实验报告

班级：电信1809 学号：U201813496 姓名：张浩

题目：编写一个哈夫曼的编/译码系统。

（基本要求） 一个完整的系统应具备以下功能

1. 初始化。从终端读入字符集大小n，以及n个字符和n个权值，建立哈夫曼树。
2. 编码。利用建立好的哈夫曼树对正文编码，将编码存在树结点。
3. 译码。利用建立好的哈夫曼树对输入的代码进行译码。
4. 打印代码。将每个字符对应的哈夫曼编码打印。
5. 需求分析：

演示程序中，集合的元素限定为26个字母A~Z，并附有每个字母的权值，屏幕打印出每个字符对应的哈夫曼编码。当输入一串新的哈夫曼代码时，程序执行要求能输出译码后的字符串。

1. 概要设计：

选用树为抽象数据类型，将待处理的26个字母字符以树的叶子结点表示，并以数组作为树的存储结构，每一个数组单元存放一个树结点，树结点有指向左右孩子和双亲的指针。

1. 详细设计：

#include<stdio.h>

#include<stdlib.h>

#define N\_LEAVE 26

#define N\_NODE (26\*2-1)

typedef struct \_Node

{

char character;

int weight;

int lchild;

int rchild;

int parent;

}Node,\*pNode;

typedef struct \_Code

{

int HufCode[N\_LEAVE];

int Start;

char Char;

}Code,\*pCode;

void Huffman(Node Ht[],int Wt[])

{

int i,j,x1,x2;

int min1,min2;

for(i=0;i<N\_NODE;i++)

{

Ht[i].parent = -1;

Ht[i].lchild = -1;

Ht[i].rchild = -1;

if(i<N\_LEAVE)

{

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

Ht[i].character = i+65;

}

else

{

Ht[i].weight = 0;

Ht[i].character = '?';

}

}

for(i=1;i<=N\_LEAVE-1;i++)

{

min1 = min2 = 1000;

x1 = x2 = 0;

for(j=0;j<N\_LEAVE-1+i;j++)

{

if(Ht[j].parent == -1 && Ht[j].weight<min1 )

{

min2 = min1;

x2 = x1;

min1 = Ht[j].weight;

x1 = j;

}

else

{

if( Ht[j].parent == -1 && Ht[j].weight<min2 )

{

min2 = Ht[j].weight;

x2 = j;

}

}

}

Ht[x1].parent = N\_LEAVE-1+i;

Ht[x2].parent = N\_LEAVE-1+i;

Ht[N\_LEAVE-1+i].weight = Ht[x1].weight + Ht[x2].weight;

Ht[N\_LEAVE-1+i].lchild = x1;

Ht[N\_LEAVE-1+i].rchild = x2;

}

}

void Code\_Ht(Node Ht[],Code Hc[])

{

int i,d,p,j;

Code x;

for(i=0;i<N\_LEAVE;i++)

{

x.Char = Ht[i].character;

x.Start = N\_LEAVE-1;

d = i;

p = Ht[i].parent;

while(1)

{

if(Ht[p].lchild == d)

x.HufCode[x.Start] = 0;

else if(Ht[p].rchild == d)

x.HufCode[x.Start] = 1;

else

printf("ERROR!");

d = p;

p = Ht[d].parent;

if(p == -1) break;

x.Start--;

}

for(j=x.Start;j<=N\_LEAVE-1;j++)

{

Hc[i].HufCode[j] = x.HufCode[j];

}

Hc[i].Start = x.Start;

Hc[i].Char = x.Char;

}

}

void PrintCode(Code Hc[])

{

int i,j;

for(i=0;i<N\_LEAVE;i++)

{

for(j=Hc[i].Start;j<N\_LEAVE;j++)

{

printf("%d",Hc[i].HufCode[j]);

}

printf("%5c\n",Hc[i].Char);

}

}

void FindCode(Code Hc[],char s[])

{

int i,j,k=0;

printf("\n");

printf("\n");

while(s[k]!='\0'){

for(i=0;i<N\_LEAVE;i++)

{

if( s[k] == Hc[i].Char )

{

for(j=Hc[i].Start;j<N\_LEAVE;j++)

{

printf("%d",Hc[i].HufCode[j]);

}

}

}

k++;

}

}

void trans(char index[],Node hc[])

{

int i,root,p;

for(i=0;i<N\_NODE;i++)

{

if(hc[i].parent==-1)

{

root=i;

break;

}

}

p=root;

for(i=0;index[i]!='\0';i++)

{

if(index[i]=='1')

{

p=hc[p].rchild;

}

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

{

p=hc[p].lchild;

}

if(hc[p].lchild==-1&&hc[p].rchild==-1)

{

printf("%c",hc[p].character);

p=root;

}

}

}

int main()

{

Node HufTree[N\_NODE];

Code HCode[N\_LEAVE];

int Wt[N\_LEAVE] = {64,13,22,32,103,21,15,47,57,1,5,32,20,57,63,15,1,48,51,80,23,8,18,1,16,1};

char s[]={"THISPROGRAMISMYFAVORITE"};

Huffman(HufTree,Wt);

Code\_Ht(HufTree,HCode);

PrintCode(HCode);

FindCode(HCode,s);

char string[20];

scanf("%s",string);

trans(string,HufTree);

}

1. 调试分析：

程序模块划分合理。整体的时间复杂度为O(n²)，但在开辟编码数组时数组长度过大，造成了浪费，因此，空间复杂度比较大。

1. 测试结果：’’



