## 实验四 Huffman编码

## 需求分析

利用所给字母表及其权值建立Huffman树，输出所有字母的Huffman编码。

输入输出形式：

字符 哈夫曼编码

… …

字符 哈夫曼编码

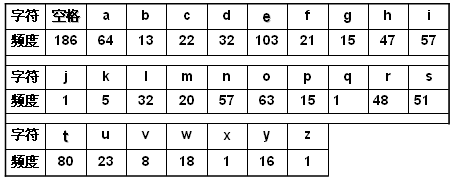
请输入一个小写字母：输入字符（a~z）

字符X的哈夫曼编码是：x的哈夫曼编码

功能：利用Huffman树建立前缀型编码并输出，输入对应字符，输出其哈夫曼编码

数据范围：空格、26个小写英文字母

测试数据：



**2.概要设计**

利用最优二叉树即哈夫曼树将含有不同权值的26个字符编码并输出，记录不同字符的哈夫曼编码，输入26个小写字母中任意一个输出其哈夫曼编码。

**3.详细设计**

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

//叶子节点个数及总节点个数

//哈夫曼树创建结构体

typedef struct \_Node

{

char character;

int weight, lchild, rchild, parent;

}Node,\*pNode;

//哈夫曼编码结构体

typedef struct \_Code

{

int HufCode[N\_LEAVE]; //叶节点最长编码位数应该为树的最长路径

int Start; //编码起始位（相对编码数组）

char Char; //编码的字符值

}Code,\*pCode;

//构造哈夫曼树函数

依次选出两个权值最小的数据合成一棵二叉树，父结点的数据域为左右子树的权值和

//获取并保存每个叶节点的哈夫曼编码供解码时查询

依次每个叶结点(在哈夫曼结点数组的最前面的空间中)寻找双亲直到根节点，记录路径,路径就是哈夫曼编码

//输出每个字符的的哈夫曼编码

传入哈夫曼编码表

循环输出字符数组及其编码

//查询字符的编码

循环查找到该字符，调用哈夫曼编码数组，输出其哈夫曼编码

//主函数

创建叶子结点数组、哈夫曼编码数组

创建叶子结点权值数组，并将权值填写到数组中

调用创建哈夫曼函数

调用哈夫曼编码函数

调用打印哈夫曼字符表及其哈夫曼编码函数

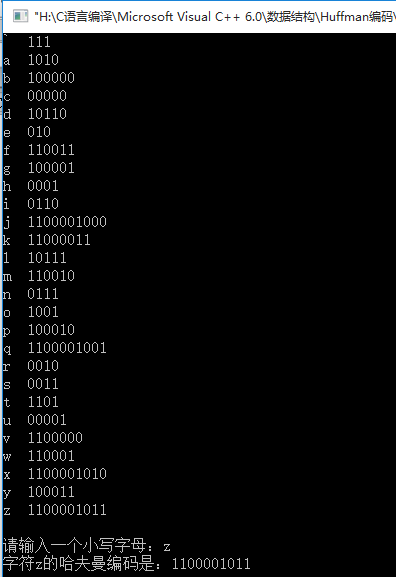
调用查找对应字母的哈夫曼编码的函数

**4.用户使用说明**

程序自动输出26个给定权值的哈夫曼树编码，当程序提示：“请输入一个小写字母：”时，用户输入26个小写字母中的任意一个，程序将输出该字母的哈夫曼编码。

**5.测试结果**

输入数据：z



**6.附录**

/\*

程序功能：创建给定权值的26个字母哈夫曼树及其编码

1、建立哈夫曼树

2、输出每个字母的哈夫曼编码

3、输入字母解码

\*/

#include<iostream>

#include<stdio.h>

#include<stdlib.h>

using namespace std;

//定义结点数据

#define N\_LEAVE 26+1

//N+1个权值,则有N+1个叶结点，树共2(N+1)-1个结点,+1表示空格

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

typedef struct \_Node

{

char character;

int weight, lchild, rchild, parent;

}Node,\*pNode;

typedef struct \_Code

{

int HufCode[N\_LEAVE]; //叶节点最长编码位数应该为树的最长路径

int Start; //编码起始位（相对编码数组）

char Char; //编码的字符值

}Code,\*pCode;

//构造哈夫曼树

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

{

int i,j,x1,x2, min1,min2;

//初始化结点数组Ht

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+96; //空格，a-z的ASCII码

}

else

{

Ht[i].weight = 0;

Ht[i].character = '?'; //生成的中间结点字符值标记为'？'

}

}

//控制n-1次结点的结合（若有n个叶结点）

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

{

//min1、min2记录当前最小、次小权值

min1 = min2 = 0x3f3f3f3f;

//x1、x2记录当前最小次小权值结点的位置(数组标号)

x1 = x2 = 0;

for(j=0;j<N\_LEAVE-1+i;j++) //在[0-j]范围内找最小次小权值结点

{

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

//parent元素的判断是为了排除已结合过的结点，结合过的结点parent有正值

{

min2 = min1;

//当前结点权值小于最小值，所以当前结点变成最小权值结点，原最小结点变成//原来的次小结点

x2 = x1;

min1 = Ht[j].weight;

x1 = j;

}

else

{

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

//当前结点权值大于最小值，小于次小值，则取代次小结点

{

min2 = Ht[j].weight;

x2 = j;

}

}

}

//将找到的最小、次小权值结点结合成树，为其父结点赋值,可见该哈夫曼树的//根节点应该是Ht数组最后一个结点Ht[N\_NODE-1]

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;

}

}

//获取并保存每个叶节点的哈夫曼编码供解码时查询

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

{

int i,d,p,j;

Code x;

//依次每个叶结点(在哈夫曼结点数组的最前面的空间中)寻找双亲直到根//节点，记录路径,路径就是哈夫曼编码

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

{

x.Char = Ht[i].character;

x.Start = N\_LEAVE-1; //默认编码起点为编码数组最后一位

d = i;

p = Ht[i].parent;

while( 1 )

{

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

x.HufCode[x.Start] = 0; //默认编码为左0右1

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

x.HufCode[x.Start] = 1;

else

printf("ERROR!");

d = p;

p = Ht[d].parent;

if(p == -1) break;

//Ht[i]为根结点退出循环，说明已经回溯到了根结点

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;

}

}

//输出每个字符的的哈夫曼编码

void PrintCode(Code Hc[])

{

int i,j;

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

{

cout<<Hc[i].Char<<" ";

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

{

cout<<Hc[i].HufCode[j];

}

cout<<endl;

}

}

//查询字符的编码

void FindCode(Code Hc[])

{

int i,j;

char x;

cout<<"\n请输入一个小写字母：";

cin>>x;

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

{

if( x == Hc[i].Char )

{

cout<<"字符"<<x<<"的哈夫曼编码是：";

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

{

cout<<Hc[i].HufCode[j];

}

cout<<endl;

return ;

}

}

}

int main()

{

Node HufTree[N\_NODE]; //存放所有结点数据

Code HCode[N\_LEAVE];

//存放叶结点权值

int Wt[N\_LEAVE] = {186,64,13,22,32,103,21,15,

47,57,1,5,32,20,57,63,15,1,

48,51,80,23,8,18,1,16,1};

Huffman(HufTree,Wt);

Code\_Ht(HufTree,HCode);

PrintCode(HCode);

FindCode(HCode);

return 0;

}