**数据结构综合实验报告**

2020/ 2021（2）

**题 目： 哈夫曼编码器的设计与实现**

**学 号：**

**班 级：**

**姓 名：**

**分数：**

目录

[1 需求分析 3](#_Toc76673684)

[1.1任务要求 3](#_Toc76673685)

[1.2 软件功能分析 3](#_Toc76673686)

[1.3 输入输出格式及示例 3](#_Toc76673687)

[2 设计分析 4](#_Toc76673688)

[2.1 功能模块图及各功能简要说明 4](#_Toc76673689)

[2.2 主程序模块 4](#_Toc76673690)

[2.3 存储结构 6](#_Toc76673691)

[2.4 各功能模块的详细设计 7](#_Toc76673692)

[2.4.1 统计词频模块 7](#_Toc76673693)

[2.4.2 创建哈夫曼树模块 7](#_Toc76673694)

[2.4.3 哈夫曼编码模块 7](#_Toc76673695)

[2.4.4 对文章进行编码的模块 8](#_Toc76673696)

[3实现和调试 8](#_Toc76673697)

[3.1主要算法 8](#_Toc76673698)

[3.1.1 统计词频 8](#_Toc76673699)

[3.1.2 创建哈夫曼树 9](#_Toc76673700)

[3.1.3 哈夫曼编码 10](#_Toc76673701)

[3.1.4 对文章进行编码 10](#_Toc76673702)

[3.2主要问题及解决 11](#_Toc76673703)

[3.3测试执行及结果 11](#_Toc76673704)

[4总结及改进 13](#_Toc76673705)

[4.1实验解决的问题 13](#_Toc76673706)

[4.2实验不足及改进措施 13](#_Toc76673707)

[5 参考文献 13](#_Toc76673708)

# 1 需求分析

## 任务要求

信息在通讯线路中是以编码形式进行传输的，在接收端采取一种译码的机制将信息“译”出来。

要求：

（1）读取一篇英文文章，先对文章中以空格分隔的英文单词或符号统计词频；

（2）建立哈夫曼树；给出哈夫曼树的存储结构图；

（3）输出每个单词的哈夫曼编码；

（4）利用哈夫曼树，对模拟输入的报文进行编码，并将结果显示出来。

## 1.2 软件功能分析

（1）输入txt文件路径，从txt文件中读取一篇英文文章；

（2）统计文章中每个字符串出现的频率，字符串之间以空格分隔，如：”I am a good student.”中“good“是一个字符串，”student.“也是一个字符串；或者选择统计文章中每个字符出现的频率，字符包括字母、数字、运算符号、标点符号、空格、回车等；

（3）创建哈夫曼树，并显示哈夫曼树的参数表；

（4）对每个字符串或每个字符进行哈夫曼编码，

（5）输入文件路径，对从文件路径中读取的文章进行编码，并将结果显示出来。

## 1.3 输入输出格式及示例

进入程序界面，用户选择以下四个功能： 1. 统计词频并创建哈夫曼树 2. 进行哈夫曼编码 3. 对文章进行编码 4. 退出。

（1）输入1，进行统计词频并创建哈夫曼树。输入文件路径如：C:\Users\fsrm0\Desktop\test1.txt，从路径中读取文件，再选择编码方式（1.对以文章中空格分隔的字符串进行编码 2.对每个字符进行编码)，如选择1后，会对文章中以空格分隔的所有字符串统计词频，建立哈夫曼树，并输出哈夫曼树的参数表如：

序号 单词 词频 父结点序号 左孩子序号 右孩子序号

1 and 1 7 0 0

2 are 4 9 0 0

3 are. 1 7 0 0

4 is 4 10 0 0

5 me 3 9 0 0

6 you 2 8 0 0

7 2 8 1 3

8 4 10 6 7

9 7 11 5 2

10 8 11 4 8

11 15 0 9 10

（2）接着输入2，进行哈夫曼编码。编码成功后，输出每个单词编码：

单词 哈夫曼编码

and 1110

are 01

are. 1111

is 10

me 00

you 110

（3）输入3，对文章进行编码。先输入要进行编码的文章的文件路径：C:\Users\fsrm0\Desktop\test2.txt，文章编码成功后，输出文章编码结果(在编码表中不存在的用“#”代替）：

111011011000000010101010010101011111

（4）输入4，退出程序。

# 2 设计分析

## 2.1 功能模块图及各功能简要说明

如图1所示，哈夫曼编码器具有以下四个功能：

（1）统计词频：输入文件路径，从文件中读取一篇英文文章，统计文章中每个单词的词频或每个字符的出现频率；

（2） 创建哈夫曼树：根据统计的频率创建出哈夫曼树，并显示哈夫曼树的各个参数：序号，单词/字符，权重，父节点序号，左孩子序号，右孩子序号 ；

（3） 进行哈夫曼编码 ：显示出每个单词字符的编码；

（4） 对文章进行编码。显示出文章编码后的结果。

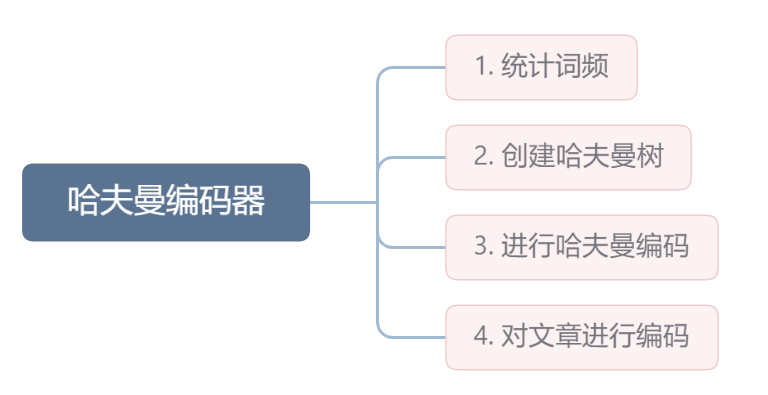


图1 功能模块图

## 2.2 主程序模块

int main() {

map<string, int> m;

HuffmanTree HT = NULL;

HuffmanCode HC = NULL;

int choice;

while (true) {

system("cls");

int num;

cout << "请选择功能 1.统计词频并创建哈夫曼树 2.进行哈夫曼编码 3.对文章进行编码 4.退出：";

cin >> num;

switch (num) {

case 1: {

cout << "请输入文件路径:";

stringstream ss;

readFile(ss); // 读文件

cout << "请选择编码方式 (1.单词 2.字符):";

cin >> choice;

Count(ss, m, choice); //统计词频

CreateHuffmanTree(HT, m); // 创建哈夫曼树

cout << "创建哈夫曼树成功！下面是哈夫曼树的参数输出：" << endl;

if (choice == 1) cout << "序号\t单词\t词频\t父结点\t左孩子\t右孩子\n";

else cout << "序号\t字符\t词频\t父结点\t左孩子\t右孩子\n";

ShowHuffmanTree(HT, m); // 展示哈夫曼树的存储图

system("pause");

break;

}

case 2: {

HuffmanCoding(HT, HC, m.size()); // 哈夫曼编码

if (choice == 1) {

cout << "哈夫曼编码成功！下面是每个单词编码：" << endl;

cout << "单词\t哈夫曼编码" << endl;

}

else {

cout << "哈夫曼编码成功！下面是每个字符的编码：" << endl;

cout << "字符\t哈夫曼编码" << endl;

}

ShowHuffmanCode(m, HC); // 显示各单词/字符的编码

system("pause");

break;

}

case 3: {

stringstream ss2;

cout << "请输入要编码的文章的文件路径:";

readFile(ss2); // 读文件

cout << "文章编码成功！下面是文章编码结果(在编码表中不存在的用“#”代替）：" << endl;

PaperEncode(HC, m, ss2, choice); //对文章编码并输出

system("pause");

break;

}

case 4: return 0;

}

}

}

## 2.3 存储结构

（1）用一个string类型的动态数组存储所有字符串的哈夫曼编码：typedef string \* HuffmanCode；

（2）哈夫曼树有n个叶子结点，那么共有2n-1个结点，可以将所有结点存储在一个大小为2n-1的一维数组中。每个结点定义如下：

typedef struct

{

int weight;

int parent, lchild, rchild;

}HTNode, \* HuffmanTree;

Weight为权重，即词频，parent为其父结点的序号，lchild为其左孩子的序号，rchild为其右孩子的序号，parent 为0说明是此结点是根节点，lchild, rchild为0说明此节点是叶结点。

例如，在一篇文章中有7个单词，单词存入的顺序是：and, are, is, me, you。各单词及其出现次数为： and 1, you 2, me 3, is 4, are 5。 那么其存储结构如图2中深色部分所示：（parent为其父结点的序号，lchild为其左孩子的序号，rchild为其右孩子的序号）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 单词 | 序号 | Weight | Parent | Lchild | Rchild |
|  | 0 |  |  |  |  |
| and | 1 | 1 | 6 | 0 | 0 |
| are | 2 | 5 | 8 | 0 | 0 |
| is | 3 | 4 | 8 | 0 | 0 |
| me | 4 | 3 | 7 | 0 | 0 |
| you | 5 | 2 | 6 | 0 | 0 |
|  | 6 | 3 | 7 | 1 | 5 |
|  | 7 | 6 | 9 | 4 | 6 |
|  | 8 | 9 | 9 | 3 | 2 |
|  | 9 | 15 | 0 | 7 | 8 |

图2 存储结构图

## 2.4 各功能模块的详细设计

### 2.4.1 统计词频模块

函数名称： void Count(stringstream& ss, map<string, int>& map, int choice)；

函数参数：ss为串流，map为存储单词和词频的容器，choice是选择对字符串还是对字符统计频率；

实现方法：

将串流中以空格分隔的字符串存入map容器中，每读入一个单词时用find()函数查找容器中是否已存在该单词，如果找到了，second++，没有找到就insert（）插入。具体如下：

for ( i = 1; ss >> word; ++i) {

auto pos = map.find(word);

if (pos == map.end()) map.insert(make\_pair(word, 1));

else pos->second += 1;

}

### 2.4.2 创建哈夫曼树模块

函数名称：CreateHuffmanTree

函数参数：HuffmanTree& HT, map<string, int> map)

返回值：无；

实现方法：

先判断map.size()是否合理，size要大于1；再为哈夫曼树HT开辟m+1个内存，（因为还要有一个空间要存储0号单元）。初始化HT所有结点，weight等于map对应的second的值，parent和左右孩子下标都是0。

接着建立哈夫曼树。i从n+1到m，循环执行以下步骤：在HT[ 1……i-1 ] 中选择parent为0的权值最小的2个结点，其数组下标分别为s1、s2。S1 和 s2 的parent等于 i；i 的lchild、rchild 等于s1、s2，i的weight 是s1、s2的weight的和；如下：

for (int i = n + 1; i <= m; i++) {

Select(HT, i, s1, s2);

HT[s1].parent = i;

HT[s2].parent = i;

HT[i].lchild = s1;

HT[i].rchild = s2;

HT[i].weight = HT[s1].weight + HT[s2].weight;

}

### 2.4.3 哈夫曼编码模块

函数名称： HuffmanCoding

函数参数：HuffmanTree& HT, HuffmanCode& HC, int n

返回值：无

实现方法：

从叶子到根逆着求每个单词的哈夫曼编码

for ( i=1; i<=n; ++I )

{

c = i;

for ( f = HT[ c ].parent； f != 0；c = f，f = HT[ c ].parent)

{

if ( HT[ f ]. lchild == c ) // 左0右1

HC[ i ] . append('0');

else

HC[ i ]. append( '1');

}

Reverse( HC[ i ].begin(), HC[ i ].end() )

}

### 2.4.4 对文章进行编码的模块

函数名称： PaperEncode

函数参数：HuffmanCode HC, map<string, int>& map, stringstream& ss, int choice

函数返回值：无

实现方法：

读入每个单词，在map容器中查找该单词的位置i，找到之输出HC[i]，没有找到输出“#“。如下：

while (ss >> str) {

auto it = map.begin();

for (int i = 1; it != map.end(); ++i, ++it)

if (it->first == str) { cout << HC[i]; break; }

if (it == map.end()) cout << "#";

}

# 实现和调试

## 主要算法

### 3.1.1 统计词频

void Count(stringstream& ss, map<string, int>& map, int choice) {

map.clear();

if (choice == 1) {

string word;

for (int i = 1; ss >> word; ++i) {

auto pos = map.find(word);

if (pos == map.end()) map.insert(make\_pair(word, 1));

else pos->second += 1;

}

}

else {

char c;

ss.unsetf(ios::skipws);

for (int i = 1; ss >> c; ++i) {

string s(1, c);

auto pos = map.find(s);

if (pos == map.end()) map.insert(make\_pair(s, 1));

else pos->second += 1;

}

}

}

### 3.1.2 创建哈夫曼树

void CreateHuffmanTree(HuffmanTree& HT, map<string, int> map) {

int n, m, i, s1, s2;

n = map.size();

if (n <= 1) return;

m = 2 \* n - 1;

HT = new HTNode[m + 1];

auto it = map.begin();

for (i = 1; i <= m; ++i) {

if (it != map.end()) {

HT[i].weight = it->second;

++it;

}

HT[i].parent = 0;

HT[i].lchild = 0;

HT[i].rchild = 0;

}

for (int i = n + 1; i <= m; i++) {

Select(HT, i, s1, s2);

HT[s1].parent = i;

HT[s2].parent = i;

HT[i].lchild = s1;

HT[i].rchild = s2;

HT[i].weight = HT[s1].weight + HT[s2].weight;

}

}

### 3.1.3 哈夫曼编码

void HuffmanCoding(HuffmanTree& HT, HuffmanCode& HC, int n) {

int i, c, f;

HC = new string[n + 1];

for (i = 1; i <= n; ++i) {

c = i;

for (f = HT[c].parent; f != 0; c = f, f = HT[c].parent) {

if (HT[f].lchild == c) HC[i].append("0");

else HC[i].append("1");

}

reverse(HC[i].begin(), HC[i].end());

}

}

### 3.1.4 对文章进行编码

void PaperEncode(HuffmanCode HC, map<string, int>& map, stringstream& ss, int choice) {

char c;

string str;

if (choice == 1) {

while (ss >> str) {

auto it = map.begin();

for (int i = 1; it != map.end(); ++i, ++it)

if (it->first == str) { cout << HC[i]; break; }

if (it == map.end()) cout << "#";

}

}

else {

ss.unsetf(ios::skipws);

while (ss >> c) {

string str(1, c);

auto it = map.begin();

for (int i = 1; it != map.end(); ++i, ++it)

if (it->first == str) { cout << HC[i]; break;}

if (it == map.end()) cout << "#";

}

}

}

## 主要问题及解决

（1）字符串在已存在的编码表中找不到匹配的问题。

当输入一篇文章，基于该文章的所有单词字符串进行编码形成编码表后，再输入另一篇文章，可能这篇文章中会出现一些新的单词，而在编码表中找不到对应的编码。对于这种情况，当输出结果时，在编码表中不存在的会用“#”代替。

另外，为了减少找不到不匹配的情况，可选择对字符进行编码，由于单个字符的种类有限，能减少再输入新的文章时找不到对应的编码的情况。用户在选择进行哈夫曼编码时可有两种选择：1. 根据文章中以空格为分隔的字符串进行编码 2. 根据文章中的单个字符进行编码。后者形成的编码表较为全面，能较好地解决无法匹配的问题。

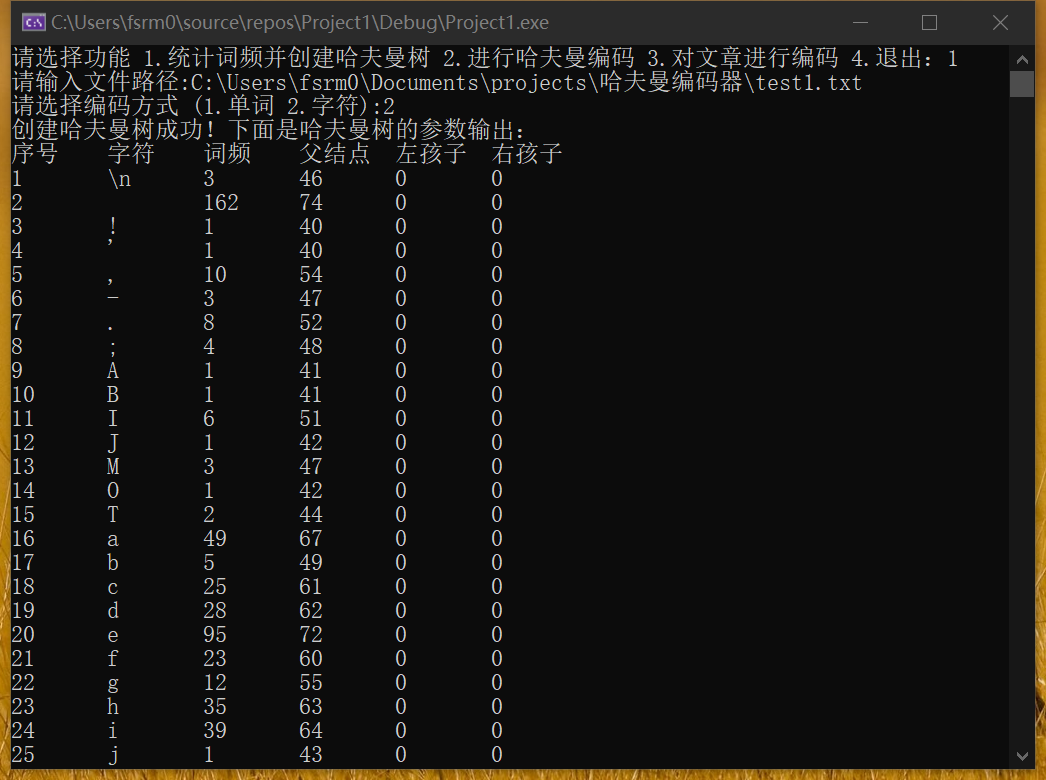
（2）如何读入空格、回车并作为单个字符存入容器中。

空格、回车也可做为单个字符进行编码，由于C++默认输入时会跳过空格，以空格为分隔，因此可以使用cin.unsetf(ios::skipws)；取消输入时跳过空白的默认设置。

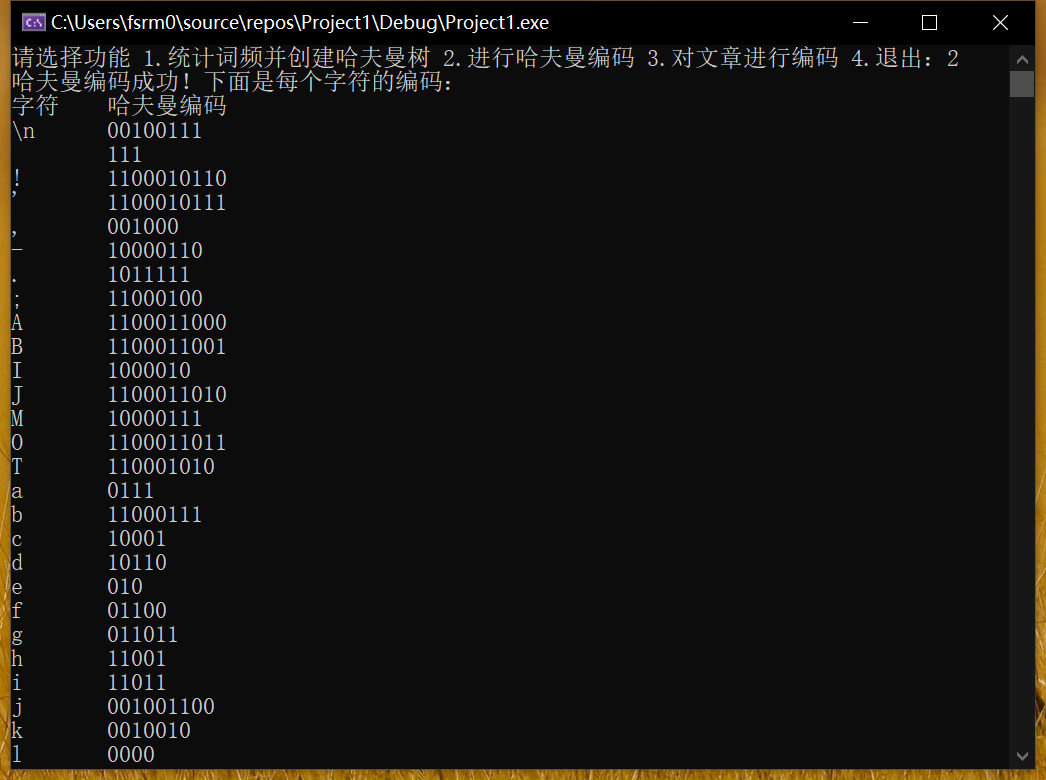
另外回车在作为字符进行显示时可用”\n”进行代替。

## 测试执行及结果

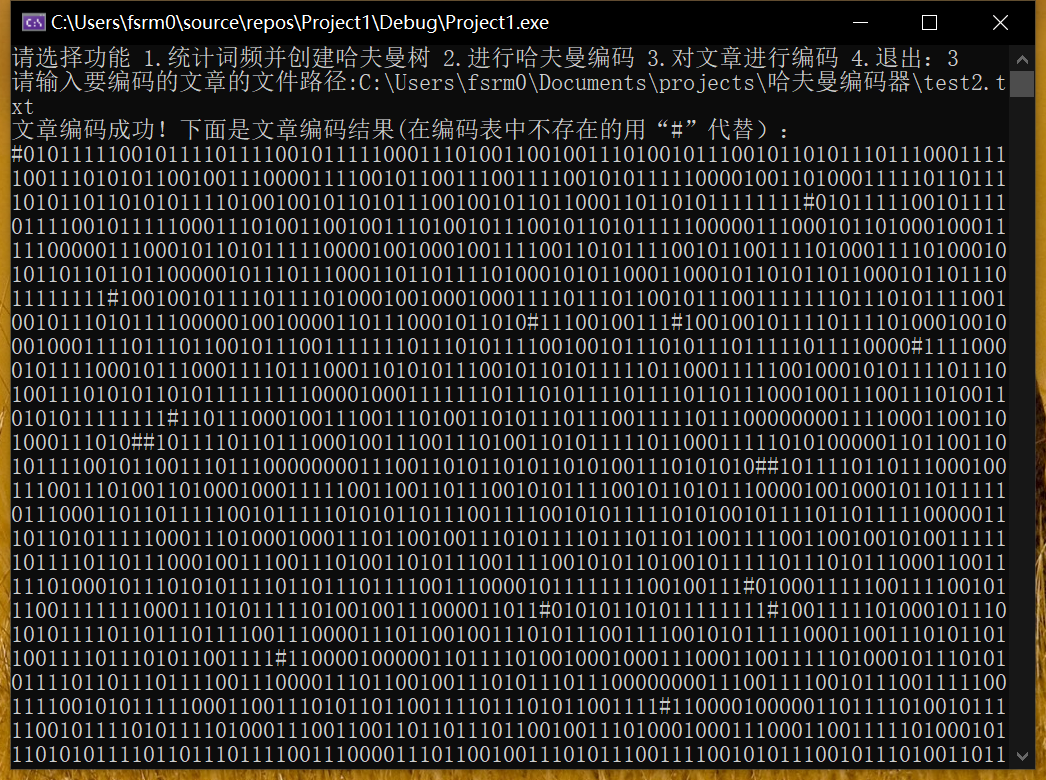
（1）统计词频并创建哈夫曼树，输出哈夫曼树的各个参数。



（2）进行哈夫曼编码，输出每个字符或字符串的哈夫曼编码



（3）输入一篇文章，对该文章进行编码后输出。



# 总结及改进

## 实验解决的问题

本实验设计并实现了哈夫曼编码器。并有如下功能：

（1）输入txt文件路径，从txt文件中读取一篇英文文章；统计文章中每个字符串出现的频率，或者统计文章中每个字符出现的频率；

（3）创建哈夫曼树，并显示哈夫曼树的参数表；

（4）对每个字符串或每个字符进行哈夫曼编码，

（5）输入文件路径，对从文件路径中读取的文章进行编码，并将结果显示出来。

## 实验不足及改进措施

针对于字符串在已存在的编码表中找不到匹配的问题。为了尽量减少此种找不到不匹配的情况，可为用户提供另一种选择，选择对字符进行编码，由于单个字符的种类有限，能减少再输入新的文章时找不到对应的编码的情况。当用户提供的原始文章所包含的字符较为充分、丰富时，这种方法形成的编码表会较为全面，能较好地解决无法匹配的问题。

# 5 参考文献

[1] 严蔚敏, 吴伟民.数据结构(C语言版): 清华大学出版社