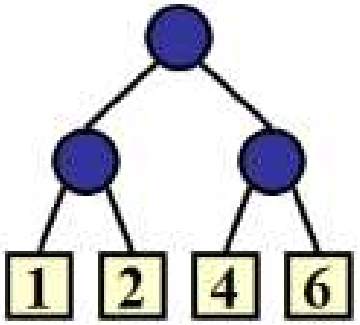
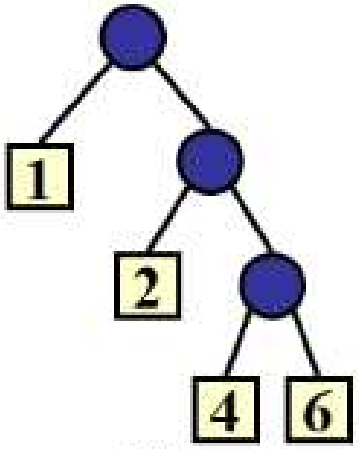
例　W={1,2,4,6}，可构造出如下的二叉树：



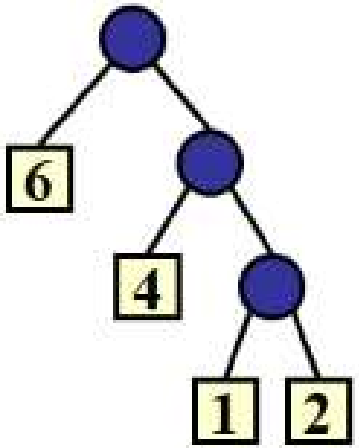
(1)

WPL
=(1+2+4+6)*2=26



(2)

WPL
=1+2*2+(4+6)*3=35



(3)

WPL
=6+4*2+(1+2)*3=23

根据定义求Huffman树的方法是：对给定的n个叶子结点（外部结点），构造出全部二叉树并求出其WPL，然后找出WPL最小的树。

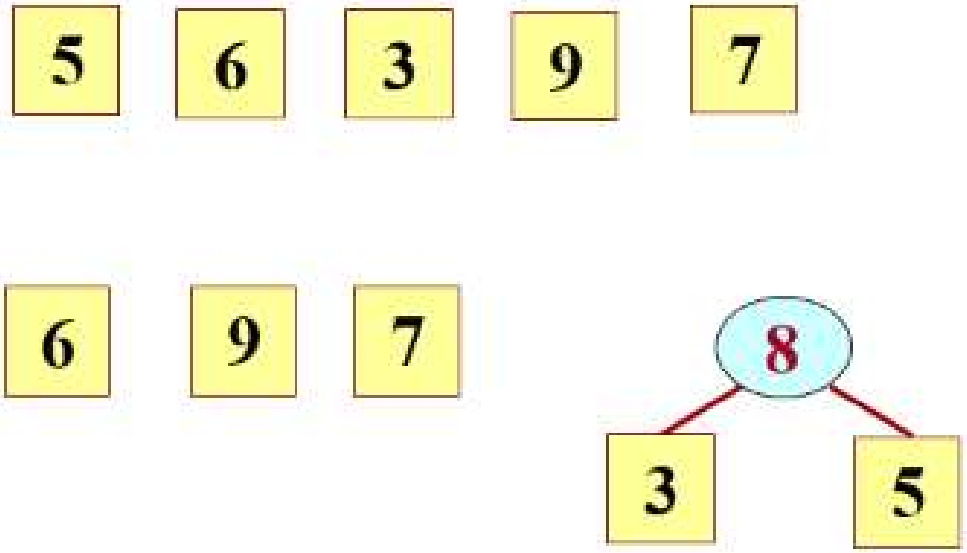
当n较大时，显然这种方法是不可取的。

二、构造huffman树

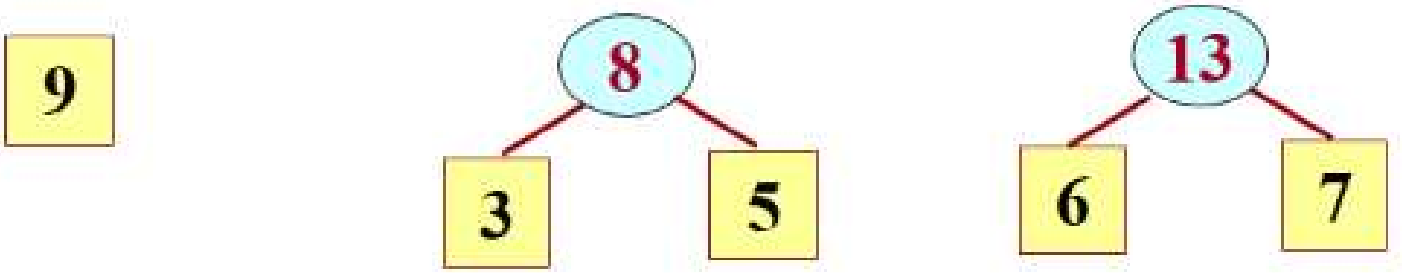
算法思想：

1. 根据权值{w_1 , w_2 ,…, w_n}构造n个二叉树F={T_1 , T_2 , …, T_n}，其中 T_i中是只含权值为w_i 的结点。
2. 从F中选两个权值最小的二叉树T_i和T_j，构造一个根结点R, R的权w_R为$w_i + w_j$ 。
3. 从F中删除T_i和T_j ，加入新的树R到F中。
4. 重复2，3 直到F中只有一棵树(或执行n-1次)为止。

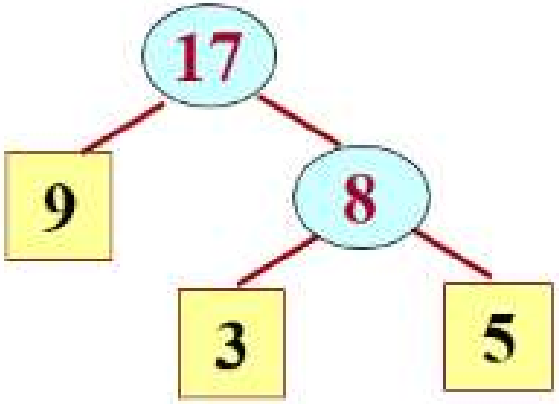
例：已知权值 W={ 5, 6, 3, 9, 7 }

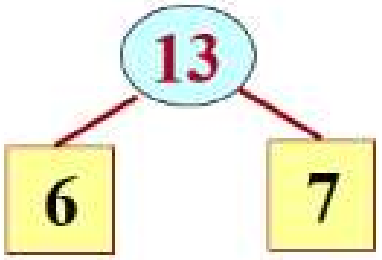


例：已知权值 W={ 5, 6, 3, 9, 7 }

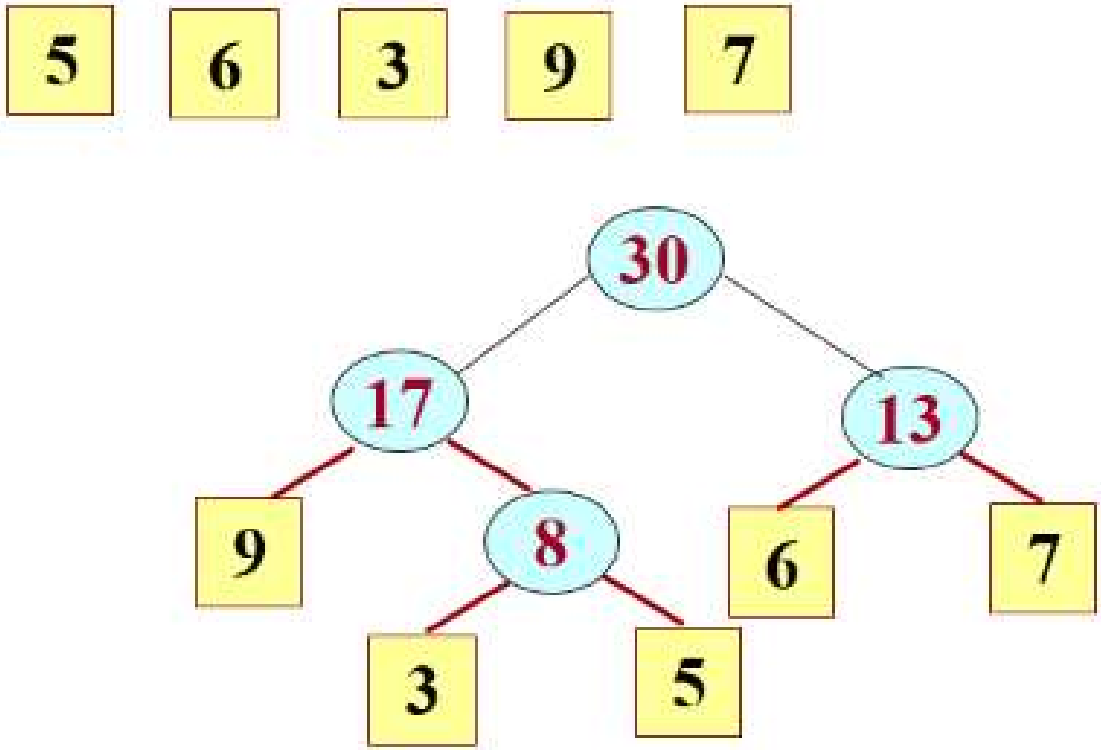


例：已知权值 W={ 5, 6, 3, 9, 7 }





例：已知权值 W={ 5, 6, 3, 9, 7 }



Huffman 算法的实现：

用n个叶结点构造出的最优二叉树共有几个结点？

- 构造最优树时共执行**n-1**次循环，每次增加一个新结点，共增加了**n-1**个结点，所以结点总数一定是**n+n-1=2n-1**
- 因为$n_0=n_2+1$，所以$n_2=n_0-1$，又由于在最优二叉树中没有度为1的结点,所以在最优二叉树中总的结点数为 **n+n-1=2n-1**

Huffman编码：

设字符集为 $\{c_1, c_2, \ldots, c_n\}$，看作叶结点

出现概率为 $\{w_1, w_2, \ldots, w_n\}$，叶结点的权

（1）构造Huffman树

（2）左分支标0，右分支标1

（3）根到叶结点c_i的路径上的二进制编码就是c_i的编码

编码长度为$\{l_1, l_2, \ldots, l_n\}$是叶结点的路径长度，

则树的带权路径长度*WPL*是：

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

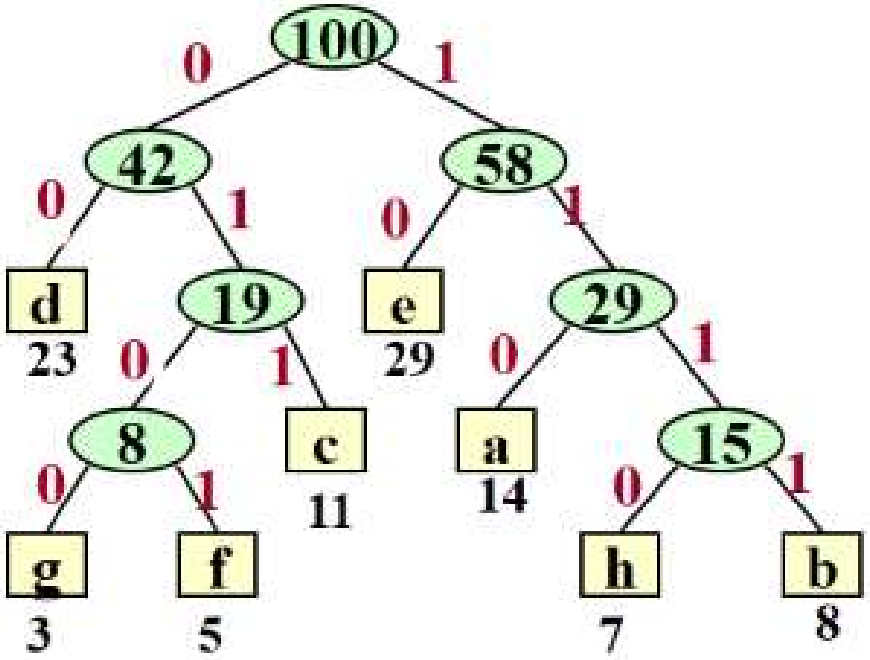
Huffman编码：

根到任何c_i的路径都不会经过任何其它c_k，因此Huffman编码是无前缀编码，即任何一个编码都不会是另一个编码的前缀。

例：字符集{a,b,c,d,e,f,g,h}出现概率分别是

{0.14, 0.08, 0.11, 0.23, 0.29, 0.05, 0.03, 0.07}

(1) 构造Huffman 树：



(2)编码：

a: 110

b: 1111

c: 011

d: 00

e: 10

f: 0101

g: 0100

h: 1110

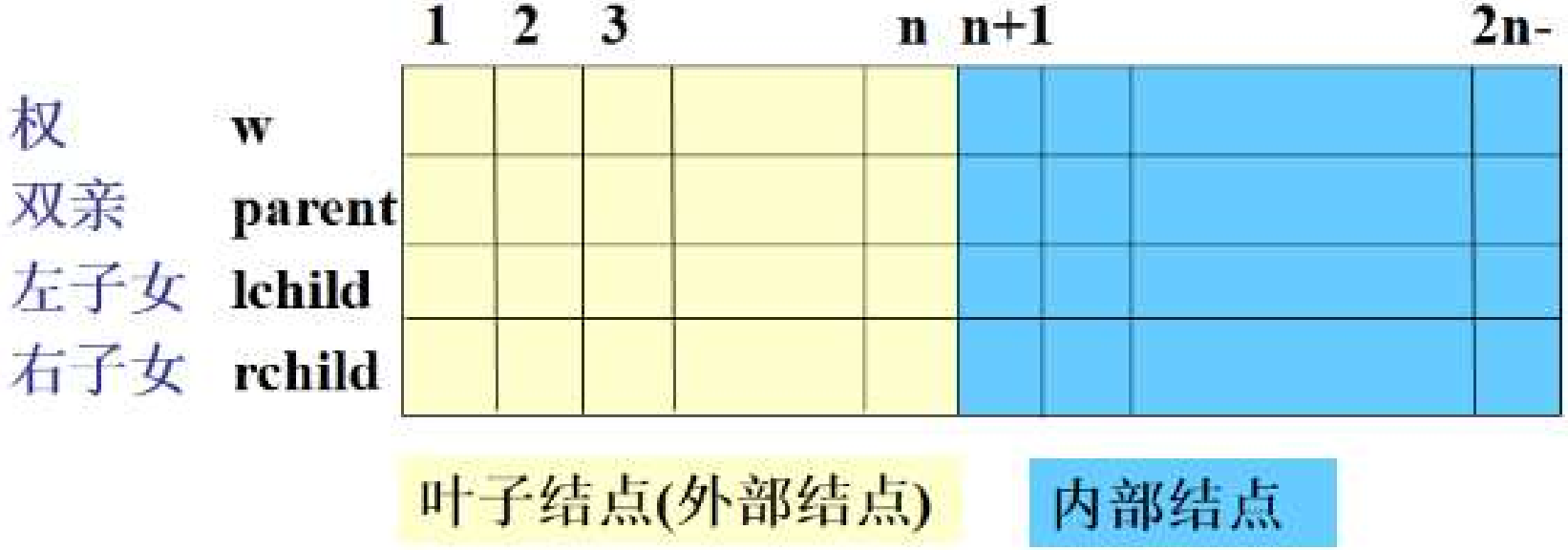
(3)编码长度：

WPL= 2*0.23+4*0.03+ 4*0.05+ 3*0.11+

2*0.29+3*0.14+4*0.08+4*0.07=2.71

Huffman 算法的实现：

用数组存储Huffman 树，每个数组元素HTNode包括4个域：



```
//——哈夫曼树和哈夫曼编码的存储表示——
typedef struct{
   unsigned int   weight;
   unsigned int   parent，lchild，rchild;
}HTNode，*HuffmanTree;  //动态分配数组存储哈夫曼树

typedef  char  **HuffmanCode;  //动态分配数组存储哈夫曼编码表
```

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：

（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)

（2）修改HT[s1]和HT[s2]的parent值： parent=i

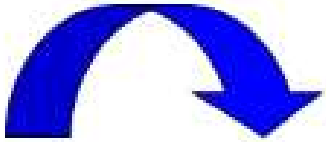
（3）置HT[i]：weight=HT[s1].weight + HT[s2].weight ，

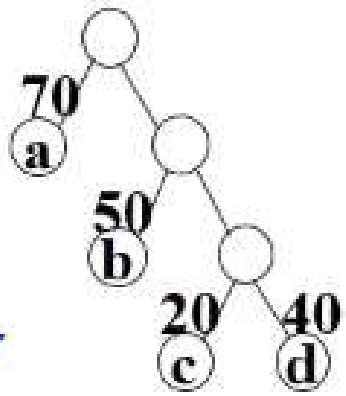
lch=s1, rch=s2

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号单元未用,HT[m]表示根结点
for(i=1;i<=m;++i)
  {HT[i].lch=0;HT[i].rch=0;HT[i].parent=0;}
for(i=1;i<=n;++i)cin>>HT[i].weight;
```

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
2	29	0	0	0
3	7	0	0	0
4	8	0	0	0
5	14	0	0	0
6	23	0	0	0
7	3	0	0	0
8	11	0	0	0
9		0	0	0
10		0	0	0
11		0	0	0
12		0	0	0
13		0	0	0
14		0	0	0
15		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:

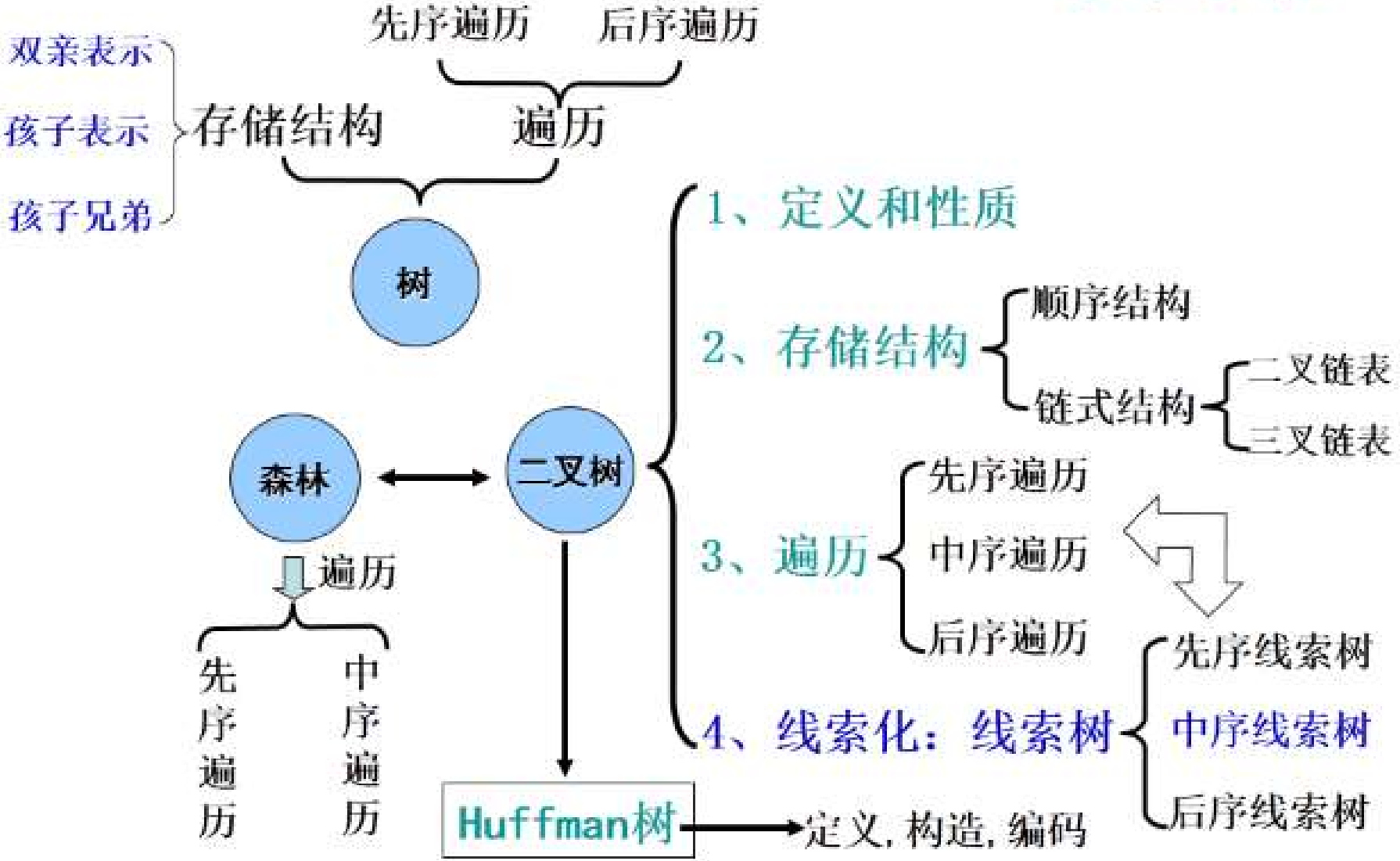
	weight	parent	lch	rch
1	5	9	0	0
	29	14	0	0
.	7	10	0	0
	8	10	0	0
.	14	12	0	0
	23	13	0	0
.	3	9	0	0
8	11	11	0	0
9	8	11	1	7
	15	12	3	4
.	19	13	8	9
	29	14	5	10
.	42	15	6	11
	58	15	2	12
15	100	0	13	14

```
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].lchild= =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

本章小结


双亲表示
孩子表示
孩子兄弟
存储结构
先序遍历
后序遍历
遍历
树
森林
二叉树
遍历
先序遍历
中序遍历
Huffman树
定义，构造，编码
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
先序线索树
中序线索树
后序线索树

12
1