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。

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 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

1)初始化HT[1..2n-1]：lch=rch=parent=0

2)输入初始n个叶子结点：置HT[1..n]的weight值

3)进行以下n-1次合并，依次产生HT[i]，i=n+1..2n-1：

3.1)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)

3.2)修改HT[s1]和HT[s2]的parent值： parent=i

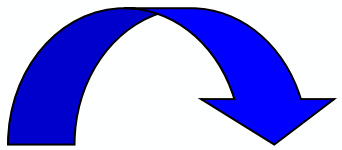
3.3)置HT[i]：weight=HT[s1].weight + HT[s2].weight ，

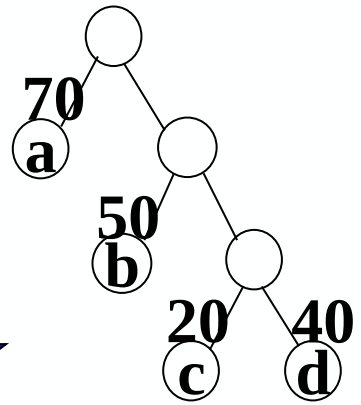
lch=s1，rch=s2

例:设n=4，w={70,50,20,40}

试设计 huffman code (m=2*4-1=7)

	weight	parent	lch	rch
1	70	0	0	0
2	50	0	0	0
3	20	0	0	0
4	40	0	0	0
5				
6				
7				





	weight	parent	lch	rch
1	70	7	0	0
2	50	6	0	0
3	20	5	0	0
4	**40**	**5**	**0**	**0**
5	**60**	**6**	**3**	**4**
6	**110**	**7**	**2**	**5**
7	**180**	**0**	**1**	**6**

算法

```
void CreatHuffmanTree (HuffmanTree & HT,int n){
if(n<=1)return;
m=2*n-1;
HT=new HTNode[m+1];//0号单
for(i=1;i<=m;++i)
  {HT[i].lch=0;HT[i].rch=0;HT[i].parer
for(i=1;i<=n;++i)cin>>HT[i].wei
```

例:设n=8, w={5,29,7,8,14,23,3,11}

试设计 huffman code (m=2*8-1=15)

	weight	parent	lch	rch
1	5	0	0	0
.	29	0	0	0
.	7	0	0	0
.	8	0	0	0
8	14	0	0	0
	23	0	0	0
	3	0	0	0
	11	0	0	0
9		0	0	0
.		0	0	0
.		0	0	0
15		0	0	0
		0	0	0
		0	0	0
		0	0	0

```
for( i=n+1;i<=m;++i)       //构造  Huffman树
 { Select(HT,i-1, s1, s2);
     //在HT[k](1≤k≤i-1)中选择两个其双亲域为0,
     // 且权值最小的结点,
     // 并返回它们在HT中的序号s1和s2
   HT[s1].parent=i;   HT[s2] .parent=i;
    //表示从F中删除s1,s2
   HT[i].lch=s1;    HT[i].rch=s2 ;
    //s1,s2分别作为i的左右孩子
   HT[i].weight=HT[s1].weight + HT[s2] .weight;
    //i 的权值为左右孩子权值之和
    }
}
```

构造Huffman tree后,HT为:

	weight	parent	lch	rch
1	5	9	0	0
	29	14	0	0
.	7	10	0	0
	8	10	0	0
.	14	12	0	0
	23	13	0	0
.	3	9	0	0
8	11	11	0	0
9	8	11	1	7
	15	12	3	4
.	19	13	8	9
	29	14	5	10
.	42	15	6	11
	58	15	2	12
15	100	0	13	14

```
void CreatHuffmanCode(HuffmanTree HT, HuffmanCode &HC, int n){
//从叶子到根逆向求每个字符的赫夫曼编码，存储在编码表HC中
HC=new char *[n+1];                    //分配n个字符编码的头指针矢量
cd=new char [n];                       //分配临时存放编码的动态数组空间
cd[n-1]='\0';          //编码结束符
for(i=1; i<=n; ++i){              //逐个字符求赫夫曼编码
   start=n-1; c=i; f=HT[i].parent;
   while(f!=0){       //从叶子结点开始向上回溯，直到根结点
       --start;                                          //回溯一次start向前指一个位置
       if (HT[f].lch= =c)  cd[start]='0'; //结点c是f的左孩子，则生成代码0
       else cd[start]='1';                               //结点c是f的右孩子，则生成代码1
       c=f; f=HT[f].parent;                              //继续向上回溯
   }                                     //求出第i个字符的编码
   HC[i]= new char [n-start];            // 为第i 个字符编码分配空间
   strcpy(HC[i], &cd[start]);      //将求得的编码从临时空间cd复制到HC的当前行中
 }
 delete cd;                        //释放临时空间
} // CreatHuffanCode
```

哈夫曼编码的几点结论

- 哈夫曼编码是不等长编码
- 哈夫曼编码是前缀编码，即任一字符的编码都不是另一字符编码的前缀
- 哈夫曼编码树中没有度为1的结点。若叶子结点的个数为n，则哈夫曼编码树的结点总数为 2n-1
- 发送过程：根据由哈夫曼树得到的编码表送出字符数据
- 接收过程：按左0、右1的规定，从根结点走到一个叶结点，完成一个字符的译码。反复此过程，直到接收数据结束

5.8 案例分析与实现

案例5.2： 利用二叉树求解表达式的值

【案例实现】

- 假设运算符均为双目运算符，则表达式对应的表达式树中叶子结点均为操作数，分支结点均为运算符。
- 由于创建的表达式树需要准确的表达运算次序，因此在扫描表达式创建表达式树的过程中，当遇到运算符时不能直接创建结点，而应将其与前面的运算符进行优先级比较，根据比较的结果再进行处理。
- 借助一个运算符栈OPTR，来暂存已经扫描到的还未处理的运算符。
- 每两个操作数和一个运算符就可以建立一棵表达式二叉树，而该二叉树又可以作为另一个运算符结点的一棵子树。
- 另外借助一个表达式树栈EXPT，来暂存已建立好的表达式树的根结点，以便其作为另一个运算符结点的子树而被引用。

表达式树的创建---【算法步骤】

① 初始化OPTR栈和EXPT栈，将表达式起始符“#”压入OPTR栈。

② 扫描表达式，读入第一个字符ch，如果表达式没有扫描完毕至“#”或OPTR的栈顶元素不为“#”时，则循环执行以下操作：

- 若ch不是运算符，则以ch为根创建一棵只有根结点的二叉树，且将该树根结点压入EXPT栈，读入下一字符ch；
- 若ch是运算符，则根据OPTR的栈顶元素和ch的优先级比较结果，做不同的处理：
  - 若是小于，则ch压入OPTR栈，读入下一字符ch；
  - 若是大于，则弹出OPTR栈顶的运算符，从EXPT栈弹出两个表达式子树的根结点，以该运算符为根结点，以EXPT栈中弹出的第二个子树作为左子树，以EXPT栈中弹出的第一个子树作为右子树，创建一棵新二叉树，并将该树根结点压入EXPT栈；
  - 若是等于，则OPTR的栈顶元素是“(”且ch是“)”，这时弹出OPTR栈顶的“(”，相当于括号匹配成功，然后读入下一字符ch。

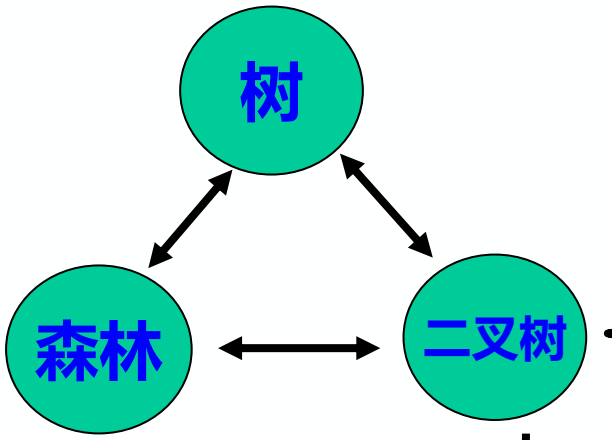
表达式树的求值---【算法步骤】

① 设变量lvalue和rvalue分别用以记录表达式树中左子树和右子树的值，初始均为0。

② 如果当前结点为叶子（结点为操作数），则返回该结点的数值，否则（结点为运算符）执行以下操作：

- 递归计算左子树的值记为lvalue；
- 递归计算右子树的值记为rvalue；
- 根据当前结点运算符的类型，将lvalue和rvalue进行相应运算并返回。

小结



1、定义和性质

2、存储结构
- 顺序结构
- 链式结构
  - 二叉链表
  - 三叉链表

3、遍历
- 先序遍历
- 中序遍历
- 后序遍历

4、线索化：线索树

哈夫曼树

哈夫曼编码

小结

1. 掌握二叉树的基本概念、性质和存储结构
2. 熟练掌握二叉树的前、中、后序遍历方法
3. 了解线索化二叉树的思想
4. 熟练掌握：哈夫曼树的实现方法、构造哈夫曼编码的方法
5. 了解：森林与二叉树的转换，树的遍历方法