实验报告（week-08）

2020111235 马靖淳

一、实验任务

1. 二叉树遍历的应用
2. 赫夫曼编码

二、实验上机时间 6h

三、知识点

1. 二叉树遍历的应用
   （1） 计算二叉树叶子结点的个数
   （2） 计算二叉树的深度
2. 赫夫曼编码
   （1） 赫夫曼树和赫夫曼编码的存储结构
   （2） 赫夫曼编码算法
   （3） 译码算法

四、源代码及结果截屏

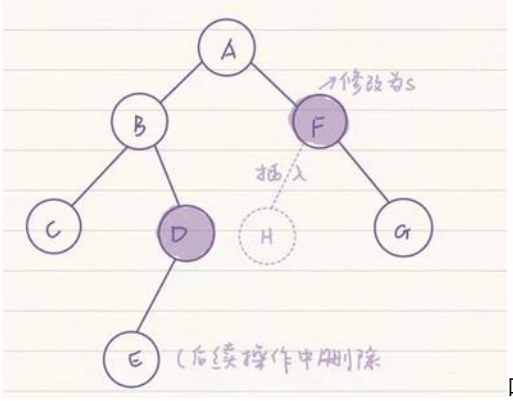
1. 二叉树遍历的应用

因为直接在上次作业 BiTree.c 中继续写的代码，全复制过来太长了，所以只摘取了本周所需要的代码

```
void CountLeaf(BiTree T, int* count)
{
    if (T)
    {
        if ((!T->lchild) && (!T->rchild))
            (*count)++;//对叶子结点计数
        CountLeaf(T->lchild, &(*count));
        CountLeaf(T->rchild, &(*count));
    }//if
}//CountLeaf

int BiTreeDepth(BiTree T)
{
    int lchildHeight, rchildHeight;
    if (T == NULL)
        return 0;
    lchildHeight = BiTreeDepth(T->lchild);
    rchildHeight = BiTreeDepth(T->rchild);
    return (lchildHeight > rchildHeight) ? (1 + lchildHeight) : (1 + rchildHeight);
}

int main()
{
    BiTree T;
    //InitBiTree test
```

```
    if (InitBiTree(&T))
        printf("InitBiTree success!\n");
    else
        printf("InitBiTree unsuccess!\n");

    //CreateBiTree test
    char ch1, ch2, ch3;
    printf("请输入一个二叉树T:");
    if (CreateBiTree(&T))
        printf("CreateBiTree success!\n");
    else
        printf("CreateBiTree unsuccess!\n");
    ch1 = getchar();//吸收换行符

    //CountLeaf test
    int a = 0;
    CountLeaf(T, &a);
    printf("叶子节点个数为：%d\n", a);

    //BiTreeDepth test
    printf("S 的深度为：%d\n", BiTreeDepth(T));

    return 0;
}
```

```
Microsoft Visual Studio 调试控制台
InitBiTree success!
请输入一个二叉树T:ABC##DE###F#G##
CreateBiTree success!
叶子节点个数为：3
S的深度为：4

C:\Users\DELL\source\repos\BiTree\Debug\BiTree.exe (进程 17476)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```



叶子节点为 C、E、G，深度为 4

2. 赫夫曼编码

（1） 赫夫曼树和赫夫曼编码的存储结构

（2） 赫夫曼编码算法

（3） 译码算法

（1）Huffmantree.h

```
#ifndef HUFFMANTREE_H
#define HUFFMANTREE_H

#include <stdlib.h>
#include "Status.h"

typedef int TElemType;
typedef struct
{
    unsigned char ch;//叶子结点字符（根据应用需要添加此项）
    unsigned int weight;
    unsigned int parent, lchild, rchild;
}HTNode,*HuffmanTree;//动态分配数组存储赫夫曼树

typedef char** HuffmanCode;//动态分配数组存储赫夫曼编码表

void Select(HuffmanTree HT, int n, int* s1, int* s2);
void HuffmanCoding(HuffmanTree* HT, HuffmanCode* HC, int* w, int n, char* str);
void Output_HuffmanCode(HuffmanCode HC, int n);
void Decoding(HuffmanTree HT, int m, char* buff);
void ShowHuffmanTree(HuffmanTree HT, int n);
void ShowHuffmanCode(HuffmanTree HT, HuffmanCode* HC, int n);
#endif
```

(2)Huffmantree.c

```
#include "Huffmantree.h"
#include "Status.h"
#include <stdio.h>
#include <malloc.h>

void Select(HuffmanTree HT, int n, int* s1, int* s2)
{
    //找赫夫曼树HT中1~n的parent为0的最小的两个权值的下标（HT的下标从1开始）
    int min1 = 1000, min2 = 1000;
    for (int i = 1; i <= n; i++)
    {
        if (HT[i].parent == 0 && HT[i].weight < min1)
        {
            min1 = HT[i].weight;
```

```
            *s1 = i;
        }//if
    }//for
    for (int i = 1; i <= n; i++)
    {
        if (HT[i].parent == 0 && i != *s1 && HT[i].weight < min2)
        {
            //i!=s1用来排除s1的情况，找除最小值外的第二个最小值
            min2 = HT[i].weight;
            *s2 = i;
        }//if
    }//for
}

void HuffmanCoding(HuffmanTree* HT, HuffmanCode* HC, int* w, int n, char* str)
{
    //w存放n个字符权值，构造赫夫曼树HT，求出n个字符的编码HC
    if (n <= 1)
        return;
    int m = 2 * n - 1;//树中结点个数
    *HT = (HuffmanTree)malloc((m + 1) * sizeof(HTNode));
    int i;
    for (i = 1; i <= n; i++)
    {
        //初始叶子，从1号存储
        (*HT)[i].weight = w[i - 1];
        (*HT)[i].parent = (*HT)[i].lchild = (*HT)[i].rchild = 0;
        (*HT)[i].ch = str[i - 1];
    }//for
    for (; i <= m; i++)
    {
        //初始化非叶子，m=2*n-1
        (*HT)[i].ch = '\0';
        (*HT)[i].parent = (*HT)[i].lchild = (*HT)[i].rchild = 0;
    }//for
    //至此赫夫曼树完成初始化
    int s1, s2;
    for (i = n + 1; i <= m; i++)
    {
        //构造赫夫曼树
        //从HT[1..i-1]中选择parent为0且weight最小的两个结点，其序号记为s1和s2
        Select((*HT), i - 1, &s1, &s2);
        //生成存储在第i个位置的新结点
        (*HT)[s1].parent = i;
```

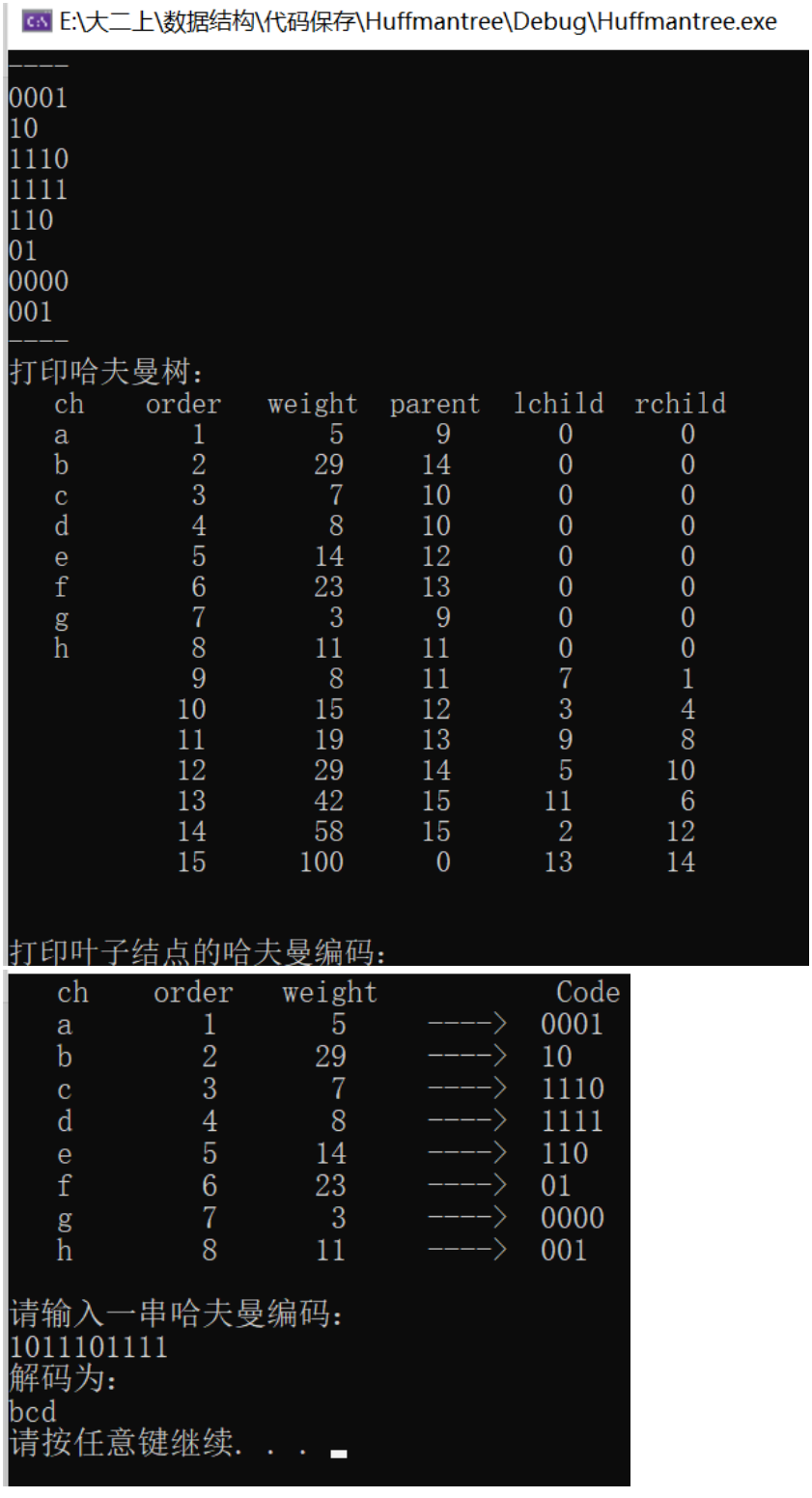
```
        (*HT)[s2].parent = i;
        (*HT)[i].lchild = s1;
        (*HT)[i].rchild = s2;
        (*HT)[i].weight = (*HT)[s1].weight + (*HT)[s2].weight;
    }//for

    //从叶子结点到根逆向求每个字符的赫尔曼编码
    *HC = (HuffmanCode)malloc((n + 1) * sizeof(char*));
    //分配n个字符编码的头指针向量
    char* cd;
    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
    int c, f, start;
    for (i = 1; i <= n; i++)
    {
        //逐个字符编码
        start = n - 1;//编码当前编制0/1的位置
        //从叶子到根逆向求编码
        for (c = i, f = (*HT)[i].parent; f != 0; c = f, f = (*HT)[f].parent)
            if ((*HT)[f].lchild == c)
                cd[--start] = '0';
            else
                cd[--start] = '1';
        (*HC)[i] = (char*)malloc((n - start) * sizeof(char));
        strcpy((*HC)[i], &cd[start]);
    }//for
    free(cd);
}

void Output_HuffmanCode(HuffmanCode HC, int n)//HC下标从1开始
{
    printf("----\n");
    for (int i = 1; i <= n; i++)
    {
        for (int j = 0; HC[i][j] != '\0'; j++)
            printf("%d", HC[i][j]-48);
        printf("\n");
    }
    printf("----\n");
}

void Decoding(HuffmanTree HT, int m, char* buff)
{
```

```
    //将二进制编码buff翻译回信息原文，m是树根的编号
    int p = m;
    while (*buff != '\0' && p != 0)
    {
        if ((*buff) == '0')
            p = HT[p].lchild; //进入左分支
        else
            p = HT[p].rchild; //进入右分支
        buff++;
        if (!HT[p].lchild && !HT[p].rchild)
        {
            //进入叶子结点
            printf("%c", HT[p].ch);
            p = m; //重新从树根出发进行译码
        }//if
    }//while
}

void ShowHuffmanTree(HuffmanTree HT, int n)
{
    int i;
    printf("   ch    order   weight  parent  lchild  rchild \n");
    for (i = 1; i <= n; i++)
        printf("   %c       %2d      %3d     %2d      %2d      %2d   \n",
            HT[i].ch, i, HT[i].weight, HT[i].parent, HT[i].lchild, HT[i].rchild);
    printf("\n");

}

void ShowHuffmanCode(HuffmanTree HT, HuffmanCode* HC, int n)
{
    int i;
    printf("   ch    order   weight          Code  \n");
    for (i = 1; i <= n; i++)
        printf("   %c       %2d      %2d      ---->  %-8s\n",
            HT[i].ch, i, HT[i].weight, HC[i]);
    printf("\n");
}
```

(3)test.c

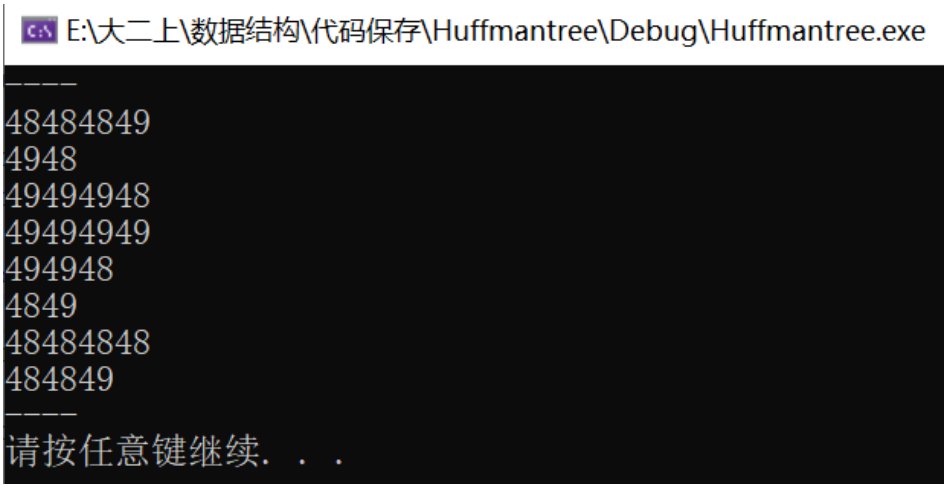
```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include "Huffmantree.h"
```

```
int main()
{
    HuffmanTree HT;
    HuffmanCode HC;
    int n = 8;
    int a[8] = { 5,29,7,8,14,23,3,11 };//8个字符的权值
    char str[8] = "abcdefgh";
    //char str[30] = "a   b   c   d   e   f   g   h";

    HuffmanCoding(&HT, &HC, a, n, str);
    Output_HuffmanCode(HC, n);

    printf("打印哈夫曼树：\n");
    ShowHuffmanTree(HT, 2 * n - 1);
    printf("\n打印叶子结点的哈夫曼编码：\n");
    ShowHuffmanCode(HT, HC, n);

    printf("请输入一串哈夫曼编码：\n");
    char buff[50];
    scanf("%s", buff);
    printf("解码为：\n");
    Decoding(HT, 2 * n - 1, buff);
    printf("\n");

    system("pause");
    return 0;
}
```


E:\大二上\数据结构\代码保存\Huffmantree\Debug\Huffmantree.exe
0001
10
1110
1111
110
01
0000
001
打印哈夫曼树：
ch order weight parent lchild rchild
a 1 5 9 0 0
b 2 29 14 0 0
c 3 7 10 0 0
d 4 8 10 0 0
e 5 14 12 0 0
f 6 23 13 0 0
g 7 3 9 0 0
h 8 11 11 0 0
9 8 11 7 1
10 15 12 3 4
11 19 13 9 8
12 29 14 5 10
13 42 15 11 6
14 58 15 2 12
15 100 0 13 14
打印叶子结点的哈夫曼编码：
ch order weight Code
a 1 5 ----> 0001
b 2 29 ----> 10
c 3 7 ----> 1110
d 4 8 ----> 1111
e 5 14 ----> 110
f 6 23 ----> 01
g 7 3 ----> 0000
h 8 11 ----> 001
请输入一串哈夫曼编码：
1011101111
解码为：
bcd
请按任意键继续. . .

五、实验总结

1. 出现错误



修改后

```
printf("%d", HC[i][j]-48);
```

正常输出

2. **出现错误**

无法解码

```
while (*buff != '0')
  {
      if ((*buff) == '0')
          p = HT[p].lchild;//进入左分支
      else
          p = HT[p].rchild;//进入右分支
      buff++;
      if (!HT[p].lchild && !HT[p].rchild)
      {
          //进入叶子结点
          printf("%c", HT[p].ch);
```

```
            p = m;//重新从树根出发进行译码
        }
            }
```

发现加粗位置写错

更改为：'\0'

3. 出现错误

```
void HuffmanCoding(HuffmanTree* HT, HuffmanCode* HC, int* w, int n, char* str)
```

对字符串理解出现错误，传入字符串名称代表传入字符串首地址，不需要再传入地址

4. 查询函数 HT 不需要传入地址，因为不对 HT 进行修改