#include <iostream> // Huffman Tree and Code

using namespace std;

a 0.05

b 0.32

c 0.18

d 0.07

e 0.25

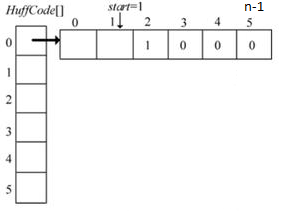
f 0.13

#define MAXBIT 100

#define MAXVALUE 10000

#define MAXLEAF 30

#define MAXNODE MAXLEAF\*2 -1



typedef struct // 结点结构体

{

double weight; // 权值

int parent;

int lchild;

int rchild;

char value; // 该节点表示的字符

}HNodeType;

// bit[]存放结点的编码，start记录编码开始下标，逆向译码（从叶子到根）。

// 存储时，start从n-1开始依次递减，从后向前存储；

// 读取时，从start+1开始到n-1，从前向后输出，即为该字符的编码。

typedef struct // 编码结构体

{

int bit[MAXBIT]; // 存放结点编码

int start; // 编码开始下标

}HCodeType;

HNodeType HuffNode[MAXNODE]; // 定义一个结点结构体数组

HCodeType HuffCode[MAXLEAF]; // 定义一个编码结构体数组

//———————————————————————————————构造哈夫曼树

void HuffmanTree(HNodeType HuffNode[MAXNODE], int n)

{

int i, j; // 循环变量

int x1, x2; // 构造哈夫曼树不同过程中两个最小权值结点在数组中的序号

double m1,m2; // 构造哈夫曼树不同过程中两个最小权值结点的权值

for(i=0; i<2\*n-1; i++) // 初始化存放哈夫曼树数组 HuffNode[] 中的结点

{

HuffNode[i].weight = 0; // 权值

HuffNode[i].parent =-1;

HuffNode[i].lchild =-1;

HuffNode[i].rchild =-1;

}

for(i=0; i<n; i++) // 输入 n 个叶子结点的权值

{

cout<<"Please input value and weight of leaf node "<<i + 1<<endl;

cin>>HuffNode[i].value>>HuffNode[i].weight;

}

for(i=0; i<n-1; i++) // 构造 Huffman 树，执行n-1次合并

{

m1=m2=MAXVALUE; // m1、m2中存放两个无父结点且结点权值最小的两个结点

x1=x2=-1; // 找出所有结点中权值最小、无父结点的两个结点，并合并之为一棵二叉树

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

{

if(HuffNode[j].weight < m1 && HuffNode[j].parent==-1)

{

m2 = m1;

x2 = x1;

m1 = HuffNode[j].weight;

x1 = j;

}

else if(HuffNode[j].weight < m2 && HuffNode[j].parent==-1)

{

m2=HuffNode[j].weight;

x2=j;

}

}

// 设置找到的两个子结点 x1、x2 的父结点信息

HuffNode[x1].parent = n+i;

HuffNode[x2].parent = n+i;

HuffNode[n+i].weight = m1+m2;

HuffNode[n+i].lchild = x1;

HuffNode[n+i].rchild = x2;

cout<<"x1.weight and x2.weight in round "<<i+1<<" "; // 用于测试

cout<<HuffNode[x1]. weight<<" "<<HuffNode[x2].weight<<endl; }

}

void HuffmanCode(HCodeType HuffCode[MAXLEAF], int n) // 哈夫曼树编码

{

HCodeType cd; // 定义一个临时变量来存放求解编码时的信息

int i,j,c,p;

for(i = 0;i < n; i++) // 从0号结点开始，逐个结点构建，i代表各结点编号

{

cd.start = n-1; // 每个结点的编码数组从数组最后一个元素开始构建

c = i;

p = HuffNode[c].parent; // 找父结点

while(p != -1) // p=-1时为根结点

{

if(HuffNode[p].lchild == c) // 左子树为0，右子树为1

cd.bit[cd.start] = 0;

else

cd.bit[cd.start] = 1;

cd.start--; // 求编码的低一位

c = p; // c,p↓变量上移，准备下一个循环

p = HuffNode[c].parent; // 设置下一循环条件

}

// 把叶子结点的编码信息从临时编码cd中复制出来，放入编码结构体数组

for(j=cd.start+1; j<n; j++)

HuffCode[i].bit[j] = cd.bit[j];

HuffCode[i].start = cd.start;

}

}

int main()

{

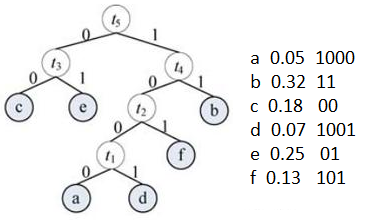
int i,j,n;

cout<<"Please input n："<<endl;

cin>>n;

HuffmanTree(HuffNode, n); // 构造哈夫曼树

HuffmanCode(HuffCode, n); // 哈夫曼树编码

 for(i = 0;i < n;i++) // 输出已保存好的所有存在编码的哈夫曼编码

{

cout<<HuffNode[i].value<<": Huffman code is: ";

for(j=HuffCode[i].start+1; j < n; j++)

cout<<HuffCode[i].bit[j];

cout<<endl;

}

return 0;

}

1952年，数学家D.A.Huffman提出了根据字符在文件中出现的频率，用0、1的数字串表示各字符的最佳编码方式，称为哈夫曼（Huffman）编码。哈夫曼编码很好地解决了编码尽可能短和不能有二义性的问题，被广泛应用于数据压缩，尤其是远距离通信和大容量数据存储方面，常用的JPEG图片就是采用哈夫曼编码压缩的（图片的许多像素值是相同的，如果出现频率高的像素值，采用较短的编码，则需要的字节数较少）。

哈夫曼编码的基本思想是以字符的使用频率作为权构建一棵哈夫曼树，然后利用哈夫曼树对字符进行编码。构造一棵哈夫曼树，是将所要编码的字符作为叶子结点，该字符在文件中的使用频率作为叶子结点的权值，以自底向上的方式，通过n-1次的“合并”运算后构造出的一棵树，核心思想是权值越大的叶子离根越近。

哈夫曼算法采取的贪心策略是每次从树的集合中取出没有双亲且权值最小的两棵树作为左右子树，构造一棵新树，新树根节点的权值为其左右孩子结点权值之和，将新树插入到树的集合中，求解步骤如下。

1 确定合适的数据结构。

编写程序前需要考虑的情况有：哈夫曼树中没有度为1的结点，则一棵有n个叶子结点的哈夫曼树共有2n-1个结点（n-1次的“合并”，每次产生一个新结点），构成哈夫曼树后，为求编码，需从叶子结点出发走一条从叶子到根的路径。译码需要从根出发走一条从根到叶子的路径，那么我们需要知道每个结点的权值、双亲、左孩子、右孩子和结点的信息。

2 初始化。构造n棵结点为n个字符的单结点树集合T={t1，t2，t3，…，tn}，每棵树只有一个带权的根结点，权值为该字符的使用频率。

3 如果T中只剩下一棵树，则哈夫曼树构造成功，跳到步骤（6）。否则，从集合T中取出没有双亲且权值最小的两棵树ti和tj，将它们合并成一棵新树zk，新树的左孩子为ti，右孩子为tj，zk的权值为ti和tj的权值之和。

4 从集合T中删去ti，tj，加入zk。

5 重复以上（3）～（4）步。

6 约定左分支上的编码为“0”，右分支上的编码为“1”。从叶子结点到根结点逆向求出每个字符的哈夫曼编码，从根结点到叶子结点路径上的字符组成的字符串为该叶子结点的哈夫曼编码。算法结束。

首先是六个字符的编码没有二义性，任何一个字符的编码都不是其它编码的前缀，这样字符排列在一起就不会引起歧义，如0010001001，只能解码为cad。

其次，使用频率最高的采用二位编码，使用频率最低的采用四位编码，体现了“频率越高、编码越短”的原则。