Chapter5 树

BinTreeNode：二叉树节点　BinaryTree：二叉树　ThreadTree：线索化二叉树

Heap：堆

5.1 树和森林的概念

5.1.1 有根树

一棵有根树 T，简称为树，它是n (n≥0) 个结点的有限集合。当n = 0时，T 称为空树；否则，T 是非空树。记作：

$$T=\begin{cases}\Phi, & n=0\\ \{r,T_1,T_2,...,T_m\}, & n>0\end{cases}$$

r 是一个特定的称为**根(root)**的结点，它**只有直接后继**，但**没有直接前驱**；

根以外的其他结点划分为 m (m >= 0) 个互不相交的有限集合T_1, T_2, ..., T_m，每个集合又是一棵树，并且称之为**根的子树。**

每棵子树的根结点**有且仅有一个**直接前驱，但可以**有0个或多个直接后继。**

5.1.2 树的基本术语

子女、双亲、兄弟、祖先、子孙

度：**结点的子女个数即为该结点的度**；**树中各个结点的度的最大值称为树的度**

分支结点：**度不为0的结点即为分支结点**，亦称为非终端结点。

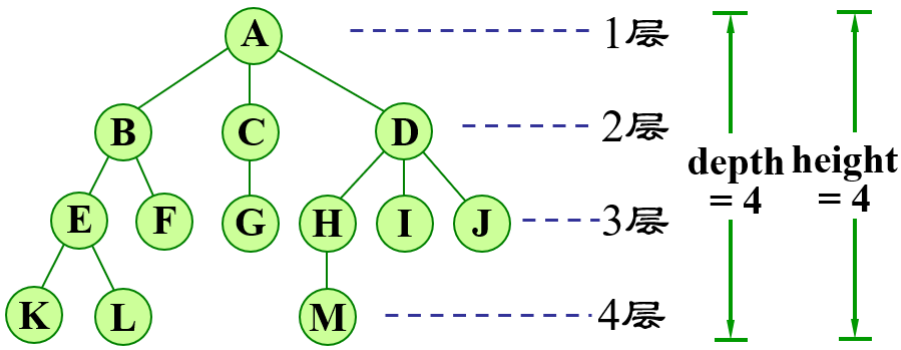
叶结点：**度为0的结点即为叶结点**，亦称为终端结点。

结点的层次：规定**根结点在第一层**，其**子女结点的层次等于它的层次加一。**以下类推。

深度：结点的深度即为结点的层次；离根最远结点的层次即为树的深度。 **（从根定义）**

高度：规定叶结点的高度为1，其双亲结点的高度等于它的高度加一。 **（从叶定义）**

树的高度：等于根结点的高度，即根结点所有子女高度的最大值加一。



有序树：树中结点的各棵子树 T_0, T_1, ...是有次序的，即为有序树。

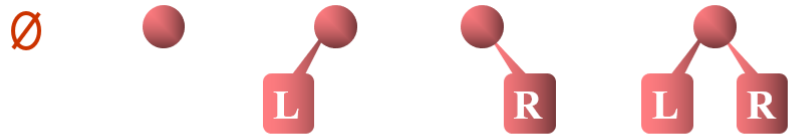
无序树：树中结点的各棵子树之间的次序是不重要的，可以互相交换位置。

森林：森林是m（m ≥ 0）棵树的集合

5.2 二叉树

5.2.1 二叉树的定义

一棵二叉树是结点的一个有限集合，该集合或者为空，或者是由一个根结点加上两棵分别称为左子树和右子树的、互不相交的二叉树组成。



二叉树的五种不同形态

度为2的树是否为二叉树？　　不是。一个节点仅有一棵子树时，不能区分其为左子树还是右子树

5.2.2 二叉树的性质

性质1 若二叉树结点的层次从 1 开始, 则在二叉树的第 i 层最多有 $2^i - 1$ 个结点。(i ≥ 1)

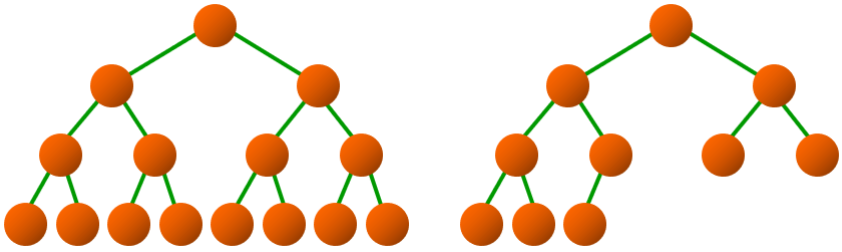
性质2 深度为 k 的二叉树最少有 k 个结点，最多有 $2^k - 1$个结点。(k ≥ 1)

性质3 对任何一棵二叉树，如果其叶结点有 n_0 个, 度为 2 的非叶结点有 n_2 个，则有$n_0 = n_2 + 1$

定义1 满二叉树 (Full Binary Tree)

定义2 完全二叉树 (Complete Binary Tree)

若设二叉树的深度为 k，则共有 k 层。除第 k 层外，其它各层 (1 ~ k-1) 的结点数都达到最大个数，第k层从右向左连续缺若干结点，这就是完全二叉树。



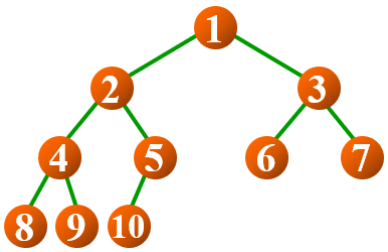
性质4 具有 n (n ≥ 0) 个结点的完全二叉树的深度为 $log_2(n+1)$ ，向上取整

性质5 如将一棵有n个结点的完全二叉树自顶向下，同一层自左向右连续给结点编号1, 2, ..., n，则有以下关系：

若i = 1, 则 i 无双亲；若i > 1, 则 i 的双亲为 i / 2，向下取整

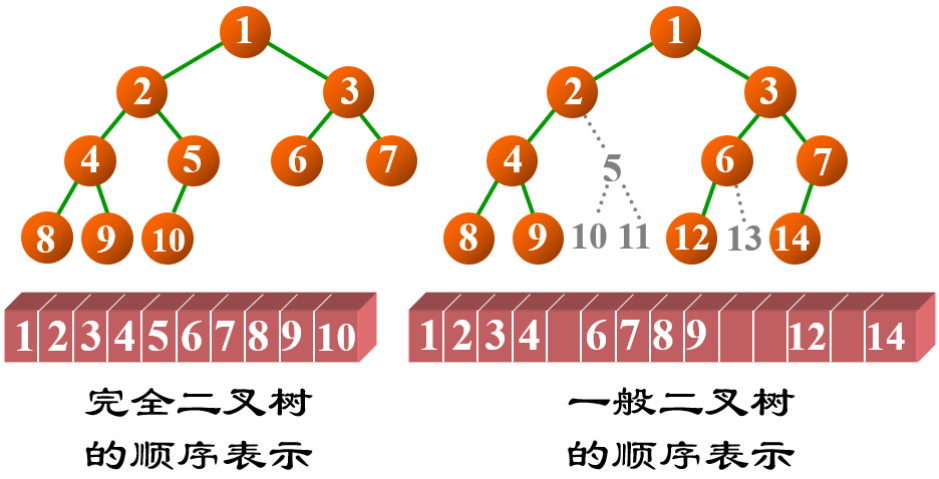
若2*i <= n, 则 i 的左子女为 2*i；若2*i+1 <= n, 则 i 的右子女为2*i+1

若 i 为奇数, 且i != 1, 则其左兄弟为i-1；若 i 为偶数, 且i != n,则其右兄弟为i+1



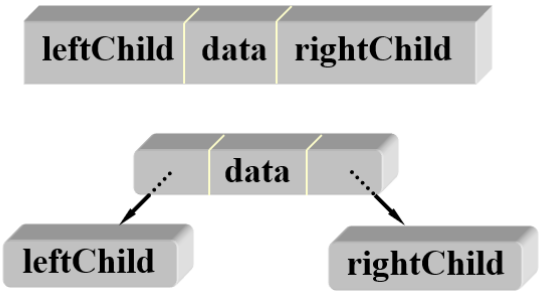
5.3 二叉树的存储表示

5.3.1 二叉树的顺序表示

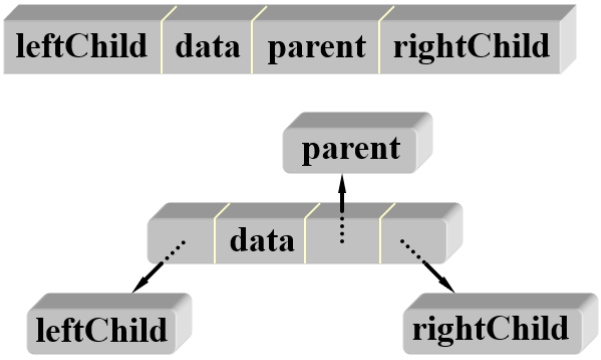


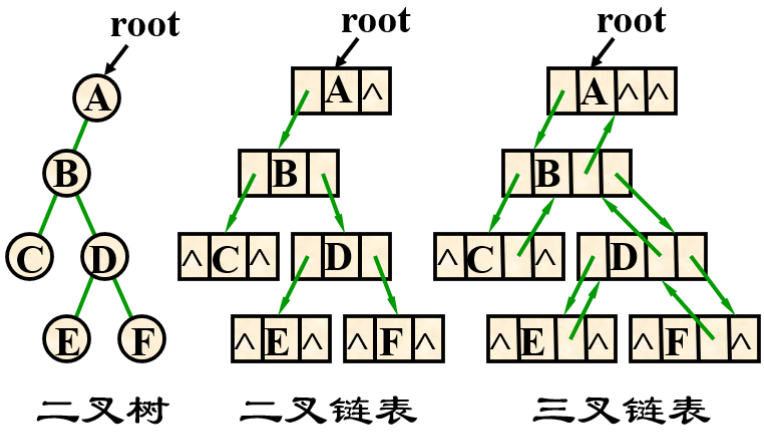
5.3.2 二叉树的链表表示（二叉链表）

二叉树结点定义：每个结点有3个数据成员，data域存储结点数据，leftChild和rightChild分别存放指向左子女和右子女的指针。

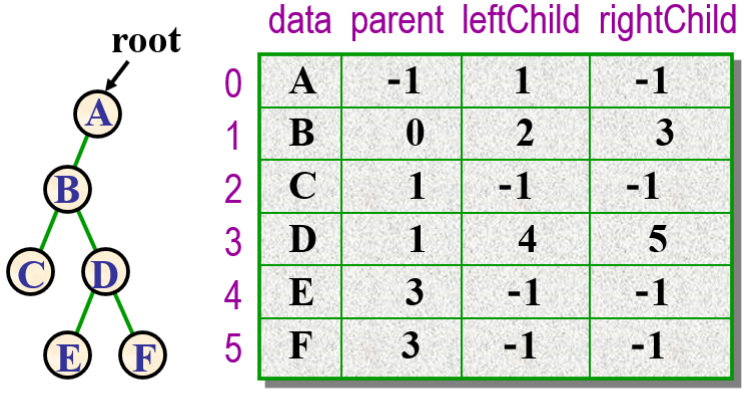


每个结点增加一个指向双亲的指针parent，使得查找双亲也很方便





二叉树链表表示的示例



	data	parent	leftChild	rightChild
0	A	-1	1	-1
1	B	0	2	3
2	C	1	-1	-1
3	D	1	4	5
4	E	3	-1	-1
5	F	3	-1	-1

二叉链表的静态结构

5.4 二叉树的遍历

5.4.1 二叉树遍历的递归实现

前序遍历，中序遍历，后序遍历（递归实现）

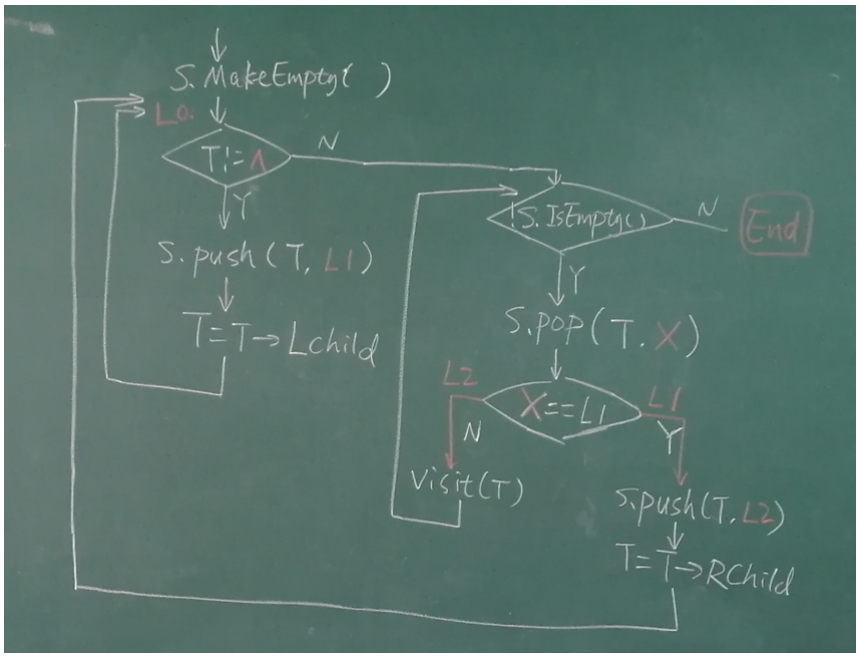
5.4.2 二叉树遍历的非递归实现

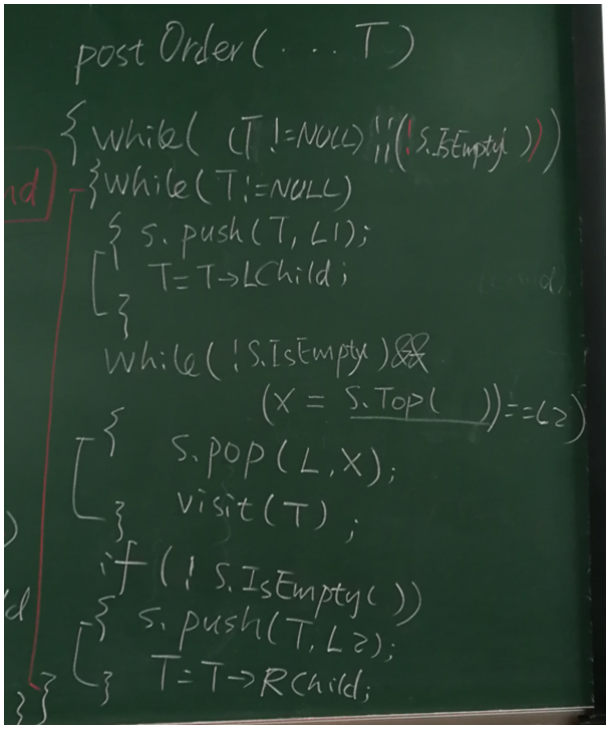
前序遍历，中序遍历，后序遍历（利用栈非递归实现）

具体算法看Chapter5课件，注意递归函数到非递归函数的转换：

二叉树递归的后序遍历算法

```
template <class T>
void BinaryTree<T>::PostOrder (BinTreeNode<T> *
   subTree, void (*visit) (BinTreeNode<T> *t ) {
L0:    if (subTree != NULL ) {
          PostOrder (subTree->leftChild, visit);
L1:       PostOrder (subTree->rightChild, visit);
L2:       visit (subTree);
       }
};
```





5.5 线索化二叉树

5.5.1 基本定义

又称为穿线树。

通过二叉树的遍历，可将二叉树中所有结点的数据排列在一个线性序列中，可以找到某数据在这种排列下它的前驱和后继。

希望不必每次都通过遍历找出这样的线性序列。只要事先做预处理，将某种遍历顺序下的前驱、后继关系记在树的存储结构中，以后就可以高效地找出某结点的前驱、后继。

5.5.2 实现方法

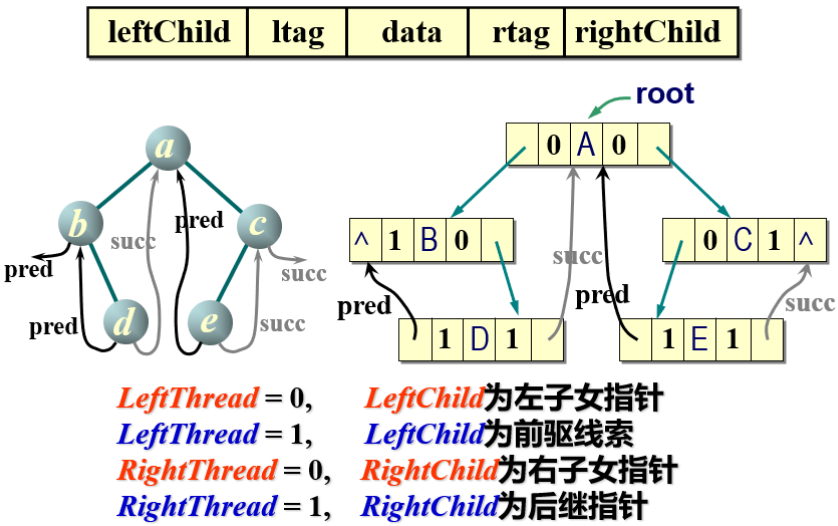
增加**左右线索标志**的二叉树

改造树结点，将 pred 指针和 succ 指针压缩到 leftChild 和 rightChild 的空闲指针中，并增设两个标志 ltag 和 rtag，指明指针是指示子女还是前驱 / 后继。后者称为线索。

leftChild	ltag	data	rtag	rightChild

ltag (或rtag) = 0，表示相应指针指示左子女（或右子女结点）；当ltag (或rtag) = 1, 表示相应指针为前驱（或后继）线索。

图例：



5.5.3 线索化二叉树的建立

```
template <class T>
void ThreadTree<T>::
createInThread (ThreadNode<T> *current, ThreadNode<T> *& pre)
{//通过中序遍历，对二叉树进行线索化
     if (current == NULL) return;
     createInThread (current->leftChild, pre);//递归，左子树线索化
     if (current->leftChild == NULL)
     {
          current->leftChild = pre;   current->ltag = 1;
     }//建立当前结点的前驱线索
     if (pre != NULL && pre->rightChild == NULL)
     {
         pre->rightChild = current;   pre->rtag = 1;
     }//建立前驱结点的后继线索
     pre = current;                                  //前驱跟上,当前指针向前遍历
     createInThread (current->rightChild, pre);//递归，右子树线索化
};
```

5.5.4 前序线索树与后序线索树

前序下找p前驱：

1. P无左孩子 Lchild
2. P有左孩子，P不是根

（1）P是其父的左孩子 Parent

（2）P是其父的右孩子，无左兄弟 Parent

（3）P是其父的右孩子，有左兄弟 其左兄弟最右下的叶子（右优先级高）

3. P是根节点 无前驱

前序下找p后继：

1. P无右孩子 Rchild
2. P有左孩子 Lchild
3. P无左孩子，有右孩子 Rchild

后序下找p前驱：

1. P无左孩子 Lchild

2. P有右孩子 Rchild
3. P无右孩子，有左孩子 Lchild

后序下找p后继：

1. P无右孩子 Rchild
2. P有右孩子，P不是根

（1）P是其父的右孩子 Parent

（2）P是其父的左孩子，无右兄弟 Parent

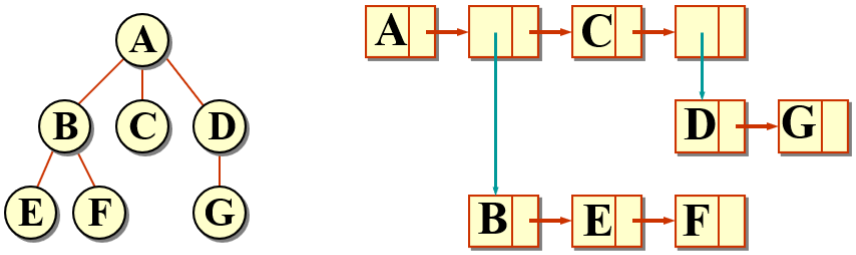
（3）P是其父的左孩子，有右兄弟 其右兄弟最左下的叶子（左优先级高）

3. P是根节点 无后继

5.6 树与森林

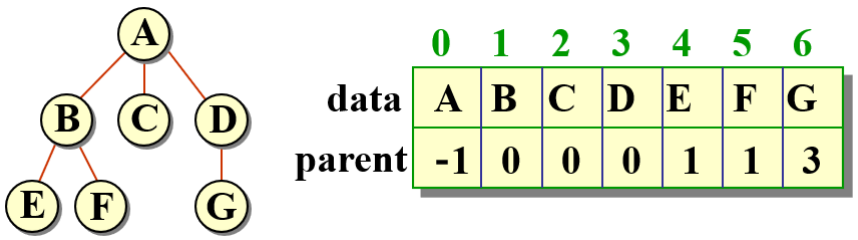
5.6.1 树的存储表示

广义表表示：



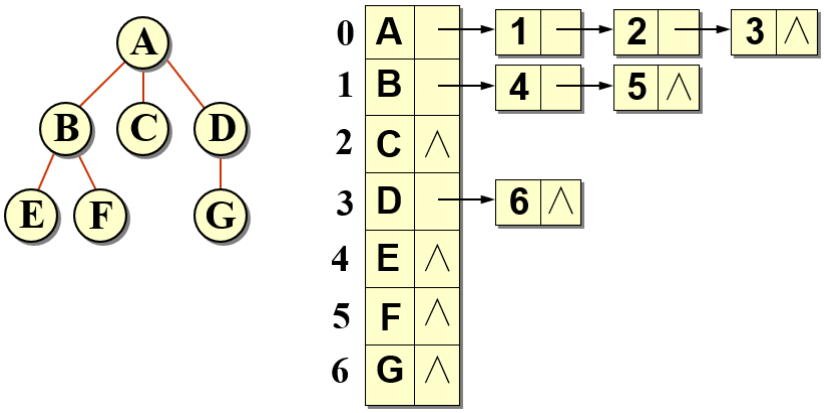
A(B(E, F), C, D(G)) 结点的utype域没有画出

双亲表示：



树中结点的存放顺序一般不做特殊要求，但为了操作实现的方便，有时也会规定结点的存放顺序。例如，可以规定按树的前序次序存放树中的各个结点，或规定按树的层次次序安排所有结点

子女链表表示：



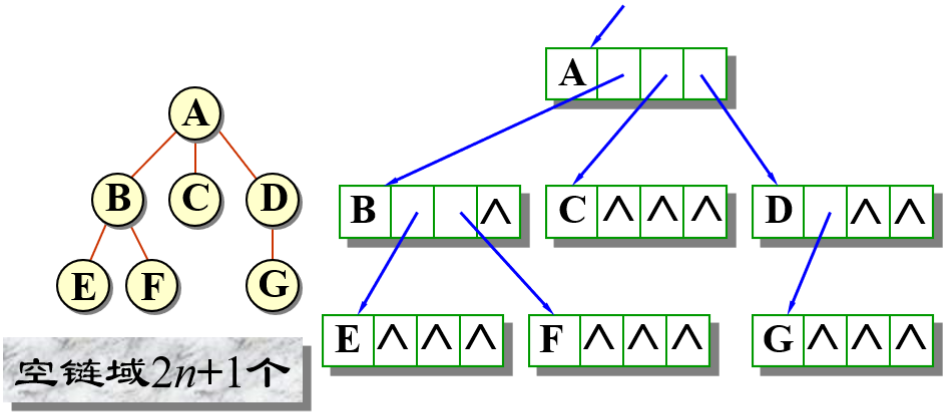
无序树情形链表中各结点顺序任意，有序树必须自左向右链接各个子女结点。

子女指针表示：

一个合理的想法是在结点中存放指向每一个子女结点的指针。但由于各个结点的子女数不同，每个结点设置数目不等的指针，将很难管理。

为此，设置等长的结点，每个结点包含的指针个数相等，等于树的度（degree）。

这保证结点有足够的指针指向它的所有子女结点。但可能产生很多空闲指针，造成存储浪费。



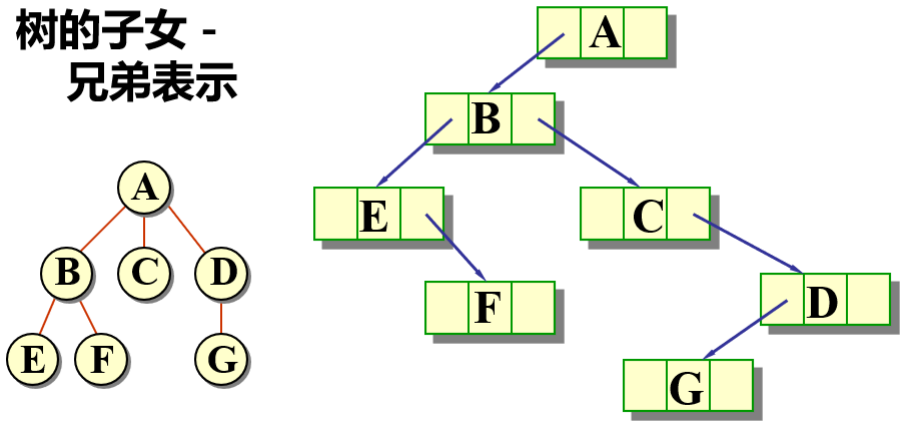
等数量的链域

子女-兄弟表示

firstChild 指向该结点的第一个子女结点。无序树时，可任意指定一个结点为第一个子女。

nextSibling 指向该结点的下一个兄弟。任一结点在存储时总是有顺序的。

若想找某结点的所有子女，可先找firstChild,再反复用 nextSibling 沿链扫描。



```
template <class T>
struct TreeNode {                                   //树的结点类
     T data;                                        //结点数据
     TreeNode<T> *firstChild, *nextSibling;         //子女及兄弟指针
     TreeNode (T value = 0, TreeNode<T> *fc = NULL,
        TreeNode<T> *ns = NULL)                     //构造函数
        : data (value), firstChild (fc), nextSibling (ns) { }
};
```

5.6.2 树的遍历

- 深度优先遍历

  先根次序遍历

  后根次序遍历

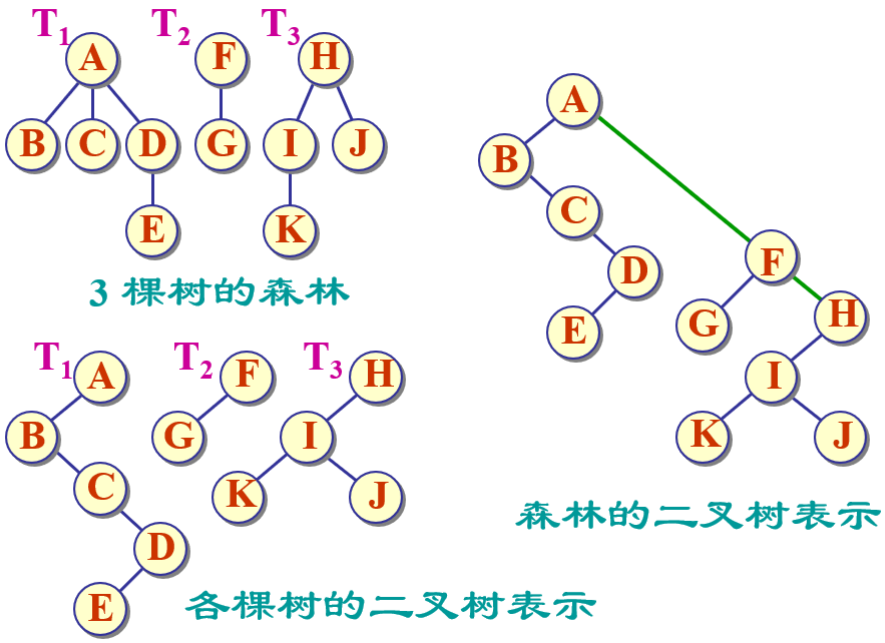
- 广度优先遍历（层次次序）遍历

利用队列实现

5.6.3 森林与二叉树的转换

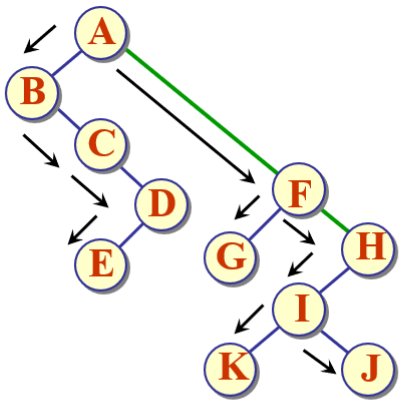
将一般树化为二叉树表示就是用树的子女-兄弟表示来存储树的结构。

森林与二叉树表示的转换可以借助树的二叉树表示来实现。



5.6.4 森林的遍历

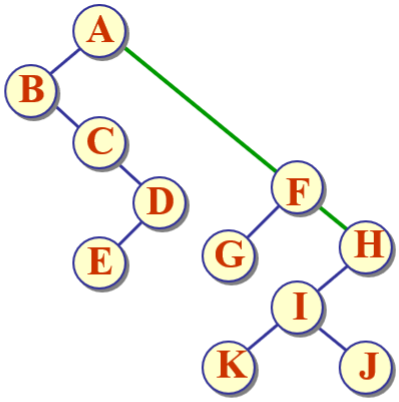
森林的遍历也分为深度优先遍历和广度优先遍历，深度优先遍历又可分为先根次序遍历和后根次序遍历。



- **森林的先根次序遍历的结果序列**

  ABCDE FG HIKJ

- **这相当于对应二叉树的前序遍历结果。**



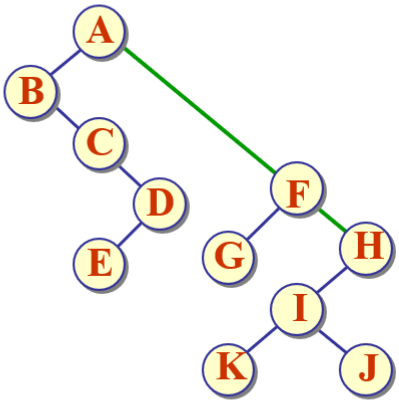
- 森林的后根次序遍历的结果序列

  BCEDA GF KIJH

- 这相当于对应二叉树中序遍历的结果。

广度优先遍历（层次序遍历）

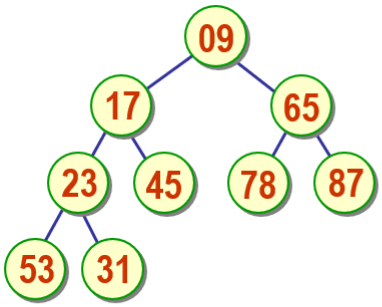
- 若森林 F 为空，返回；否则
  - ✓ 依次遍历各棵树的根结点；
  - ✓ 依次遍历各棵树根结点的所有子女；
  - ✓ 依次遍历这些子女结点的子女结点；
  - ✓ ……



AFH BCDGIJ EK

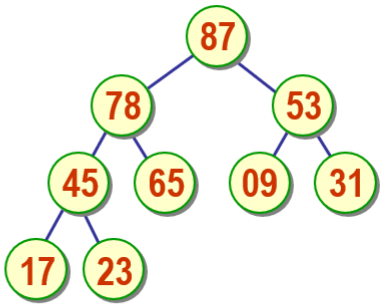
5.7 堆

5.7.1 堆的定义



完全二叉树顺序表示

$K_i \leqslant K_{2i+1}$ && $K_i \leq K_{2i+2}$



完全二叉树顺序表示

$K_i \geqslant K_{2i+1}$ && $K_i \geq K_{2i+2}$

5.7.2 堆的元素下标计算

·由于堆存储在下标从 0 开始计数的一维数组中，因此在堆中给定下标为 i 的结点时

a)如果 i = 0，结点 i 是根结点，无双亲；否则结点 i 的父结点为结点 (i-1)/2)；向下取整

b)如果 2i+1 > n-1，则结点 i 无左子女；否则结点 i 的左子女为结点 2i+1；

c)如果 2i+2 > n-1，则结点 i 无右子女；否则结点 i 的右子女为结点 2i+2。

5.7.3 堆的建立与调整

```
template <class T, class E>
MinHeap<T>::MinHeap (E arr[], int n) {
     maxHeapSize = (DefaultSize < n) ? n : DefaultSize;
     heap = new E[maxHeapSize];
     if (heap == NULL) {
        cerr << “堆存储分配失败！” << endl;  exit(1);
     }
     for (int i = 0; i < n; i++) heap[i] = arr[i];
     currentSize = n;                              //复制堆数组，建立当前大小
     int currentPos = (currentSize-2)/2;           //找最初调整位置:最后分支结点
     while (currentPos >= 0) {                     //逐步向上扩大堆
           siftDown (currentPos, currentSize-1); //局部自上向下下滑调整
           currentPos--;
     }
};
```

调整算法：自上而下

```
template <class T, class E>
void MinHeap<T>::siftDown (int start, int m ) {
//私有函数: 从结点start开始到m为止，自上向下比较，
//如果子女的值小于父结点的值，  则关键码小的上浮，
//继续向下层比较，将一个集合局部调整为最小堆。
    int i = start,   j = 2*i+1;                          //j是i的左子女位置
    E temp = heap[i];
    while (j <= m) {                                     //检查是否到最后位置
        if ( j < m && heap[j] > heap[j+1] ) j++;         //让j指向两子女中的小者
        if ( temp <= heap[j] ) break;                    //小则不做调整
        else { heap[i] = heap[j];  i = j; j = 2*j+1; } //否则小者上移，i，j下
移
     }
     heap[i] = temp;                                    //回放temp中暂存的元素
};
```

调整算法：自下而上

```cpp
void MinHeap<T>::ShiftUp (int start) {
//私有函数: 从结点start开始到结点0为止，自下向上
//比较，如果子女的值小于父结点的值，则相互交换，
//这样将集合重新调整为最小堆
     int j = start,  i = (j-1)/2;   E temp = heap[j];
     while (j > 0)
     {                                                    //沿父结点路径向上直达根
        if (heap[i] <= temp) break;                       //父结点值小，不调整
        else { heap[j] = heap[i];  j = i;  i = (i-1)/2; }//父结点结点值大，调整
     }
}
```

5.8 Huffman树

5.8.1 路径长度 (Path Length)

两个结点之间的路径长度 PL 是连接两结点的路径上的分支数。

树的外部路径长度是各叶结点（外结点）到根结点的路径长度之和 EPL。

树的内部路径长度是各非叶结点（内结点）到根结点的路径长度之和 IPL。

树的路径长度 PL = EPL + IPL。

- **n 个结点的二叉树的路径长度不小于下述数列前 n 项的和，即**

$$PL = \sum_{i=1}^{n} \lfloor \log_2 i \rfloor$$
$$= 0+1+1+2+2+2+2+3+3+\cdots$$

- **其路径长度最小者为**

$$\mathrm{PL} = \sum_{\mathrm{i}=1}^{\mathrm{n}} \lfloor \log_2 \mathrm{i} \rfloor$$

- **完全二叉树或理想平衡树满足这个要求。**

5.8.2 带权路径长度

在很多应用问题中为树的叶结点赋予一个权值，用于表示出现频度、概率值等。因此，在问题处理中把叶结点定义得不同于非叶结点，把叶结点看成“外结点”，非叶结点看成“内结点”。这样的二叉树称为扩充二叉树。

扩充二叉树中只有度为 2 的内结点和度为 0的外结点。根据二叉树的性质，有 n 个外结点就有 n-1 个内结点，总结点数为2n-1。

若一棵扩充二叉树有 n 个外结点，第 i 个外结点的权值为w_i，它到根的路径长度为l_i，则该外结点到根的带权路径长度为$w_i * l_i$。

扩充二叉树的带权路径长度定义为树的各外结点到根的带权路径长度之和。

$$WPL = \sum_{i=1}^{n} w_i * l_i$$

对于同样一组权值，如果放在外结点上，组织方式不同，带权路径长度也不同。

5.8.3 Huffman树

带权路径长度达到最小的扩充二叉树即为Huffman树。

在Huffman树中，权值大的结点离根最近。