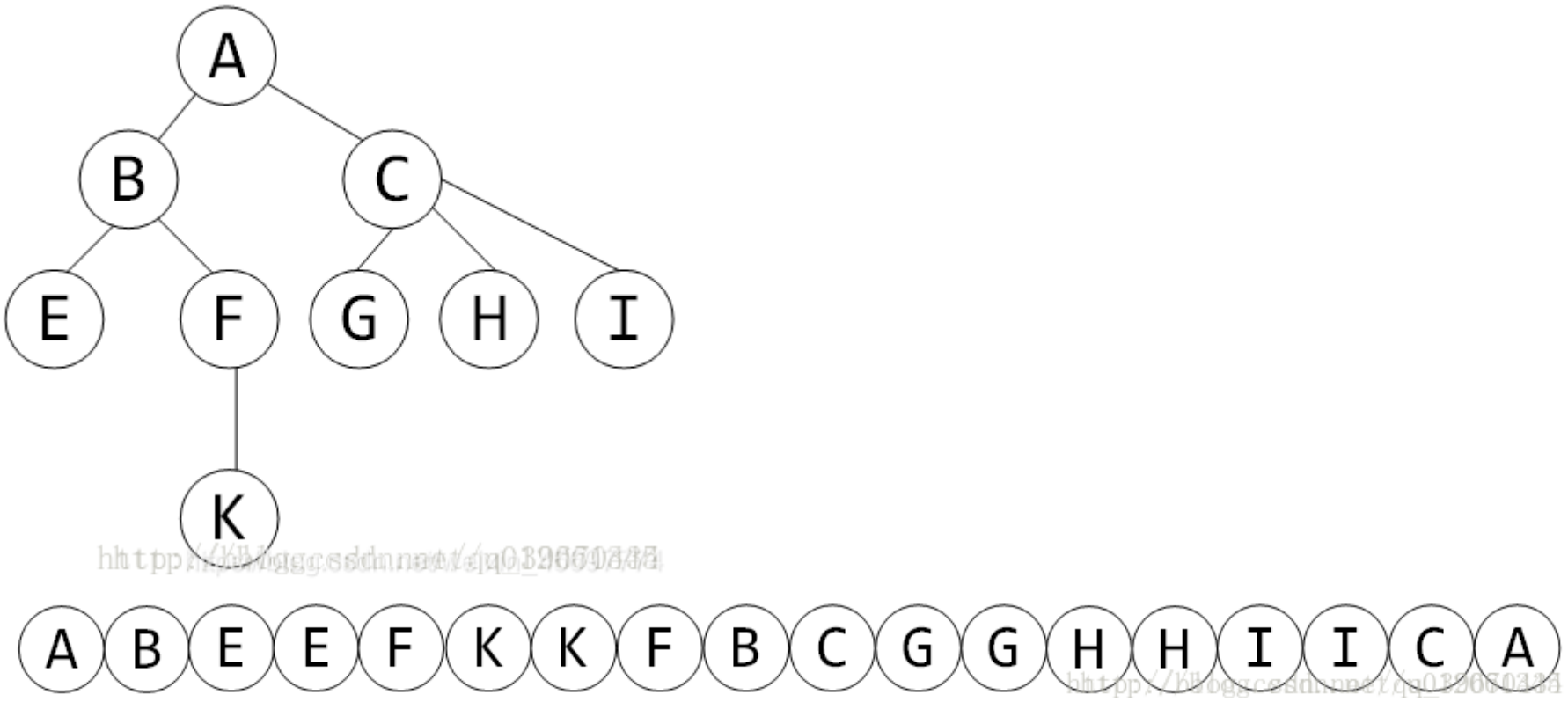
按照上述的深度优先遍历的过程，以每一个结点第一次被访问的顺序，依次赋值1~N的整数标记，该标记就被称为时间戳。
标记了每一个结点的访问顺序。

3.树的DFS

一般来说，我们在对树的进行深度优先时，对于每个节点，在刚进入递归时和回溯前各记录一次该点的编号，最后会产生一个长度为$2N$的序列，就成为该树的DFS序。

```
int a[N],cnt;
int dfs(int u){
     a[++cnt] = u;//用a数组存DFS序
     vis[u] = 1;
     for(int i = head[u]; i;i = nex[i]){
        int v = ver[i];
        if(vis[v])
            continue;
        dfs(v);
     }
     a[++cnt] = u;
}
```

DFS序的特点时：每个节点的x的编号在序列中恰好出现两次。设这两次出现的位置时$L[x], R[x]$，那么闭区间$[L[x], R[x]]$就是以x为根的子树的DFS序。
dfs序可以把一棵树区间化，即可以求出每个节点的管辖区间。
对于一棵树的dfs序而言，同一棵子树所对应的一定是dfs序中连续的一段。



放一个博客。

dfs序的七个基本问题

4.树的深度

树中各个节点的深度是一种**自顶向下**的统计信息

起初，我们已知根节点深度是0.若节点x的深度为$d[x]$,则它的子结点 y 的深度就是$d[y] = d[x] + 1$

```

int dep[N];
void dfs(int u){
    vis[u] = 1;
    for(int i = head[u];i;i = nex[i]){
        int v = ver[i];
        if(vis[v])
            continue;
        dep[v] = dep[u]+1;//父结点 u 到子结点 v  递推
        dfs(v);
    }
}
```

5.树的重心与*size*

树的重心是自底向上统计的

树的重心也叫树的质心。对于一棵树n个节点的无根树，找到一个点，使得把树变成以该点为根的有根树时，最大子树的结点数最小。

【树形DP】树的重心详解+多组例题详解

```

int vis[N];
int Size[N];
int ans = INF;
int id;
```

```
void dfs(int u){
    vis[u] = 1;
    Size[u] = 1;//子树的大小
    int max_part = 0;
    for(int i = head[u];i;i = nex[i]){
        int v = ver[i];
        if(vis[v])
            continue;
        dfs(v);
        Size[u] += Size[v];
        max_part = max(max_part,Size[v]);//比较儿子的size因为这里是假设以u为重心
    }
    max_part = max(max_part,n-Size[u]);//n为整棵树的结点数
    if(max_part<ans){//更新
        ans = max_part;//记录重心对应的max_part的值
        id = u;//记录重心位置
    }
}
```

6.图的连通块划分

若在一个无向图中的一个子图中任意两个点之间都存在一条路径（可以相互到达），并且这个子图是“极大的”（不能在扩展），则称该子图是原图的一个联通块

如下代码所示，cnt是联通块的个数，v记录的是每一个点属于哪一个联通块
经过连通块划分，可以将森林划分出每一颗树，或者将图划分为各个连通块。

```
int cnt;
void dfs(int u){
    vis[u] = cnt;//这里存的是第几颗树或者是第几块连通图
    for(int i = head[u];i;i = nex[i]){
        int v = ver[i];
        if(vis[v])
            continue;
        dfs(v);
    }
}
int main()
{
    for(int i = 1;i≤n;++i){
        if(!vis[i])//如果是颗新树就往里面搜
            ++cnt,dfs(i);
    }
}

```

DFS算法效率分析

用邻接矩阵来表示图，遍历图中每一个顶点都要从头扫描该顶点所在行，时间复杂度为$O(n^2)$。

用邻接表来表示图，虽然有 2e 个表结点，但只需扫描 e 个结点即可完成遍历，加上访问 n个头结点的时间，时间复杂度为$O(n+e)$。

结论：
稠密图适于在邻接矩阵上进行深度遍历；

稀疏图适于在邻接表上进行深度遍历。

2.）树与图的广度优先搜索

树与图的广度优先遍历，顺便求d数组（树结点的深度/图结点的层次）。

```
void bfs(){
    memset(d,0,sizeof d);
    queue<int>q;
    q.push(1);
    d[1] = 1;
    while(q.size()){
        int u = q.front();
        q.pop();
        for(int i = head[u];i;i = nex[i]){
            int v = ver[i];
            if(d[v])continue;
            d[v] = d[u]+1;
            q.push(v);
        }
    }
}
```

广度优先遍历是一种按照层次顺序访问的方法。
它具有两个重要的性质：

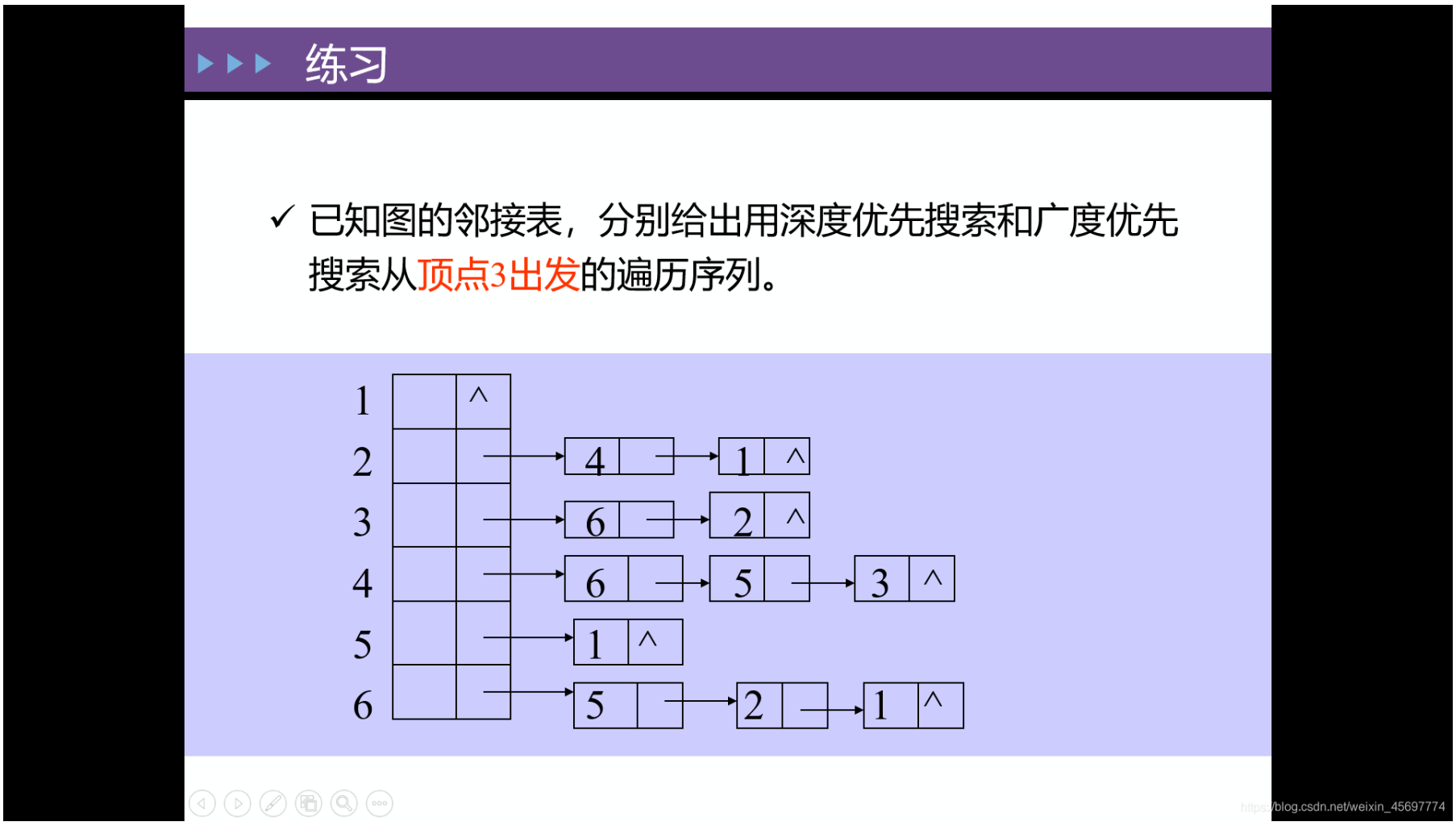
1. 在访问完所有的第i层结点后，才会访问第i+1层结点。
2. 任意时刻，队列中只会有两个层次的结点，满足“两段性”和“单调性”。

BFS算法效率分析

如果使用邻接矩阵，则BFS对于每一个被访问到的顶点，都要循环检测矩阵中的整整一行（ n 个元素），总的时间代价为$O(n^2)$。

用邻接表来表示图，虽然有 2e 个表结点，但只需扫描 e 个结点即可完成遍历，加上访问 n个头结点的时间，时间复杂度为$O(n+e)$。

练习



答案:

深度：3，6，5，1，2，4

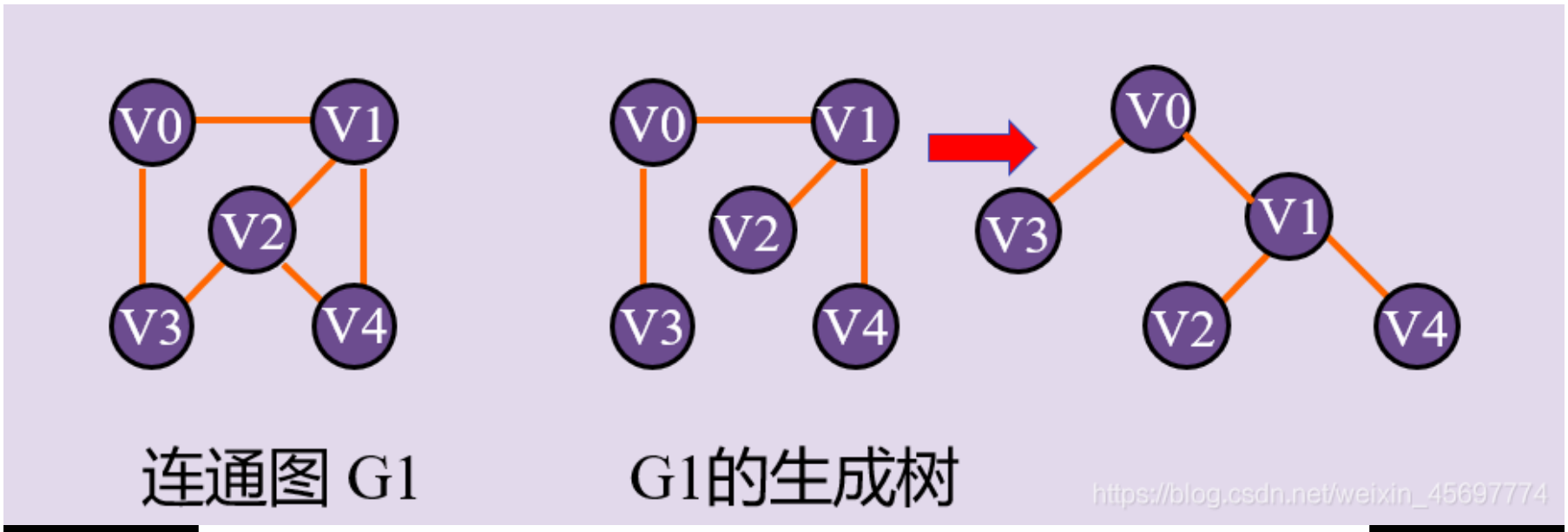
广度：3，6，2，5，1，4

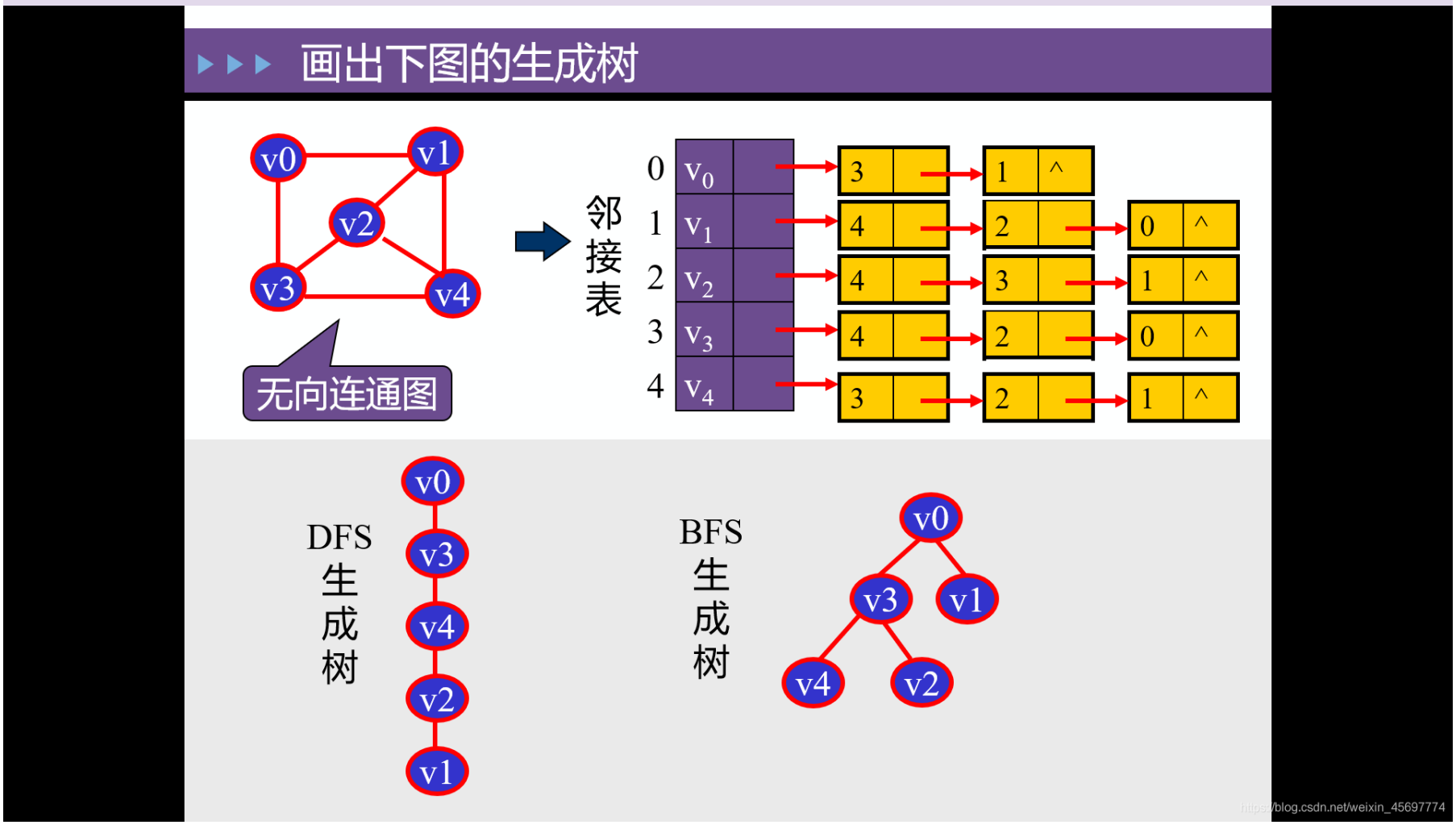
四、图的应用

本ACMer狂喜

1.最小生成树

极小连通子图：该子图是G 的连通子图，在该子图中删除任何一条边，子图不再连通。
生成树：包含图G所有顶点的极小连通子图（n-1条边）。





首先明确：

1. 使用不同的遍历图的方法，可以得到不同的生成树
2. 从不同的顶点出发，也可能得到不同的生成树。
3. 按照生成树的定义，n 个顶点的连通网络的生成树有 n 个顶点、n-1 条边。

目标：
在网的多个生成树中，寻找一个**各边权值之和最小**的生成树。

$Kruskal$算法可以简单理解为按边贪心。
$Prim$算法是以更新过的节点的连边找最小值

Prim算法适用于稠密图
Kruskal适用于稀疏图

1.$Kruskal$算法

每次选择权值最小的边，若该边两点没有加入集合，就将他加入。
起初每个点的都是一个独立的集合，把边权从小到达排序，按照边权枚举边，用并查集判断两个是否在同一个集合，如果在一个集合就跳过当前边，反之就联通这两个集合。
时间复杂度：$O(mlogm)$

给出C++代码:

```cpp
#include<iostream>
#include<algorithm>
#include<cstdio>
#include<cstring>
#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 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);
}

```

2.$Prim$算法

每次选择当前点所连的边的最小值，然后把它连起来
有些类似$Dijkstra$就是一个
普通版本的时间复杂度为$O(n^2)$
堆优化的算法时间复杂度为$O(nlogn)$
给出C++代码：

```
#include<iostream>
#include<algorithm>
#include<cstdio>
#include<cstring>
#include<bitset>
#include<vector>
#include<queue>

#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 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();
    printf("%d\n",ans);
    return 0;
}

```

2.最短路算法

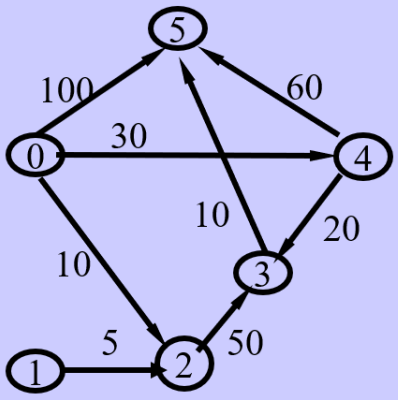
1.Dijkstra算法

经典的最短路算法，基于贪心思想的，适用于非负权值图的经过优先队列或者线段树优化后的$O(mlogn)$的优秀算法。（m是边数，n是点数）

其实也超级简单，就是从起点开始，用一个dis数组存从起点到每一个点的最短距离，每次在当前点更新dis数组（可能经过当前点u到达的v点的总距离dis[u]+edge[v]是小于dis[v]就更新），然后往下走。
最后得到一个dis数组。

从v0到其余各点的最短路径--按路径长度递增次序求解

源　点	终　点	最短路径	路径长度
v_0	v_2	(v_0, v_2)	10
	v_4	(v_0, v_4)	30
	v_3	(v_0, v_4, v_3)	50
	v_5	(v_0, v_4, v_3, v_5)	60
	v_1	无	∞



0	0	∞	10	∞	30	100
1	∞	0	5	∞	∞	∞
2	∞	∞	0	50	∞	∞
3	∞	∞	∞	0	∞	10
4	∞	∞	∞	20	0	60
5	∞	∞	∞	∞	∞	0

Dijkstra算法的思想

1.初始化：先找出从源点v_0到各终点v_k的直达路径（v_0,v_k），即通过一条弧到达的路径。

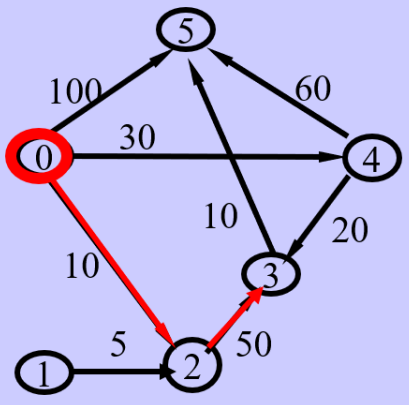
2.选择：从这些路径中找出一条长度最短的路径（v_0,u）。

3.更新：然后对其余各条路径进行适当调整：

若在图中存在弧（u,v_k），且（v_0,u）+（u,v_k）<（v_0,v_k），

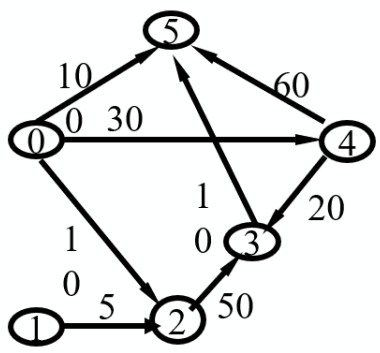
则以路径（v_0,u,v_k）代替（v_0,v_k）。

在调整后的各条路径中，再找长度最短的路径，依此类推。



存储结构（顶点个数为n）

- 主：邻接矩阵G[n][n] (或者邻接表)
- 辅：
  - 数组S[n]：记录相应顶点是否已被确定最短距离
  - 数组D[n]：记录源点到相应顶点路径长度
  - 数组Path[n]：记录相应顶点的前驱顶点



算法初始化结果

	$v=0$	$v=1$	$v=2$	$v=3$	$v=4$	$v=5$
S	true	false	false	false	false	false
D	0	∞	10	∞	30	100
$Path$	−1	−1	0	−1	0	0

【算法思想】

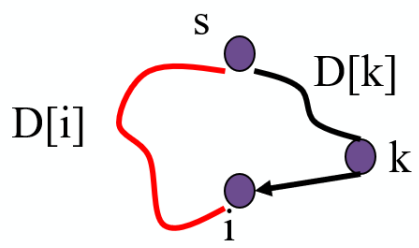
① 初始化：

- 将源点$v0$加到S中，即S[$v0$] = true；
- 将$v0$到各个终点的最短路径长度初始化为权值，即D[i] = G.arcs[$v0$][vi]，($vi \in V-S$)；
- 如果$v0$和顶点vi之间有弧，则将vi的前驱置为$v0$，即Path[i] = $v0$，否则Path[i] = −1。

② 选择下一条最短路径的终点vk，使得：

$$D[k] = \mathrm{Min}\{D[i] | vi \in V-S\}$$

▶▶▶【算法思想】



③ 将vk加到S中，即S[vk] = true。

④ 更新从$v0$出发到集合$V-S$上任一顶点的最短路径的长度，同时更改vi的前驱为vk。

若S[i]=false 且 D[k]+G.arcs[k][i]<D[i]，则D[i]=D[k]+G.arcs[k][i]; Path [i]=k;。

⑤ 重复②～④ $n-1$次，即可按照路径长度的递增顺序，逐个求得从$v0$到图上其余各顶点的最短路径。

然后就是代码了。

Dijkstra算法时间复杂度为$O(nm)$，算法瓶颈是每次查找值最小的路径，下面代码使用优先队列，直接实现自动排序，将整个算法的时间复杂度优化到$O(mlogn)$，普通的C语言写法可以写一个循环，$O(n)$来每次查找当前点的最小路径。

```
#include<bits/stdc++.h>
using namespace std;
#define debug(x) cout<<"#  "<<x<<" "<<endl;
typedef long long ll;
const ll mod=2147483647000;
const ll N=500007;
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;
}

```

3.拓扑排序

4.关键路径

我根据自己的理解改了一点ppt

这里的关键路径实际上跟最短路正好相反。

这里每到达一个结点（事件）就必须要花所有指向该结点的边的权值最大（活动可以同时进行），即所有边（活动）都完成才能触发该事件。

关键路径

- 用途：估算工程项目完成时间

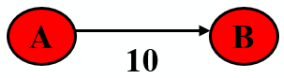
AOE网络：定义结点为事件，有向边的指向表示事件的执行次序。单位是时间（时刻）。有向边定义为活动，它的权值定义为活动进行所需要的时间。

- 术语：

源点：表示整个工程的开始点，也称起点。

收点：表示整个工程的结束点，也称汇点。

事件结点：单位时间，表示的是时刻。



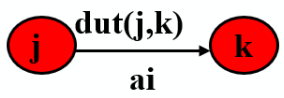
活动（有向边）：它的权值定义为活动进行所需要的时间。方向表示起始结点事件先发生，而终止结点事件才能发生。

事件的最早发生时间（Ve(j)）：从起点到本结点的最长的路径。意味着事件最早能够发生的时刻。

事件的最迟发生时间（V l (j)）：不影响工程的如期完工，本结点事件必须发发生的时刻。

活动的最早开始时间：e(ai) = Ve(j)

活动的最迟开始时间：l(ai) = V l(k) - dut(j , k)

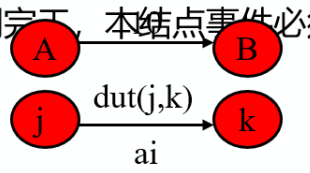


关键路径

这里结点是事件，边是活动，所有活动全部完成才能到达结点，触发事件。

事件的最早发生时间（Ve(j)）：从起点到本结点的最长的路径。意味着事件最早能够发生的时刻。

事件的最迟发生时间（V l (j)）：不影响工程的如期完工，本结点事件必须发发生的时刻。

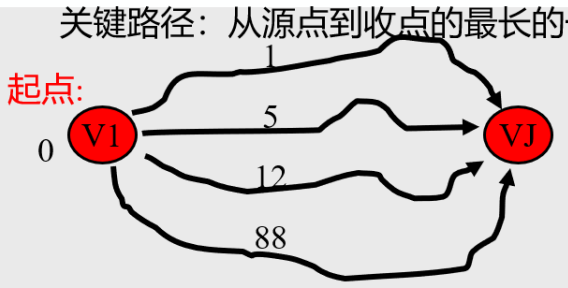


活动的最早开始时间：e(ai) = Ve(j)

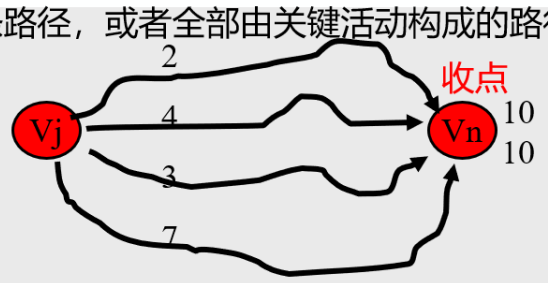
活动的最迟开始时间： l (ai) = V l(k) - dut(j , k)

关键活动：最早开始时间 = 最迟开始时间的活动

关键路径：从源点到收点的最长的一条路径，或者全部由关键活动构成的路径。



Ve(Vj) =0+ 88 取 1、5、12、88的最大值 88
（四条路代表四个活动全部完成才能到达VJ）



Vl(Vj) = 取 10-2、10-4、10-3、10-7的最小值 3;或 10 - 最长路径 7

计算其实非常简单，算好$Ve(j)$（取最大值）和$Vl(j)$（取最小值），关键是把握好下面的公式:

事件的最早发生时间（$Ve(j)$）

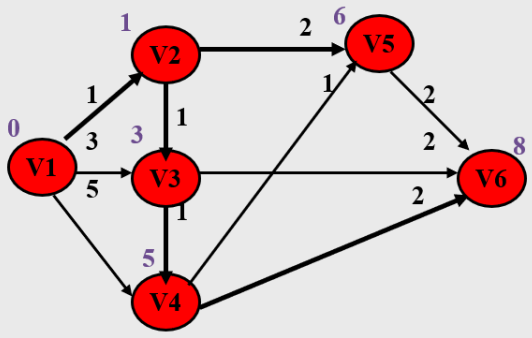
事件的最迟发生时间（$Vl(j)$）

活动的最早开始时间：$e(ai) = Ve(j)$

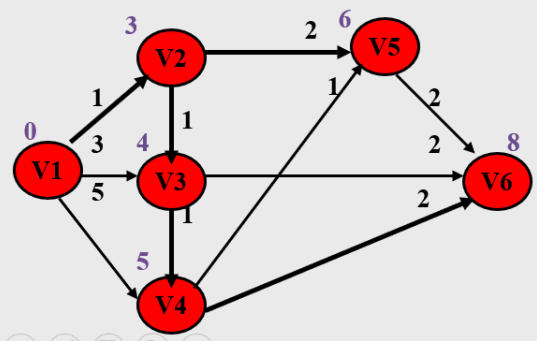
活动的最迟开始时间：$l(ai) = Vl(k) - dut(j,k)$

实例：求事件结点的最迟发生时间

- 实例的事件结点的最早发生时间



- 实例的事件结点的最迟发生时间



边	最早发生时间 Ve(j)	最迟发生时间 V l(k) - dut(j , k)	
V1 → V2	0	2	
V1 → V3	0	1	
V1 → V4	0	0	关键活动
V2 → V3	1	3	
V2 → V5	1	4	
V3 → V4	3	4	
V3 → V6	3	6	
V4 → V5	5	5	关键活动
V4 → V6	5	6	
V5 → V6	6	6	关键活动

然后就是我最喜欢的代码环节了：

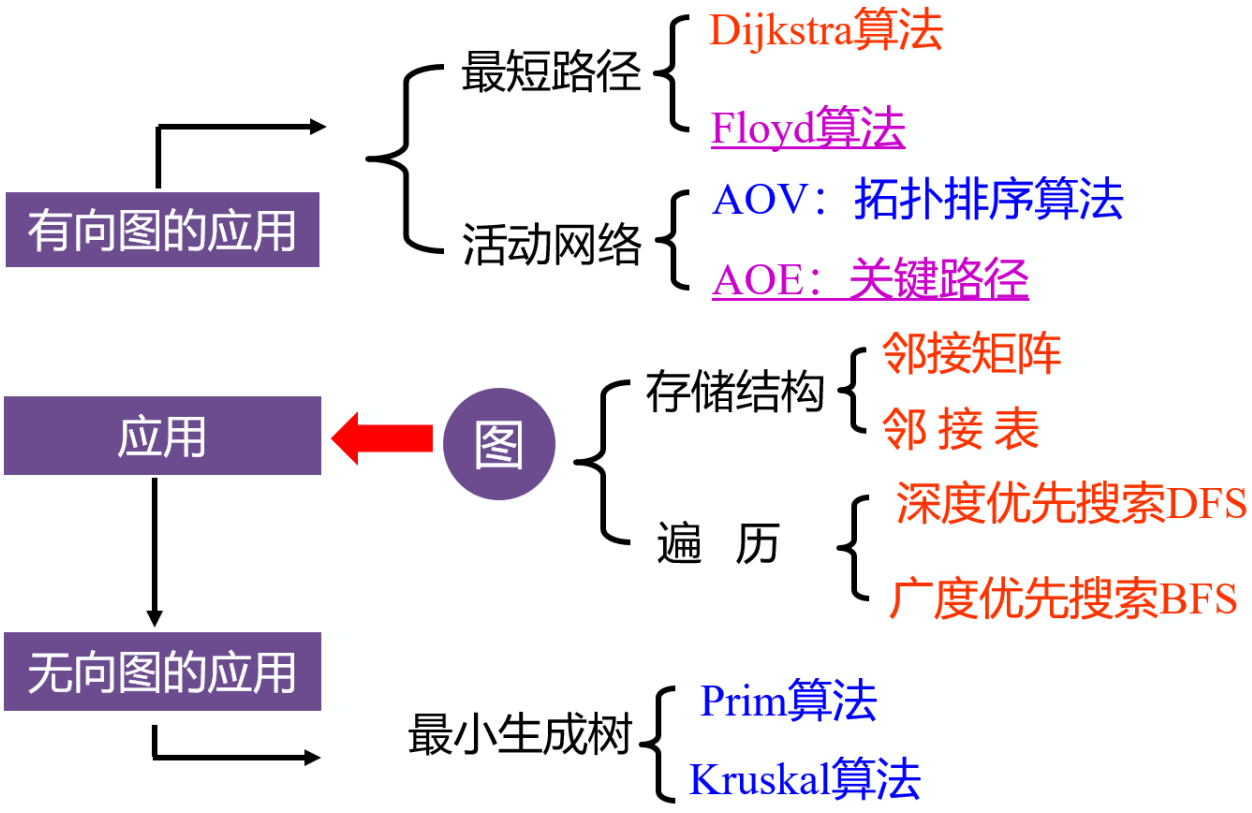
实例：求事件结点的最早发生时间

```
Status  Topologicalsort（ ALGraph G,  Stack &T）
{  FindinDegree（G，indegree）；        // 对各顶点求入度，建立入度为零的栈 S，
Initstack（T）；count = 0;
    ve [ 0 .. G.vexnum - 1 ] = 0;
while  （! StackEmpty（S））
       {  Pop（S，j）;Push（T，j）；++count;
           for  （p=G.vertices[i]. firstarc; p; p=p→nextarc）；
    { k = p→adjnexr;
       if  （!（- - indegree [ k ]））  Push（S，k）；
       if  （ve[ j ]+ *（ p→info）> ve[  k ] ）
                                ve[ k ]  = ve[ j ] +  *（ p→info）；}
                    }
        }
      if  （count < G.vexnum）return ERROR;
      else return OK;
} //  栈 T 为求事件的最迟发生时间的时候用。

```

实例：求事件结点的最迟发生时间

$$vl(i) = \min_{j}\{vl(j) - dut(\langle i,j\rangle)\}$$
$$\langle i,j\rangle \in S, \quad i = n-2,\cdots,0$$

小结



小结

01 OPTION 掌握：图的基本概念及相关术语和性质

02 OPTION 熟练掌握：图的邻接矩阵和邻接表两种存储表示方法

03 OPTION 熟练掌握：图的两种遍历方法DFS和BFS， DFS算法的实现

04 OPTION 熟练掌握：最短路算法（Dijkstra算法）的实现

05 OPTION 掌握：最小生成树的两种算法的思想及拓扑排序算法的思想

五、作业习题详解

1.选择判断

1.下面（ ）方法可以判断出一个有向图是否有环。 (2分)

A.深度优先遍历

B.拓扑排序

C.求最短路径

D.求关键路径

答案：B

对于有向图的拓扑排序,

1. 计算图中所有点的入度，把入度为0的点加入栈
2. 如果栈非空：取出栈顶顶点a，输出该顶点值，删除该顶点
3. 从图中删除所有以a为起始点的边，如果删除的边的另一个顶点入度为0，则把它入栈
4. **如果图中还存在顶点，则表示图中存在环**；否则输出的顶点就是一个拓扑排序序列

2.在图采用邻接表存储时,求最小生成树的Prim算法的时间复杂度为()
A.$O(n)$
B.$O(n+e)$
C.$O(n^2)$
D.$O(n^3)$

Prim算法的时间复杂度
邻接表存储	$O(n+e)$
邻接矩阵	$O(n^2)$

3.图中有关路径的定义是
A.由顶点和相邻顶点序偶构成的边所形成的序列
B.由不同顶点所形成的序列
C.由不同边所形成的序列
D.上述定义都不是

答案：A

A（正确）. 序偶：两个具有固定次序的客体组成一个序偶。由顶点和相邻顶点序偶构成的边的序列-----一条边对应两个端点，每条边的两个端点之间都有序偶关系----则一系列边的序列，构成有次序关系的一系列顶点的序列-----路径的定义：一个vp vi1 vi2 ... vq的顶点序列就是一条路径。 所以很清楚了，A的确能反映路径。

B. 路径分为简单路径和复杂路径，该选项只是简单路径的性质。

C. 这一系列的边之间是否有连接关系？如果只是很多不相连的线段呢?

4.若从无向图的任意一个顶点出发进行一次深度优先搜索可以访问图中所有的顶点，则该图一定是（ ）图。
A．非连通　　B．连通　　C．强连通　　D．有向

答案：B
解释：即从该无向图任意一个顶点出发有到各个顶点的路径，所以该无向图是连通图。
强连通图是指的所有的点都连通。
5.n个顶点的连通图用邻接距阵表示时，该距阵至少有（ ）个非零元素。
A．n　　B．2(n-1)　　C．n/2　　D．n2

答案：B

6.关键路径是事件结点网络中（ ）。 (2分)
A.从源点到汇点的最长路径
B.从源点到汇点的最短路径
C.最长回路
D.最短回路
答案：A
关键路径在实际应用中被当做“参考路径”，即deadline 长度最长的路径

7.下列关于AOE网的叙述中，不正确的是（　　）。

A.所有的关键活动提前完成，那么整个工程将会提前完成

B.关键活动不按期完成就会影响整个工程的完成时间

C.任何一个关键活动提前完成，那么整个工程将会提前完成

D.某些关键活动提前完成，那么整个工程将会提前完成

答案：C

2.编程题