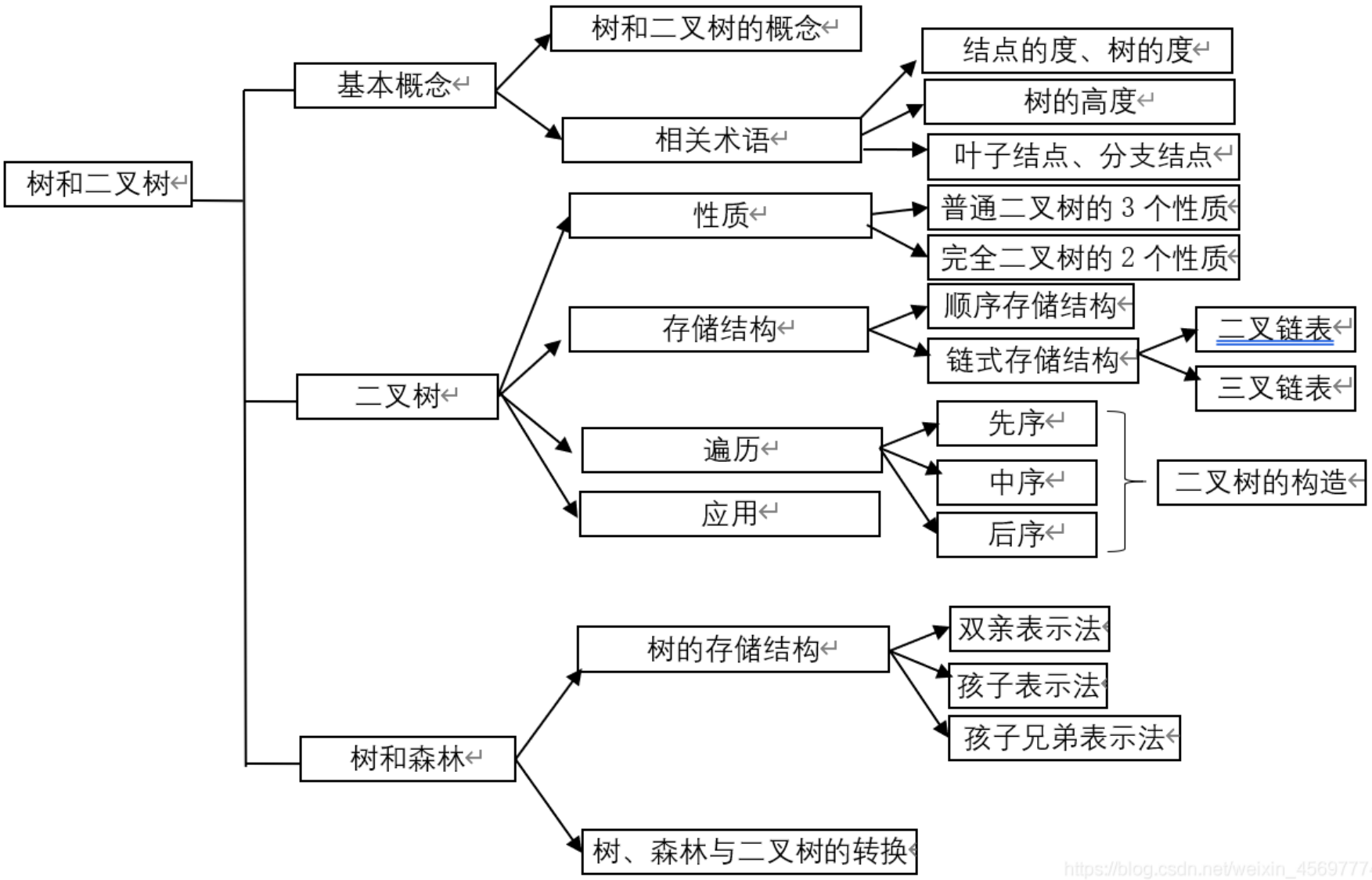
> **本系列博客为《数据结构》（C语言版）的学习笔记（上课笔记），仅用于学习交流和自我复习**
>
> ---
>
> 数据结构合集链接： 《数据结构》C语言版（严蔚敏版） 全书知识梳理（超详细清晰易懂）



树和二叉树

一.树

树：是N(N≥0)个结点的有限集合，N=0时，称为空树，这是一种特殊情况。在任意一棵非空树中应满足：

- 有且仅有一个特定的称为根的结点。
- 当N>1时，其余结点可分为m(m>0)个互不相交的有限集合T1,T2,...,Tm，其中每一个集合本身又是一棵树，并且称为根结点的子树。

基本术语

结点 ——即树的数据元素

结点的度 ——结点挂接的子树数

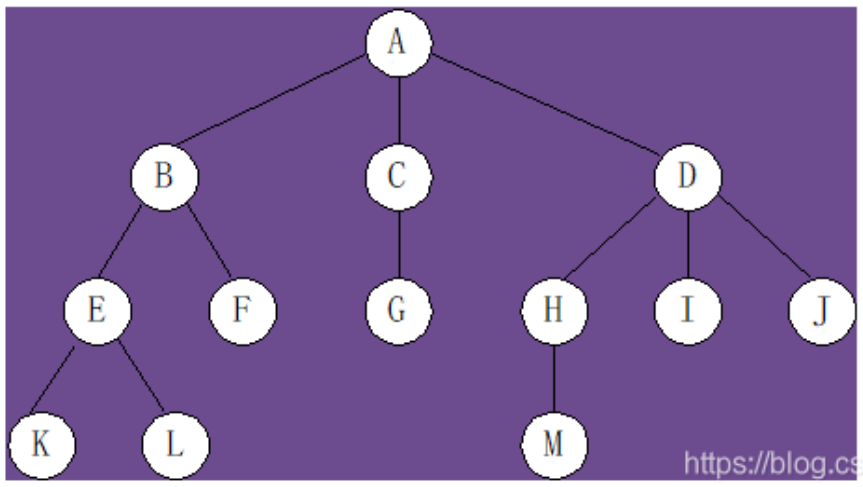
结点的层次 ——从根到该结点的层数（根结点算第一层）

终端结点 ——即度为0的结点，即叶子

分支结点 ——即度不为0的结点（也称为内部结点）

树的度 ——所有结点度中的最大值

树的深度(或高度) ——指所有结点中最大的层数



显然树的定义是递归的，是一种递归的数据结构。树作为一种逻辑结构，同时也是一种分层结构，具有以下两个特点：

1. 树的根结点没有前驱结点，除根结点之外的所有结点有且仅有一个前驱结点。
2. 树中所有结点可以有零个或者多个后继结点。

树适合于表示具有层次结构的数据。

树中的某个结点（除了根结点之外）最多之和上一层的一个结点（其父结点）有直接关系，根结点没有直接上层结点，因此在n个结点的树中最多只有`n-1`条边。而树中每个结点与其下一层的零个或者多个结点（即其子女结点）有直接关系。

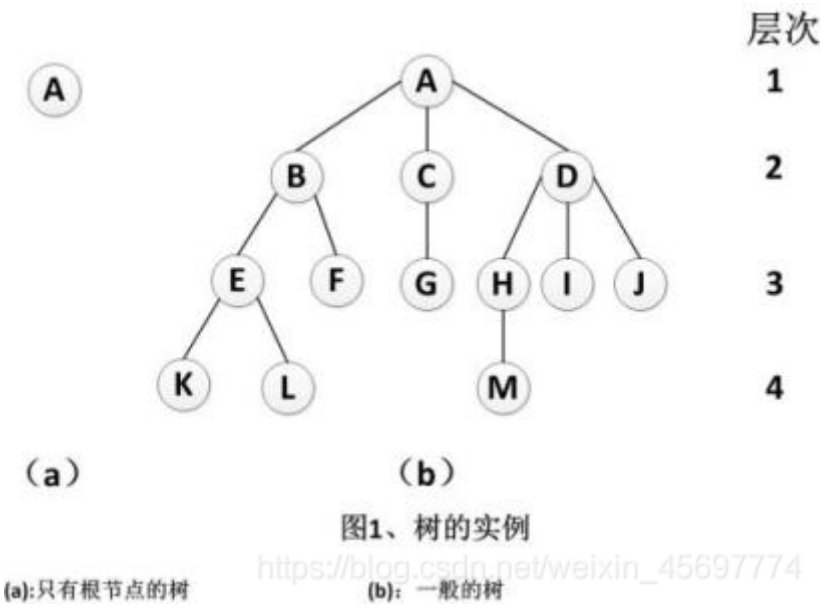


图1、树的实例

(a):只有根节点的树　　(b)：一般的树

对K来说：根结点A到K的唯一路径上的任意结点，称为K的**祖先结点**。如结点B是K的祖先节点，K是B的子孙结点。路径上最接近K的结点E称为K的**双亲结点**，K是E的**孩子结点**。根A是树中唯一没有双亲的结点。有相同双亲的结点称为**兄弟节点**，如K和L有相同的双亲结点E，即K和L是兄弟结点。

树中一个结点的子结点个数称为该结点的**度**， **树中结点最大度数称为树的度**。如B的度为2，但是D的度为3，所以该树的度为3.

度大于0的结点称为**分支结点**（又称为非终端结点）；度为0（没有子女结点）的结点称为**叶子结点**（又称终端结点）。在分支结点中，每个结点的分支数就是该节点的度。

结点的高度，深度和层次。

结点的层次从树根开始定义，根节点为第一层（有些教材将根节点定义为第0层），它的子结点为第2层，以此类推。
结点的深度是从根节点开始自顶向下逐层累加的。
结点的高度是从叶节点开始自底向上逐层累加的。

树的高度

（又称深度）是树中结点的**最大层数**。

2.有序树和无序树

树中结点的子树从左到右是有次序的，不能交换，这样的树称为**有序树**。有序树中，一个结点其子结点从左到右顺序出现是有关联的。反之称为**无序树**。在上图中，如果将子结点的位置互换，则变为一棵不同的树。
路径和路径长度：树中两个结点之间的路径是由这两个节点之间所经过的结点序列构成的，而路径长度是路径上所经过的边的个数。A和K的路径长度为3.路径为B，E。

3.森林

森林是m棵互不相交的树的集合。森林的概念和树的概念十分相近，因为只要把树的根节点删掉之后就变成了森林。反之，只要给n棵独立的树加上一个结点，并且把这n棵树作为该结点的子树，那么森林就变成了树。

4.树的基本性质

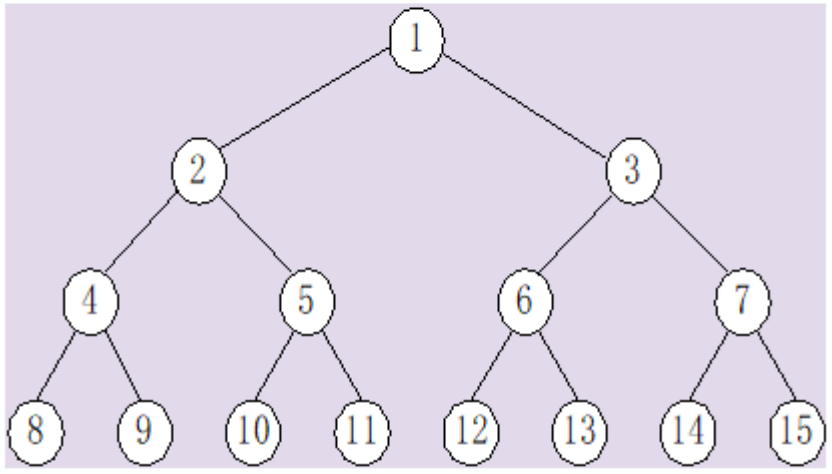
- 树中结点数等于所有节点的度数+1.
- 度为m的树中第i层上之多有mi−1个结点(i≥1)
- 高度为h的m叉树至多有mh−1m−1个结点。
- 具有n个结点的m叉树的最小高度为logm(n(m−1)+1)。

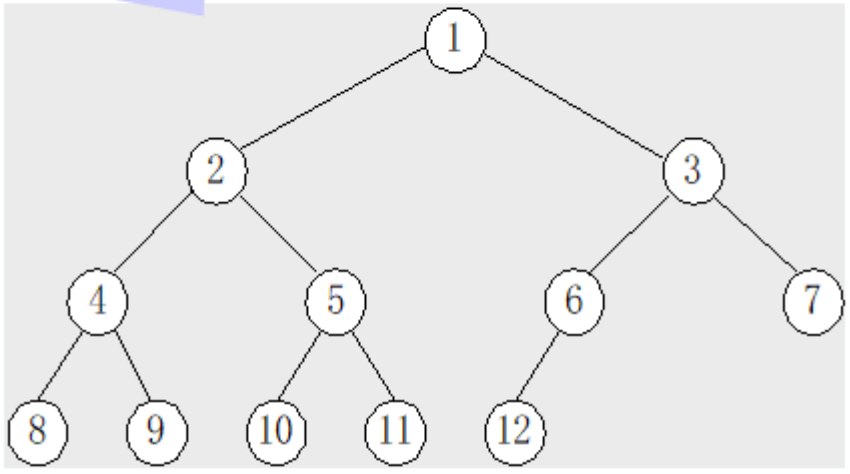
二、二叉树

- 性质1: 在二叉树的第 i 层上至多有$2i-1$个结点 ， 第i层上至少有 1个结点?
- 性质2: 深度为k的二叉树至多有2k-1个结点，深度为k时至少有 k 个结点?
- 性质3: 对于任何一棵二叉树，若2度的结点数有n2个，则叶子数n0必定为n2＋1 （即n0=n2+1）

特殊形态的二叉树

只有最后一层叶子不满，且全部集中在左边





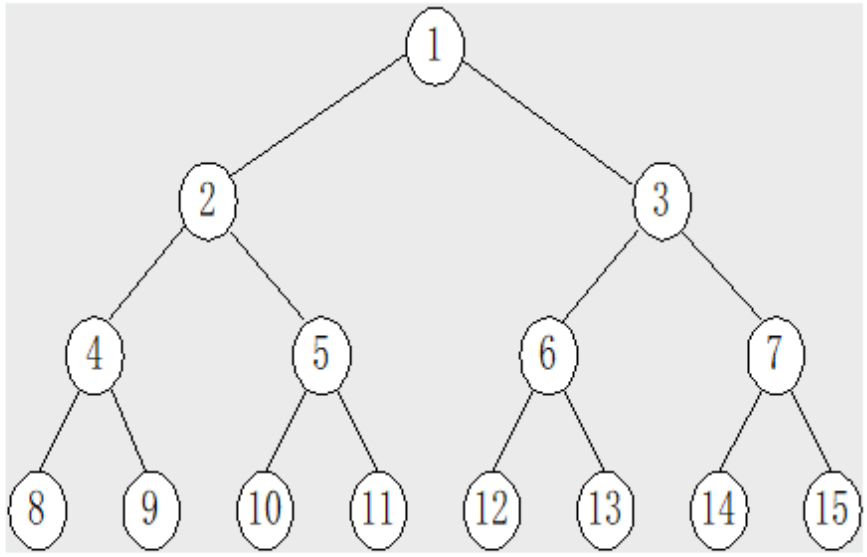
满二叉树：一棵深度为k且有2^k-1个结点的二叉树。

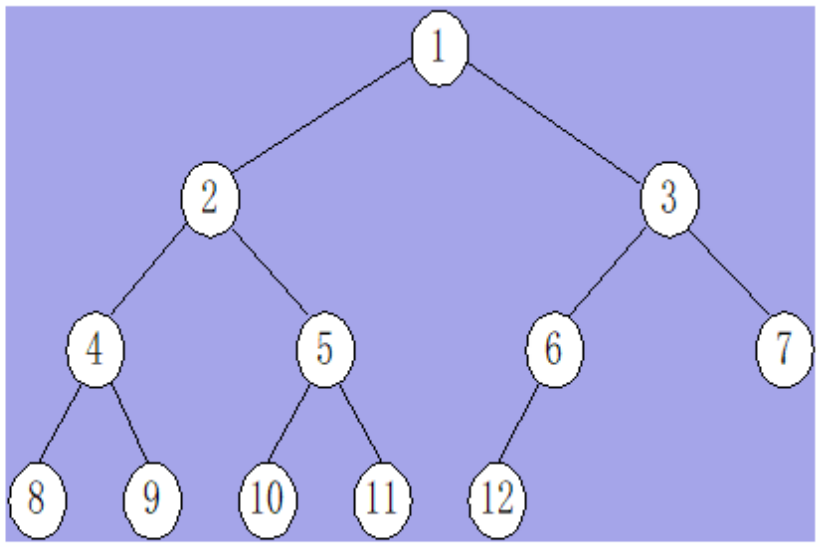
（特点：每层都“充满”了结点）

完全二叉树：深度为k的，有n个结点的二叉树，当且仅当其每一个结点都与深度为k的满二叉树中编号从1至n的结点一一对应

满二叉树和完全二叉树的区别

满二叉树是叶子一个也不少的树，而完全二叉树虽然前n-1层是满的，但最底层却允许在右边缺少连续若干个结点。满二叉树是完全二叉树的一个特例。

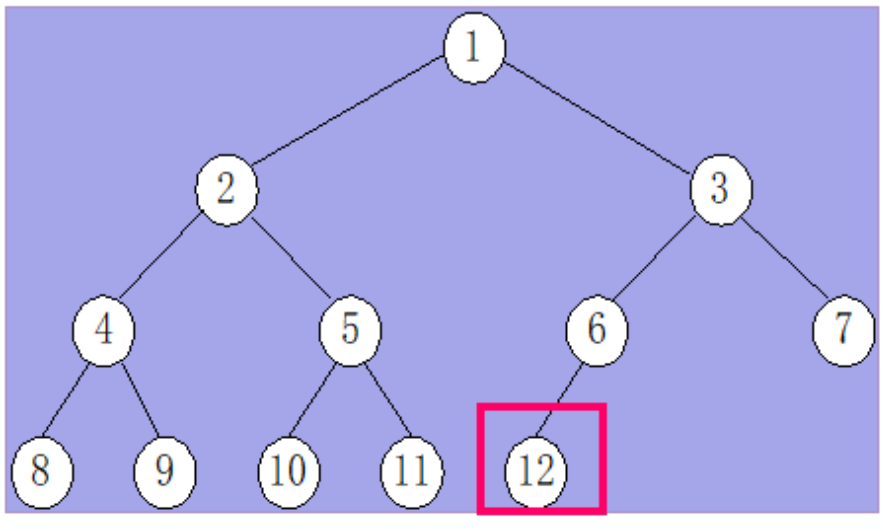




一棵完全二叉树有5000个结点，可以计算出其叶结点的个数是（ **2500**）。

二叉树的性质和存储结构

性质4: 具有n个结点的完全二叉树的深度必为$[\log_2 n] + 1$



k-1层 $2^{k-1}-1$

k层 $2^{k}-1$

n

$2^{k-1}-1 < n \le 2^{k}-1$ 或 $2^{k-1} \le n < 2^{k}$

$k-1 \le \log_2 n < k$，因为k是整数

所以$k = \lfloor \log_2 n \rfloor + 1$

性质5: 对完全二叉树，若从上至下、从左至右编号，则编号为i 的结点，其左孩子编号必为2i，其右孩子编号必为2i＋1；其双亲的编号必为i/2。

二叉树的顺序存储

实现：按满二叉树的结点层次编号，依次存放二叉树中的数据元素。

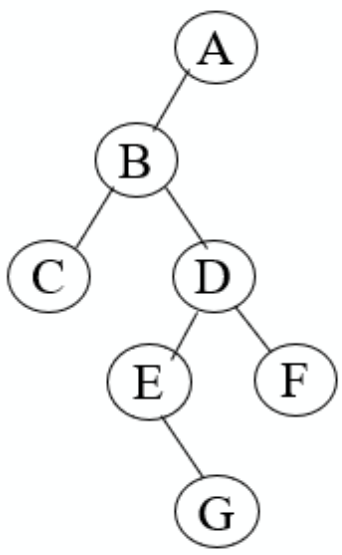
特点：

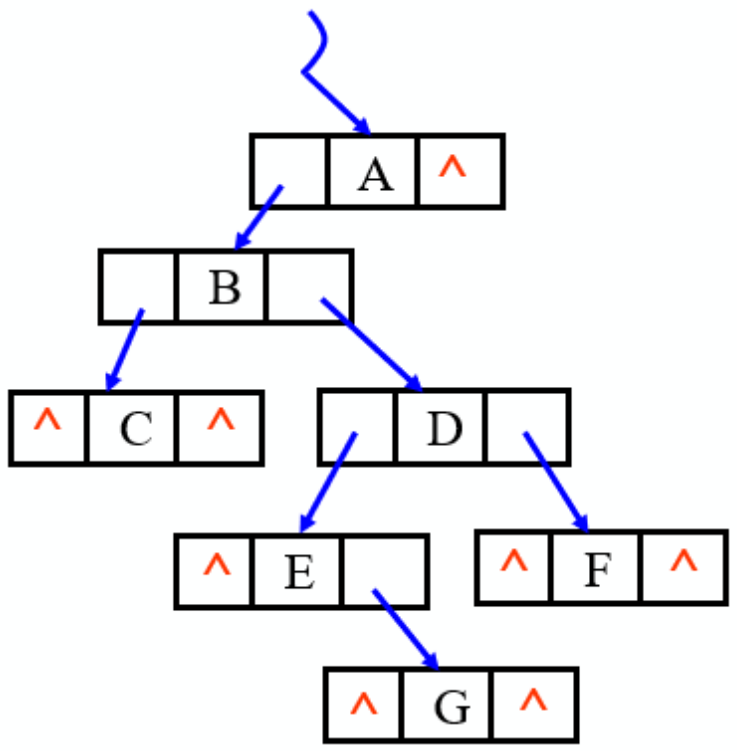
结点间关系蕴含在其存储位置中

浪费空间，适于存满二叉树和完全二叉树

二叉树的链式存储

二叉链表





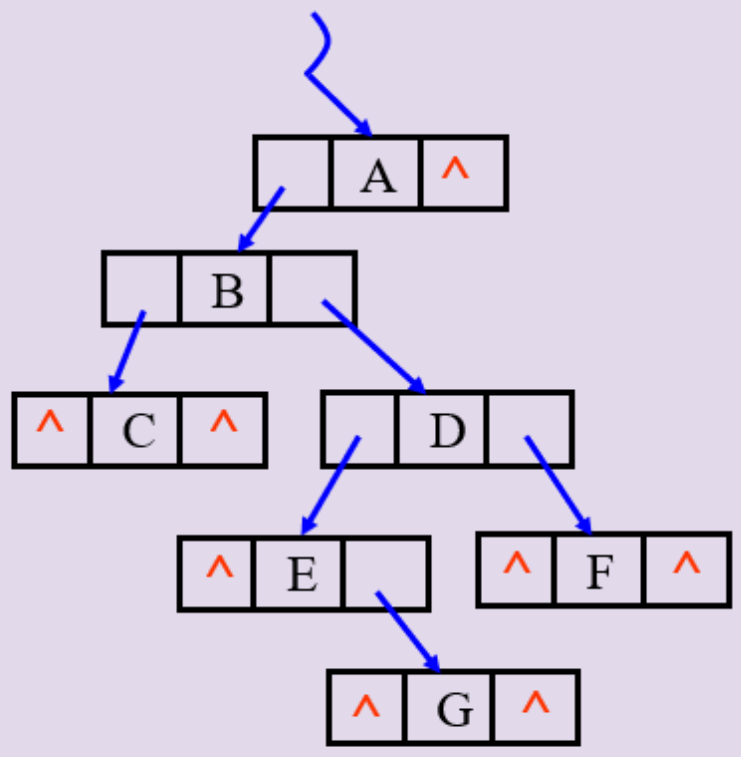

```
typedef struct BiNode{
   TElemType   data;
   struct  BiNode   *lchild,*rchild; //左右孩子指针
}BiNode,*BiTree;
```


练习

在n个结点的二叉链表中，有 **n+1** 个空指针域

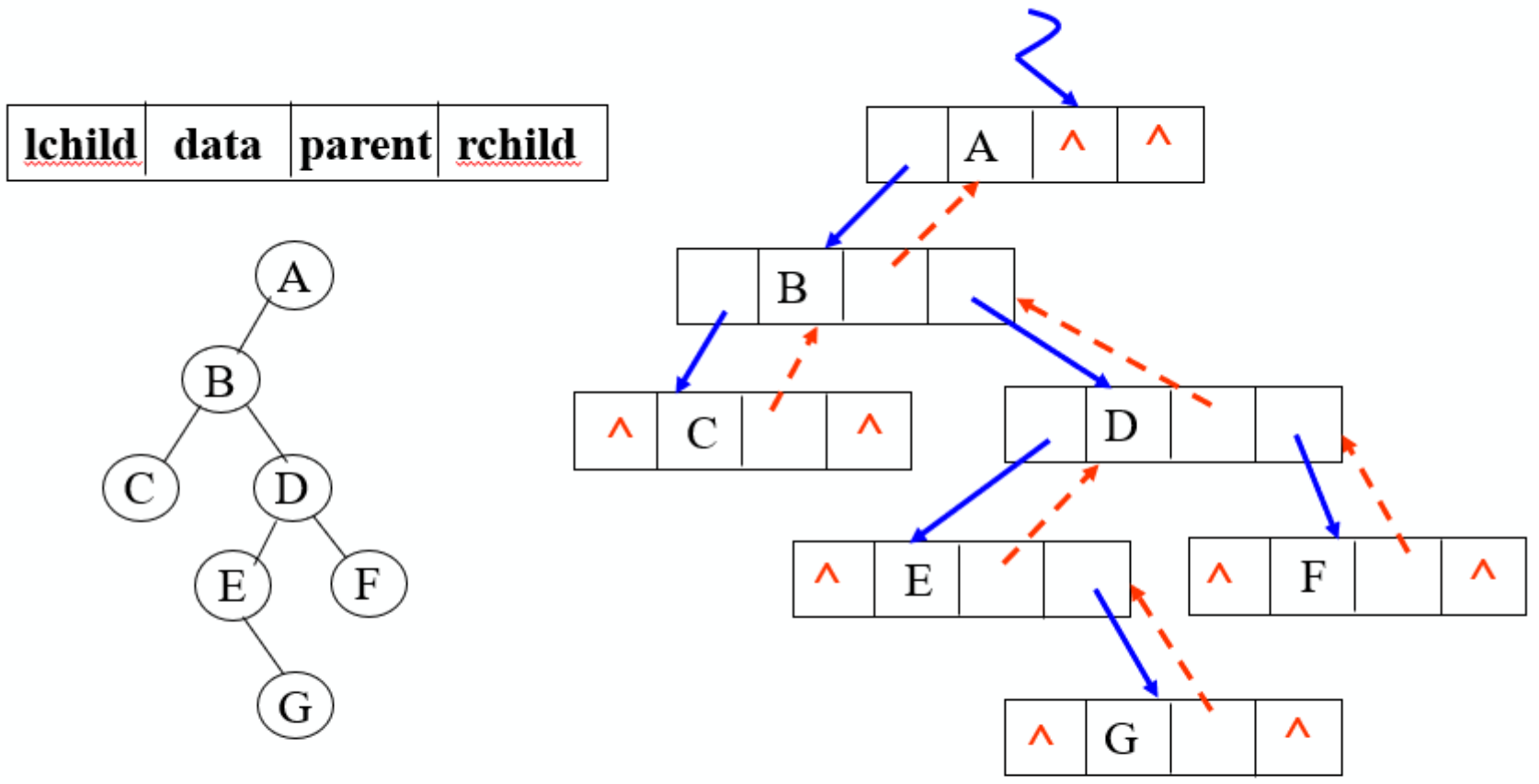
分析：必有2n个链域。除根结点外，每个结点有且仅有一个双亲，所以只会有n - 1个结点的链域存放指针，指向非空子女结点。

空指针数目 = 2n - (n-1)=n+1



三叉链表

三叉链表




```
typedef struct TriTNode
{  TelemType data;
   struct TriTNode *lchild,*parent,*rchild;
 }TriTNode,*TriTree;
```


三、树的遍历（二叉树）

遍历定义

指按某条搜索路线遍访每个结点且不重复（又称周游）。

遍历用途

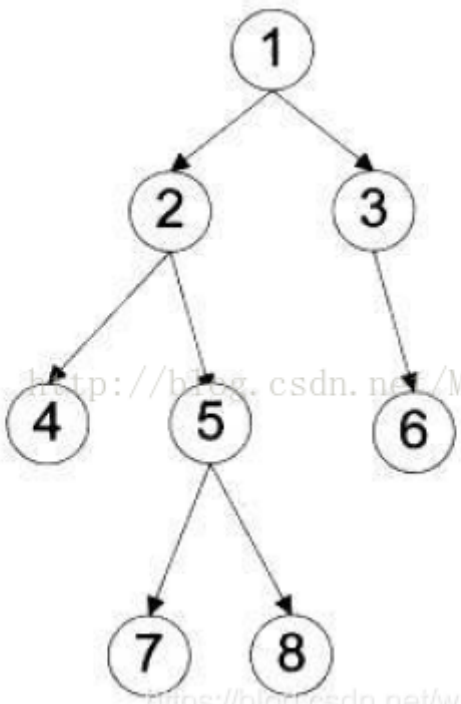
它是树结构插入、删除、修改、查找和排序运算的前提，是二叉树一切运算的基础和核心。

先(前)序遍历：对访问到的每个结点,先访问根结点,然后是左结点,然后是右结点。

中序遍历：对访问到的每个结点,先访问左结点,然后是根结点,然后是右结点。

后序遍历：对访问到的每个结点,先访问左结点,然后是右结点,然后是根结点。

例如：



先(前)序遍历：

```
1 1  2  4  5  7  8  3  6
```

中序遍历：

```
1 4  2  7  5  8  1  3  6
```

后序遍历：

```
1 4  7  8  5  2  6  3  1
```

层次遍历：

```
1 1  2  3  4  5  6  7  8
```

1.先序遍历

(1)递归写法

```
1 void PreOrder(BitTree T)
2 {
3     if(T≠NULL)
4     {
5         cout<<T→val<<"  ";
6         PreOrder(T→l);
7         PreOrder(T→r);
8     }
9 }
```

(2)先序非递归写法

根据先序遍历的顺序，先访问根结点，然后再访问左子树和右子树。所以，对于任意结点`BitTree`，第一部分即直接访问根节点，之后在判断左子树是否为空，不为空时即重复上面的步骤，直到其为空。若为空，则需要访问右子树。利用栈的先进后出，每次访问左子树时都把整个节点放到栈里面，左子树访问完了就往上走，判断父节点是否有右子树，有就访问，没有或者访问完毕就继续往上找父节点，直到将树按先序遍历方式全部遍历完毕。

```
void PreOrder (BitTree T)
{
    stack<BitTree>s;
    BitTree p = T;
    while(p || !s.empty)
    {
        if(p)
        {
            cout<<p→val<<"  ";
            s.push(p);
            p = p→l;
        }else
        {
            p = s.top();
            p = p→r;
            p = p.pop();
        }
    }
}
```

中序遍历和后序遍历的非递归方法和先序遍历一样，改一下先后顺序即可

2.中序遍历

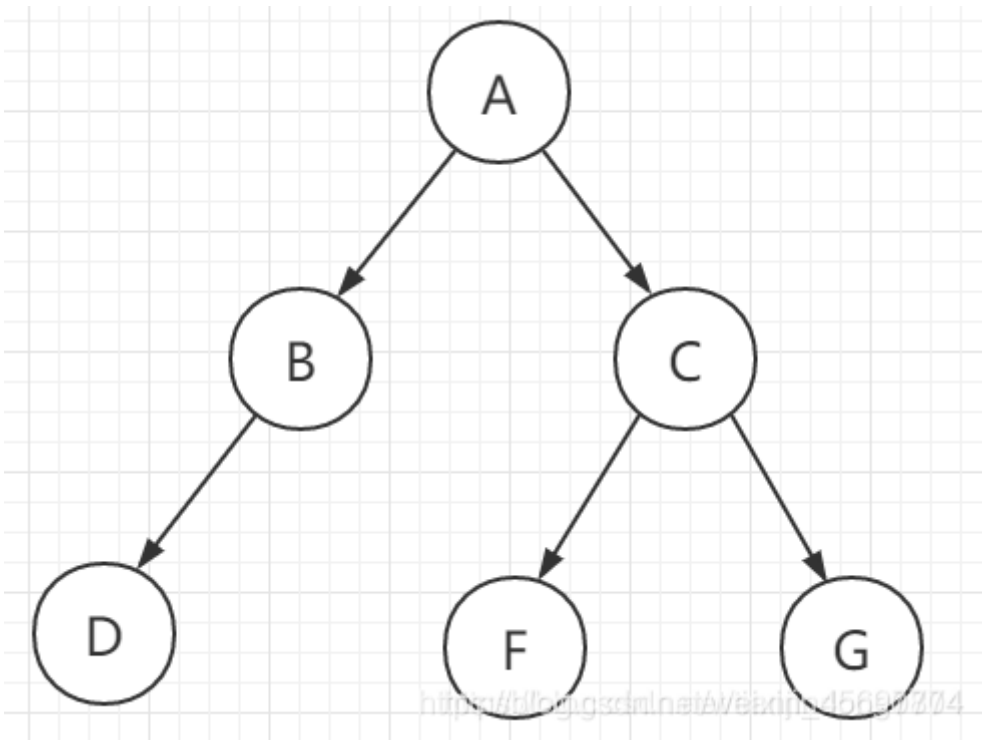
```
void InOrder(BitTree T)
{
    if(T≠NULL)
    {
        InOrder(T→l);
        printf("%d\n",T→val);
        InOrder(T→r)
    }
}
```

3.后序遍历

后序遍历的性质：对于一颗树，最后的那一个结点是**根节点**

```
void PostOrder(BitTree T)
{
    if(T≠NULL)
    {
        PostOrder(T→l);
        PostOrder(T→r);
        printf("%d\n",T→val);
    }
}
```

4.层序遍历



这棵二叉树的层序遍历次序为：A、B、C、D、F、G

每次出队一个元素，就将该元素的子节点加入队列中，直至队列中元素个数为0时，出队的顺序就是该二叉树的层次遍历结果。不懂点这里

```
void LevelOrder(BitTree T)
{
    queue<int>q;
    if (T == NULL){return;}
    q.push(T);
    while (!q.empty())
    {
        BitTree p=q.front();
        cout<<p.val;
        q.pop();
        if(p.l)
            q.push(p.l);
        if(p.r)
            q.push(p.r);
    }
}
```

以上四种种遍历算法中递归遍历左子树和右子树的顺序都是固定的，只是访问根节点的顺序不同。不管采用何种遍历方法，每个结点都是访问一次，所以时间复杂度就是 `O(n)` 。

在递归遍历中，递归工作栈的深度恰巧是树的深度，所以在最坏的情况下，二叉树是有n个结点且深度为n的单支树，递归遍历算法的时间复杂度是`O(n)`。

四、二叉树遍历算法的应用

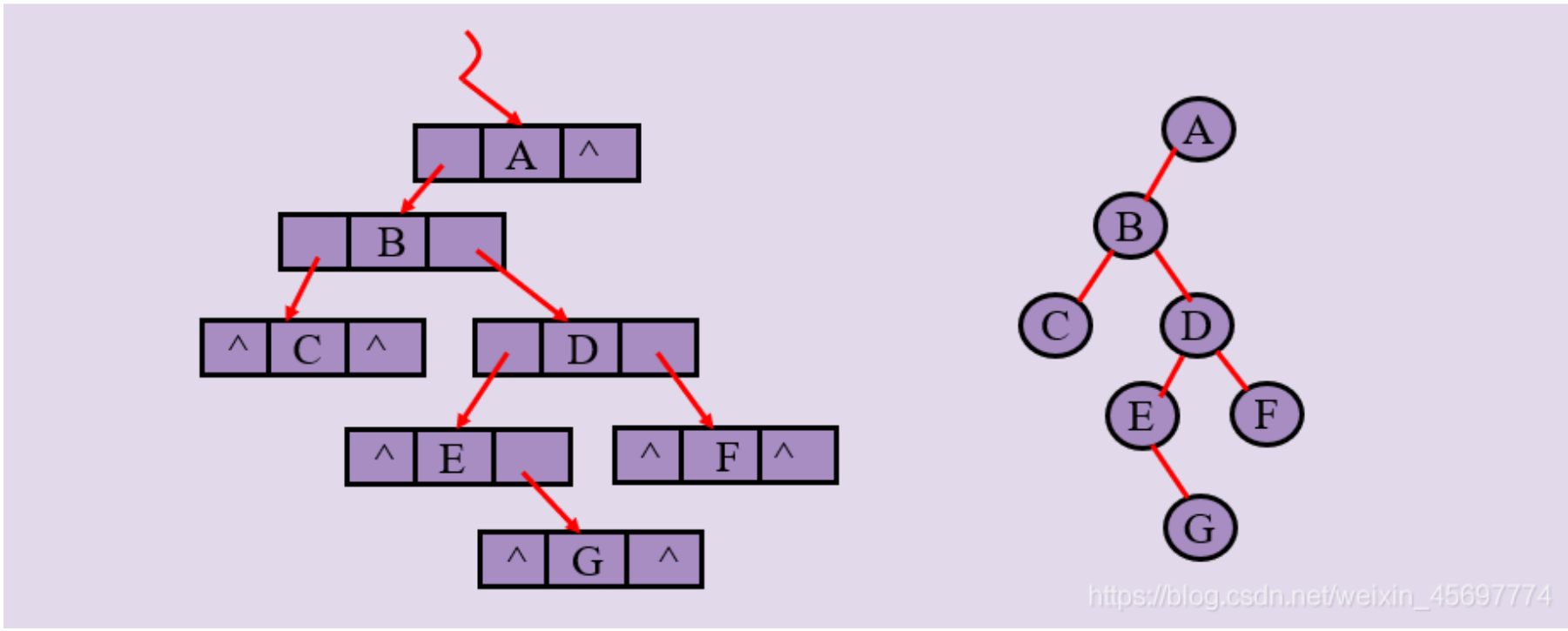
1.二叉树的建立

▶▶▶ 二叉树的建立（算法5.3）

按先序遍历序列建立二叉树的二叉链表

例：已知先序序列为：

A B C Φ Φ D E Φ G Φ Φ F Φ Φ Φ （动态演示）




```
1  void CreateBiTree(BiTree &T) {
2      cin>>ch;
3      if (ch==’#’)
4          T=NULL;      //递归结束，建空树
5      else {
6          T=new BiTNode;
7          T->data=ch;      //生成根结点
8          CreateBiTree(T->lchild);  //递归创建左子树
9          CreateBiTree(T->rchild); //递归创建右子树
10     }
11 }
12
```

2.计算二叉树结点总数

如果是空树，则结点个数为0；
否则，结点个数为左子树的结点个数+右子树的结点个数再+1。

```
int NodeCount(BiTree T){
    if(T == NULL ) return 0;
    else return NodeCount(T→lchild)+NodeCount(T→rchild)+1;
}
```

~~有线段树那味子~~

3.计算二叉树叶子结点总数

> 如果是空树，则叶子结点个数为0； 否则，为左子树的叶子结点个数+右子树的叶子结点个数。

```
int LeadCount(BiTree T){
    if(T==NULL)      //如果是空树返回0
        return 0;
    if (T→lchild == NULL && T→rchild == NULL)
        return 1; //如果是叶子结点返回1
    else return LeafCount(T→lchild) + LeafCount(T→rchild);
}
```

4.计算二叉树深度

- 如果是空树，则深度为0；
- 否则，递归计算左子树的深度记为m，递归计算右子树的深度记为n，二叉树的深度则为m与n的较大者加1。

```

int get_deep(tree *p){
    if(p == NULL)
        return 0;
    return max(deep(p→l),deep(p→r))+1;
}
```

5. 重要结论

> 若二叉树中各结点的值均不相同，则： 由二叉树的前序序列和中序序列，或由其后序序列和中序序列均能唯一地确定一棵二叉树，
> 但由前序序列和后序序列却不一定能唯一地确定一棵二叉树。

由中序遍历可以知道左右子树的结点个数。

6.由中序遍历和后序遍历建树

已知一棵二叉树的中序序列和后序序列分别是BDCEAFHG和 DECBHGFA，请画出这棵二叉树。

①由后序遍历特征，根结点必在后序序列尾部（A）；

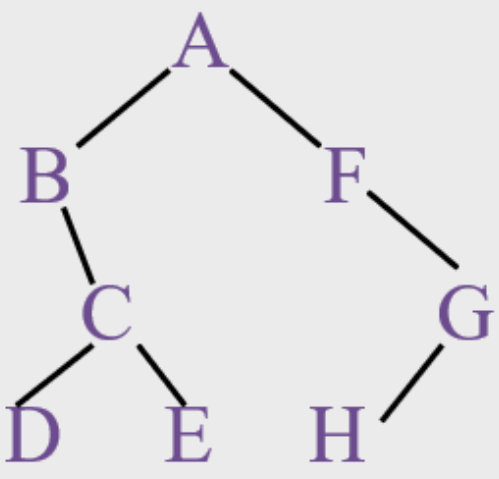
②由中序遍历特征，根结点必在其中间，而且其左部必全部是左子树子孙（BDCE），其右部必全部是右子树子孙（FHG）；

③继而，根据后序中的DECB子树可确定B为A的左孩子，根据HGF子串可确定F为A的右孩子；以此类推。

练习

中序遍历：B D C E A F H G

后序遍历：D E C B H G F A



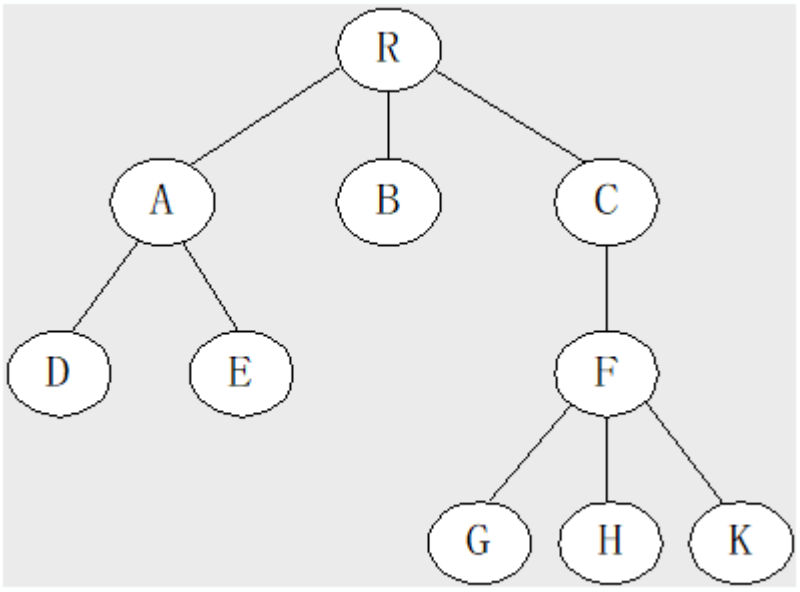
(D C E)　　(H G)

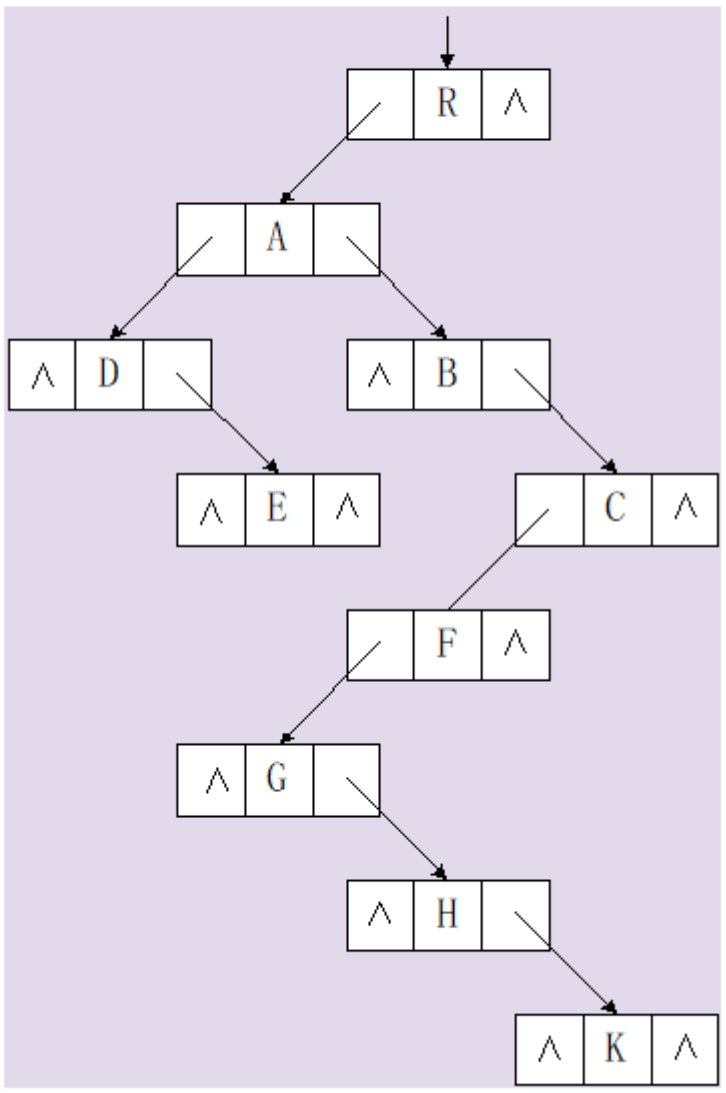
五、森林

左指针指的是左孩子，右指针指的是兄弟，这样就避免了一颗多叉树需要像二叉树一样的每一个结点使用多个指针的情况了。

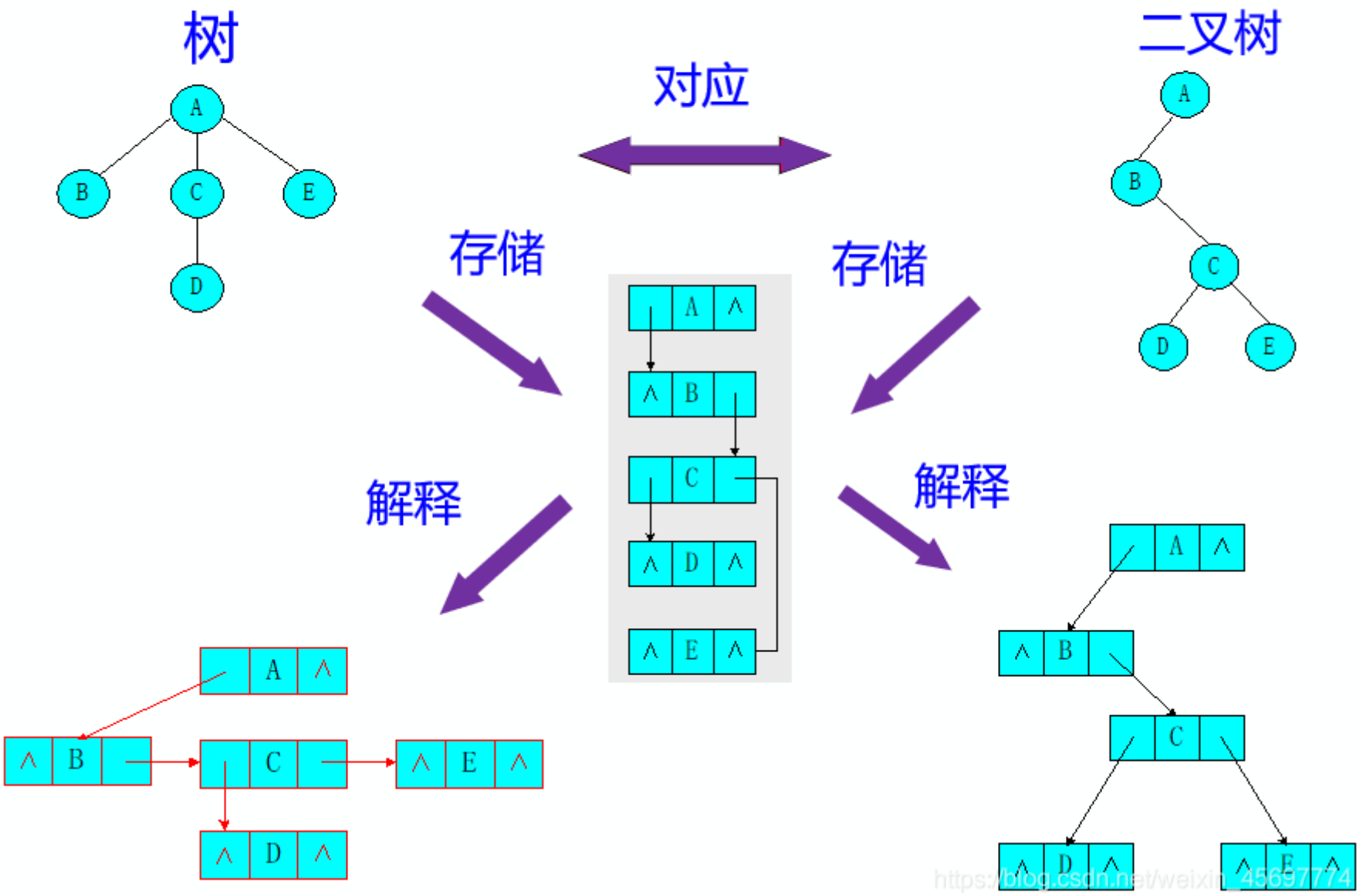
树的存储结构 - - 二叉链表表示法

```
typedef struct CSNode{
  ElemType        data;
  struct CSNode
     *firstchild,*nextsibling;
}CSNode,*CSTree;
```



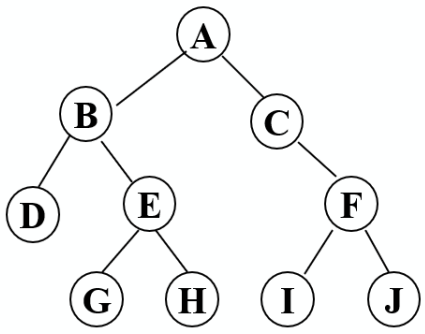


树的存储结构 - - 二叉链表表示法

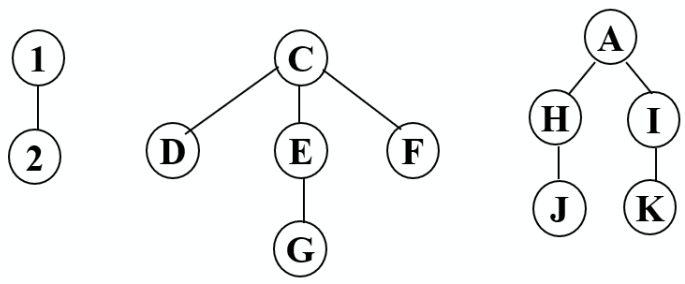


练习：

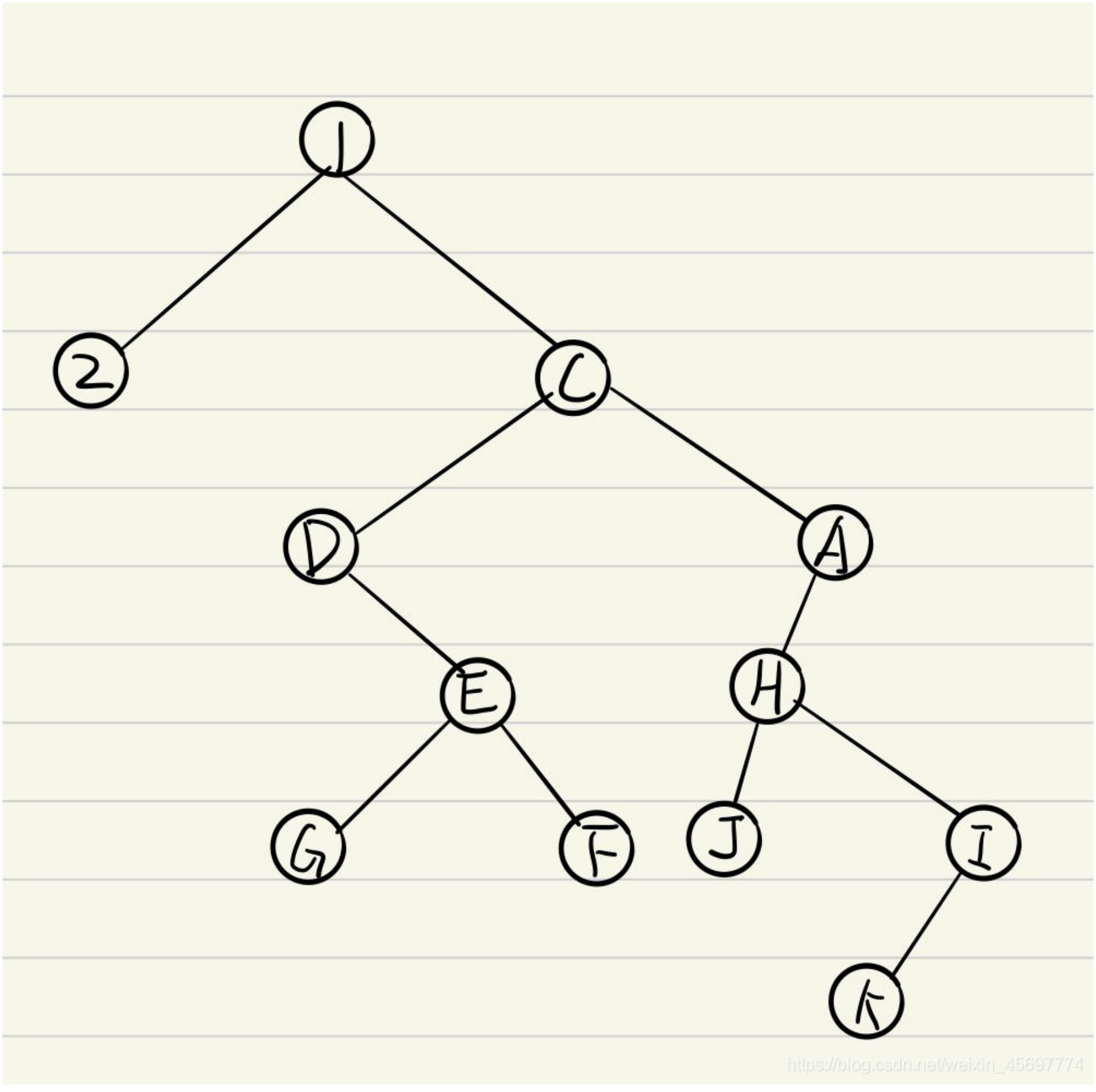
1、将二叉树转换为森林。



2、将森林转换为二叉树。



2.答案：



六、Huffman树

哈夫曼树应用实例 - - 哈夫曼编码

在远程通讯中，要将待传字符转换成二进制的字符串，怎样编码才能使它们组成的报文在网络中传得最快?

A	00
B	01
C	10
D	11

ABACCDA

A	0
B	00
C	1
D	01

0001100101011 00等长编码　　不等长编码000011010

出现次数较多的字符采用尽可能短的编码

哈夫曼树应用实例 - - 哈夫曼编码

ABACCDA

A	0
B	00
C	1
D	01

0000 → AAAA　ABA　BB

重码

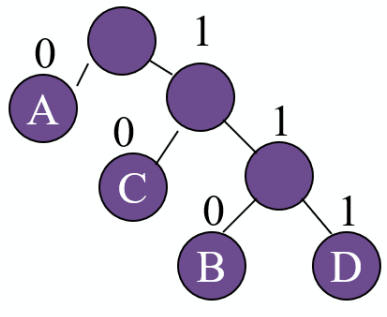
0000 11010

关键：要设计长度不等的编码，则必须使任一字符的编码都不是另一个字符的编码的前缀 - 前缀编码

https://blog.csdn.net/weixin_45697774

哈夫曼编码的译码过程

分解接收字符串：遇“0”向左，遇“1”向右；一旦到达叶子结点，则译出一个字符，反复由根出发，直到译码完成。



0110010101110

↓

ABACCDA

特点：每一码都不是另一码的前缀，绝不会错译! 称为前缀码

https://blog.csdn.net/weixin_45697774

哈夫曼树的构造

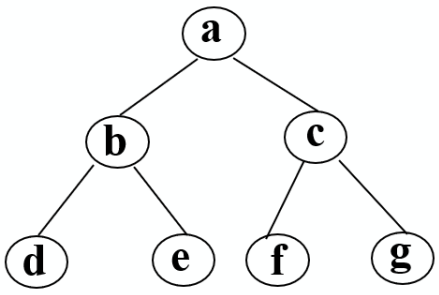
路　径 01

由一结点到另一结点间的分支所构成

路径长度 02

路径上的分支数目

a→e的路径长度 = 2



带权路径长度 03

结点到根的路径长度与结点上权的乘积

树的带权路径长度 04

树中所有叶子结点的带权路径长度之和

$$WPL = \sum_{k=1}^{n} w_k l_k$$

哈 夫 曼 树 05

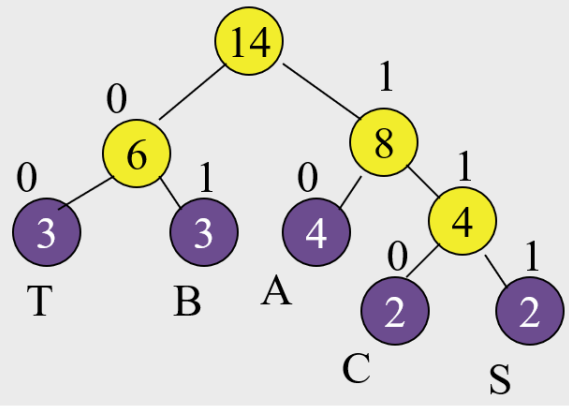
带权路径长度最小的树

哈夫曼编码的构造

基本思想：概率大的字符用短码，小的用长码，构造哈夫曼树

例：某系统在通讯时，只出现C，A，S，T，B五种字符，其出现频率依次为2，4，2，3，3，试设计Huffman编码。

T	00
B	01
A	10
C	110
S	111



例5.2

哈夫曼树的构造过程

- ✓根据给定的n个权值$\{w_1,w_2,\ldots\ldots w_n\}$，构造n棵只有根结点的二叉树。
- ✓在森林中选取两棵根结点权值最小的树作左右子树，构造一棵新的二叉树，置新二叉树根结点权值为其左右子树根结点权值之和。
- ✓在森林中删除这两棵树，同时将新得到的二叉树加入森林中。
- ✓重复上述两步，直到只含一棵树为止，这棵树即哈夫曼树。

huffman树不唯一！

哈夫曼树构造算法的实现（算法5.10）

一棵有n个叶子结点的Huffman树有 **2n-1** 个结点

- ✓采用顺序存储结构——一维结构数组
- ✓结点类型定义

```
typedef  struct
{  int weght;
   int parent,lch,rch;
}*HuffmanTree;
```

哈夫曼树构造算法的实现

01 OPTION 初始化HT[1..2n-1]：lch=rch=parent=0

02 OPTION 初始化HT[1..2n-1]：lch=rch=parent=0

03 OPTION 进行以下n-1次合并，依次产生HT[i]，i=n+1..2n-1：

- 在HT[1..i-1]中选两个未被选过的weight最小的两个结点HT[s1]和HT[s2] (从parent = 0 的结点中选)
- 修改HT[s1]和HT[s2]的parent值： parent=i
- 置HT[i]：weight=HT[s1].weight + HT[s2].weight ,lch=s1，rch=s2

七、相关作业习题

1.选择判断

1.完全二叉树一定存在度为1的结点。

答案：×
满二叉树也是完全二叉树，其叶子结点的度为0。

2.哈夫曼树的结点个数不能是偶数

答案：√

假设哈夫曼树是二叉的话，则度为0的结点个数为N，度为2的结点个数为N-1，则结点总数为2N-1。

二叉树可以为空

3.树形结构中元素之间存在一个对多个的关系。

答案：√
树形结构元素之间存在一对多的关系，线性结构元素之间存在一对一的关系，图形结构元素之间存在多对多的关系。

4.用一维数组存储二叉树时,总是以前序遍历顺序存储结点

答案：×
总是以层次遍历的顺序存储，并且按照完全二叉树的方式建立，所以有很多空节点，会浪费存储空间，完全二叉树可以非常方便地找到孩子兄弟和双亲

5.若一个二叉树的叶子结点是某子树的**中序**遍历序列中的**最后**一个结点，则它必是该子树的**前序**序列中的**最后**一个结点。

若一个二叉树的树叶是某子树的**中序**遍历序列中的**第一**个结点，则它必是该子树的**后序**遍历序列中的**第一**个结点。

6.以下说法错误的是（）
A哈夫曼树是带权路径长度最短的树，路径上权值较大的结点离根较近。
B若一个二叉树的树叶是某子树的中序遍历序列中的第一个结点，则它必是该子树的后序遍历序列中的第一个结点。
C已知二叉树的前序遍历和后序遍历序列并不能惟一地确定这棵树，因为不知道树的根结点是哪一个。
D在前序遍历二叉树的序列中，任何结点的子树的所有结点都是直接跟在该结点的之后。
答案：C

7.给定一棵树，可以找到唯一的一棵二叉树与之对应。

答案：√

8.二叉树是一种特殊的树。

答案：×
9.二叉树的遍历只是为了在应用中找到一种线性次序。

答案：√

++

10.完全二叉树中,若一个结点没有左孩子,则它必是树叶()

答案：√

> 若深度为1，则该结点即为根结点，又为叶子结点
>
> 11.已知一棵二叉树的前序遍历结果为ABCDEF，中序遍历结果为CBAEDF，则后序遍历的结果为?
> A.CBEFDA
> B.FEDCBA
> C.CBEDFA
> D.不定

答案：A
12.先序遍历ABCDEF可知A为根节点，然后中序遍历CBAEDF可知CB为A的左子树，EDF为A的右子树，对于左子树，先序遍历为BC，可知B为根节点，中序遍历为CB，C为其左子树，右子树分析一样。
8.若一棵二叉树具有10个度为2的结点，5个度为1的结点，则度为0的结点个数是( )。

A．9
B．11
C．15
D．不确定

正确答案
B
答案解析
对任何一棵二叉树，如果终端结点数为n0，度为2的结点数为n2，则一定有n0=n2+1。所以n0=10+1=11，而与n1无关。

13.一棵二叉树的高度为h，所有结点的度为0，或为2，则此树最少有( )个结点
2h-1
2h+1
h+1

正确答案：B

根的高度是0

2.编程题

11.建立与遍历二叉树 (25分)

以字符串的形式定义一棵二叉树的先序序列，若字符是‘#’，表示该二叉树是空树，否则该字符是相应结点的数据元素。读入相应先序序列，建立二叉链式存储结构的二叉树，然后中序遍历该二叉树并输出结点数据。
输入格式:
字符串形式的先序序列（即结点的数据类型为单个字符）
输出格式:
中序遍历结果
输入样例:
在这里给出一组输入。例如：
ABC##DE#G##F###

输出样例:
在这里给出相应的输出。例如：
CBEGDFA

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

//#define ls (p<<1)
//#define rs (p<<1|1)

#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 = 1e5+7;
const int M = 2007;

char ch;
```

```
26
27 struct node{
28     char val;
29     node *ls;
30     node *rs;
31 };
32
33 void build(node *&t){//注意要用引用变量
34     cin>>ch;
35     if(ch == '#')t = NULL;
36     else {
37         t = new node;
38         t->val = ch;
39         build(t->ls);
40         build(t->rs);
41     }
42 }
43
44 void print(node *t){
45     if(t){
46         print(t->ls);
47         cout<<(t->val);
48         print(t->rs);
49     }
50 }
51
52 void solve(){
53     node *t = NULL;
54     build(t);
55     print(t);
56 }
57 int main()
58 {
59     solve();
60     return 0;
61 }
62
```