高级数据结构-树与二叉树

信智学院　陈琰宏

主要内容

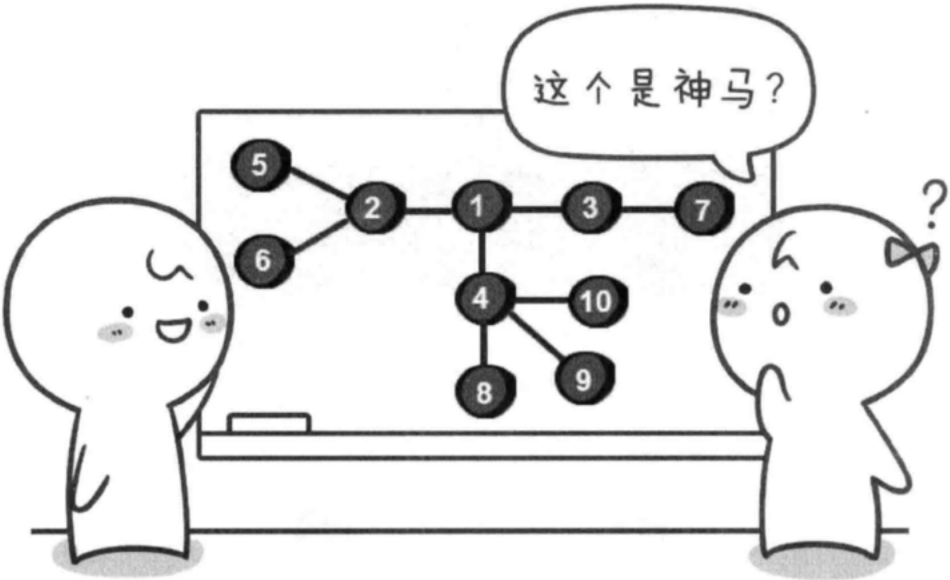
01 树的定义与理解

02 二叉树的定义与性质

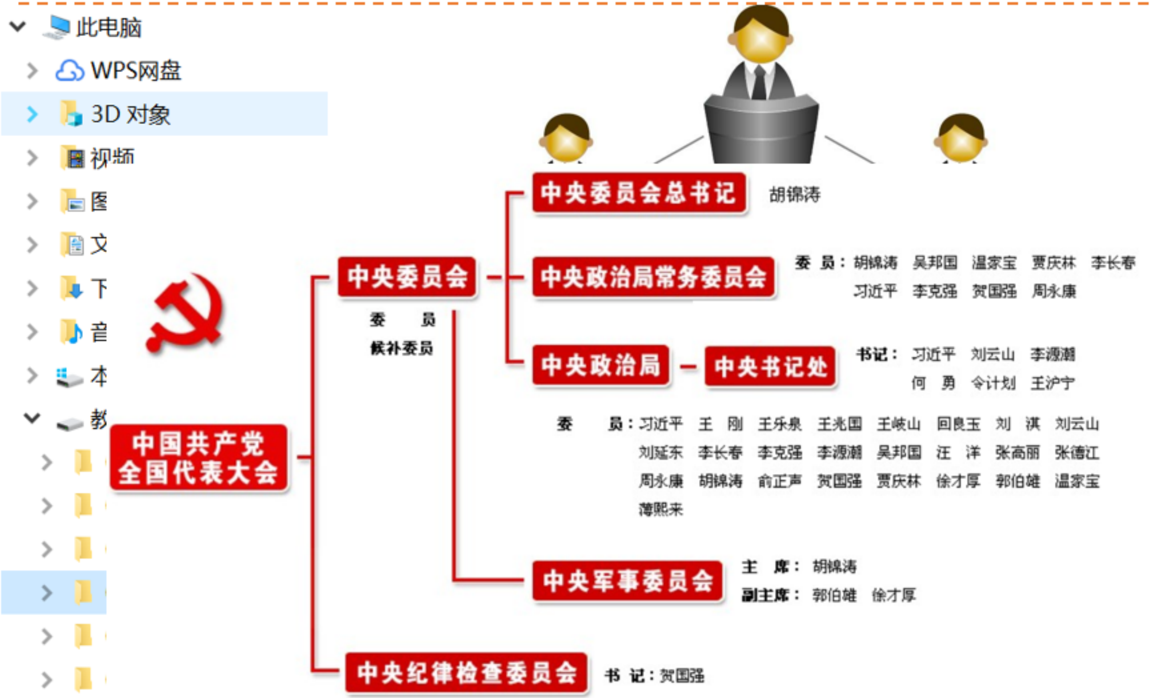
03 二叉树的操作

04 二叉树的应用

5.1 树的定义



5.1 树的定义


此电脑
WPS网盘
3D 对象
视频
中国共产党全国代表大会
中央委员会
委 员
候补委员
中央委员会总书记
胡锦涛
中央政治局常务委员会
委 员：胡锦涛 吴邦国 温家宝 贾庆林 李长春 习近平 李克强 贺国强 周永康
中央政治局
中央书记处
书记：习近平 刘云山 李源潮 何 勇 令计划 王沪宁
委 员：习近平 王 刚 王乐泉 王兆国 王岐山 回良玉 刘 淇 刘云山 刘延东 李长春 李克强 李源潮 吴邦国 汪 洋 张高丽 张德江 周永康 胡锦涛 俞正声 贺国强 贾庆林 徐才厚 郭伯雄 温家宝 薄熙来
中央军事委员会
主 席：胡锦涛
副主席：郭伯雄 徐才厚
中央纪律检查委员会
书 记：贺国强

5.1 树的定义

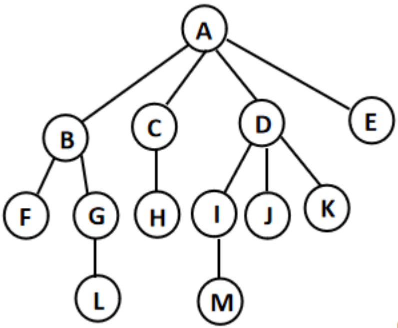
- 树（Tree）是n（n≥0）个结点构成的有限集合。当n=0时，称为空树；对于任一棵非空树（n> 0），它具备以下性质：

1. 树中有一个称为“根（Root）”的特殊结点，用 r 表示；
2. 其余结点可分为m(m>0)个互不相交的有限集T_1，T_2，…，T_m，其中每个集合本身又是一棵树，这些树称为原来树的“子树（SubTree）”。每个子树的根结点都与 r 有一条相连接的边，r 是这些子树根结点的“父结点(Parent)”。

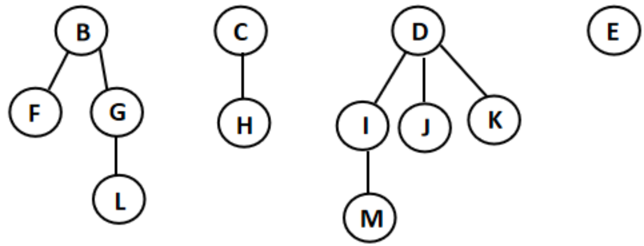
  - 子树是不相交的；
  - 除了根结点外，每个结点有且仅有一个父结点；
  - 一棵N个结点的树有N-1条边。

5.1 树的定义

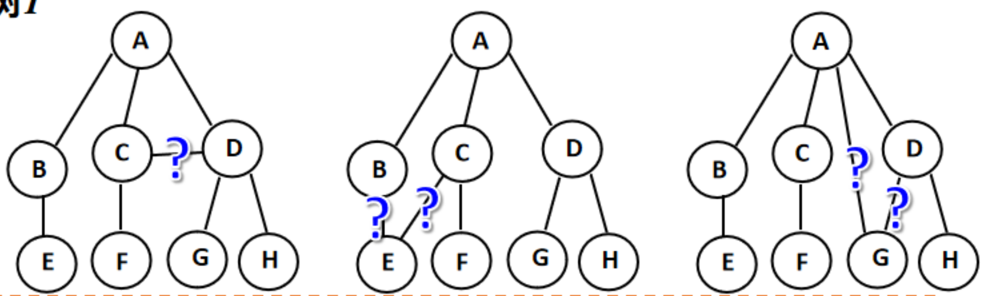
❖ 树与非树



(a) 树T

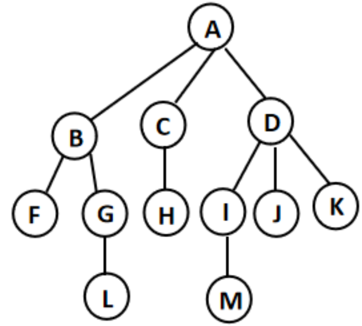


(b) 子树T_{A1} (c) 子树T_{A2} (d) 子树T_{A3} (e)子树T_{A4}



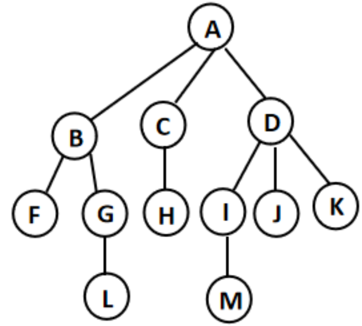
5.1.1 树的基本术语

1. **结点的度**（Degree）：一个结点的度是其**子树的个数**。
2. **树的度**：树的所有结点中最大的度数。
3. **叶结点**（Leaf）：是**度为0**的结点；叶结点也可称为端结点。
4. **父结点**（Parent）：有子树的结点是其子树的根结点的父结点。
5. **子结点**（Child）：若A结点是B结点的父结点，则称B结点是A结点的子结点；子结点也称**孩子结点**。
6. **兄弟结点**（Sibling）：具有同一父结点的各结点彼此是兄弟结点。



5.1.1 树的基本术语

7. **分支**：树中两个相邻结点的连边称为一个分支。
8. **路径和路径长度**：从结点n_1到n_k的路径被定义为一个结点序列$n_1, n_2, \ldots, n_k$，对于$1 \le i \le k$, n_i是n_{i+1}的父结点。一条路径的长度为这条路径所包含的边**（分支）的个数**。
9. **祖先结点**(Ancestor)：沿**树根到某一结点路径**上的所有结点都是这个结点的祖先结点。
10. **子孙结点**(Descendant)：某一结点的**子树中的所有结点**是这个结点的子孙。
11. **结点的层次**（Level）：规定**根结点在1层**，其它任一结点的层数是其父结点的层数加1。
12. **树的高度**（Height）：树中所有结点中的**最大层次**是这棵树的高度（也有把根定义成高度为1的）。



5.1.2 树的简单理解-二分查找

根据某个给定的关键字*K*，从集合*R*中找出关键字与*K*相同的记录，这个过程称为“查找”。

- 当线性表中数据元素是按大小排列存放时，可以改进顺序查找算法，以得到更高效率的新算法——二分法（折半查找)。

- 二分查找是每次在要查找的数据集合中取出中间元素关键字K_{mid}与要查找的关键字K进行比较，根据比较结果确定是否要进一步查找。当 $K_{mid}=K$，查找成功；否则，将在K_{mid}的左半部分（当$K_{mid}>K$）或者右半部分（当$K_{mid}<K$）继续下一步查找。以此类推，每步的查找范围都将是上一次的一半。

5.1.2 树的简单理解-二分查找

[例5.1] 假设有13个数据元素，它们的关键字为 51，202，16，321，45，98，100，501，226，39，368，5，444。若按关键字由小到大顺序存放这13个数，二分查找关键字为444的数据元素过程如下：

5	16	39	45	51	98	100	202	226	321	368	444	501
1	2	3	4	5	6	7	8	9	10	11	12	13

left mid right

1、left = 1, right = 13; mid = (1+13)/2 = 7: 100 < 444;

2、left = mid+1=8, right = 13; mid = (8+13)/2 = 10: 321 < 444;

3、left = mid+1=11, right = 13; mid = (11+13)/2 = 12: 444 = 444查找结束。

5.1.2 树的简单理解-二分查找

[例5.2] 仍然以上面13个数据元素构成的有序线性表为例，二分查找关键字为 43 的数据元素如下：

5	16	39	45	51	98	100	202	226	321	368	444	501
1	2	3	4	5	6	7	8	9	10	11	12	13
↑ left						↑ mid						↑ right

1、left = 1, right = 13; mid = (1+13)/2 = 7:   100 > 43;

2、 left = 1, right = mid-1= 6; mid = (1+6)/2 = 3:   39 < 43;

3、left = mid+1=4, right = 6; mid = (4+6)/2 = 5:   51 > 43;

4、left = 4, right = mid-1= 4; mid = (4+4)/2 = 4:   45 > 43;

5、left = 4, right = mid-1= 3; left > right ?   查找失败，结束。

二分查找算法

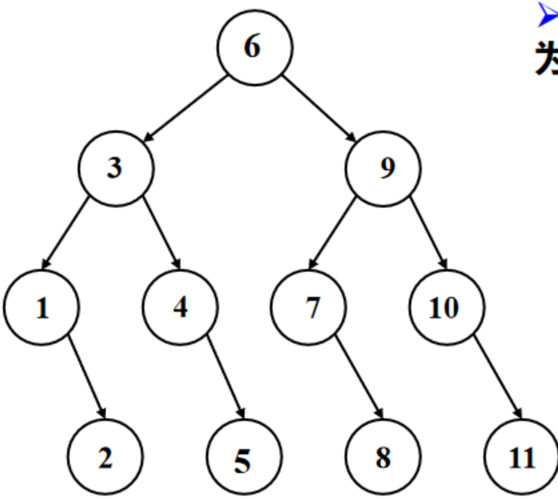
```
int  BinarySearch ( StaticTable * Tbl, ElementType K)
{  /*在表Tbl中查找关键字为K的数据元素*/
   int left, right, mid, NoFound=-1;

   left = 1;                /*初始左边界*/
   right = Tbl->Length;     /*初始右边界*/
   while ( left  <= right )
   {
      mid = (left+right)/2;  /*计算中间元素坐标*/
      if( K < Tbl->Element[mid])        right = mid-1;   /*调整右边界*/
      else if( K > Tbl->Element[mid])  left = mid+1;    /*调整左边界*/
      else  return  mid;  /*查找成功，返回数据元素的下标*/
   }
   return  NotFound;      /*查找不成功，返回-1*/
}
```

➢ **二分查找算法的时间复杂度为O(logN)**

5.1.2 树的简单理解-二分查找

- 查找的效率主要用“平均查找长度”（Average Search Length， ASL ）来衡量。



11个元素的判定树

- 判定树上每个结点需要的查找次数刚好为该结点所在的层数;
- 查找成功时查找次数不会超过判定树的深度
- ASL = (4*4+4*3+2*2+1)/11 = 3
- n个结点的判定树的深度为$[\log_2 n]+1$
- 折半查找的算法复杂度为$O(\log_2 n)$

5.2 二叉树的定义

【定义】一棵二叉树T是一个有穷的结点集合。这个集合可以为空，若不为空，则它是由根结点和称为其左子树T_L和右子树T_R的两个不相交的二叉树组成。可见左子树和右子树还是二叉树。

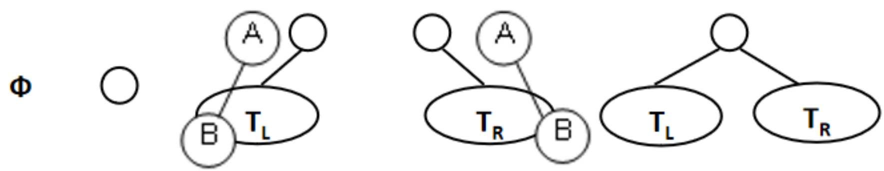
递归的定义形式

➢ 二叉树具体五种基本形态

(1) 空二叉树；
(2) 只有一个根结点的二叉树；
(3) 只有根结点和左子树T_L的二叉树；
(4) 只有根结点和右子树T_R的二叉树；
(5) 具有根结点、左子树T_L和右子树T_R的二叉树。

➢ 二叉树与树不同，除了每个结点至多有两棵子树外，子树有左右顺序之分。

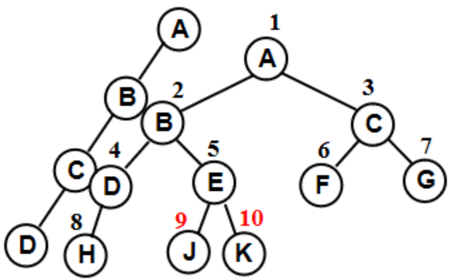
➢ 例如，下面两个树按一般树的定义它们是同一个树；而对于二叉树来讲，它们是不同的两个树。

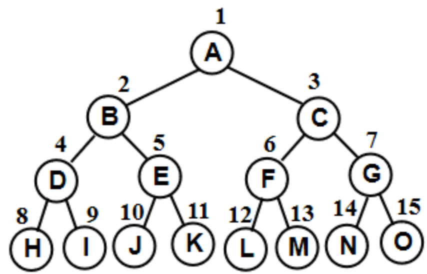


(a)　(b)　(c)　(d)　(e)

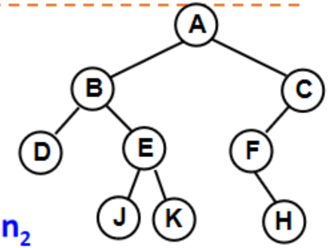
5.2 二叉树的定义-特殊二叉树

- 二叉树的深度小于结点数*N*，可以证明平均深度是 $O(\sqrt{N})$
- “斜二叉树(Skewed Binary Tree)”（也称为退化二叉树）；
- “完美二叉树(Perfect Binary Tree)”。（也称为满二叉树）。
- 一棵深度为k的有n个结点的二叉树，对树中的结点按从上至下、从左到右的顺序进行编号，如果编号为i（$1 \le i \le n$）的结点与满二叉树中编号为 i 的结点在二叉树中的位置相同，则这棵二叉树称为“完全二叉树(Complete Binary Tree)”。





5.2.1 二叉树的性质



- **一个二叉树第 i 层的最大结点数为：** 2^{i-1}，$i \geq 1$。
- **深度为k的二叉树有最大结点总数为：** $2^{k}-1$，$k \geq 1$。
- 对任何非空的二叉树 T，若n_0表示叶结点的个数、n_2是度为2的非叶结点个数，那么两者满足关系$n_0 = n_2 + 1$。
- n个结点的完全二叉树的深度为 $\lfloor \log_2 n \rfloor + 1$

证明：设 n_1 为度为1的结点数，n 是它的总结点数，那么

$$n = n_0 + n_1 + n_2 \quad ①$$

- $n_0 = 4$，$n_1 = 2$
- $n_2 = 3$
- $n_0 = n_2 + 1$

设 B 是全部分枝数. 则 $n \sim B?$　　$n = B + 1.$ ②

因为所有分枝都来自度为1或2的结点, 所以 $B \sim n_1 \,\&\, n_2$?

$$B = n_1 + 2\,n_2. \quad ③$$

$$\Rightarrow \qquad n_0 = n_2 + 1$$

5.2.2 二叉树的存储结构

1. 顺序存储结构

- 完全二叉树最适合这种存储结构。
- n个结点的完全二叉树的结点父子关系，简单地由序列号决定：

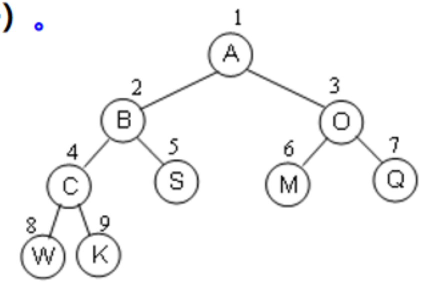
1、非根结点（序号 $i > 1$）的父结点的序号是 $\lfloor i / 2 \rfloor$;

2、结点（序号为 i ）的左孩子结点的序号是 $2i$，
（若 $2i <= n$，否则没有左孩子）；

3、结点（序号为 i ）的右孩子结点的序号是 $2i+1$，
（若 $2i+1 <= n$，否则没有右孩子）。

数据	A	B	O	C	S	M	Q	W	K
编号	1	2	3	4	5	6	7	8	9

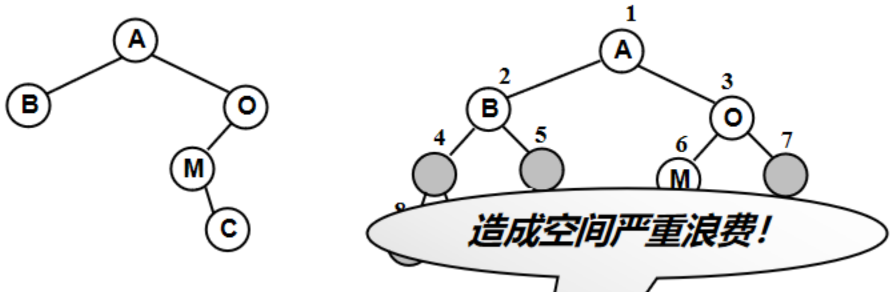
(a)相应的顺序存储结构



(b) 完全二叉树

5.2.2 二叉树的存储-顺序存储

➢**一般二叉树采用这种结构将造成空间浪费**

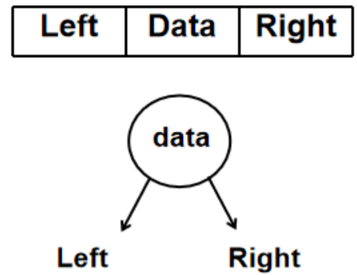


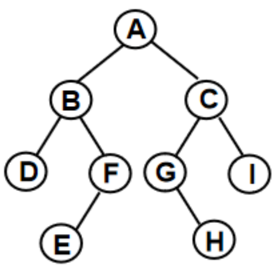
(a)一般二叉树　　(b) 对应(a)……全二叉树

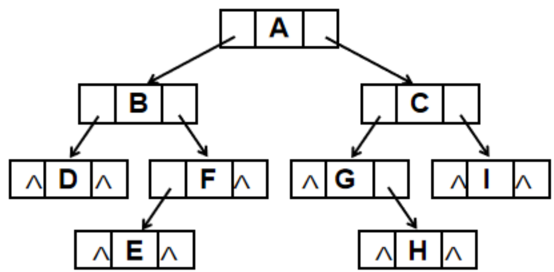
数据	A	B	O	∧	∧	M	∧	∧	∧	∧	∧	∧	C
编号	1	2	3	4	5	6	7	8	9	10	11	12	13

5.2.2 二叉树的存储-链表存储

```
typedef struct TreeNode *BinTree;
typedef BinTree Position;
struct TreeNode{
        ElementType Data;
        BinTree Left;
        BinTree Right;
}
```







5.3 二叉树的操作

1、Boolean IsEmpty(BinTree BT)

若BT为空返回TRUE; 否则返回FALSE;

2、BinTree CreatBinTree( )

创建一个二叉树。

3、void Traversal(BinTree BT)

二叉树的遍历,

3.1、void InOrderTraversal(BinTree BT)

根结点的访问次序在左、右子树之间;

3.2、void PreOrderTraversal(BinTree BT)

根结点的访问次序在左、右子树之前;

3.3、 void PostOrderTraversal(BinTree BT)

根结点的访问次序在左、右子树之后。

3.4、 void LevelOrderTraversal(BinTree BT)

按层从小到大、从左到右的次序遍历

5.3.1 二叉树的中序遍历

❖ 树的遍历是指访问树的每个结点，且每个结点仅被访问一次。二叉树的遍历可按二叉树的构成以及访问结点的顺序分为四种方式，即先序遍历、中序遍历、后序遍历和层次遍历。

(1)　中序遍历

其遍历过程为：

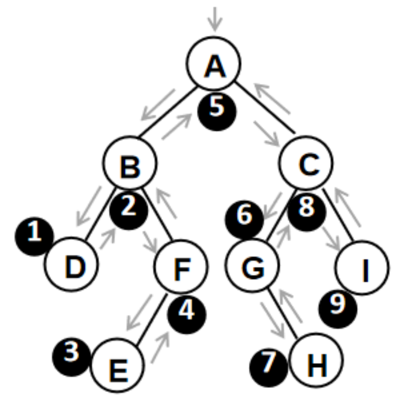
① 中序遍历其左子树；

② 访问根结点；

③ 中序遍历其右子树。

(D B E F) A (G H C I)

中序遍历=>　D B E F A G H C I




```
void InOrderTraversal( BinTree BT )
{
    if( BT ) {
        InOrderTraversal( BT->Left );
        printf("%d", BT->Data);
        InOrderTraversal( BT->Right );
    }
}
```

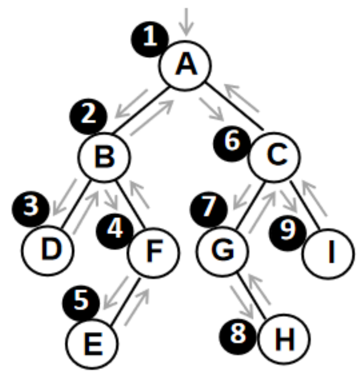
5.3.2 二叉树的先序遍历

其遍历过程为:
① 访问根结点;
② 先序遍历其左子树;
③ 先序遍历其右子树。

A (B D F E) (C G H I)

先序遍历=> A B D F E C G H I

```
void PreOrderTraversal( BinTree BT )
{
    if( BT ) {
        printf("%d", BT->Data);
        PreOrderTraversal( BT->Left );
        PreOrderTraversal( BT->Right );
    }
}
```



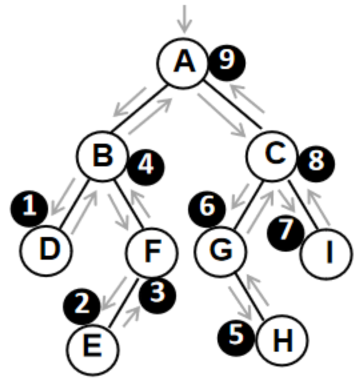
5.3.3 二叉树的后序遍历

其遍历过程为：
① 后序遍历其左子树；
② 后序遍历其右子树；
③ 访问根结点。

(D E F B) (H G I C) A

后序遍历=> D E F B H G I C A

```
void PostOrderTraversal( BinTree BT )
{
    if( BT ) {
        PostOrderTraversal( BT->Left );
        PostOrderTraversal( BT->Right);
        printf("%d", BT->Data);
    }
}
```

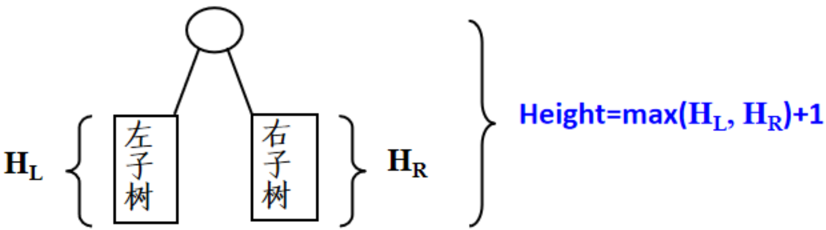


【例3】输出二叉树中的叶子结点。

- 在二叉树的遍历算法中增加检测结点的“左右子树是否都为空”。

```
void PreOrderPrintLeaves( BinTree BT )
{
    if( BT ) {
        if ( !BT-Left && !BT->Right )
            printf(“%d”, BT->Data );
        PreOrderPrintLeaves ( BT->Left );
        PreOrderPrintLeaves ( BT->Right );
    }
}
```

【例4】求二叉树的高度。




```
void PostOrderGetHeight( BinTree BT )
{   int HL, HR, MaxH;
    if( BT ) {
        HL = PostOrderGetHeight(BT->Left);  /*求左子树的深度*/
        HR = PostOrderGetHeight(BT->Right); /*求右子树的深度*/
        MaxH = HL > HR? HL : HR; /*取左右子树较大的深度*/
        return ( MaxH + 1 ); /*返回树的深度*/
    }
    else  return 0; /* 空树深度为0 */
}
```

5.4.1 [3298]二叉树的建立

括号表示法表示的二叉树字符串A(B(D,F(E,)),C(G(,H),I))，建立这棵树，并用先序、中序和后序遍历输出结果。

样例输入

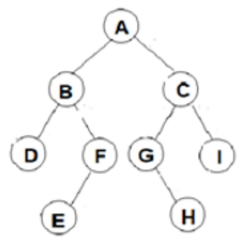
A(B(D,F(E,)),C(G(,H),I))

样例输出

ABDFECGHI

DBEFAGHCI

DEFBHGICA



A(B(D,F(E,)),C(G(,H),I))

5.4.1 [3298] 分析

- 若ch='(': 表示前面刚创建的p结点存在着孩子结点，需将其进栈，以便建立它和其孩子结点的关系（如果一个结点刚创建完毕，其后一个字符不是'('，表示该结点是叶子结点，不需要进栈）。然后开始处理该结点的左孩子，因此置k=1，表示其后创建的结点将作为这个结点（栈顶结点）的左孩子结点；
- 若ch=')': 表示以栈顶结点为根结点的子树创建完毕，将其退栈；
- 若ch=',': 表示开始处理栈顶结点的右孩子结点，置k=2；
- 其他情况：只能是单个字符，表示要创建一个新结点p，根据k值建立p结点与栈顶结点之间的联系，当k=1时，表示p结点作为栈顶结点的左孩子结点，当k=2时，表示p结点作为栈顶结点的右孩子结点。

5.4.1 [3298] 方法1-指针栈模拟

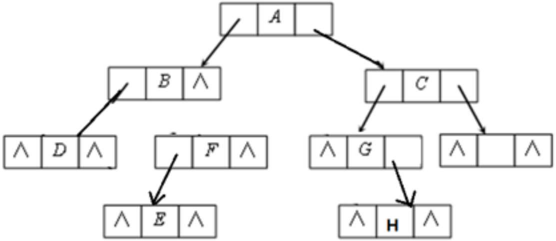
```
#include<iostream>
#include<cstring>
using namespace std;
struct BTNode
{
    char data;
    BTNode *lchild;
    BTNode *rchild;
};//代表一个结点
BTNode *CreateBTNode(char *str)//生成树
{

void DispBTNode1(BTNode *t)
{
    if(t!=NULL)
    {
        printf("%c",t->data);
        DispBTNode1(t->lchild);//遍历左孩子
        DispBTNode1(t->rchild);//遍历右孩子
    }
}
int main()
{
    cin>>str;
    BTNode *bt=CreateBTNode(str);
    DispBTNode1(bt);
    return 0;
}
```

5.4.1 [3298] 方法1-指针栈模拟

```
BTNode *CreateBTNode(char *str)//生成树
{
    BTNode *r=NULL;
    BTNode *st[100000];//建立一个顺序栈
    BTNode *p;
    int top=-1,k=0,j=0;//k代表左孩子还是右孩子，1为左，2为右，
                       //top表示栈顶，j表示第几个字符
    char ch;
    while(str[j]!='\0')//扫描str中每个字符
    {
    return r;
}
```

```
    while(str[j]!='\0')//扫描str中每个字符
    {
        ch=str[j];
        switch(ch)
        {
        case'(':top++;st[top]=p;k=1; break;//新建结点有孩子，将其进栈
        case')':top--;break;
        case',':k=2;break;//开始处理右孩子结点
        default:
            p=new BTNode();//新建一个结点
            p->lchild=p->rchild=NULL;
            p->data=ch;
            if(r==NULL)
                r=p;//若尚未建立根节点，点*p作为根节点
            else
            {
                switch(k)
                {
                    case 1://新建结点作为栈顶结点的左孩子
                        st[top]->lchild=p;break;
                    case 2://新建结点作为栈顶结点的右孩子
                        st[top]->rchild=p;break;
                    default:break;
                }
            }
            break;
        }
        j++;//继续扫描其他字符
    }
    return r;
```



5.4.1 [3298] 方法2-栈模拟

```
#include<bits/stdc++.h>
using namespace std;
struct node{
    int left,right,father,id;
    char ch;//数据域
}t[105];
void preInorder(int rt){
    cout<<t[rt].ch;
    if(t[rt].left)preInorder(t[rt].left);
    if(t[rt].right)preInorder(t[rt].right);
}
void midInorder(int rt){
void postInorder(int rt){
stack<int> st;
int main(){// A(B(D,F(E,)),C(G(,H),I))
    string str;
    cin>>str;
    int k,id=0,len=str.length();
    for(int i=0;i<len;i++){
        if(str[i]=='(')st.push(id),k=1;
        else if(str[i]==')')st.pop();
        else if(str[i]==',')k=2;
        else{
    }
    preInorder(1);cout<<endl;
    midInorder(1);cout<<endl;
    postInorder(1);
    return 0;
}
```

5.4.1 [3298] 方法2-栈模拟

```
    for(int i=0;i<len;i++){
        if(str[i]=='(')st.push(id),k=1;
        else if(str[i]==')')st.pop();
        else if(str[i]==',')k=2;
        else{
            if(id==0){//表示根节点
                id++;t[id].id=id,t[id].ch=str[i];
                t[id].father=-1;
            }
            else{//非根节点
                if(k==1){//左孩子
                    id++;t[id].id=id,t[id].ch=str[i];
                    t[id].father=st.top();
                    t[st.top()].left=id;
                }
                else if(k==2){
                    id++;t[id].id=id,t[id].ch=str[i];
                    t[id].father=st.top();
                    t[st.top()].right=id;
                }
            }
        }
    }
```

5.4.1 [3298] 方法3-递归

```
#include<bits/stdc++.h>
using namespace std;
char s[10000];
void dfs(int l,int r)
{
    int x,start;
    for(int i=l;i<=r;i++)
    {
        if(s[i]=='(')
        {
            x=1,start=i;
            while(x!=0)
            {//x==0表示找到了跟（ 匹配的 ） 的位置
                i++;
                if(s[i]=='(') x++;
                else if(s[i]==')') x--;
            }
            dfs(start+1,i-1);
        }
        if(s[i]!=','&&s[i]!=')')
            cout<<s[i];
    }
}
int main()
{
    scanf("%s",s);//A(B(D,F(E,)),C(G(,H),I))
    dfs(0,strlen(s)-1);
    return 0;
}
```

5.4.2 [1530] 完全二叉树的高度

已知完全二叉树的结点数，求其高度。
输入文件中包含多个测试数据。每个测试数据占1行，为一个整数n(1≤n≤100)，表示完全二叉树的结点数。输入文件中最后一行为0，表示测试数据结束。
对输入文件中的每个测试数据，输出完全二叉树的高度。
样例输入
10
20
0
样例输出
4

5.4.2 [1530] 完全二叉树的高度

```cpp
#include<iostream>
using namespace std;
int main(){
	int n,h;
	while(cin>>n&&n!=0){
		int h=0;
		while(n!=0){
			n/=2;
			h++;
		}
		cout<<h<<endl;
	}
	return 0;
}
```

5.4.3 [1909] 二叉树1

给定完全二叉树的节点树n，要求输出完全二叉树的高度k、叶子节点个数n1和非叶节点个数n2。

输入

输入文件中包含多个测试数据。每个测试数据占一行，为一个自然数n，n≤1000。输入文件最后一行为0，表示输入结束。

输出

对输入文件中的每个测试数据，计算k、n1和n2并输出。

样例输入

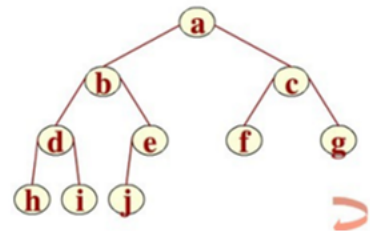
12

0

样例输出

4 6 6

5.4.3 [1909] 二叉树1



分析:

1. 观察完全二叉树来说，我们可以发现度为1的点，即n1=0或者n1=1。
2. 根据二叉树性质，n0=n2+1

当 n1=0时　n0+n2=n
n0=n2+1，即 n2=(n-1)/2　n0=(n+1)/2
当 n1=1时　n0+n2=n-1
n0=n2+1，即 n2=(n-2)/2　n0=n/2

5.4.3 [1909] 二叉树1

```
include <iostream>
#include <cmath>
using namespace std;
int main(){
int n,n2,n0;
while(cin>>n&&n!=0){
  n0=(n+1)/2;
  n2=n-n0;
  n=log(n)/log(2)+1;
cout<<n<<" "<<n0<<" "<<n2<<endl;
}
while(cin>>n&&n==0) break;
return 0;
}
```

5.4.4 [2886] 找树根和孩子

给定一棵树，输出树的根root，孩子最多的结点max以及他的孩子。
输入
第一行：n（结点个数≤100），m（边数≤200）。
以下m行：每行两个结点x和y，表示y是x的孩子(x,y≤1000)。
输出
第一行：树根：root；
第二行：孩子最多的结点max；
第三行：max的孩子。

样例输入
8 7
4 1
4 2
1 3
1 5
2 6
2 7
2 8
样例输出
4
2
6 7 8

5.4.4 [2886] 找树根和孩子

```cpp
#include<iostream>
using namespace std;
int main(){
    int parent[101]={0};
    int m,n,x,y,maxnode,sum=0,root;
    int i,j,k;
    cin>>n>>m;
    for(i=1;i<=m;i++){
        cin>>x>>y;
        parent[y]=x;//记录父亲节点

    }
    for(i=1;i<=n;i++)
        if(parent[i]==0){

    int max=0;
    for(i=1;i<=n;i++){//寻找最大
    cout<<root<<endl;
    cout<<maxnode<<endl;

    for(i=1;i<=n;i++){//寻找孩子
    return 0;
}
```

```cpp
    for(i=1;i<=n;i++)
        if(parent[i]==0){
            root=i;//寻找父亲节点
            break;
        }
    int max=0;
    for(i=1;i<=n;i++){//寻找最大
        sum=0;
        for(j=1;j<=n;j++)
            if(parent[j]==i)sum++;
        if(sum>max){
            max=sum;
            maxnode=i;
        }

    }
    cout<<root<<endl;
    cout<<maxnode<<endl;

    for(i=1;i<=n;i++){//寻找孩子
        if(parent[i]==maxnode)
            cout<<i<<" ";
    }
```

5.4.5 [7278]: 二叉树求高度

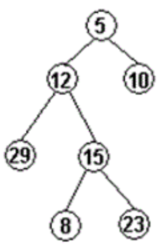
已知一棵二叉树用邻接表结构存储，求这棵树的高度。例：如图二叉树的数据文件的数据格式如下:

输入

第一行n为二叉树的结点个树，n≤100；以下第一列数据是各结点的值，第二列数据是左儿子结点编号，第三列数据是右儿子结点编号。

输出　该树的高度

7
15
5 2 3
12 4 5
10 0 0
29 0 0
15 6 7
8 0 0
23 0 0



样例输入

7 5 2 3
12 4 5
10 0 0
29 0 0
15 6 7
8 0 0
23 0 0

样例输出

4

5.4.5 [7278]: 方法1

```
#include <iostream>
using namespace std;
#define nochild 0   /* 如果没有对应孩子，设置为此 */
struct tree {
    int self;
    int lchild; // 左孩子
    int rchild; // 右孩子
} example[1001];
int getDepth(int root) {
    if (root==nochild) return 0;
    return max(getDepth(example[root].lchild),getDepth(example[root].rchild))+1;
}

int main() {
    int n,tmp;
    scanf("%d",&n);
    for (int i = 1; i <= n; i++) {
        scanf("%d%d%d",&example[i].self,&example[i].lchild,&example[i].rchild);
    }
}
```

5.4.5 [7278]: 方法2

```cpp
#include<bits/stdc++.h>
using namespace std;
struct node{
    int dat;
    int left;
    int right;
}a[105];
int n,x,ans,f;
void dfs(int r,int dep){
    ans=max(dep,ans);
    if(a[r].left) dfs(a[r].left,dep+1);
    if(a[r].right) dfs(a[r].right,dep+1);
}
int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;++i){
        scanf("%d%d%d",&a[i].dat,&a[i].left,&a[i].right);
    }

    dfs(1,1);
    printf("%d\n",ans);
    return 0;
}
```

今天的课程结束啦……

下课了…
同学们再见！