**课程设计题目：**设计实现一个Huffman编码与解码系统

**1、问题描述**

利用哈夫曼编码进行信息通讯可以大大提高信道的利用率，缩短信息传输时间,降低传输成本。但是，这要求在发送端通过一个编码系统对待传输数据预先编码;在接受端将传来的数据进行译码。对于双工信道（即可以双向传输信息的信道），每端都需要一个完整的编/译码系统。试为这样的信息收发站编写一个哈夫曼码编码/译码系统。

基本要求：根据某字符文件(suurce.txt)统计各字符的出现频度，构造Human 树，编制Huffman编码．并将给定字符文件进行编码，生成二进制编码文件;再将给定编码文件解码，生成字符文件。要求按二进制位表示编码文件,并分析字符文件的压缩比。

提高要求：实现一个哈夫曼编码/解码系统，包括输入、编码、解码，并要求产生两种以上的二进制编码方案，且系统界面友好。对同一组测试数据，采用不同的编码方案，并从文件长度、算法复杂度等方面进行比较分析。

**2．需求分析**

软件的基本功能：程序可以读入一个英文的文本文件，经过Huffman编码后写出一个二进制的编码文件。程序可以再读入该二进制文件并进行解码，输出原先的文本。

输入/输出形式：文本文件，二进制文件

输入形式：

1. 纯英文文本文件
2. 二进制编码文件

输出形式：

① 二进制编码文件

② 控制台输出文本

测试数据要求：一个不少于2000字符的英文文章（包括字母、数字和符号)。

**3．概要设计**

**（1）抽象数据类型**：

哈夫曼编码的过程中，需要构建哈夫曼树。哈夫曼树是带权路径长度最短的二叉树。

1. **struct** HuffmanNode {
2. **char** data;
3. unsigned frequency;
4. HuffmanNode\* left;
5. HuffmanNode\* right;
7. HuffmanNode(**char** data, unsigned frequency) : data(data), frequency(frequency), left(nullptr), right(nullptr) {}
8. };

以上定义了哈夫曼树节点的结构体，其中data是该节点的字符（若为非叶子节点则为‘\0’），frequency是该字符的在文本中出现频率，left与right分别是左右子节点。

**（2）主程序流程**：

开始

以字符串形式读入文本文件

初始化哈夫曼编码对象

以字符串形式读入文本文件

将数据字符串传入哈夫曼编码对象

进行哈夫曼编码

编码写入二进制文件

将二进制文件读取为编码字符串

利用哈夫曼编码信息进行解码

结束

**（3）模块调用关系**

本程序中函数包括：main函数，这是程序的入口，实现了一个菜单。readFileToString函数，其读入一个文本文件，返回字符串。code2file函数，其输入一个只含01的字符串，并将其以bit而非字符的形式写入二进制文件。file2code函数，其读入一个二进制文件，输出只含01的字符串。

本程序中类：HuffmanNode类，实现了构建哈夫曼树、编码、解码等功能，将在第4节中详述。

函数调用关系如下：

main函数

code2file

ReadFileToString

file2code

HuffmanNode类

HuffmanNode类

编码

解码

**4．详细设计**

**（1）实现概要设计的数据类型：**

HuffmanNode类：

1. **class** HuffmanCode {
2. **private**:
3. std::unordered\_map<**char**, **int**> freq;
4. std::unordered\_map<**char**, std::string> huffman\_codes;
6. **size\_t** dataLen;
8. HuffmanNode\* generate\_tree();
9. **void** generate\_codes(HuffmanNode\* root, **const** std::string &code);
11. **public**:
13. std::string encoding(**const** std::string& data);
14. std::string decoding(**const** std::string& code);
16. **void** printCodes();
17. };

freq是一个哈希表，用于记录每个字符的出现次数。huffman\_code是一个哈希表，用于记录每个字符的编码。dataLen记录了要编码的文本数据的长度。

generate\_tree函数用于生成哈夫曼树，generate\_codes函数则根据哈夫曼树来得到各个字符的哈夫曼编码。encoding函数则会调用上面两个函数计算出各个字符的哈夫曼编码后，将输入的文本字符串编码为01字符串。decoding函数会根据哈夫曼编码对应的每个字符将01字符串解码为文本。

**（2）主程序以及主要模块的算法描述：**

构建哈夫曼树：

开始

结束，返回该节点

将所有字符构造为哈夫曼树节点，存入队列

将队列按字符的频数排序

是否队列长度为1

拿出队列中最小的两个节点

将新节点按照排序插入队列

是

否

构造一个新的哈夫曼节点，data为\0，频数为两节点频数之和，左右子节点为那两个节点（左边的更小）

generate\_tree函数：

1. HuffmanNode\* HuffmanCode::generate\_tree() {
2. std::priority\_queue<HuffmanNode\*, std::vector<HuffmanNode\*>, compare> minHeap;
3. **for** (auto pair : freq) {
4. minHeap.push(**new** HuffmanNode(pair.first, pair.second));
5. }
7. **while** (minHeap.size() != 1) {
8. HuffmanNode\* left = minHeap.top();
9. minHeap.pop();
11. HuffmanNode\* right = minHeap.top();
12. minHeap.pop();
14. HuffmanNode\* newNode = **new** HuffmanNode('\0', left->frequency + right->frequency);
15. newNode->left = left;
16. newNode->right = right;
18. minHeap.push(newNode);
19. }
21. **return** minHeap.top();
22. }

代码中，使用了优先队列来存储所有哈夫曼节点，它能够方便的拿出最小的节点。

generate\_codes函数：

1. **void** HuffmanCode::generate\_codes(HuffmanNode\* root, **const** std::string &code) {
2. **if** (root) {
3. **if** (!root->left && !root->right) {
4. huffman\_codes[root->data] = code;
5. }
6. generate\_codes(root->left, code + "0");
7. generate\_codes(root->right, code + "1");
8. }
9. }

对哈夫曼树进行先序遍历，左子节点记0，右子节点记1，遍历至叶节点将对应字符的编码记录至哈希表中。

**（3）其它模块的算法描述**

code2file函数（01字符串写入二进制文件）：

1. **void** code2file(**const** std::string &code) {
2. std::ofstream outputFile("output.bin", std::ios::out | std::ios::binary);
4. // 将01信息写入二进制文件
5. **char** buffer = 0; // 用于存储每个比特位
6. **int** count = 0; // 计数器
8. **for** (**char** bit : code) {
9. **if** (bit == '1') {
10. buffer = (buffer << 1) | 1; // 将1写入buffer
11. } **else** {
12. buffer = buffer << 1; // 将0写入buffer
13. }
14. count++;
16. // 当buffer中的比特位达到8个时，将buffer写入文件
17. **if** (count == 8) {
18. outputFile.write(&buffer, 1);
19. buffer = 0;
20. count = 0;
21. }
22. }
24. // 如果buffer中还有剩余的比特位，将其写入文件
25. **if** (count > 0) {
26. buffer = buffer << (8 - count); // 将剩余的比特位移到最高位
27. outputFile.write(&buffer, 1);
28. }
30. outputFile.close();
31. }

文件写入的最小单位为字节（1字节=8bit），因此在写入时，用一个char类型（1字节）的变量buffer存储每个比特位，并通过位运算写入0或1.每8个01字符，便将buffer写入文件并清空。

**5、编码与调试分析**

**编码与调试过程中遇到的问题及解决办法**：

【问题一】程序解码时结尾出现多余字符。

分析：文件写入时以字节为单位，而编码的长度未必是8的倍数，因此会在结尾补0，而若某字符的编码若正好全为0，那么解码时则会将补的0也解码为字符。

解决方法：编码时记录字符串长度，若解码出的字符串已达到长度，则直接停止。

**6、使用说明**

进入菜单，根据提示进行选择：

(a).若要打印哈夫曼编码，选择‘1’，若未先进行编码，则无输出。

(b).若要编码文件，选择‘2’，文件为工作目录下的data.txt文件，并写出为output.bin文件。

(c).若要解码output.bin，选择3，若未先进行编码，则无输出。

(d).选择4，退出。

**7、测试结果**





