树结构及其应用

1. 需求分析

树状图是一种数据结构，它是由 n（n>=1）个有限节点组成一个具有层次关系的集合。把它叫做“树”是因为它看起来像一棵倒挂的树，也就是说它是根朝上，而叶朝下的。

(1) 输入的形式和输入值的范围：int 整型

(2) 输出的形式：int 整型

(3) 程序所能达到的功能：从键盘接受输入(先序),以二叉链表作为存储结构,建立二叉树(以先序来建立),并采用递归算法对其进行遍历(先序、中序和后序),将遍历结果打印输出。除此之外，实现建立赫夫曼树，赫夫曼编码，赫夫曼译码。

(4) 测试数据：ABC□□DE□G□□F□□□(其中□表示空格字符),则输出结果为:先序为 ABCDEGF,中序为 CBEGDFA,后序为 CGBFDBA。

2. 概要设计

(1)二叉树的基本操作：

void InitTree(Tree &T)、void input(Tree &T)

操作结果：构造了一个二叉树，按顺序输入节点数据，如果结点不存在输入空格。

void preOrderTraverse(Tree T)

初始条件：二叉树已存

操作结果：先序遍历二叉树

void inOrderTraverse(Tree T)

初始条件：二叉树已存

操作结果：中序遍历二叉树

void postOrderTraverse(Tree T)

初始条件：二叉树已存

操作结果：后序遍历二叉树

(2)二叉树简单测试

main()

操作结果：构建二叉树，依次实现先序、中序、后序遍历二叉树。

(3) 赫夫曼树、赫夫曼编码、赫夫曼译码

void select(Tree T,int end,int &s1,int &s2)

初始条件：二叉树已存在。

操作结果：从已生成的树中选择出父母为零且权重最小和次小的两个

结点，返回他们的序号

void Huffmantree(int sum,Tree &T)

初始条件：二叉树已存在。

操作结果:用静态链表实现赫夫曼树,并将信息存入文件HfmTree.txt。

void HuffmanCoding(int sum,Tree T,HuffmanCode &HC)

初始条件：赫夫曼树已存在。

操作结果：对赫夫曼树实现赫夫曼编码，并将译码结果存入文件CodeFile.txt。

void Decode(int sum,Tree T,HuffmanCode HC)

初始条件：赫夫曼树已存在。

操作结果:对赫夫曼编码译码,并将译码结果存入文件TextFile.txt。

void PrintCode(int num)

初始条件：赫夫曼树已存在。

操作结果：将赫夫曼译码结果从文件 TextFile.txt 提取出来，并在控制台打印。

（4）赫夫曼编码数据测试

main()

操作结果：构建赫夫曼树实现赫夫曼编码、译码

3. 详细设计

（1）二叉树基本内容———头文件

```
#include <iostream>

using namespace std;



typedef struct Node
{
    char data;
    Node *lchild;
    Node *rchild;
}Node,*Tree;



void input(Tree &T)
{
     char data;
    data=getchar();
    if(data == ' ')
    {
        T = NULL;
        return ;
    }
    else{
        Node *p = (Node *)malloc(sizeof(Node));
```

```
        T = p;
        p->data=data;
        input(T->lchild);
        input(T->rchild);
}


}



void InitTree(Tree &T)
{

    cout<<"请按顺序输入节点数据，如果结点不存在请输入空格"<<endl;

      input(T);


    if(!T)cout<<"    树 为 空 ！"<<endl;




}



void preOrderTraverse(Tree T)//先序遍历
    {
        if(T)
        {
            putchar(T->data);
            preOrderTraverse(T->lchild);
            preOrderTraverse(T->rchild);
                }
    }

void inOrderTraverse(Tree T)//中序遍历
{
    if(T)
```

```
    {
        inOrderTraverse(T->lchild);
        putchar(T->data);
        inOrderTraverse(T->rchild);
    }
}


void postOrderTraverse(Tree T)//后序遍历
{
    if(T)
    {
        postOrderTraverse(T->lchild);
        postOrderTraverse(T->rchild);
        putchar(T->data);

    }
}
```

（2）二叉树测试———可执行 cpp 文件

```
#include <iostream>
#include "Tree.h"
using namespace std;

int main() {


    Tree T;



    InitTree(T);
    cout<<endl;
    cout<<"先序遍历二叉树：";
    preOrderTraverse(T);
   cout<<endl;
    cout<<"中序遍历二叉树：";
    inOrderTraverse(T);
    cout<<endl;
```

```
    cout<<"后序遍历二叉树：";
    postOrderTraverse(T);
    cout<<endl;



    return 0;
}
```

（3）赫夫曼树的生成、编码以及译码———头文件

```
#include <iostream>
#include <fstream>
#include <string>
using std::string;
using namespace std;



typedef struct{

    char code;
    int weight;
    int parent,lchild,rchild;



}Node,*Tree;


typedef char ** HuffmanCode;


//从已生成的树中选择出父母为零且权重最小和次小的两个结点，返回他们的序号
void select(Tree T,int end,int &s1,int &s2)
{
    int min=65535;
    int min_no = -1;
    int second_min_no=-1;
    for(int i=0;i<end;i++)//选择最小值
    {
        if(T[i].weight<min && T[i].parent==0)
        {
           min=T[i].weight;
```

```
            min_no=i;
        }

    }


    int second_min = 65535;

    for(int i=0;i<end;i++)//选择最小值
    {
        if(T[i].weight<second_min && i!=min_no && T[i].parent==0)
        {
           second_min=T[i].weight;
            second_min_no=i;
        }

    }


    s1=min_no;
    s2=second_min_no;


}




void Huffmantree(int sum,Tree &T)
{
    ofstream outfile("HfmTree.txt",ios::out);

    int m = 2*sum-1;

    T =(Node*)malloc((2*sum-1)*sizeof(double));
    int i ;
    for( i=0;i<sum;i++)//对原始编码初始化，每个编码为一棵树
    {
        cout<<"请输入第"<<i+1<<"个编码：";
        cin>>T[i].code;
        cout<<endl;
        cout<<"请输入第"<<i+1<<"个编码的权重：";
        cin>>T[i].weight;
```

```
        cout<<endl;
        T[i].parent=0;
        T[i].lchild=0;
        T[i].rchild=0;
        outfile<<T[i].code<<"   "<<T[i].weight<<"   "<<T[i].lchild<<"
"<<T[i].rchild<<" "<<T[i].parent<<endl;
    }

    for(;i<m;i++)//对树中其他结点初始化
    {
        T[i].weight=0;
        T[i].parent=0;
        T[i].lchild=0;
        T[i].rchild=0;
    }
    for(int j=sum;j<m;j++)
    {
        int s1;
        int s2;

        select(T, j, s1, s2);

        T[j].lchild = s1;
        T[j].rchild = s2;
        T[s1].parent = j;
        T[s2].parent = j;
        T[j].weight = T[s1].weight+T[s2].weight;
        outfile<<T[j].code<<"   "<<T[j].weight<<"   "<<T[j].lchild<<"
"<<T[j].rchild<<" "<<T[j].parent<<endl;

    }


    outfile.close();


}




void HuffmanCoding(int sum,Tree T,HuffmanCode &HC)//Huffman 译码
{
    ofstream outfile("CodeFile.txt",ios::out);
```

```
    HC = (HuffmanCode)malloc(sum*sizeof(char*));
    char * cd = (char*)malloc(sum*sizeof(char));
    cd[sum-1]='\0';



    for(int i=0;i<sum;i++)
    {
        int j,k;
        int start = sum-1;//方便逆向记录下编码
        for(j=i,k=T[i].parent;k!=0;j=k,k=T[j].parent)//从叶子到根逆向
求编码
        {
            if(T[k].lchild==j)cd[--start]='0';
            else if(T[k].rchild==j)cd[--start]='1';
        }
        HC[i]=(char*)malloc((sum-start)*sizeof(char));
        strcpy(HC[i],&cd[start]);
    }

    cout<<endl;
    for(int i=0;i<sum;i++)
    {cout<<T[i].code<<"---->"<<HC[i]<<endl;
        outfile<<T[i].code<<"---->"<<HC[i]<<endl;}


     outfile.close();


}



int Decode(int sum,Tree T,HuffmanCode HC)
{
    int num=0;
    ofstream outfile("TextFile.txt",ios::out);
    cout<<"请依次输入字符：(输入$表示终止)";
    char c;
    cin>>c;
    while(c!='$')
    { num++;
        for(int i=0;i<sum;i++)
```

```
            if(T[i].code==c)outfile<<HC[i]<<" ";

        cin>>c;
    }

    return num;

}


void PrintCode(int num)
{

    ifstream infile("TextFile.txt");
    string str;
    for(int i=0;i<num;i++)
    {
       infile>>str;
        cout<<str;

    }


}
```

（4）赫夫曼编码测试

```
#include <iostream>
#include "HuffamnTree.h"

using namespace std;

int main() {
    int sum,num;
    Tree T;
    HuffmanCode HC;

    while(true)
```

```
    {   cout<<endl;
        cout<<"=======<<"HuffmanTree"<<"=======<<endl;
        cout<<"         1.初始化"<<endl<<endl;
        cout<<"         2.Huffman 编码"<<endl<<endl;
        cout<<"         3.Huffman 译码"<<endl<<endl;
        cout<<"         4.打印译码结果"<<endl<<endl;
        cout<<"         5.退出"<<endl<<endl;
        cout<<"=======<<"=======<<"============<<endl;
        int i;
        cin>>i;
        switch (i) {
            case 1:
                cout<<"请输入你总共要输入的编码数量：";

                cin>>sum;
                cout<<endl;
                Huffmantree(sum, T);
                break;
            case 2: HuffmanCoding(sum, T, HC);
                break;

            case 3:num=Decode(sum,T,HC);
                break;
            case 4:PrintCode(num);

                break;

            case 5:exit(1);

            default:continue;

        }


    }

    return 0;

}
```

4. 调试分析

1、从键盘输入 ABC□□DE□G□□F□□□(其中□表示空格字符)，

```
请按顺序输入节点数据，如果结点不存在请输入空格
ABC   DE G   F
```

2、先序、中序、后序遍历输出：

```
请按顺序输入节点数据，如果结点不存在请输入空格
ABC   DE G   F

先序遍历二叉树：ABCDEGF
中序遍历二叉树：CBEGDFA
后序遍历二叉树：CGEFDBA
```

测试完毕，二叉树可正常使用。

<1>调试过程中遇到的问题：

第一次运行的时候，发现空格无法输入，才想起 cin 不支持空格输入，于是改成了 getchar 函数，这样子就能成功输入空格了。

<2>时间和空间分析；

先序、中序、后序遍历时间相同，均为 0（n）

二叉树占用空间不定，为每个结点所占用空间之和。

<3>改进设想：

考虑到有些高级语言不支持指针（例如 Java），可以考虑用静态链表实现二叉树，此功能在后面 HuffmanTree 中实现。

<4>经验、心得和体会：

第一次二叉树这种有层次感的存储结构，不单单是用脑子来思考了，还得在草稿纸上画出基本构架，试了后发现很多时候都得这样打个草稿，这样思路更佳明确。

5. 使用说明

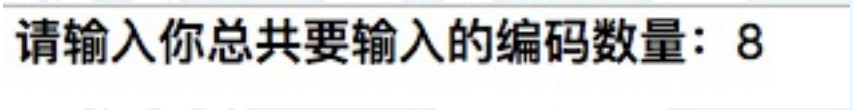
1、输入所需编码的总数。

2、依次输入每个编码的信息和权重。

6. 测试程序的运行结果

（1）主界面：

（2）选择 1，输入所需编码的总数：



（3）依次输入每个编码的信息和权重。

请输入第1个编码：1

请输入第1个编码的权重：5

请输入第2个编码：2

请输入第2个编码的权重：29

请输入第3个编码：3

请输入第3个编码的权重：7

请输入第4个编码：4

请输入第4个编码的权重：8

请输入第5个编码：5

请输入第5个编码的权重：14

请输入第6个编码：6

请输入第6个编码的权重：23

请输入第7个编码：7

请输入第7个编码的权重：3

请输入第8个编码：8

请输入第8个编码的权重：11

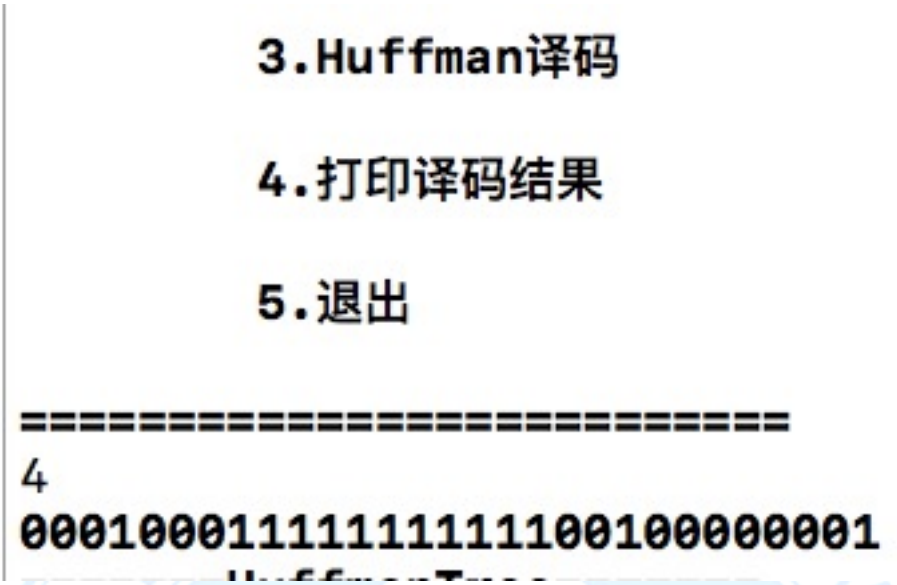
（4）选择 2，实现编码

```
          4.打印译码结果

          5.退出

============================
2

A----->0001
B----->10
C----->1110
D----->1111
E----->110
F----->01
G----->0000
H----->001
```

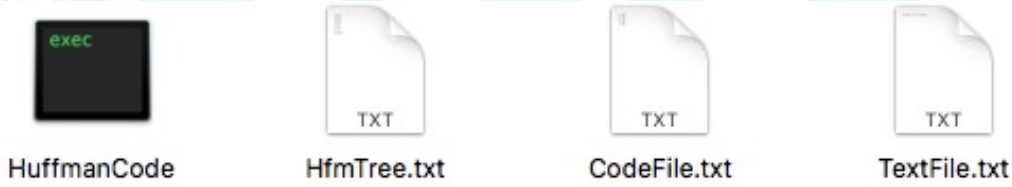
（5）选择 3，输入一段字符，实现译码：

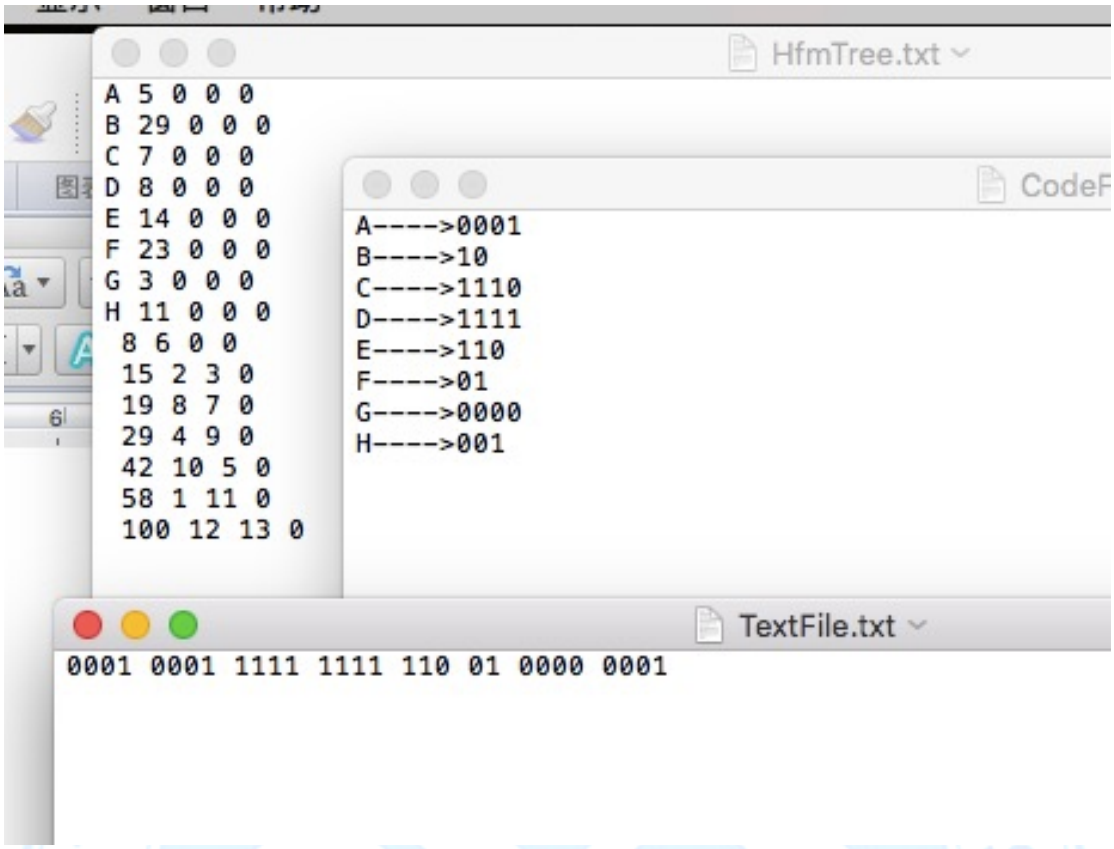
```
============================
3
请依次输入字符：（输入$表示终止）a
a
d
d
e
f
g
a
$
```

（6）选择 4，显示译码结果



（7）文件以及文件内的信息





7. 心得体会

跟以前的实验比起来，这次可以算是最难的了，也覆盖了很多以前没学过的 C 语言知识，这个实验编写了很久，包括从各种书上参考的思路的代码，最后经过不断的调试，终于成功实现了赫夫曼编码，这次实验对我来说是一次不小的挑战，树比较难写，每个节点都有两个分支，如果遍历的话经常需要走到底然后返回到原来分叉的地方去走另一条路，但是用递归写遍历非常简单，这就是递归的妙处吧，总是能化复杂为简单。

附录：源程序文件清单

各程序源代码文件随本实验报告电子版一起打包，存放在 bin 子目录。

文件清单如下：

Tree.h……基本二叉树定义及基本操作，头文件

HuffmanTree.h……赫夫曼树、赫夫曼编码、赫夫曼译码的定义，头文件

Test1.cpp ……二叉树三种遍历测试

Test2.cpp……赫夫曼树测试文件

HfmTree.txt……保存赫夫曼树

CodeFile.txt……保存赫夫曼编码结果

TextFile.txt……保存赫夫曼译码结果