实习报告

题目:哈夫曼树。

班级：电信9班 姓名：王嘉成 学号：U201813481 日期：2019.04.22

**需求分析。**

1. 本次程序中，用户需要给出一个字符集及其使用频度，然后程序中的InitHfmTree函数会根据该字符集和频度创建出相应的哈夫曼树，之后的initHfmCode函数会将每个字符转化为不同的二进制码，且每个二进制码都不为其他字符二进制码的前缀。之后用户输入一串字符，Encoding函数将其能转化为一串二进制码，再由Decoding函数解码，可以将一串二进制码解码成一串字符串。
2. 测试数据：

用户输入字符集及频度：

{' ',186},{'A',64},{'B',13},{'C',22},{'D',32},{'E',103},{'F',21},{'G',15},{'H',47},

{'I',57},{'J',1},{'K',5},{'L',32},{'M',20},{'N',57},{'O',63},{'P',15},{'Q',1},{'R',48},{'S',51},

{'T',80},{'U',23},{'V',8},{'W',18},{'X',1},{'Y',16},{'Z',1}

字符编码结果： 111

A1011

B100000

C000000

D10101

E0010

F110011

G100010

H00001

I00111

J1100001001

K11000011

L10100

M110010

N00110

O1001

P100001

Q1100001000

R00010

S00011

T1101

U000001

V1100000

W110001

X1100001011

Y100011

Z1100001010

用户输入字符集：THIS PROGRAM IS MY FAVORITE;

程序输出二进制码：11010000100111000111111000010001010011000100001010111100101110011100011111110010100011111110011101111000001001000100011111010010

程序解码字符串：THIS PROGRAM IS MY FAVORITE。

**概要设计。**

1. 本程序抽象数据类型有树以及一个顺序表，但此程序中只是用顺序表存放字符和频度，故其抽象数据类型就不赘述了。

typedef int Elemtype ;

ADT BinaryTree{

数据对象:D=D是具有相同特性的数据元素集和。

数据关系R：

若D为空集，则R也为空集，此时为空二叉树；

若D不为空集，则R为{H}，H关系如下：

1. D中有一称为根的数据元素，其无前驱
2. 若D-{root}不为空，则存在D-{root}={Dl,Dr},且Dl与Dr没有交集；
3. （Dl，{H}）是一颗符合题目的二叉树，称为左子树，（Dr，{H}）是一颗符合题目的二叉树，称为右子树；

基本操作：

InitBiTree(&T)

操作结果：构造一个空二叉树T。

DestroyBiTree(&T)

初始条件：二叉树T存在。

操作结果：将树摧毁。

BiTreeEmpty(&T)

初始条件：二叉树T存在

操作结果：判断二叉树T是否为空，为空，返回0，不为空，返回1。

本程序中用到二叉树的基本操作很少，故在此不赘述多余的基本操作。

}ADT BinaryTree

**详细设计。**

以下使是该程序的完整代码：

#include <stdio.h>

#include <stdlib.h>

#include<string.h>

typedef char \*\* HfmCode;

typedef struct HfmTree//哈夫曼树的结构体，由于其需要知道双亲和左右孩子的位置，故设置parents，lchild ，rchild三个值记录该信息

{

char ch;

int weight;

int parents;

int lchild;

int rchild;

}HfmTree,\*pHfmTree;

typedef struct list//该顺序表存字符和频度

{

char ch;

int weight;

}list,\*plist;

pHfmTree InitHfmTree(pHfmTree Ht,plist c,int n)//创建相应的哈夫曼树

{

Ht=(pHfmTree)malloc((2\*n)\*sizeof(HfmTree));

int i,j,k,m,flag=0;

for(i=0;i<n;i++)//将前n个数组元素初始化，用来存字符和频度信息

{

Ht[i].ch=c[i].ch;

Ht[i].weight=c[i].weight;

Ht[i].parents=0;

Ht[i].lchild=0;

Ht[i].rchild=0;

}

for(i=n;i<2\*n-1;i++)//将后面的数组元素也初始化，当开始创建哈夫曼树时，用来作为双亲元素

{

Ht[i].ch='&';

Ht[i].weight=0;

Ht[i].parents=0;

Ht[i].lchild=0;

Ht[i].rchild=0;

}

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

{

for(j=0;j<n+i;j++)//找到还未建入哈夫曼树的元素，用来作为比较的数，来找出下一个建入哈夫曼树的元素

{

if(Ht[j].parents==0)

{

k=j;

flag=1;

break;

}

}

for(j=0;j<n+i;j++)//先找一个最小的数

{

if(Ht[j].parents==0)

{

if(Ht[j].weight<=Ht[k].weight)

{

k=j;

}

}

}

Ht[k].parents=n+i;//找到一个最小的数就给其parent值赋值，以免影响第二个最小值的查找

for(j=0;j<n+i;j++)// 再到还未建入哈夫曼树的元素，用来作为比较的数，来找出下第二个建入哈夫曼树的元素

{

if(Ht[j].parents==0)

{

m=j;

flag=1;

break;

}

}

for(j=0;j<n+i;j++)//查找第二个最小值

{

if(Ht[j].parents==0)

{

if(Ht[j].weight<=Ht[m].weight)

{

m=j;

}

}

}

Ht[m].parents=n+i;//将这两个权值最小的元素分作左右子树，连到一个父元素上，此时父元素的权值变为两元素权值之和

Ht[n+i].lchild=k;

Ht[n+i].rchild=m;

Ht[n+i].weight=Ht[k].weight+Ht[m].weight;

}

return Ht;

}

HfmCode initHfmCode(pHfmTree Ht,int n,HfmCode Hc)//将各字符化为相互独立的二进制码

{

Hc=(HfmCode)malloc(n\*sizeof(char\*));

char\* tem=(char\*)malloc((n+1)\*sizeof(char));

int start,i,j,f;

tem[n]='\0';

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

{

start=n-1；由于是从叶子开始向上遍历的，所以从后往前来转换成二进制码，

for(f=i,j=Ht[i].parents;j!=0;f=j,j=Ht[j].parents)//左子树赋0，右子树赋1

{

if(Ht[j].lchild==f)

{

tem[start]='0';

start--;

}

else

{

tem[start]='1';

start--;

}

}

tem[start--]=Ht[i].ch;

Hc[i]=(char\*)malloc((n-start)\*sizeof(char));

strcpy(Hc[i],tem+start+1);//将转换结果存入Hc结构体中

puts(Hc[i]);

}

free(tem);

return Hc;

}

char\* Encoding(char\* ToBeTran,HfmCode Hc,char\* code,int n)//将字符串按编码编码成二进制码

{

int i,j,length;

length=strlen(ToBeTran);

code=(char\*)malloc(n\*length\*sizeof(char));

for(i=0;ToBeTran[i]!='\0';i++)//按字符串中每个字符，再Hc表中找出对应的编码，赋给code即可

{

for(j=0;j<n;j++)

{

if(Hc[j][0]==ToBeTran[i])

{

strcat(code,Hc[j]+1);

break;

}

}

}

return code;

}

char\* Decoding(char\* code,pHfmTree Ht,int n,char\* ToBeTran,int length)//将编码解码

{

int i,j,k,m;

m=0;

j=2\*n-1;

char ch;

ToBeTran=(char\*)malloc(length\*sizeof(char));

for(i=0;code[i]!='\0';i++)//遇0进左子树，遇1进右子树，直到到叶子，读出该叶子所存的字符，即为该二进制码对应的字符

{

if(code[i]=='0')

{

k=j;

j=Ht[j].lchild;

}

else

{

k=j;

j=Ht[j].rchild;

}

if(Ht[k].lchild==0&&Ht[k].rchild==0)

{

ch=Ht[k].ch;

ToBeTran[m]=ch;

m++;

j=2\*n-1;

i--;

}

}

return ToBeTran;

}

int main()

{

pHfmTree Ht;

HfmCode Hc;

char\* code;

char\* BeTran;

char \*ToBeTran="THIS PROGRAM IS MY FAVORITE";

int n=27;

list c[27]={{' ',186},{'A',64},{'B',13},{'C',22},{'D',32},{'E',103},{'F',21},{'G',15},{'H',47},

{'I',57},{'J',1},{'K',5},{'L',32},{'M',20},{'N',57},{'O',63},{'P',15},{'Q',1},{'R',48},{'S',51},

{'T',80},{'U',23},{'V',8},{'W',18},{'X',1},{'Y',16},{'Z',1}};

Ht=InitHfmTree(Ht,c,n);

Hc=initHfmCode(Ht,n,Hc);

code=Encoding(ToBeTran,Hc,code,n);

puts(code);

BeTran=Decoding(code,Ht,n,BeTran,strlen(ToBeTran));

puts(BeTran);

puts(ToBeTran);

}

**调试分析。**

1. 在建哈夫曼树的时候没有分析清楚对于元素的parent值的判定条件，导致数一直建错，最后通过分析程序的输出，找到错误，成功建成哈夫曼树；
2. 对于转码，它开始的一个字符总为？，而经过分析发现程序应该没错，可能是code初始化时code【0】上有一个之前的残留数据；
3. 算法的时空分析：

InitHfmTree函数时间复杂度为O（n\*n），initHfmCode函数时间复杂度为O（n），Encoding函数时间复杂度为O（n\*n），Decoding函数时间复杂度为O（n），故程序的总时间复杂度为O（n\*n）。

**用户使用说明。**

1. 程序的运行环境为Windos操作系统，执行文件为Huffman.c。
2. 进入程序用户给出一个字符集及其使用频度，然后程序中的InitHfmTree函数会根据该字符集和频度创建出相应的哈夫曼树，之后的initHfmCode函数会将每个字符转化为不同的二进制码，且每个二进制码都不为其他字符二进制码的前缀。之后用户输入一个字符串，Encoding函数将其能转化为一串二进制码，再由Decoding函数解码，可以将一串二进制码解码成一串字符串。

**测试结果。**

用户输入字符集及频度：

{' ',186},{'A',64},{'B',13},{'C',22},{'D',32},{'E',103},{'F',21},{'G',15},{'H',47},

{'I',57},{'J',1},{'K',5},{'L',32},{'M',20},{'N',57},{'O',63},{'P',15},{'Q',1},{'R',48},{'S',51},

{'T',80},{'U',23},{'V',8},{'W',18},{'X',1},{'Y',16},{'Z',1}

字符编码结果： 111

A1011

B100000

C000000

D10101

E0010

F110011

G100010

H00001

I00111

J1100001001

K11000011

L10100

M110010

N00110

O1001

P100001

Q1100001000

R00010

S00011

T1101

U000001

V1100000

W110001

X1100001011

Y100011

Z1100001010

用户输入字符集：THIS PROGRAM IS MY FAVORITE;

程序输出二进制码：11010000100111000111111000010001010011000100001010111100101110011100011111110010100011111110011101111000001001000100011111010010

程序解码字符串：THIS PROGRAM IS MY FAVORITE。

**附录。**

#include <stdio.h>

#include <stdlib.h>

#include<string.h>

typedef char \*\* HfmCode;

typedef struct HfmTree//哈夫曼树的结构体，由于其需要知道双亲和左右孩子的位置，故设置parents，lchild ，rchild三个值记录该信息

{

char ch;

int weight;

int parents;

int lchild;

int rchild;

}HfmTree,\*pHfmTree;

typedef struct list//该顺序表存字符和频度

{

char ch;

int weight;

}list,\*plist;

pHfmTree InitHfmTree(pHfmTree Ht,plist c,int n)//创建相应的哈夫曼树

{

Ht=(pHfmTree)malloc((2\*n)\*sizeof(HfmTree));

int i,j,k,m,flag=0;

for(i=0;i<n;i++)//将前n个数组元素初始化，用来存字符和频度信息

{

Ht[i].ch=c[i].ch;

Ht[i].weight=c[i].weight;

Ht[i].parents=0;

Ht[i].lchild=0;

Ht[i].rchild=0;

}

for(i=n;i<2\*n-1;i++)//将后面的数组元素也初始化，当开始创建哈夫曼树时，用来作为双亲元素

{

Ht[i].ch='&';

Ht[i].weight=0;

Ht[i].parents=0;

Ht[i].lchild=0;

Ht[i].rchild=0;

}

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

{

for(j=0;j<n+i;j++)//找到还未建入哈夫曼树的元素，用来作为比较的数，来找出下一个建入哈夫曼树的元素

{

if(Ht[j].parents==0)

{

k=j;

flag=1;

break;

}

}

for(j=0;j<n+i;j++)//先找一个最小的数

{

if(Ht[j].parents==0)

{

if(Ht[j].weight<=Ht[k].weight)

{

k=j;

}

}

}

Ht[k].parents=n+i;//找到一个最小的数就给其parent值赋值，以免影响第二个最小值的查找

for(j=0;j<n+i;j++)// 再到还未建入哈夫曼树的元素，用来作为比较的数，来找出下第二个建入哈夫曼树的元素

{

if(Ht[j].parents==0)

{

m=j;

flag=1;

break;

}

}

for(j=0;j<n+i;j++)//查找第二个最小值

{

if(Ht[j].parents==0)

{

if(Ht[j].weight<=Ht[m].weight)

{

m=j;

}

}

}

Ht[m].parents=n+i;//将这两个权值最小的元素分作左右子树，连到一个父元素上，此时父元素的权值变为两元素权值之和

Ht[n+i].lchild=k;

Ht[n+i].rchild=m;

Ht[n+i].weight=Ht[k].weight+Ht[m].weight;

}

return Ht;

}

HfmCode initHfmCode(pHfmTree Ht,int n,HfmCode Hc)//将各字符化为相互独立的二进制码

{

Hc=(HfmCode)malloc(n\*sizeof(char\*));

char\* tem=(char\*)malloc((n+1)\*sizeof(char));

int start,i,j,f;

tem[n]='\0';

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

{

start=n-1；由于是从叶子开始向上遍历的，所以从后往前来转换成二进制码，

for(f=i,j=Ht[i].parents;j!=0;f=j,j=Ht[j].parents)//左子树赋0，右子树赋1

{

if(Ht[j].lchild==f)

{

tem[start]='0';

start--;

}

else

{

tem[start]='1';

start--;

}

}

tem[start--]=Ht[i].ch;

Hc[i]=(char\*)malloc((n-start)\*sizeof(char));

strcpy(Hc[i],tem+start+1);//将转换结果存入Hc结构体中

puts(Hc[i]);

}

free(tem);

return Hc;

}

char\* Encoding(char\* ToBeTran,HfmCode Hc,char\* code,int n)//将字符串按编码编码成二进制码

{

int i,j,length;

length=strlen(ToBeTran);

code=(char\*)malloc(n\*length\*sizeof(char));

for(i=0;ToBeTran[i]!='\0';i++)//按字符串中每个字符，再Hc表中找出对应的编码，赋给code即可

{

for(j=0;j<n;j++)

{

if(Hc[j][0]==ToBeTran[i])

{

strcat(code,Hc[j]+1);

break;

}

}

}

return code;

}

char\* Decoding(char\* code,pHfmTree Ht,int n,char\* ToBeTran,int length)//将编码解码

{

int i,j,k,m;

m=0;

j=2\*n-1;

char ch;

ToBeTran=(char\*)malloc(length\*sizeof(char));

for(i=0;code[i]!='\0';i++)//遇0进左子树，遇1进右子树，直到到叶子，读出该叶子所存的字符，即为该二进制码对应的字符

{

if(code[i]=='0')

{

k=j;

j=Ht[j].lchild;

}

else

{

k=j;

j=Ht[j].rchild;

}

if(Ht[k].lchild==0&&Ht[k].rchild==0)

{

ch=Ht[k].ch;

ToBeTran[m]=ch;

m++;

j=2\*n-1;

i--;

}

}

return ToBeTran;

}

int main()

{

pHfmTree Ht;

HfmCode Hc;

char\* code;

char\* BeTran;

char \*ToBeTran="THIS PROGRAM IS MY FAVORITE";

int n=27;

list c[27]={{' ',186},{'A',64},{'B',13},{'C',22},{'D',32},{'E',103},{'F',21},{'G',15},{'H',47},

{'I',57},{'J',1},{'K',5},{'L',32},{'M',20},{'N',57},{'O',63},{'P',15},{'Q',1},{'R',48},{'S',51},

{'T',80},{'U',23},{'V',8},{'W',18},{'X',1},{'Y',16},{'Z',1}};

Ht=InitHfmTree(Ht,c,n);

Hc=initHfmCode(Ht,n,Hc);

code=Encoding(ToBeTran,Hc,code,n);

puts(code);

BeTran=Decoding(code,Ht,n,BeTran,strlen(ToBeTran));

puts(BeTran);

puts(ToBeTran);

}