# 实验报告

题目：哈夫曼编/译码器

班级：2019211316

姓名：宿文宇

学号：2018211503

完成日期：2020.11.17

### 需求分析

1. 本演示程序中，输入形式为以“回车符”为结束标志的字符串，字符串不允许多余空格。
2. 演示程序以用户和计算机的对话方式执行，即由用户在键盘上输入演示程序中规定的命令，相应的结果显示在其后或输入在文件中。
3. 程序执行的命令包括：

I (Initialization):读入字符集生成哈夫曼树并存入指定文件

E (Encoding):利用哈夫曼树对指定文件编码并存入指定文件

D (Decoding):利用哈夫曼树对指定文件解码并存入指定文件

P (Print):显示指定文件中的编码并存入指定文件

T (Tree Printing):以直观的形式显示哈夫曼树并存入指定文件

W (Word Frequency Analysis):分析指定文件中的字符集数量及每个字符的频率并存入指定文件

Q (Quit):结束程序

### 概要设计

为实现上述程序功能，应以二叉链表实现哈夫曼树，用优先队列生成哈夫曼树，为此，需要的自定抽象数据结构。

struct symbol\_table

{

    char symbol;

    int weight;

};

struct HTNode

{

    symbol\_table data;

    string Huffman\_code;

    HTNode \*left, \*right;

    HTNode(char s, int w)//生成字符为s，权值为w的结点

    HTNode(HTNode \*lc, HTNode \*rc)//生成左儿子为lc，右儿子为rc，权值为两个儿子节点权值之和的结点

};

struct cmp  //for pque

{

    bool operator()(HTNode \*x, HTNode \*y) const

    {

        return x->data.weight > y->data.weight;

    }

};

本程序包含模块：

1. main：主循环
2. menu\_select：打印菜单并读入指令
3. Initialization：读入字符集存入指定文件并生成（字符-权重）对
4. Tree\_build：利用优先队列将（字符-权重）对转为哈夫曼树
5. Read\_hfmTreefile：读入指定字符集文件并生成（字符-权重）对
6. Mapping：对哈夫曼树进行遍历，生成（字符-编码）对
7. Encoding：利用（字符-编码）对对指定文件编码并以二进制形式存入指定文件
8. Decoding：利用（字符-编码）对对指定二进制文件解码并存入指定文件
9. Print：以字符形式显示指定二进制文件中的编码并存入指定文件
10. Tree\_printing：以直观的形式显示哈夫曼树并存入指定文件
11. Node\_printing：以带连线的凹入表的形式打印当前节点
12. Max\_depth：计算当前哈夫曼树的深度
13. Word\_Frequency\_Analysis：分析指定文件中的字符集数量及每个字符的频率并存入指定文件
14. Q (Quit):结束程序

### 重点设计与分析

**选做1:**

为了实现二进制储存，可先将编码的0-1串拼接到一起，因为文件只能以字节为单位读写，所以文件末尾处可能有多余信息，会被错误解码，因此引入EOF标志，在读入字符集创建哈夫曼树时自动加入权值为1的EOF标志，用来指示该段编码结束：

HTNode \*tmp = new HTNode(EOF, EOF\_WEIGHT);

        pque.push(tmp);

因为只能以字节为单位读写文件，因此可以在编码时将字符串末尾补0至长度为8的倍数，然后以字节为单位存入文件

while(code.length() % 8)

        code += '0';

    char buf = 0;

    for(int i = 0; i < code.length(); i++)

    {

        buf <<= 1;

        buf += code[i] - '0';

        if((i + 1) % 8 == 0)

        {

            encoded.write(&buf, 1);

            buf = 0;

        }

    }

在解码时先将二进制文件按字节读出，拼接成一个01字符串，再从头匹配串中的编码，若读到EOF则结束解码

while(encoded.read((char\*)(&buf), 1))

        for(int i = 7 ; i >= 0; i--)

            code += '0' + ((buf >> i) & 1);

    for(int i = 0; i < code.length(); i++)

    {

        tmp += code[i];

        if(to\_text.find(tmp) != to\_text.end())

        {

            if(to\_text.find(tmp)->second == EOF)

                break;

            decoded << to\_text.find(tmp)->second;

            tmp = "";

        }

    }

**选做2:**

对源代码的编码译码与示例有两点不同，一是要自己创建包括源代码所有字符且权值正确的字符集，二是要处理CR和LF

对于第一个问题，可增加一个对文本文件进行字频分析的的模块，其核心内容如下：

    while(getline(text\_file, text))

    {

        for(int i = 0; i < text.length(); i++)

            bucket[text[i]]++;

        bucket['\n']++;

    }

这样就能够正确的创建对该文件中的字符来说正确且最优的的哈夫曼树

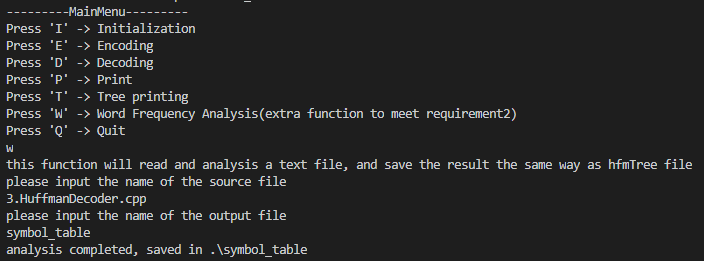
对于第二个问题，可用getline()分行读入，每读一行都在行尾自动添加一个换行符

    while(getline(uncoded, tmp))

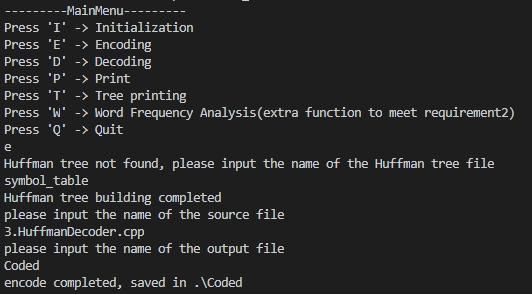
        text += tmp + '\n';

    text += EOF;

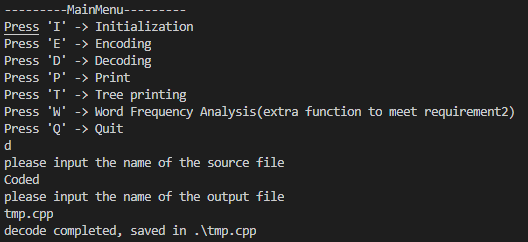
**示例：**



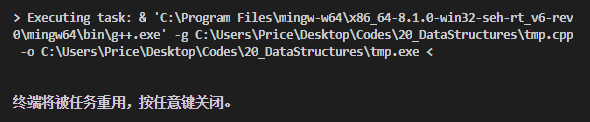
**进行字频分析并创建文件**



**读入文件并创建哈夫曼树，用哈夫曼编码对源代码进行编码**



**对二进制文件进行解码，并将其保存为.cpp格式**



**源代码编码后解码的文本文件可被编译，证明编码译码过程中无错误**

**选做3:**

为了实现用户指定输入输出文件，在每个有文件操作的模块中加入输入文件名的提示并获取文件名打开文件，例如Encoding模块中：

string src, des;

    cout << "please input the name of the source file" << endl;

    cin >> src;

    cout << "please input the name of the output file" << endl;

    cin >> des;

    ifstream uncoded(src);

ofstream encoded(des, ios::binary);

**一些使TreePrint更加直观的尝试:**

凹入表的形式并不是非常直观，可以在树的每一层有关系的节点间添加连线使其更美观，然而并不是每行的每层都要添加连线，例如某结点A的左子树的左结点和其右子树的右结点都无需画出A与A的子节点的连线，分析可知，该结点所在行数的第i列有连线当且仅当其哈夫曼编码串s中s[i]!=s[i+1]，核心代码如下：

    for(int i = 0; i < root->Huffman\_code.length(); i++)

    {

        if(root->Huffman\_code[i] != root->Huffman\_code[i+1])

        {

            cout << setw(8) << "|";

            out << setw(8) << "|";

        }

        else

        {

            cout << setw(8) << " ";

            out << setw(8) << " ";

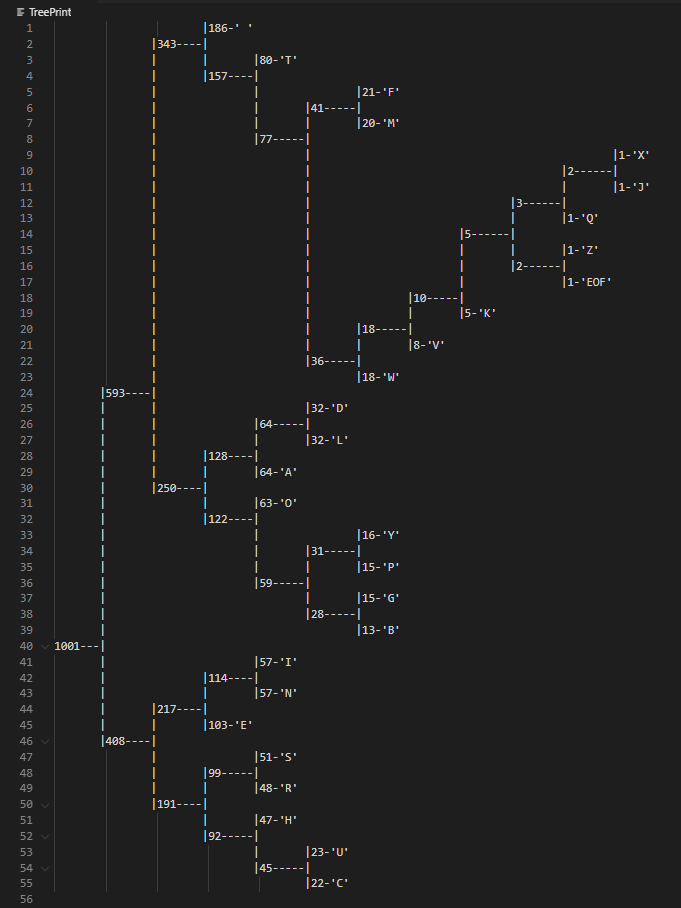
        }

    }

    cout << setw(8) << tmp << endl;

    out << setw(8) << tmp << endl;

打印示例字符集哈夫曼树（空格宽度与字符一致）：



显然更加直观。

### 附录

程序文件名：3.HuffmanDecoder.cpp