

数据结构课程设计报告

题目:哈夫曼编码

学生姓名： 聂楠

指导老师： 高永平

日期：2016/12/22

## 问题描述

哈夫曼编码也称前缀编码，它是根据每个字符出现的频率而进行编码的，要求任一字符的编码都不是其它任意字符编码的前缀且字符编码的总长度为最短。它主要应用于通信及数据的传送以及对信息的压缩处理等方面。哈夫曼编码的基础是依据字符出现的频率值而构造一棵哈夫曼树，从而实现最短的编码表示最常用的数据块或出现频率最高的数据。

某个信息所含的五种字符的出现次数分别为:

a －16，b －7，c －6，d －6，e －5．

## 问题分析

1.取根节点权值最小的两棵树作为左右子树，构造一个新的二叉树，置新二叉树根节点权值为左右子树权值之和

2.在森林中删除这两棵树并加入新构造的二叉树

3.重复上述两步直到只剩一棵树，这棵树即为哈夫曼树

4.对哈夫曼树进行编码

## 相关函数与结构

typedef struct node//节点

{

char ch;//字符

char bit[30];//编码数组

node\* left;//左孩子

node\* right;//右孩子

node\* parent;//父节点

int value;//标志是否有父节点，有为0，无为-1

double weight;//权值

}node,\*lpnode;

typedef struct hfm//哈夫曼树

{

node data[m];

int front;//指向第一个节点

int rear;//指向最后一个节点的后一位

}hfm;

Void Init（hfm& myhfm）用于初始化各节点

Void select(hfm& myhfm)选择两棵权值最小的二叉树作为左右子树，并构造一棵新二叉树，置新二叉树根节点权值为左右子树之和，在森林中删除这两棵树（并不真正删除，只是置相关节点的value值为-1）并加入新构造的二叉树

Void code(hfm& myhfm)对哈夫曼树进行编码

Void preorder(node\* root)先序遍历

Void midorder(node\* root)中序遍历

Void backorder(node\* root)后序遍历

Node\* getroot(hfm& myhfm)获取哈夫曼树的根节点

void threeorder(node\* root)输出三次遍历结果

void mainprint()主界面

## 实验代码：



#include<iostream>

#include<stdio.h>

using namespace std;

#define n 7

#define m 2\*n-1

typedef struct node

{

char ch;//字符

char bit[30];//编码数组

node\* left;//左孩子

node\* right;//右孩子

node\* parent;//父节点

int value;//标志是否有父节点，有为0，无为-1

double weight;//权值

}node,\*lpnode;

typedef struct hfm

{

node data[m];

int front;//指向第一个节点

int rear;//指向最后一个节点的后一位

}hfm;

void init(hfm& myhfm)//初始化

{

myhfm.front=myhfm.rear=0;

for(int i=0;i<n;i++)

{

myhfm.data[i].left=NULL;

myhfm.data[i].right =NULL;

myhfm.data[i].parent=NULL;

myhfm.data[i].value =-1;//value都为-1，表示都可以参与比较

cout<<"输入字符:";

cin>>myhfm.data[i].ch;

cout<<"输入权值:";

cin>>myhfm.data[i].weight ;

myhfm.rear++;

}

}

void select(hfm& myhfm)//选择最小值

{

double min1,min2;//min1,min2存储两个最小值

int temp1,temp2;//temp1,temp2存储两个最小值的下标

min1=min2=1000;

for(int i=myhfm.front;i<myhfm.rear ;i++)//选择第一个最小值

{

if(myhfm.data[i].value==0)continue;

if(min1>myhfm.data[i].weight )

{

min1=myhfm.data[i].weight ;

temp1=i;

}

else

{

min1=min1;

temp1=temp1;

}

}

myhfm.data[temp1].value=0;//下次不参与比较

for(i=myhfm.front;i<myhfm.rear ;i++)//选择第二个最小值

{

if(myhfm.data[i].value==0)continue;

if(min2>myhfm.data[i].weight )

{

min2=myhfm.data[i].weight ;

temp2=i;

}

else

{

min2=min2;

temp2=temp2;

}

}

myhfm.data[temp2].value=0;//下次不参与比较

//构造父节点并分别指向相应节点

myhfm.data[myhfm.rear].weight =min1+min2;//父节点的权值等于两最小值之和

//父节点指向子节点

myhfm.data[myhfm.rear].left=&myhfm.data[temp1];

myhfm.data[myhfm.rear].right=&myhfm.data[temp2];

//子节点指向父节点

myhfm.data[temp1].parent=&myhfm.data[myhfm.rear];

myhfm.data[temp2].parent=&myhfm.data[myhfm.rear];

myhfm.data[myhfm.rear].value =-1;//父节点的value赋值，表示下次可参与比较

myhfm.rear++;

}

void code(hfm& myhfm)//编码

{

lpnode p,q;

for(int i=0;i<n;i++)

{

p=&myhfm.data[i];//p指向子节点，q指向父节点，如果q的左孩子等于p,则编码为0，否则为1

int j=0;

do

{

q=p->parent ;

if(q->left==p)

{

myhfm.data[i].bit[j++]='0';

}

else

{

myhfm.data[i].bit[j++]='1';

}

p=q;

}while(q->value==0);

myhfm.data[i].bit[j]='\0';

//输出已编码好的第一个节点

cout<<"字符"<<myhfm.data[i].ch<<"的权值为:"<<myhfm.data[i].weight<<"编码为:";

for(j=j-1;j>=0;j--)

{

cout<<myhfm.data[i].bit[j];

}

cout<<endl;

}

}

node\* getroot(hfm& myhfm)//获取哈夫曼树根节点

{

int index=0;

for(int i=0;i<m;i++)

{

if(myhfm.data[i].value==-1)

{

index=i;

break;

}

}

node\* root;

root=&myhfm.data[index];

return root;

}

void preorder(node\* root)//先序遍历

{

if(root!=NULL)

{

printf("%6.2f",root->weight);

preorder(root->left);

preorder(root->right);

}

}

void midorder(node\* root)//中序遍历

{

if(root!=NULL)

{

midorder(root->left);

printf("%6.2f",root->weight);

midorder(root->right);

}

}

void backorder(node\* root)//后序遍历

{

if(root!=NULL)

{

backorder(root->left);

backorder(root->right);

printf("%6.2f",root->weight);

}

}

void threeorder(node\* root)

{

cout<<"先序遍历:"<<endl;

preorder(root);

cout<<endl;

cout<<"中序遍历:"<<endl;

midorder(root);

cout<<endl;

cout<<"后序遍历:"<<endl;

backorder(root);

cout<<endl;

}

void show()

{

cout<<" 哈夫曼编码 "<<endl;

cout<<" 1.输入权值与字符 2.遍历哈夫曼树"<<endl;

cout<<" 3.输出哈夫曼编码 4.退出 "<<endl;

}

void mainprint()

{

hfm myhfm;

char t;

while(1)

{

show();

fflush(stdin);

cout<<"输入您的选择:";

cin>>t;

system("cls");

switch(t)

{

case '1':

{

cout<<" 1.输入权值与字符 "<<endl;

init(myhfm);//初始化

for(int i=0;i<n-1;i++)//构造哈夫曼树

{

select(myhfm);

}

break;

}

case '2':

{

cout<<" 2.遍历哈夫曼树 "<<endl;

lpnode root;

root=getroot(myhfm);//获取哈夫曼树的根节点

threeorder(root);//三种遍历

break;

}

case '3':

{

cout<<" 3.输出编码 "<<endl;

code(myhfm);//对哈夫曼树编码

break;

}

case '4':

{

return;

}

}

}

}

int main()

{

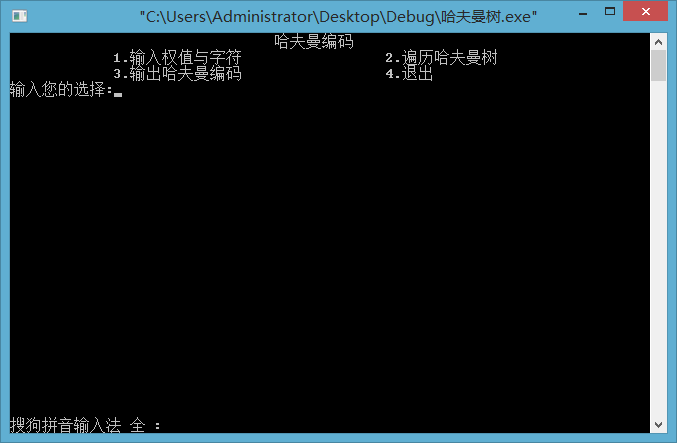
mainprint();

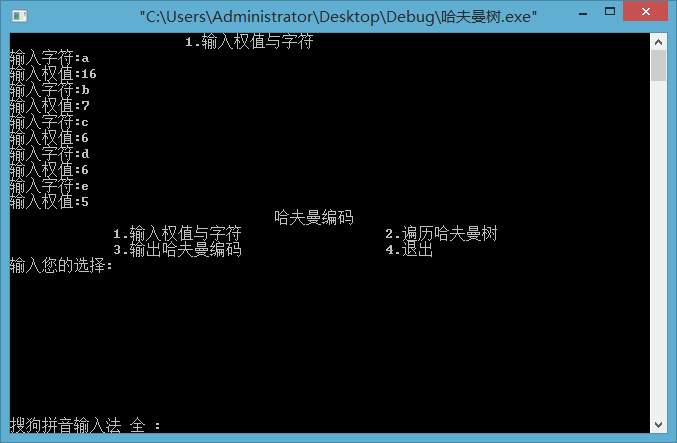
system("pause");

return 0;

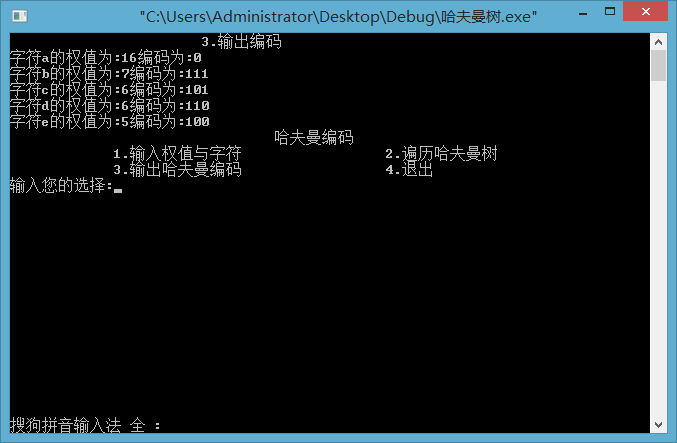
}

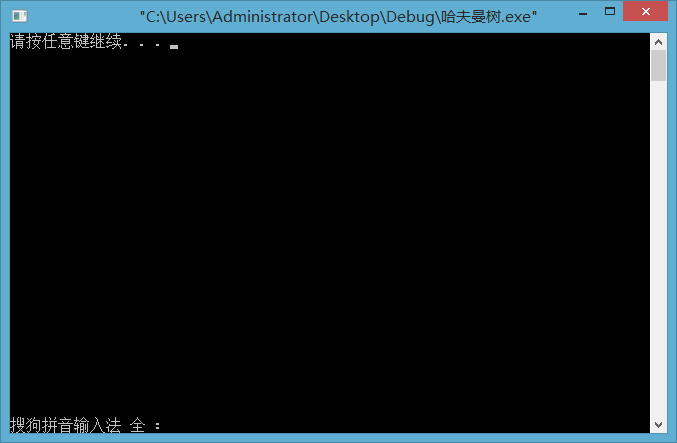
## 实验结果











## 实验小结

通过这次实验，将本学期所学的树，查找等相关知识应用到实训项目中，在算法实现的过程中，通过反复推敲，查找网络、书本知识逐步解决遇到的问题。进一步巩固了二叉树等相关知识。在建立哈夫曼树的算法实现过程中，结合书本知识，反复分析哈夫曼节点，哈夫曼树之间的关系，节点与节点之间的关系。锻炼自身在处理二叉树数据结构问题的能力。