实验六 赫夫曼树的构造和赫夫曼编码

【对应知识点】

二叉树。

【问题描述】

写出构造一棵赫夫曼树，并根据赫夫曼树求赫夫曼编码的算法。

【实验要求】

用户键盘输入若干个整数作为待编码字符的权值，程序建立赫夫曼树并输出各字符的赫夫曼编码。

【算法提示】

实现赫夫曼算法的前提是要考虑用什么样的存储结构来存储一棵赫夫曼树。在赫夫曼树中，没有度为 1 的结点，结点总数是 $n_0 + n_2$（其中 n_0 表示二叉树中度为 0 的结点数，n_2 表示度为 2 的结点数），而由二叉树的性质知道 $n_0 = n_2 + 1$，所以一棵赫夫曼树中结点总数是 $2n_0 - 1$。由此可以得出：任何 n 个字符的赫夫曼树的结点总数是 2n - 1。既然结点总数可以确定，就可以采用顺序存储结构来实现，即可以把结点信息存放在大小为 2n - 1 的一维数组（如数组 ht）中。

可定义如下赫夫曼树结点类型、存放赫夫曼编码的数据类型：

```
#define MAXSIZE 100                          /*结点允许的最大数量*/

typedef char elemtype;
typedef struct
{
    elemtype data;                   /*用户自定义类型，存放结点的值*/
    int weight;                      /*存放权值*/
    int parent;                      /*父结点下标*/
    int lchild;                      /*左孩子下标*/
    int rchild;                      /*右孩子下标*/
}huffnode;                           /*赫夫曼树结点类型*/

typedef struct
{
    char cd[MAXSIZE];
    int start;
}huffcode;                                   /*自定义存放赫夫曼编码的数据类型*/

huffnode ht[];                       /*  存放赫夫曼树的数组（自己定义数组大小）  */

huffcode hcd[];        /*  存放所有字符的赫夫曼编码的数组（自己定义数组大小）  */
```

ht[]的前 n 个元素表示叶结点，最后一个元素表示根结点。各字符的编码长度不等，但不超过 n，所以表示编码的类型 huffcode 中的 cd 数组大小为 n，start 域表示编码开始的位置。编码顺序存放在 hcd[i].cd 中下标从 hcd[i].start 到 n 的位置上。

可以编写三个子函数：构造一棵赫夫曼树、根据赫夫曼树求赫夫曼编码、输出赫夫曼编码。

【例】设权 w = {5，29，7，8，14，23，3，11}, n = 8, 2n – 1 = 15, 可构造出赫夫曼树如下图。其存储结构 ht[]的初始状态如表 1 所示，终结状态如表 2 所示。（此处的设计与课件略有不同。这里没有使用 ht[0]，因此用 0 表示不存在的单元。采用哪种设计都可以。）所得各字符的赫夫曼编码如表 3 所示。注意：求赫夫曼编码需走一条从叶子到根结点的路径。

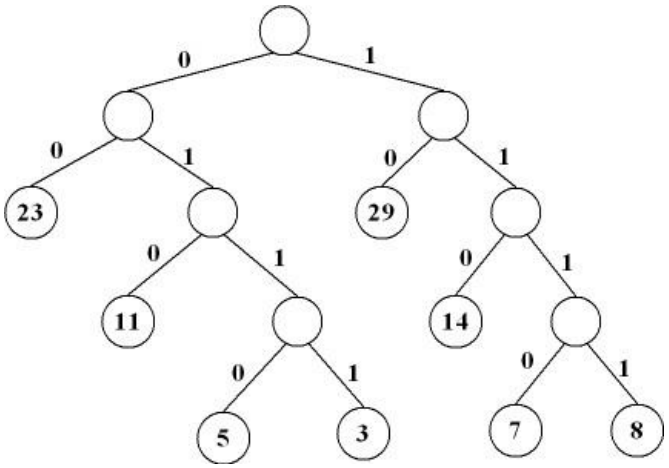

表 1　ht[]的初态

	data	weight	parent	lchild	rchild
0					
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

表 2　ht[]的终态

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

表 3 各字符的赫夫曼编码 hcd[]

	cd[]					start
	0	1	2	3	4	
0						
1					0	4
2					1	6
3					1	6
4					1	6
5					1	6
6					1	6
7					1	6

3					1	1	1	0	4
4					1	1	1	1	4
5						1	1	0	5
6							0	0	6
7					0	1	1	1	4
8						0	1	0	5