英文文献编辑与编译码系统

学 号\_19071026\_

姓 名\_\_周晟颐\_\_

指导教师\_\_杜永萍\_\_

2021年12月

1. 需求分析

1.1 基本要求

1.1.1

设计图形界面，可以实现英文文章的编辑与检索功能。

1.1.2

编辑过程包括：

（1）创建新文件；打开文件；保存文件。

（2）查找：输入单词在当前打开的文档中进行查找，并将结果显示在界面中。

（3）替换：将文章中给定的单词替换为另外一个单词，再保存等。

1.1.3

对于给定的文章片段（30<单词数量<100），统计该片段中每个字符出现的次数，然后以它们作为权值，对每一个字符进行编码，编码完成后再对其编码进行译码。在图形界面中演示该过程。

1.1.4

对于给定的多篇文章构成的文档集中，统计不同词汇的出现频率，并进行排序，在界面中显示TOP 30的排序结果。

1.1.5

对于给定的多篇文章构成的文档集中，建立倒排索引，实现按照关键词的检索（包括单个关键词；或者2个及以上关键词的联合检索，要求检索结果中同时出现所有的关键词），并在界面中显示检索的结果（如：关键词出现的文档编号以及所在的句子片段，可以将关键词高亮显示）

1.2 扩展要求

1.2.1

界面设计的优化。

1.2.2

对于编码与译码过程，可以自行设计其他算法。

1.2.3

高级检索，采用逻辑表达式来表示检索需求，例如：(关键词A) AND (关键词B) OR (关键词C) AND (!关键词D)

1.2.4

优化检索，对于检索结果的相关性排序，例如：包含关键词的数量等信息为依据。

1.2.5

可以自行根据本题目程序的实际应用情况，扩展功能。

1.3 需要处理的数据

1.3.1 英文文本

多个txt文档的导入，主要是文件路径，然后对文档中的内容进行处理。首先读取文本。然后统计单词、字符的频率以及位置，生成索引；或者直接进行查找等功能。

1.3.2 用户输入的字符串

有多个功能需要用户输入字符串，比如创建新的文档需要命名、搜索单词、修改文档、在文档中替换内容。

1.3.3 网络爬取的信息

从网站上爬取单词翻译信息，包括 HTML页。

1.3.4选择的文档信息

用户通过“导入文件”，通过窗口选择并导入文档。

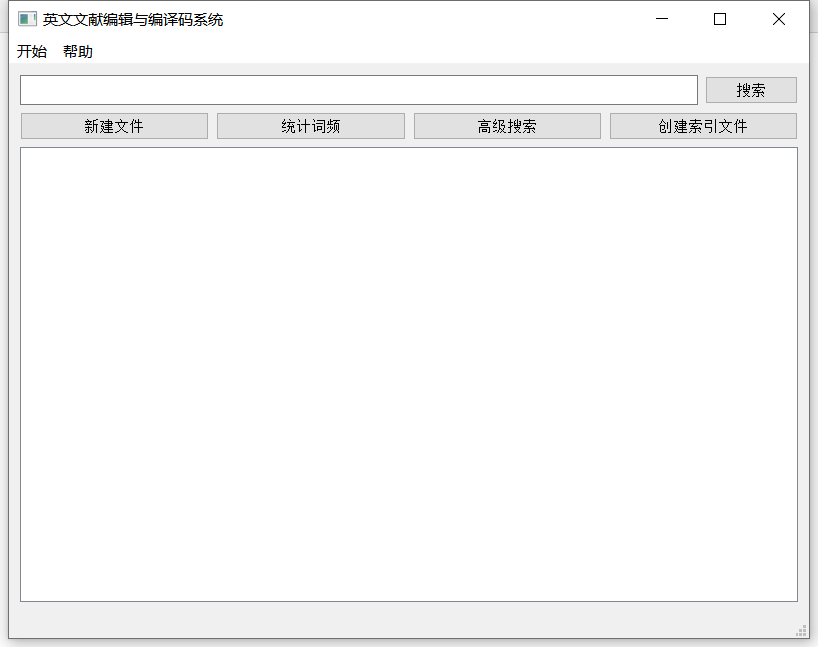
1.4 程序开发运行选用的环境

Python 3.9.1 64 bit

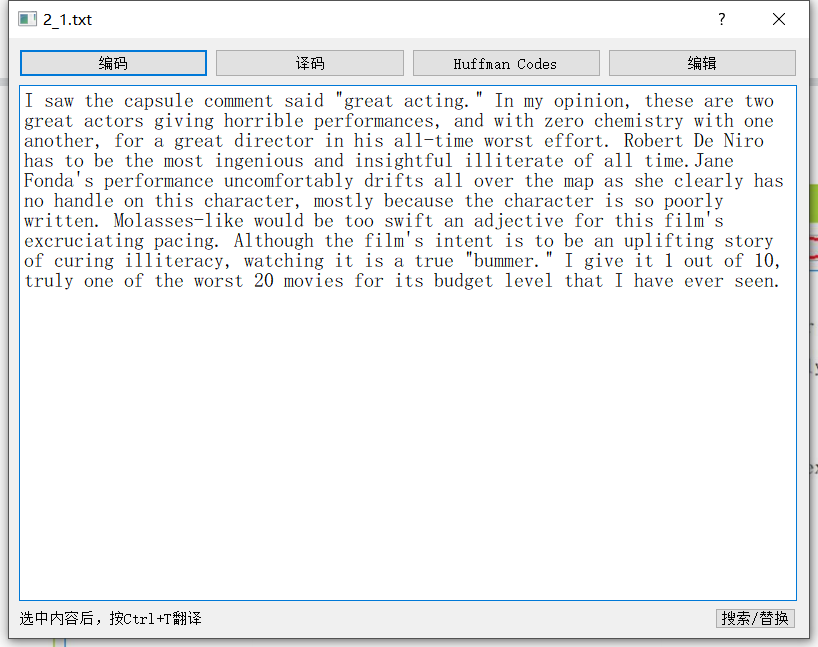
1.5 用户界面的设计

使用 PyQt5 库设计用户界面；主要窗口及功能如下：

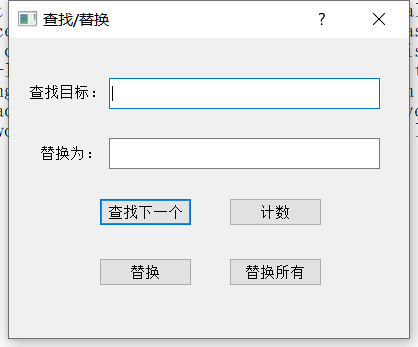
main\_UI：进入程序的 MainWindow，用来导入文件、搜索关键字等。



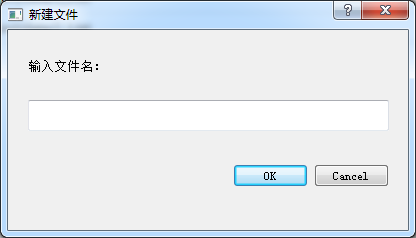
item\_UI：从main\_UI进入。主体为显示文章的文本框，同时有多个功能按钮。



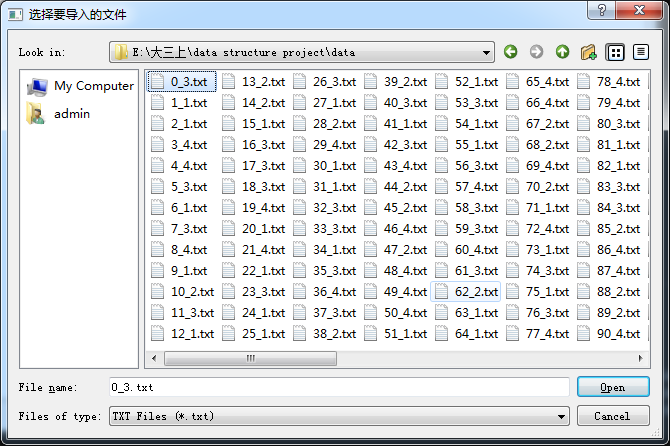
search\_UI：从item\_UI进入。可以实现搜索、替换、计数等功能。



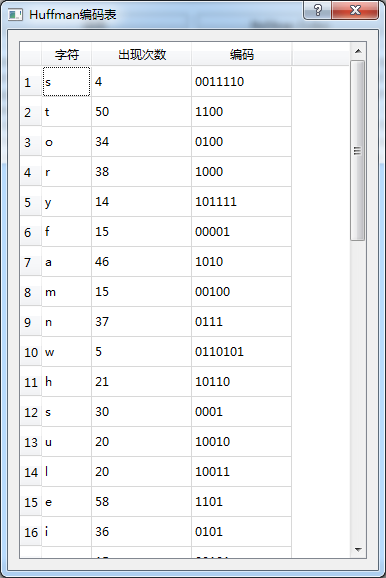
create\_file\_UI:从main\_UI进入。可以创建新文件。



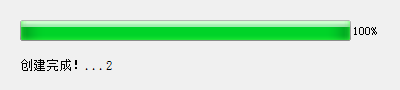
file\_UI:从main\_UI进入。可以导入文件。



huffman\_UI:从item\_UI进入。可以显示Huffman编码的具体情况。



progressbar\_UI:进度条。



2. 数据结构设计

2.1 所定义主要的数据结构

* + 1. Python内置结构

（1）list

进行列表建立，以及一些需要排序的操作，其他大部分数据结构无法排序，需要先转化成list，做排序再考虑转化回去，或者直接进行操作。用[]表示。

（2）dict

进行词典建立，使用键-值（key-value）存储，具有极快的查找速度，用于在不同目录下查找文本，不同文本下查找单词位置的功能，用牺牲空间的方式加快查找速度。用{}表示。

（3）tuple

与列表一样，也是一种序列，唯一不同的是元组不能被修改。用()表示。

* + 1. 哈夫曼树

给定n个权值作为n个叶子结点，构造一棵二叉树，若带权路径长度达到最小，称这样的二叉树为最优二叉树，也称为哈夫曼树(Huffffman Tree)。哈夫曼树是带权路径长度最短的树，权值较大的结点离根较近。本次构建的哈夫曼树的权值是文章中出现的所有字符的频率。并由此生成每个字符的编码，频率越高，其叶子结点距根越近，对应的编码也就越短。

* + 1. 索引结构

构建倒排索引，用来更快更方便的搜索单词或统计频率。索引结构如下：

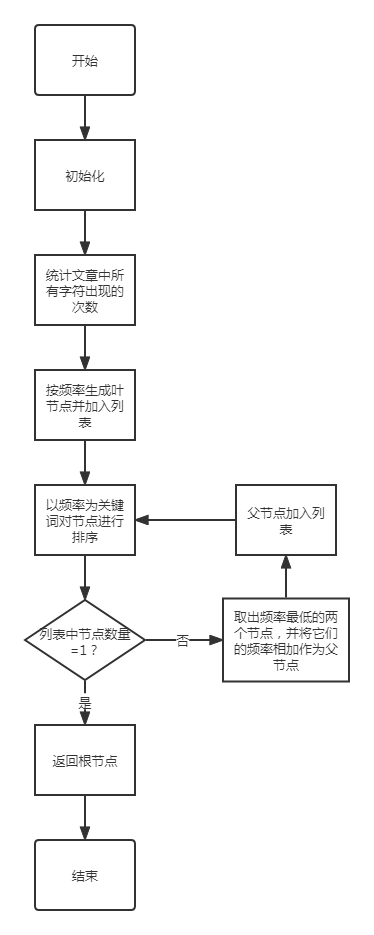
[(word1,[文件位置])(word2,[文件位置])…]

2.2程序整体结构以及各模块的功能描述

* main.py：总的main启动文件
* KMP.py:
  + - def get\_next(p)：寻找前缀后缀最长公共元素长度，计算字符串p的next表
    - def kmp(s, p)：核心计算函数，返回字符串p的位置下标
    - def positions(string, pattern)：查找pattern在string所有的出现位置的起始下标
    - def count(string, pattern)：计算pattern在string的出现次数
* Huffman.py
* class Node:
* def \_\_init\_\_(self, freq)：初始化属性left, right, father, freq
* def isLeft(self)：返回该节点是否是父节点的左节点
* def create\_Nodes(freqs)：根据频率表创建叶子结点
* def create\_Huffman\_Tree(nodes)：根据频率大小，使用队列创建哈夫曼树
* def Huffman\_Encoding(nodes, root)：用哈夫曼树生成哈夫曼对照表
* def cal\_count\_freq(content)：计算字符频率
* def cal\_Huffman\_codes(char\_freqs)：整合上述 functions
* File.py
* def search(files, keyword)：用KMP算法搜索计算所有传入的文件中keyword的数量，返回有序的文件及数量列表
* def cal\_words\_freq(files, reverse=True)：用KMP算法搜索计算所有传入的文件中所有单词各自的数量，返回有序的文件及单词列表及数量
* def cal\_words\_positions(files)：用KMP算法搜索计算所有传入的文件中所有单词各自的位置，返回有序的文件及单词列表及位置
* class File:
* def \_\_init\_\_(self, file\_pos)：初始化函数，传入文件路径作为属性
* def get\_content(self)：获得文件内容
* def set\_content(self, content)：将文件内容修改为content
* def get\_apperance\_num(self, word)：统计文件内容中word出现的次数
* def get\_huffman\_codes(self)：获得哈夫曼编码表
* def get\_encodeStr(self)：获得文件内容通过哈夫曼编码表编码后的字符串
* def get\_decodeStr(self, huffmanStr)：通过哈夫曼编码表将编码转换成原字符串
* main\_UI.py
* class UI:
* def \_\_init\_\_(self)：构造函数，将属性 search\_status 和 freqs 初始化
* def init(self)：设置事件触发
* def add\_files(self)：调用 PyQt5 的 FileDialog 选择要添加的文件
* def about(self)：“关于”窗口
* def create\_file(self)：创建文本文档，并添加文件
* def cal\_words\_freqs(self)：调用Cal\_Words\_Freq.py中的函数统计词频并生成显示降序列表
* def logic\_search(self)：实现扩展功能中的高级搜索
* def packaging(self)：调用Inverted\_Index.py中的函数生成索引文件，并保存为txt文件
* def clear\_list(self)：清空文件列表
* def get\_research\_content(self)：获得用户输入的需要检索的内容
* def get\_files\_from\_table(self)：获得当前文件列表
* def creat\_tableWidget(self, files, nums=[], poss=[])：在 GUI 中生成列表及相关信息
* def creat\_tableWidget1(self, files, nums=[])：在GUI中生成列表及相关信息
* def closeEvent(self, event)：窗口关闭时出发的关闭事件
* def search(self)：根据索引查找多个单词或词组，并显示频率及位置信息
* def buttonClicked(self)：在底部状态栏显示相关信息
* def itemClicked(self, row, col)：文件表单项目点击事件，打开新的 item 窗口
* item\_UI.py
* class item\_UI
* def \_\_init\_\_(self, file\_pos, keyword=None, keyword1=None)：初始化，将属性 file 初始化为file\_pos，同时高亮 keyword/keyword1
* def init(self, keyword, keyword1, filename)：连接按钮点击事件
* def highlight(self, pattern, color=”yellow”)：将所有 pattern（以’;’分隔）做黄色高亮处理
* def highlight1(self, pattern, color=”yellow”)：将所有 pattern做黄色高亮处理
* def hightlight\_specific(self, pos=(0, 0), color=”gray”)：按位置高亮
* def encode(self)：调用 self.file.get\_encodeStr()显示编码
* def decode(self)：调用函数进行译码
* def huffman\_codes(self)：显示哈夫曼编码表
* def edit(self)：将文本框控件变为可编辑
* def save(self)：将文本框中的文本保存到文件中
* def search\_substitute(self)：打开新窗口，查找或替换
* def translate(self)：翻译所选文本
* about\_UI.py：“关于”窗口
* freq\_UI.py：显示表单的窗口。用来显示词频统计
* huffman\_UI.py：显示表单的窗口。用来显示哈夫曼表
* file\_UI.py：用来选择文件的窗口
* create\_file\_UI.py：新建文档的窗口
* progressbar\_UI.py：进度条窗口
* search\_UI.py
* class search\_UI:
* def \_\_init\_\_(self, item\_ui)：构造函数
* def init(self)：连接按钮事件
* def prepare(self)：计算所有用户查询的单词的位置信息
* def next\_word(self)：高亮下一个搜索结果
* def count(self)：计数
* def substitute(self)：替换当前高亮的单词
* def substitute\_all(self)：替换所有符合条件的单词
* Cal\_Words\_Freq.py：计算词频并返回含有词频信息的词典
* Inverted\_Index：创建索引文件并返回索引

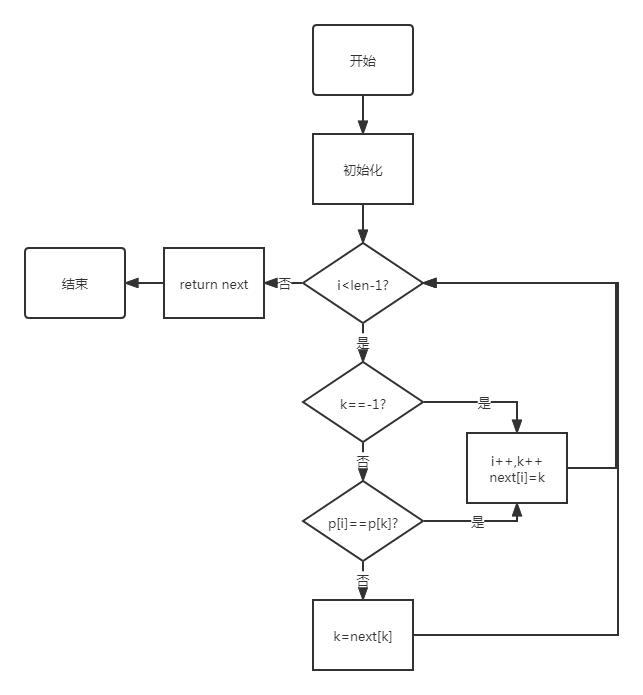
3. 详细设计

3.1 构造哈夫曼树

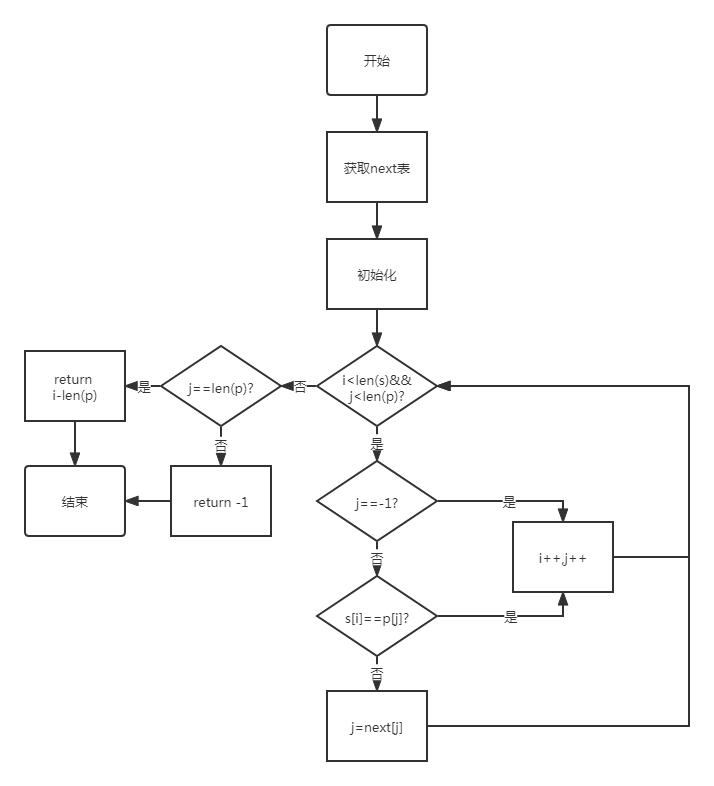


3.2 KMP算法

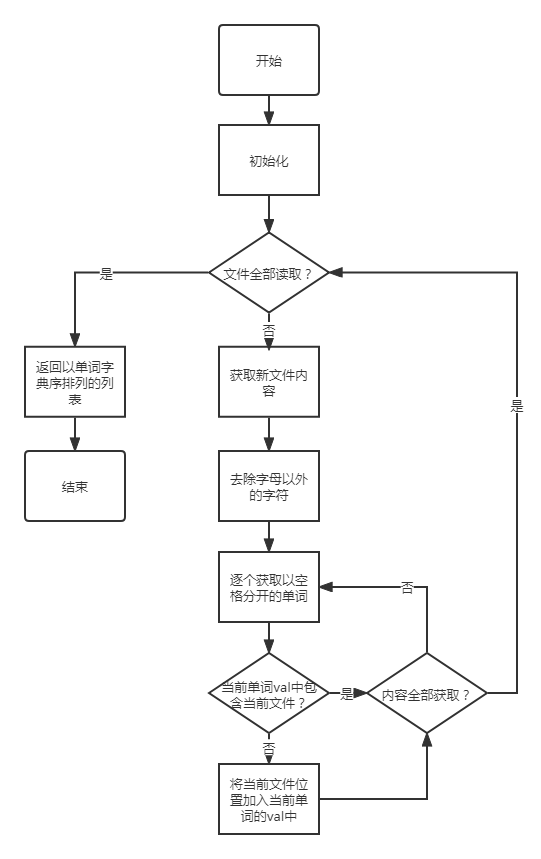
3.2.1 获取next表



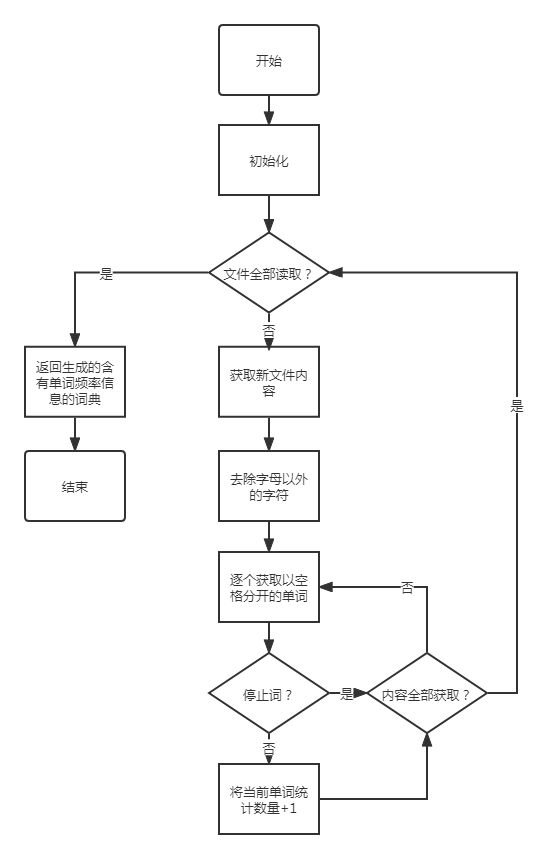
3.2.2 文本匹配



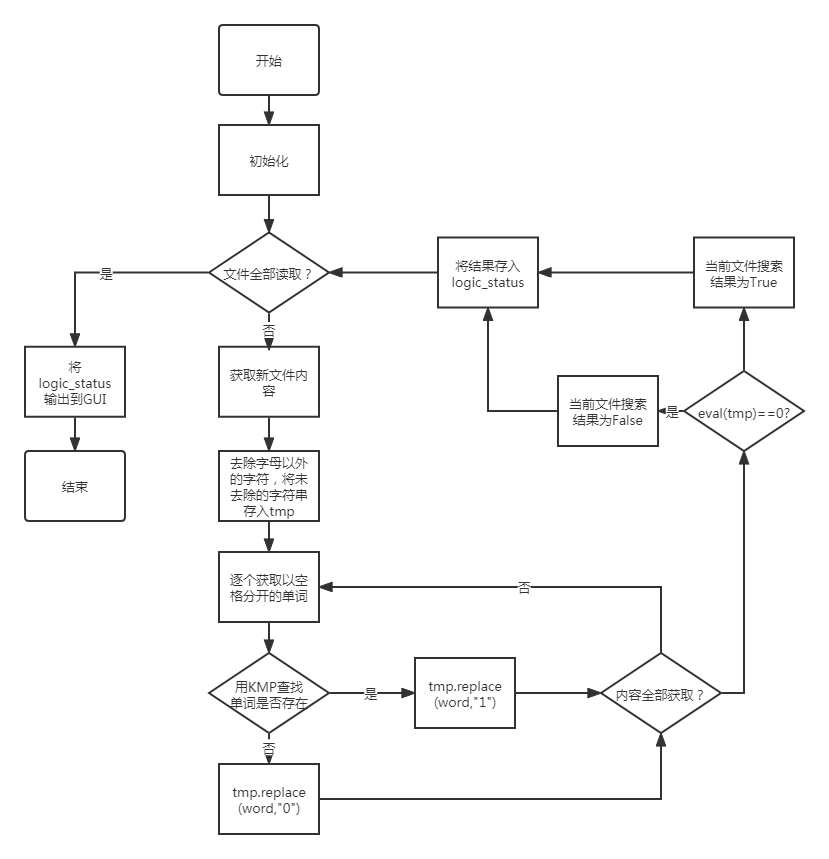
3.3 创建索引



3.4 统计词频



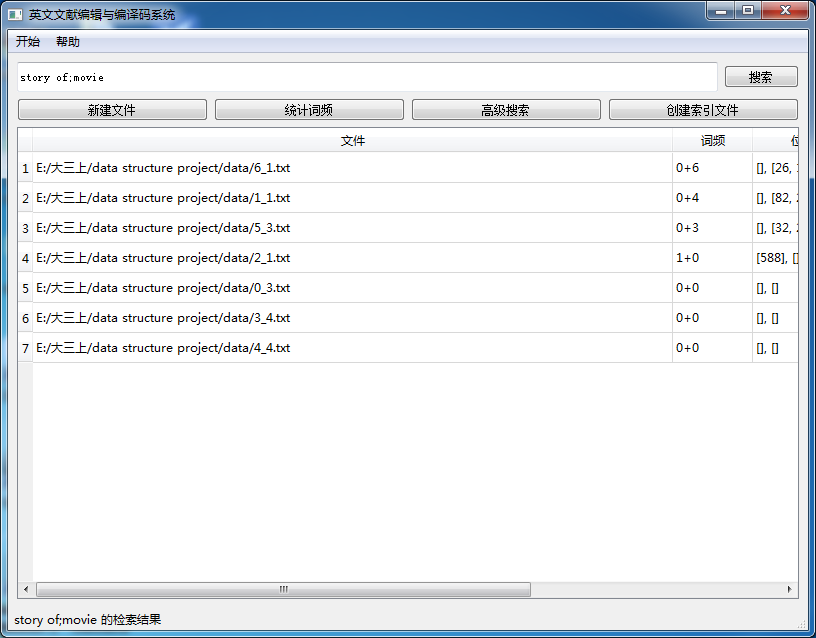
3.5 高级搜索



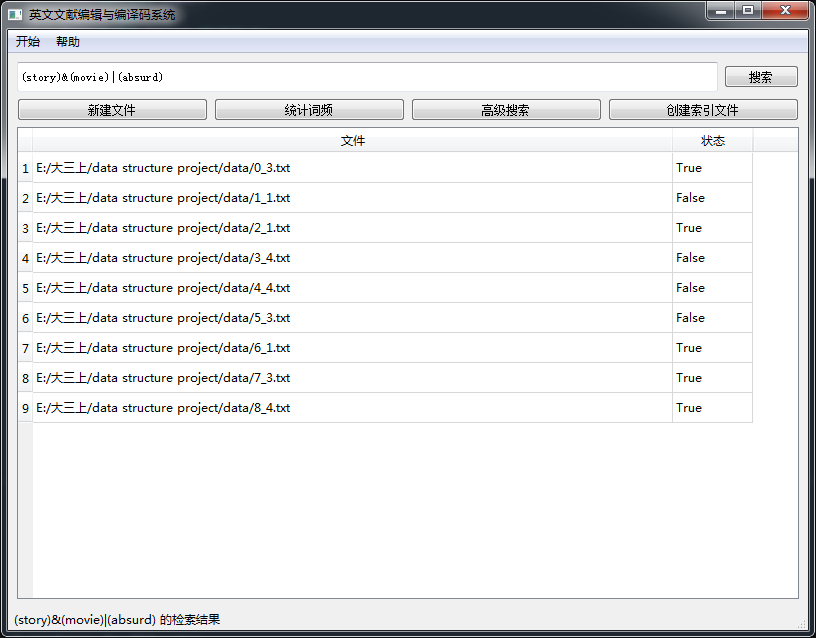
3.6 UI设计

3.6.1 主界面

搜索

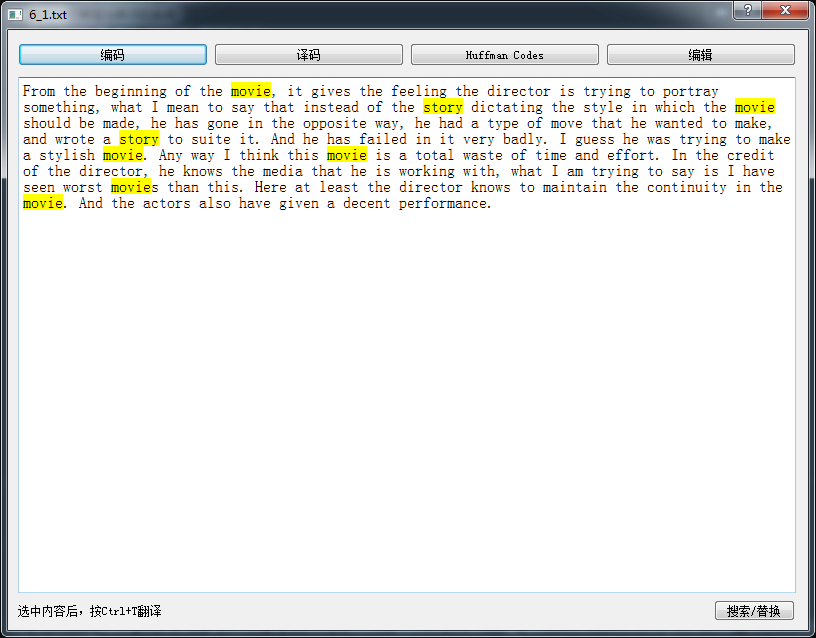


高级搜索

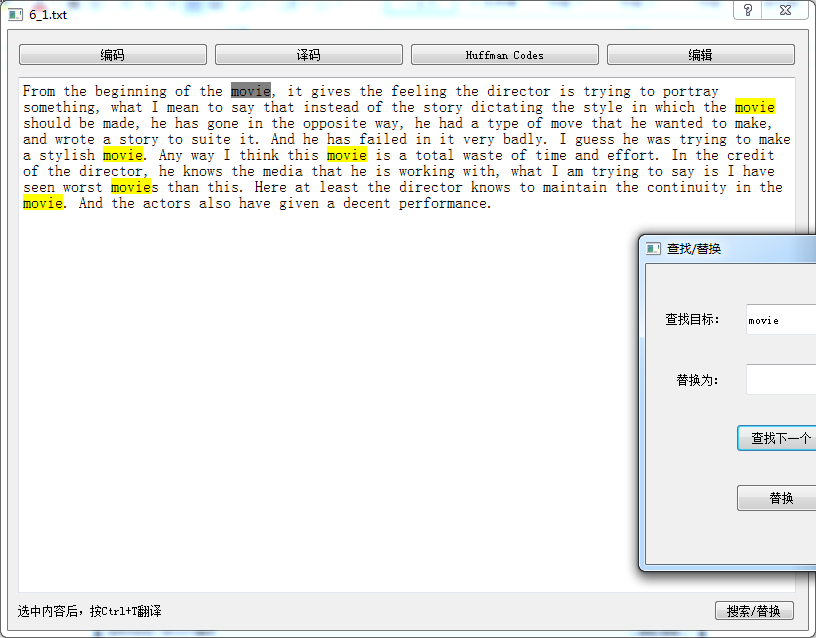


3.6.2 item窗口

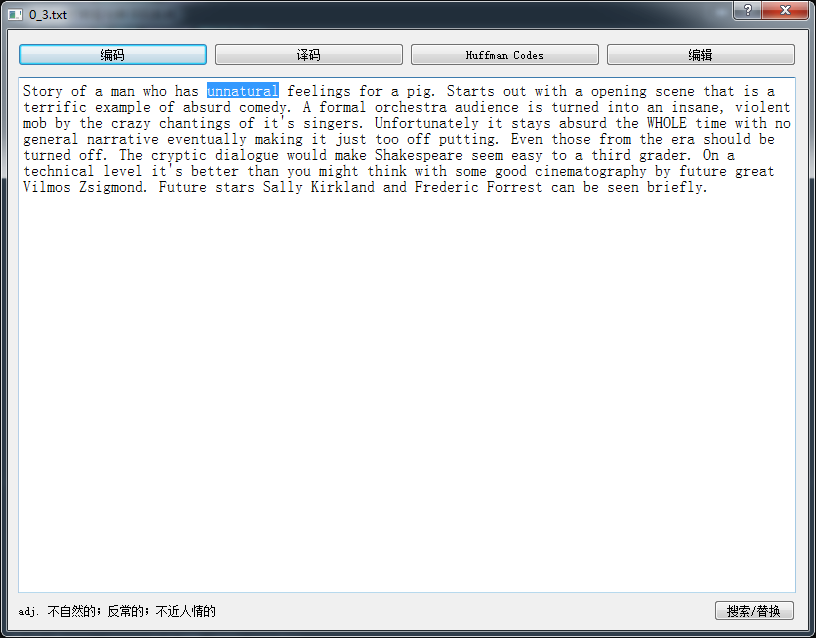
高亮



查找下一个



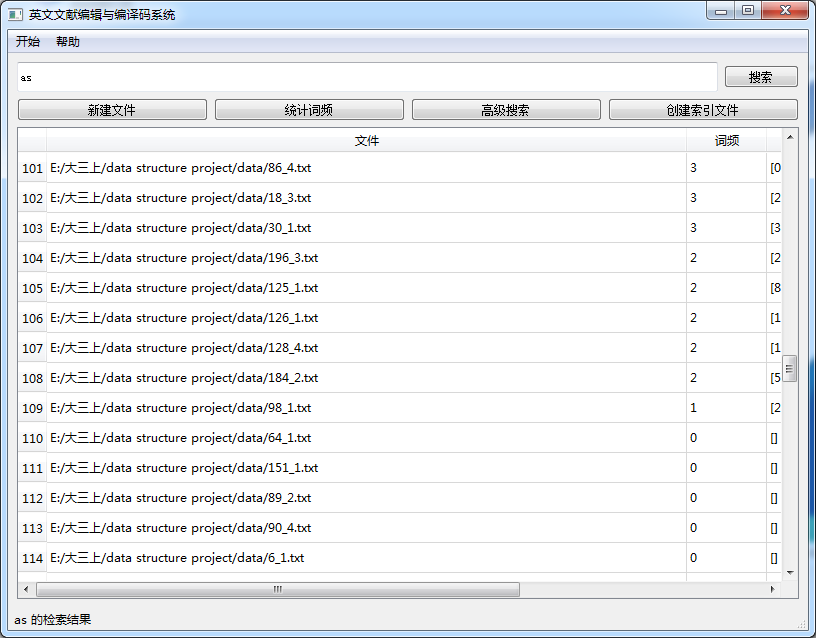
翻译

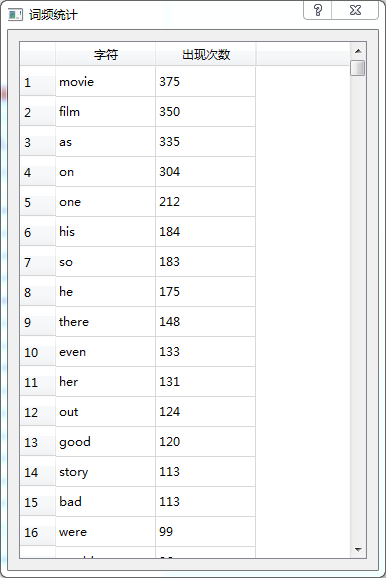


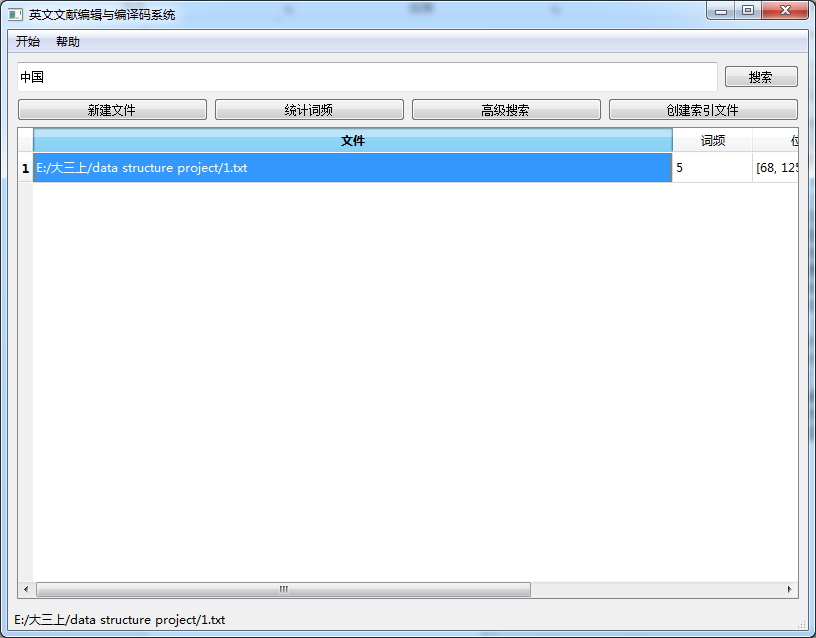
4. 测试

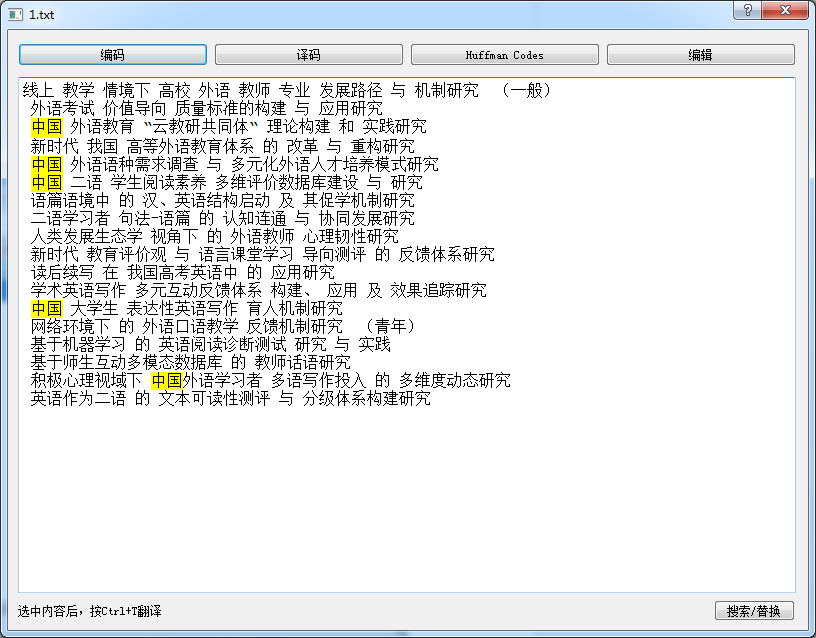
4.1 正确运行程序的用例

所有英文文本（如给定的data文件夹中的所有文本）都可以正确运行该系统所有功能。另外，以空格分隔词语的中文文本也可以正常运行。





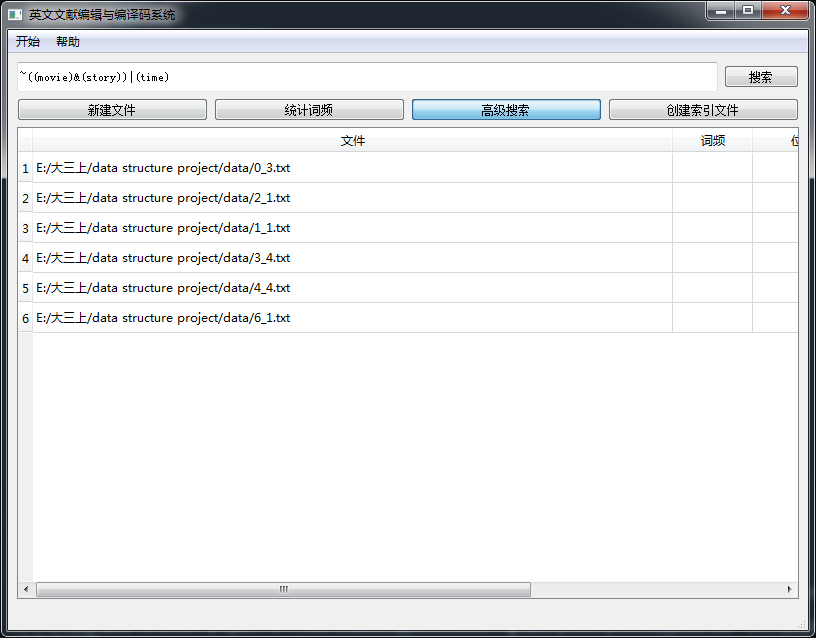


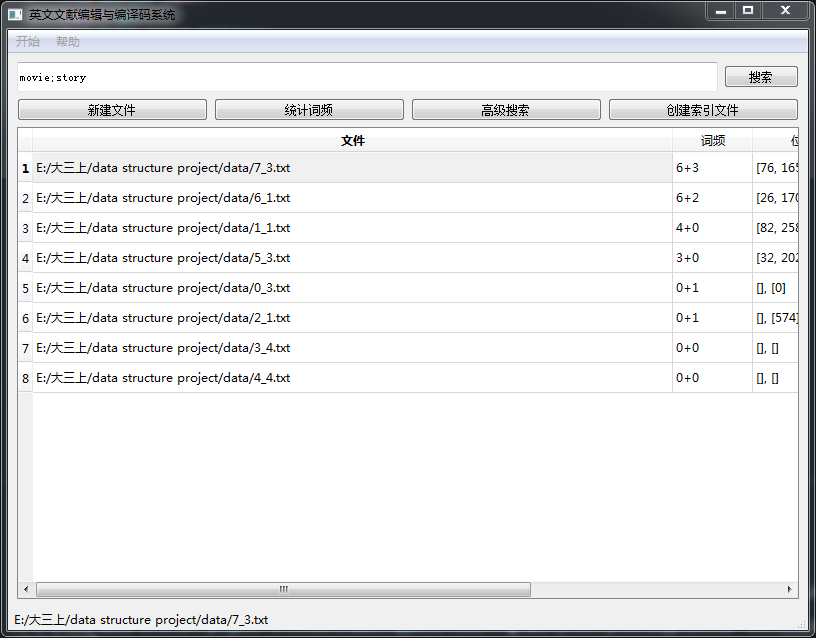


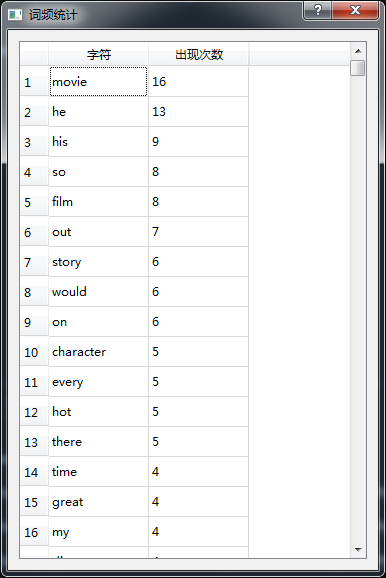


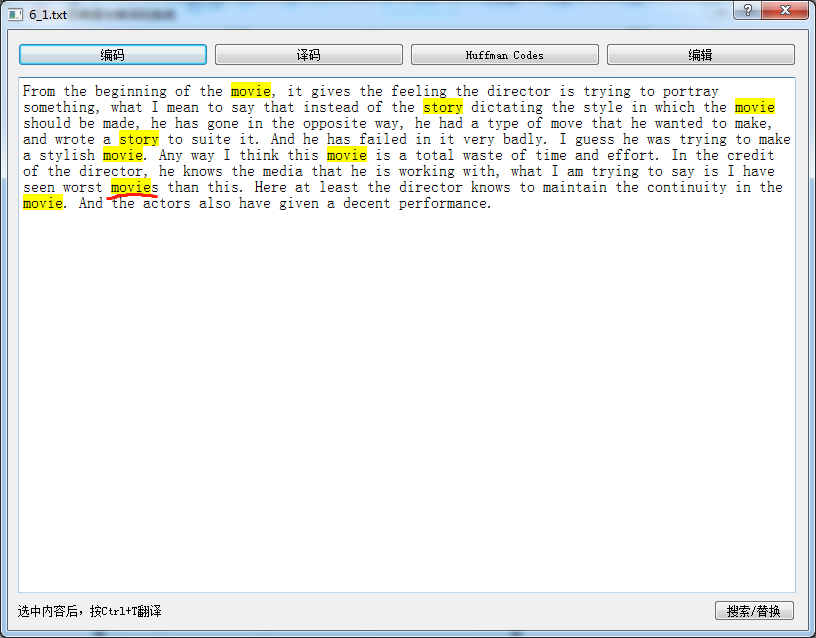
4.2导致程序运行错误的用例

* 使用高级搜索时，出现连续括号会导致程序无法正常运行
* 搜索部分带有复数的单词时，可能导致搜索结果不够准确









5. 总结与提高

当我刚拿到题目要求的时候，我的内心是抗拒的。倒不是因为算法很难实现，无论是KMP、Huffman编码还是倒排索引，在上学期的算法课和之前的算法竞赛都学习过。最让我头疼的是用户界面的实现，上一次用Java实现用户界面的过程实在是很折磨。经过一番调查和研究，最终我决定用Python的PyQt5库实现用户界面，QtDesigner可视化的编程方式真的很适合我。

本以为数据结构和算法部分进展会很顺利，没想到碰到了不少问题。首先是在KMP匹配时遇到的大小写问题。原句可能会有首字母大写的情况，这样的话就会让本来匹配的单词不再匹配。我的处理方式是把全部文本变为小写。其次，原句中的各种标点符号会对词语分割产生影响。经过各种测试后，我在预处理阶段把文本中的标点符号全部用replace函数去掉。最后就是对于单词复数的处理。我想了很多种方法，比如去除末尾s、复数和单数分别处理，都达不到很满意的效果，这也让我看到了KMP算法的局限性。如果有更多的时间，我回去更加深入的研究这个问题。

到了最麻烦的图形界面部分，最开始遇到的问题，每个程序进程只能运行一个MainWindow 类，在不知道的情况下打开多窗口一直报错，后来得以解决。随之而来的是文本的高亮问题，后来查到的方法是在文本框中控制虚拟光标选中目标然后更改其格式，以达到高亮的效果。

在完成大部分功能要求后，我开始想其他扩展功能。翻译功能是个很棒的想法。起初， 我想建立本地的词典资源文件，通过索引或二分法进行查找，但后来发现很难找到好用的文本词典资源。于是转而使用爬虫进行翻译，还好效果不错，只是需要联网。

另一个扩展功能本来是想通过自然语言识别来对文本进行一些因果分析，学院的Keystone课程我做的就是相关内容的实现。很遗憾，我们的成果并不足以分析英文文章的因果关系，只能对中文文本进行分析，且准确率并不高，最终我放弃了这个想法。

在整个编写课设的过程中，我主要学习了PyQt5的使用，对于以后编写简单的图形用户界面非常有帮助。同时，我也复习巩固了哈夫曼树和KMP算法，原来都是在竞赛中应对特定题目，这是第一次应用到自己的项目中，还是很有成就感的。我对于Python中list、dict的使用也更加熟练。不得不感叹，相比较于我常用的C语言，python的很多集成功能实在是非常方便，随着对Python的深入了解，我越来越喜欢上了这个语言。

对于自己完成课设情况的评价，我觉得我的完成度还不错，实现了老师要求的基础和拓展功能。如果要改进的话，可能更多会把注意力放在如何把各种功能的表现形式变得更美观。 非常感谢这种独自完成整个项目的机会，让我在各方面都有所提升，也很有成就感。我想，我收获的不仅是一个简单的文献系统，更重要的是，我收获了这种不断debug、敲代码最后终于实现需求的过程，相信这对于我将来的工作会有很大帮助。