# 哈夫曼压缩项目

## 理论

哈夫曼编码(Huffman Coding)，又称霍夫曼编码，是一种编码方式，哈夫曼编码是可变字长编码(VLC)的一种。Huffman于1952年提出一种编码方法，该方法完全依据字符出现概率来构造异字头的平均长度最短的码字，有时称之为最佳编码，一般就叫做Huffman编码（有时也称为霍夫曼编码）。

## 设计

### 1压缩

#### 1.1码设计

对文件中出现的所有字符进行统计，创建出哈夫曼树，对于每个叶子节，其编码为从根节点到对应叶子节点的路径，左孩子为0右孩子为1；

#### 1.2二进制编码

按顺序对文件中每个字符按哈夫曼树对其进行01编码，生成01字符串，创建新文件将01字符串按二进制转换后存入。

需要对存入的数据进行包装，目的是为了解压。

将二进制文件分为5部分

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 名称 | 哈夫曼树字符数 | 哈夫曼字符 | 哈夫曼字符权重 | 文件末偏移量（补0数） | 真正文件内容 |
| 总长度 | 4字节 | Huffman\_size字节 | Huffman\_size\*4字节 | 1字节 | —— |
| 内容 | Huffman\_size |  |  | offset |  |

## 问题、解决、思考

### 1基本算法

#### 1.1压缩：文件内中文字符问题

中文字符在c++中由两个char长度字符组成，经验证拆分为两个字节的char的整型值均为负数，而二进制写入和读取时的类型只能是char不能是unsigned char，故不能使用数组来存储。

解决：使用unodered map来存储字符编码，加快编码速度，且存储时不需要类型转换，直接可输入编码。

#### 1.2解压：二进制文件读写问题

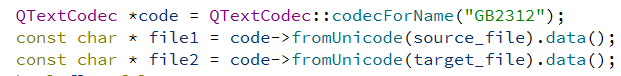
写：将01字符串转换成二进制编码的过程就是，将每8位01字符字符串转换成对应的char类型变量即可，使用移位和或1操作。

读：二进制文件中读取的是常规char类型，1.1中提到中文的char是负数，不可模2运算，无法正常获取编码顺序。使用0x00和0x01来进行与运算，每次可判断最后一位，每判断一位便进行向高位移位运算，并将结果反向存储即可得到正确的编码顺序。

### 2Qt实现

#### 2.1中文路径的编码问题

从选择框内读取的是QString格式，当存在中文路径时，QString->string->char\*的类型转换会出现编码不一的现象，原因是，当读入了中文字符时，编码并不是utf-8，而是GB2312；即使在一开始便确定编码字符也是没用的，必须每次转换前都进行编码转换



#### 2.3 程序结构

对于一个程序来说，界面上的操作不应该直接调用底层算法，这样会导致结构过于紧密，耦合程度太高，不利于后期维护和他人理解。对于应用程序来说，至少应该分离为三层比较合适。

底层为算法实现，该层只实现了关于操作最终导致的运算，该层只需完成自己的任务，不需要考虑其他任何因素，离开其他层后该层依然可以正常使用，可以使用纯c/c++代码；顶层为界面实现，该层绘制了界面，且对用户操作进行反馈，但丝毫不涉及用户操作的最终实现，该层只调用中层的函数，但并不关心中层的函数实现的变化，可以使用纯QT库编写；中层实现从操作到运算之间的连接，获取顶层的用户输入信息，返回底层的运算结果。

按这样看顶层和底层可以同步进行，而中间层则需要统筹帷幄，不可一蹴而就。

## 产品测试

文件类型：

选择文本文件时，压缩按钮可选，解压按钮不可选

选择.hm文件时，解压按钮可选，压缩按钮不可选

场景测试：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 测试序号 | 原文件 | | | 目标文件 | | | 结果 |
| 手动输入 | 窗口选择 | 文件存在 | 手动输入 | 窗口选择 | 文件存在 |
| 1 | 否 | 是 | 是 | 否 | 是 | 否 | 成功 |
| 2 | 是 | 否 | 是 | 是 | 否 | 否 | 成功 |
| 3 | 否 | 是 | 是 | 否 | 是 | 否 | 成功 |
| 4 | 否 | 是 | 否 | 否 | 是 | 否 | 失败，原文件文件不存在 |
| 5 | 否 | 是 | 是 | 否 | 是 | 是 | 弹窗，目标文件已存在是否覆盖 |

## 可优化

#### 1文件读写过程优化

目前编码和解码过程的方式是：一次性读文件->存放文件内容到字符串->字符串编码/解码->输入到文件。这个过程是分离式的，对字符串需要开辟比较大的空间，且在运行时效率不是很高。

优化方案：

压缩时，（1）仍先将原文件内容读取到字符串，因为需要构建哈夫曼树，但读取时可以每次读取一定长度的字符串，对该字符串进行统计字频，处理完成后再新读取一定长度字符串。（2）向目标文件输入完头部信息（3）将原文件指针指向开头，按（1）的方式每次读取一定长度的字符进行编码，同时将编码结果同步输入进目标文件中。这样可以节省开辟的空间并提高运行速度。

解压时，（1）读取完原文件头部信息（2）每次从原文件读取一定长度的字符串便进行转码并编码匹配，将匹配的结果输入到目标文件，字符串读完了再读取，从字符串头部继续匹配。这样做可以节省开辟的空间并提高运行速度。

#### 2多线程优化

如果可行的话，在1的基础上增加多线程优化策略。

比如在读取文件时，由于文件内是标准的按字节存储，所以可以多线程处理文件，将文件拆分，线程单独对读到的文件进行处理，但不能进行写文件操作。

#### 3代码优化

底层算法和界面代码上有部分函数出现了很多相同的操作，可以尝试将相同的操作整理成单个函数，在使用时只需传入必要的参数，可以提高可读性，但可能会导致函数之间依赖程度太高。需要进行取舍。

## 可扩展

#### 功能扩展

目前只能对单个文件进行压缩解压，可以对底层算法逻辑、文件结构进行新的处理。比如，压缩文件结构可以加入文件的总个数，相对路径信息，名字信息等。将多个文件读取，在新文件中先加入总文件个数信息，再加入各文件自身的基本信息。

#### 类型扩展

目前只可以处理文本文件、按逻辑上来说，文件中按字节存储都可以使用哈夫曼进行编码，因为每个字节都可以视作字符char型进行编码和解码。将来可以压缩扩展不同后缀的文件。

#### 界面美化

目前首页界面还是很朴素的，将来可以进行美化。