赫夫曼树的存储

- 赫夫曼树中没有度为1的结点，其叶结点为n个，总结点数为2n-1（性质3），将其存储在一个长度为2n-1的数组里面
- 每个数组分量存储一个结点信息，包括该结点的权值、双亲、左孩子、右孩子
- 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
```

赫夫曼树的算法6.12

```
void HuffmanCoding(HuffmanTree &HT, HuffmanCode &HC, int * w, int n){
    //w存放n个字符的权值（均>0），构造哈夫曼树HT，并求出n个字符的哈夫曼编码HC
    if(n <= 1)   return;
    m = 2 * n - 1;
    HT = (HuffmanTree) malloc((m + 1) * sizeof(HTNode));   // 0号单元未使用
    for(p = HT + 1, i = 1; i <= n; ++i, ++p, ++w)              *p = { *w, 0 , 0, 0 };
    for(; i <= m; ++i, ++p)           *p = { 0, 0 , 0, 0};
```

赫夫曼树的算法6.12

```
for(i = n + 1; i < = m; ++i) {            // 建赫夫曼树
    //在HT[1..i-1]中选择parent为0，且weight最小的两个结点，其序号分别为s1和s2
     Select(HT, i - 1, s1, s2);
     HT[s1].parent = i;          HT[s2].parent = i;
     HT[i].lchild =  s1;         HT[i].rchild =  s2;
     HT[i].weight = HT[s1].weight + HT[s2].weight;
}
//--- 从叶子到根逆向求每个字符的赫夫曼编码 ---//
HC =  (HuffmanCode) malloc((n + 1) * sizeof(char * )); // 分配n个字符编码的头指针向量
cd = ( char * ) malloc(n * sizeof(char));                    // 分配求编码的工作空间
cd[n - 1] =  '\0';                                           // 编码结束符
```

赫夫曼树的算法6.12

```
    for(i = 1; i <= n; ++i) {                    // 逐个字符求赫夫曼编码
        start = n - 1;                           // 编码结束符位置
        for(c = i, f = HT[i].parent; f != 0; c = f, f = HT[f].parent){   // 从叶子到根逆向求编码
            if( HT[f].lchild == c )              cd[--start] = "0";
            else  cd[--start] = "1";
        HC[i] = (char * ) malloc((n - start) * sizeof(char));  // 为第i个字符编码分配空间
        strcpy( HC[i], &cd[start] );                           // 从cd复制编码（串）到HC
    }
    free(cd);   // 释放工作空间
}// HuffmanCoding
```

赫夫曼树的算法6.13

- 向量HT的前n个分量表示叶子结点，最后一个分量表示根结点。各字符的编码长度不等，所以按实际长度动态分配空间。
- 在算法6.12中，求每个字符的哈夫曼编码是从叶子到根逆向处理的。
- 也可以从根出发，遍历整棵哈夫曼树，求得各个叶子结点所表示的字符的哈夫曼编码。
- 如算法6.13所示。

赫夫曼树的算法

```
//--- 无栈非递归遍历赫夫曼树，求赫夫曼编码--- //
HC  = (HuffmanCode) malloc((n + 1) * sizeof(char * ));
p = m;    cdlen = 0;
for(i = 1; i <= m; ++i)      HT[i].weight = 0;    //遍历哈夫曼树时用作结点状态标志
while( p ) {
     if(HT[p].weight == 0) {                                          //向左
               HT[p].weight = 1;                                      // 存在左孩子
               if( HT[p].lchild != 0 ) {  p = HT[p].lchild;   cd[cdlen++] = “0”;  }
               else if( HT[p].rchild == 0 ) {                         //登记叶子结点的字符编码
                         HC[p] = (char * ) malloc((cdlen + 1) * sizeof(char));
                         cd[cdlen] = '\0';       strcpy( HC[p], cd);   // 复制编码串
               } }
```

赫夫曼树的算法6.13

```
    } else if(HT[p].weight == 1) {          //向右
            HT[p].weight = 2;
            if(HT[p].rchild != 0) {   p = HT[p].rchild;  cd[cdlen++] = “1”;     }
    } else {
            HT[p].weight = 0;
            p = HT[p].parent;                   // 退回到父结点
            cdlen--;                            // 编码长度减1
    } //else
}//while
```

赫夫曼树解码

```
//根据给定的n个赫夫曼编码HC，计算其代表的权值
void HuffmanDecoding(HuffmanTree HT, HuffmanCode HC, int** w, int n){
        if(n <= 0) {return ;//ERROR;}
        (*w) = (int*) malloc(n * sizeof(int));          r = 2 * n - 1;    // 根结点位置
         for(i = 1; i <= n; i++) {
                s = HC[i];       k = r;   //s 为i字符的哈夫曼编码，k，为根结点位置
                for(j = 0; j < strlen(s); j++) {   // 从根结点往下找
                        if(s[j] == '0') { k = HT[k].lchild;   // 向左}
                        else if(s[j] == '1') {k = HT[k].rchild;   // 向右
                        } else {return ;//ERROR;}}
                (*w)[i - 1] = HT[k].weight;    //保存在存储n个字符权值的数组中}}
```

哈夫曼树及其应用

● 项目要求

编写程序：

1、终端输入若干字符，统计字符出现的频率，将字符出现的频率作为结点的权值，建立哈夫曼树，对各字符进行哈夫曼编码，然后对输入字符串进行编码和解码。

2.输出：　　1）中序打印输出哈夫曼树；

　　2）输出各字符的哈夫曼编码表；

　　3）解码后的字符串

3、最后提交完整的实验报告和可运行源程序（.c/.cpp格式）。