

《数据结构》上机报告

2018 年 11 月 12 日

姓名：刘思源 学号：1651390 班级：电子三班 得分：_____

实验题目	哈夫曼树	
问题描述	给定 n 个权值作为 n 个叶子结点，构造一棵二叉树，若该树的带权路径长度达到最小，称这样的二叉树为最优二叉树，也称为哈夫曼树 (Huffman Tree)。哈夫曼树是带权路径长度最短的树，权值较大的结点离根较近。	
基本要求	1. (p1) 本题练习构建哈夫曼树。求最短路径权值。 2. (p2) 本题练习用双亲表示法输入哈夫曼树，给出哈夫曼编码。 3. (p3) 本题练习对哈夫曼编码进行译码。	
	已完成基本内容（序号）：	
选做要求		
	已完成选做内容（序号）	1, 2, 3
数据结构设计	<pre>typedef struct { ElemType weight; ElemType parent; ElemType lChild; ElemType rChild; } Node, *HuffmanTree;</pre>	

功能(函数)说明	<p>1、 构建哈夫曼树。</p> <p>哈夫曼树实际上是一个多维的数组。因为构建哈夫曼树的本意在于，要使用树的结构进行一定的编码和译码的运算，所以对于树的链形式的其他运算并不是很在意。所以采用动态分配数组的方式进行存储。输入每个节点的权值，将每个双亲值，左右子节点都赋为 0。全部输入完成后，进行哈夫曼树的生成，本文采用希尔排序的方法，先对已经存在的数字进行排序，选出前两个数字进行数的生成。将新节点的权值赋给接下来一个空的结点，将左右子节点的值重置。直到所有的数组已经排满。</p> <pre> void createHuffmanTree(HuffmanTree *huffmanTree, int n) { int m = 2 * n - 1; int i; ElemType wei; huffmanTree = (HuffmanTree*)malloc((m + 1) * sizeof(Node)); for (i = 1; i <= n; i++) { cin >> wei; (*huffmanTree)[i].weight=wei; (*huffmanTree)[i].parent = 0; (*huffmanTree)[i].lChild = 0; (*huffmanTree)[i].rChild = 0; } for (i = n+1; i <= m; i++) { Node min; min = shellSort(huffmanTree, i-1); (*huffmanTree)[i].weight = min.weight; (*huffmanTree)[i].parent = min.parent; (*huffmanTree)[i].lChild = min.lChild; (*huffmanTree)[i].rChild = min.rChild; } return; } </pre> <p>2、 希尔排序</p> <p>希尔排序返回生成的新的节点。基于插入排序发展而来。希尔排序的思想基于两个原因： 1) 当数据项数量不多的时候，插入排序可以很好的完成工作。 2) 当数据项基本有序的时候，插入排序具有很高的效率。</p> <p>基于以上的两个原因就有了希尔排序的步骤：</p> <p>a. 将待排序序列依据步长(增量)划分为若干组，对每组分别进行插入排序。初始时，step=len/2，此时的增量最大，因此每个分组内数据项个数相对较少，插入排序可以很好的完成排序工作（对应 1）。</p>
----------	---

b. 以上只是完成了一次排序，更新步长 $step=step/2$ ，每个分组内数据项个数相对增加，不过由于已经进行了一次排序，数据项基本有序，此时插入排序具有更好的排序效率(对应 2)。直至增量为 1 时，此时的排序就是对这个序列使用插入排序，此次排序完成就表明排序已经完成。

```
Node shellSort(HuffmanTree *huffmanTree, int len)
{
    Node min;
    int arr[MAXSIZE];
    int i, j = 0, tmp = 0;
    for (i = 1; i < len; i++) {
        arr[i] = (*huffmanTree)[i].weight;
    }
    for (int d = len / 2; d > 0; d /= 2) {
        for (i = d; i < len; i++) {
            j = i - d;
            tmp = arr[i];
            while (j >= 0 && arr[j] > tmp) {
                arr[j + d] = arr[j];
                j -= d;
            }
            arr[j + d] = tmp;
        }
    }
    i = 0;
    while (arr[i] == -1) {
        i++;
    }
    j = i + 1;
    while (arr[j] == -1) {
        j++;
    }
    min.weight = arr[i] + arr[j];
    min.parent = 0;
    for (i = 0; i < len; i++) {
        if (arr[i] == (*huffmanTree)[i].weight) {
            min.lChild = i;
            (*huffmanTree)[i].parent = len + 1;
        }
        if (arr[j] == (*huffmanTree)[i].weight) {
            min.lChild = i;
            (*huffmanTree)[i].parent = len + 1;
        }
    }
    return min;
}
```

```
}
```

3、计算哈夫曼树的权值

遍历树，将所有非叶节点的权值相加得到哈夫曼树的最短路径。

```
int Cal(HuffmanTree *huffmanTree, int len)
{
    int i, sum = 0;
    for (i = 1; i < len; i++) {
        if ((*huffmanTree)[i].lChild == 0 && (*huffmanTree)[i].rChild == 0) {
            continue;
        }
        sum += (*huffmanTree)[i].weight;
    }
    return sum;
}
```

4、哈夫曼编码

哈夫曼编码时由叶节点向根节点进行编码。使用一个栈储存该结点的编码值，如果该结点的子节点为0，证明该结点为左子节点，将0压入栈；反之将1压入栈。去索引该结点的双亲，重复进行这一步骤直到找到根节点。最后再读出该结点的编码。

```
void creatHuffmanCode2(HuffmanTree *huffmanTree, HuffmanCode *huffmanCode, int n)
{
    int i;
    int temp;
    int p;

    for (i = 1; i <= n; i++) {
        SqStack S;
        InitStack(S);
        //cout << i << " ";
        for (p = i; (*huffmanTree)[p].parent != 0; p = (*huffmanTree)[p].parent) {
            if ((*huffmanTree)[p].Child == 0) {
                temp = 0;
            }
            else {
                temp = 1;
            }
            Push(S, temp);
        }
        cout << i << " ";
        ReadAll(S);
        //cout << endl;
    }
}
```

```
return;
```

5、由编码构建哈夫曼树

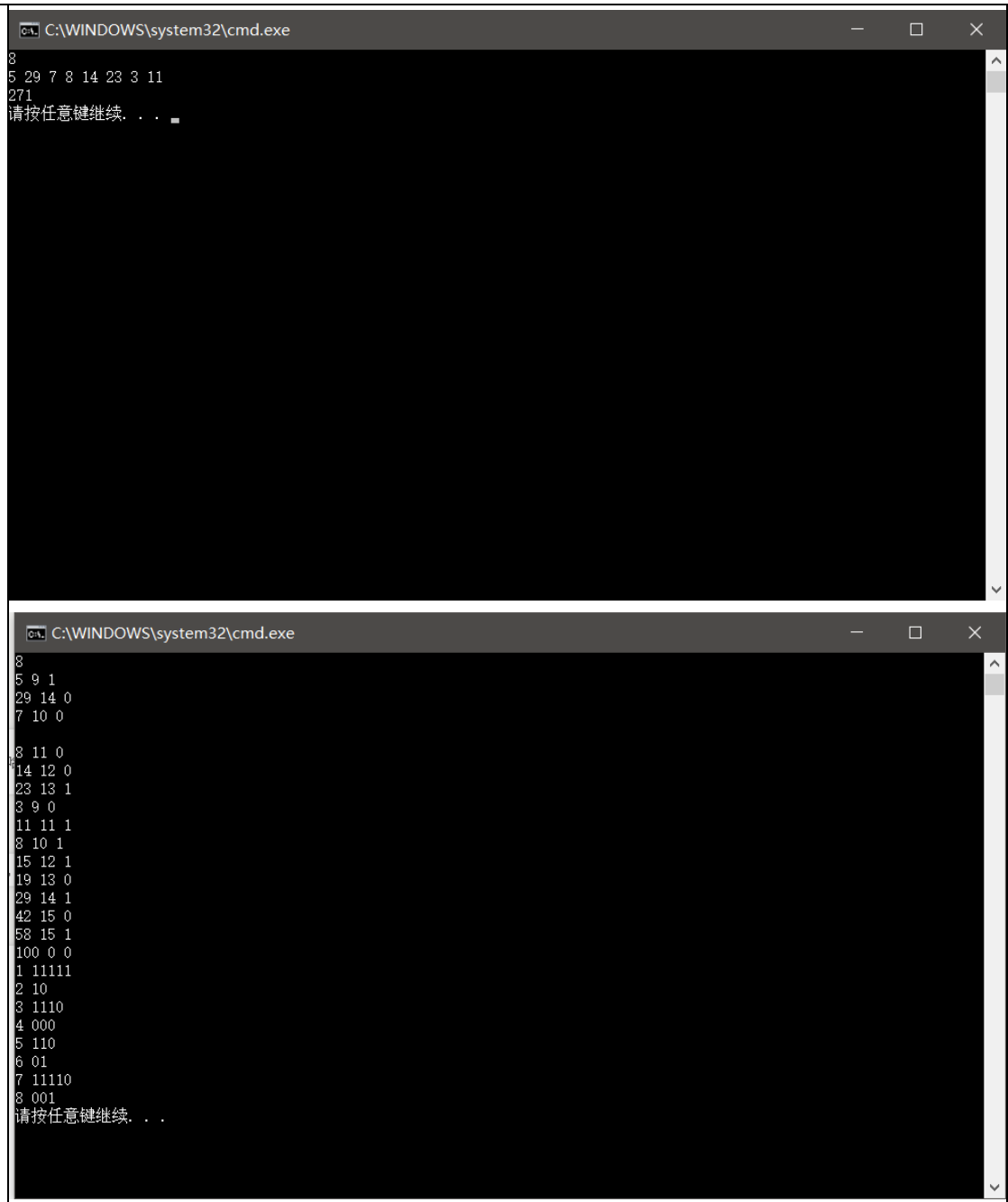
第三题中已知了哈夫曼编码，构建哈夫曼树时，应当由根节点往下构建。读入每一个节点的编码值，如果读入的数为 1，证明该结点是右子树中的结点。检索当前结点，如果右子节点存在，就直接赋值到这一节点，如果不存在就新申请一个结点。一个结点完成后，返回到根节点。直到所有的结点完成构建。

```
void CreateTree (HTree &T)
```

```
{  
    HTree Root = T;  
    T->LChild = NULL;  
    T->RChild = NULL;  
    int n,wei,i;  
    char code[MAXSIZE];  
    cin >> n;  
    while (n) {  
        cin >> wei;  
        cin >> code;  
        for (i = 0; code[i] != '\0'; i++) {  
            if (code[i] == '1') {  
                if (!T->RChild) {  
                    T->RChild = new HNode;  
                    T->RChild->LChild = NULL;  
                    T->RChild->RChild = NULL;  
                }  
                T = T->RChild;  
            }  
            else {  
                if (!T->LChild) {  
                    T->LChild = new HNode;  
                    T->LChild->LChild = NULL;  
                    T->LChild->RChild = NULL;  
                }  
                T = T->LChild;  
            }  
        }  
        T->weight = wei;  
        T = Root;  
        n--;  
    }  
    return;
```

	<pre> } 6、哈夫曼译码 由根节点向下寻找，如果读到的是 0，就向左节点继续搜索，如果读到的是 1， 就向右节点继续搜索。直到搜索到叶节点。读出这个数。 void Decode(HTree T,char Code[]) { int i=0; HTree Root = T; while (Code[i] != '\0') { if (!T->LChild && !T->RChild) { cout << char(T->weight); T = Root; continue; } if (Code[i] == '1') { T = T->RChild; } else { T = T->LChild; } i++; } cout << char(T->weight); cout << endl; return; } </pre>
开发环境	Visual studio 2017

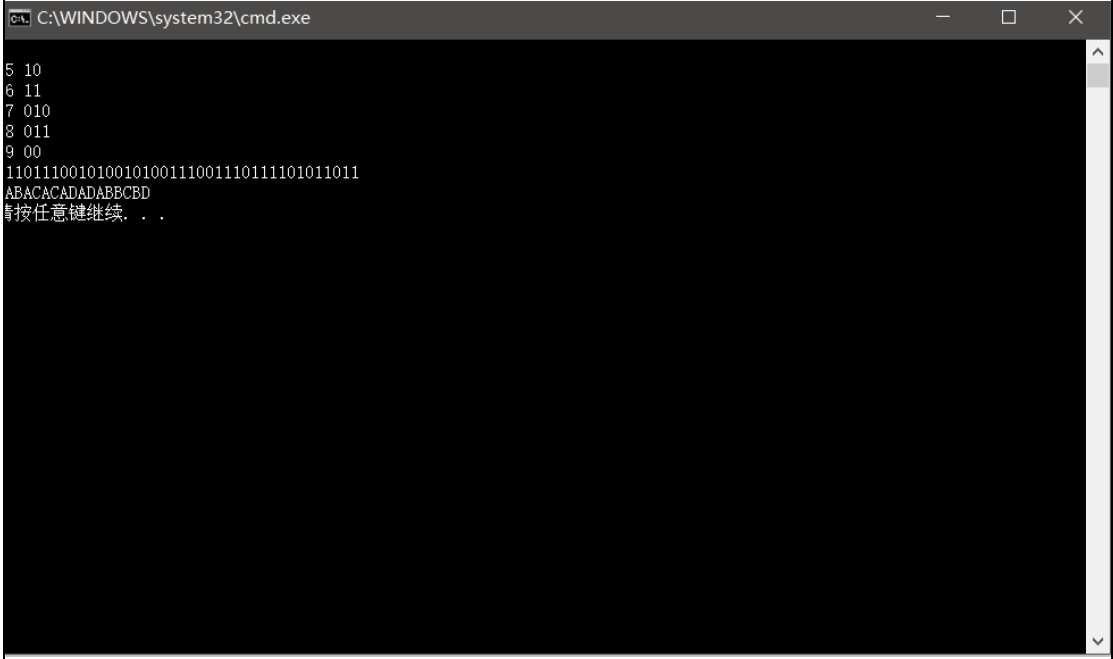
调试分析



```
C:\WINDOWS\system32\cmd.exe
8
5 29 7 8 14 23 3 11
271
请按任意键继续. . .

C:\WINDOWS\system32\cmd.exe
8
5 9 1
29 14 0
7 10 0

8 11 0
14 12 0
23 13 1
3 9 0
11 11 1
8 10 1
15 12 1
19 13 0
29 14 1
42 15 0
58 15 1
100 0 0
1 11111
2 10
3 1110
4 000
5 110
6 01
7 11110
8 001
请按任意键继续. . .
```

	 <pre> C:\WINDOWS\system32\cmd.exe 5 10 6 11 7 010 8 011 9 00 110111001010010100111001110111101011011 ABACACADADABBCED 请按任意键继续. . . </pre>
心得体会	<ol style="list-style-type: none"> 1、 哈夫曼数的形态 哈夫曼树构建的目的在于，想要使用树的特殊形态进行编码和译码。所以链表的存储形式意义不大。可以使用数组的形式进行储存，直接索引双亲结点进行构建。 2、 构建哈夫曼树的顺序 构建哈夫曼树有两种顺序。如果给定了初始的数值，就应采用从叶到根的构建方式。然后重新进行排序在进行构建，使用数组的线性储存方式。如果给定了哈夫曼节点的编码结果，就应该从根节点到叶结点的构建方式，动态申请结点，使用链表的储存方式。 3、 另外，有时判断指针是否为空的时候，应该使用(!p)的方式，而不应使用p=NULL的方式。对于不同的编译器 NULL 的值可能不同，就可能造成意想不到的失误。所以最好使用逻辑判断。