面向英文文献的编辑与检索

学　　号__15071016__

姓　　名___吴逸飞___

指导教师___杜永平___

2017 年 09 月

目录

1 分析

1.1 需要完成的功能

1.1.1 基本要求

a. 设计图形界面，可以实现英文文章的编辑与检索功能。
b. 编辑过程包括：
   - 创建新文件；打开文件；保存文件。
   - 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。
   - 替换：将文章中给定的单词替换为另外一个单词，再存盘等。
c. 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。
d. 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 20 的排序结果。
e. 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，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）。

1.1.2 扩展要求

a. 界面设计的优化。
b. 对于编码与译码过程，可以自行设计其他算法。
c. 扩展检索：例如，可以实现多于 1 个关键词的联合检索，要求检索结果中同时出现所有的关键词。
d. 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。
e. 可以自行根据本题目程序的实际应用情况，扩展功能。

1.2 需要处理的数据

1.2.1 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。

1.2.2 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。

1.2.3 网络爬取的信息

从网站上爬取单词翻译信息。包括 HTML 页面。

1.2.4 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。

1.3 程序开发运行选用的环境

操作系统：

版次：Windows 10 家庭中文版

版本：1709

OS 内部版本：16299.125

语言：Python 3.6.3

库：PyQt5==5.9.2

bs4==0.0.1

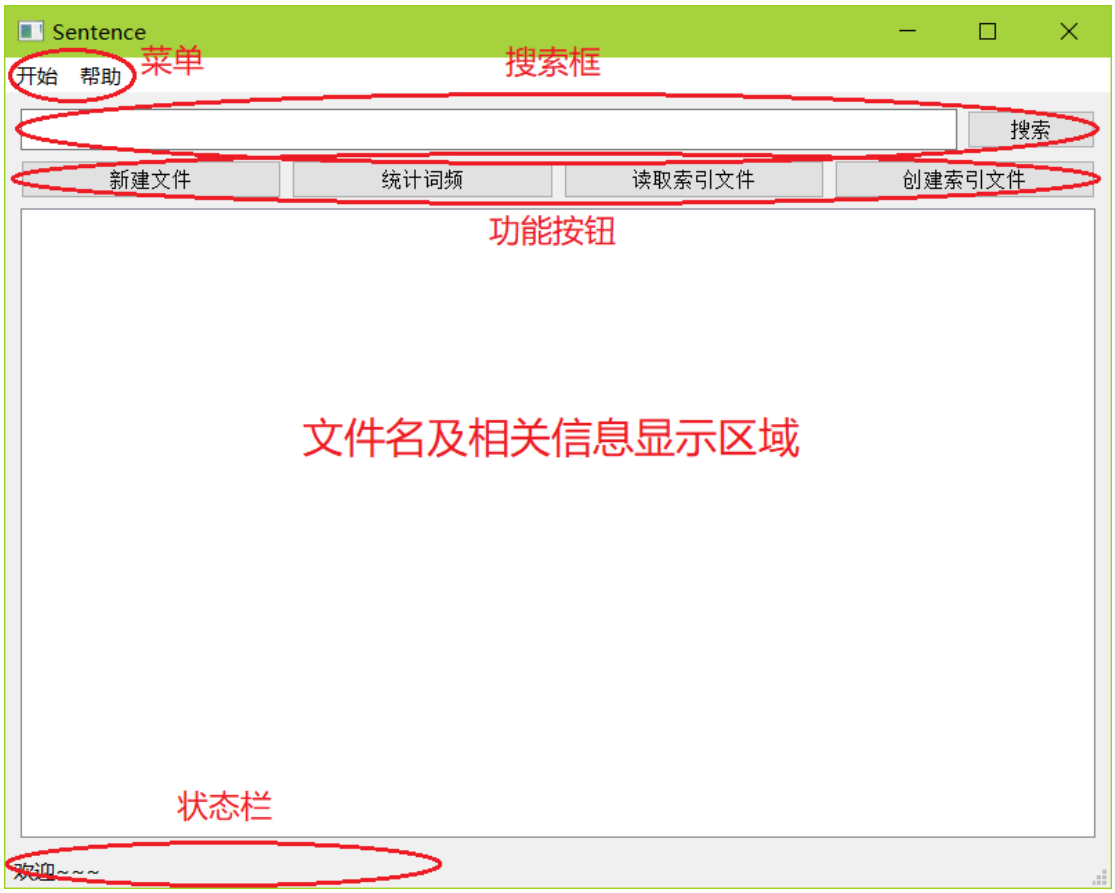
six==1.11.0

requests==2.18.4

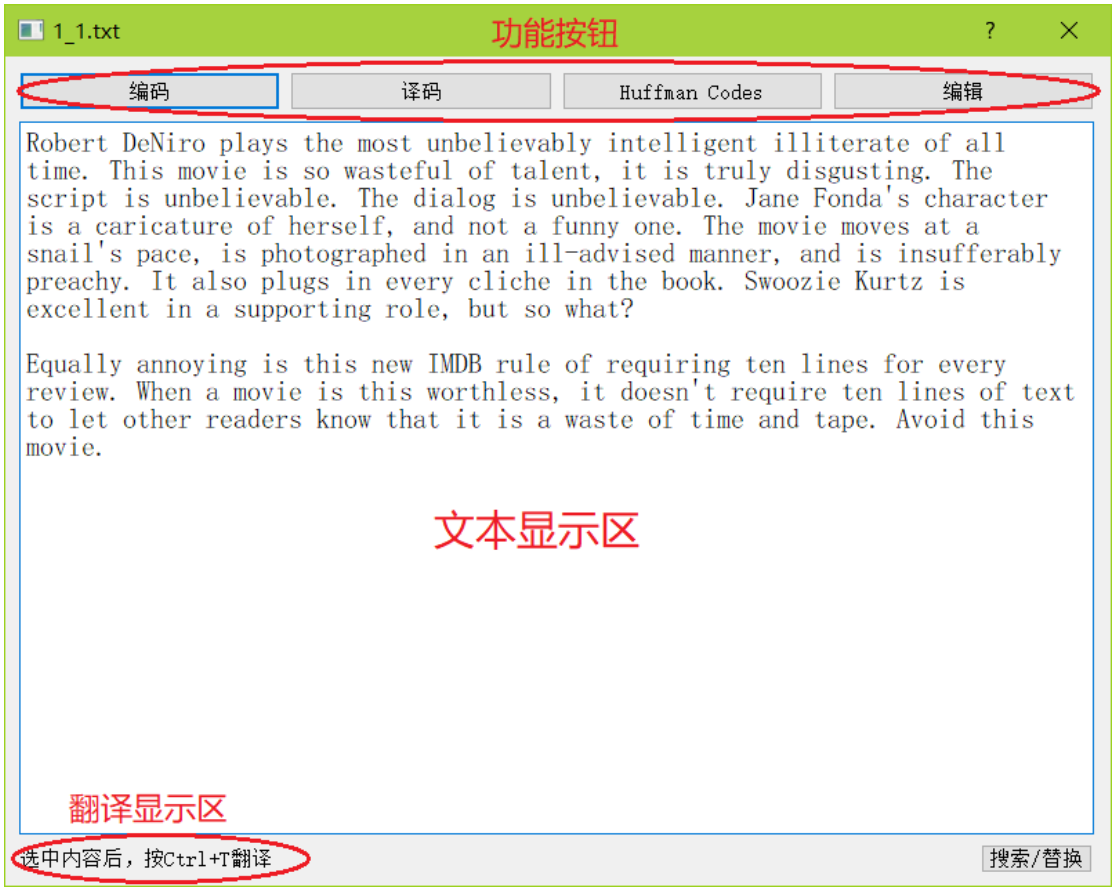
1.4 用户界面的设计

使用 PyQt5 库设计用户界面。主要窗口及功能如下：

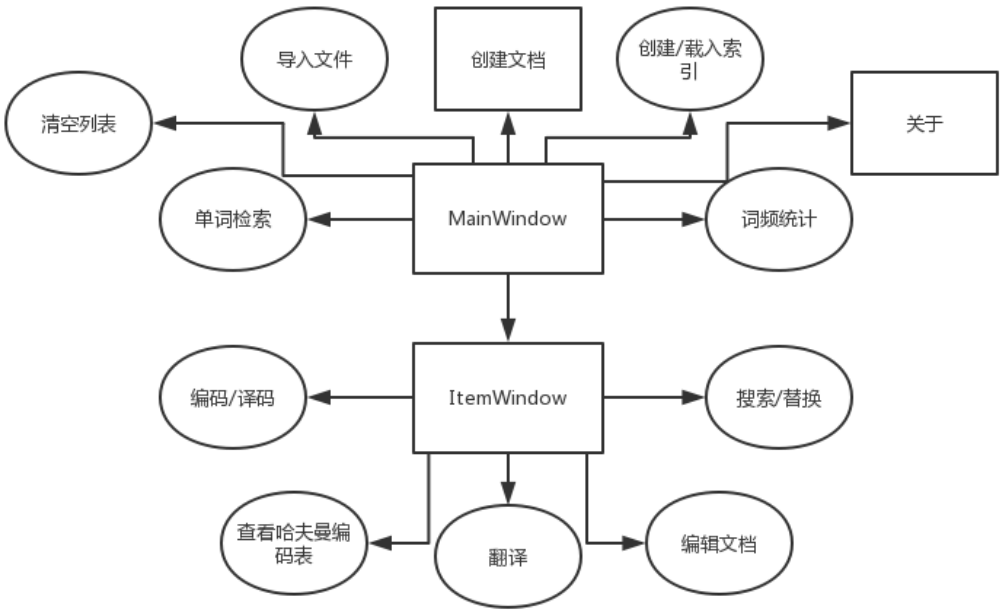
main：进入程序的 MainWindow，用来导入文件、搜索关键字等。



item：从 main 进入。主体为显示文章的文本框，同时有多个功能按钮。



1.5 主要流程图



2　数据结构设计

2.1 所定义主要的数据结构

2.1.1 Python 内置结构

a. 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(较麻烦),或者直接进行操作。用[]表示

b. 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。

c. 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

2.1.2 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(Huffman 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。

2.1.3 索引结构

构建的索引，用来更快更方便的搜索单词或统计频率。利用 Python 本身的 dict, list 以及 tuple 建立索引，结构如下：

[(文件地址 1, [{'word': 单词 1, 'pos': [位置下标]}, {'word': 单词 2, 'pos': [位置下标…]},…]), (文件地址 2, [{'word': 单词 1, 'pos': [位置下标]} , {'word': 单词 2, 'pos': [位置下标…]},…]),…]

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

img.py：资源文件。图片转为文本格式，以便于程序的封装。

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_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_huffman_codes(self)：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。

SplashScreen.py
|-class SplashScreen:
|-def __init__(self)：继承 PyQt5.QtWidgets.QSplashScreen。
|-def effect(self)：启动页的渐隐渐出特效。

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)：使用索引，统计词频并生成显示降序列表。
|-def load_package(self)：读取索引。
|-def packaging(self)：生成索引文件。使用 six 库中 pickle 打包成二进制文件。
|-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 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)：debug 所用，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)：初始化，将属性 file 初始化为 file_pos，同时高亮 keyword。

|-def init(self, keyword, filename)：连接按钮点击事件。

|-def highlight(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。

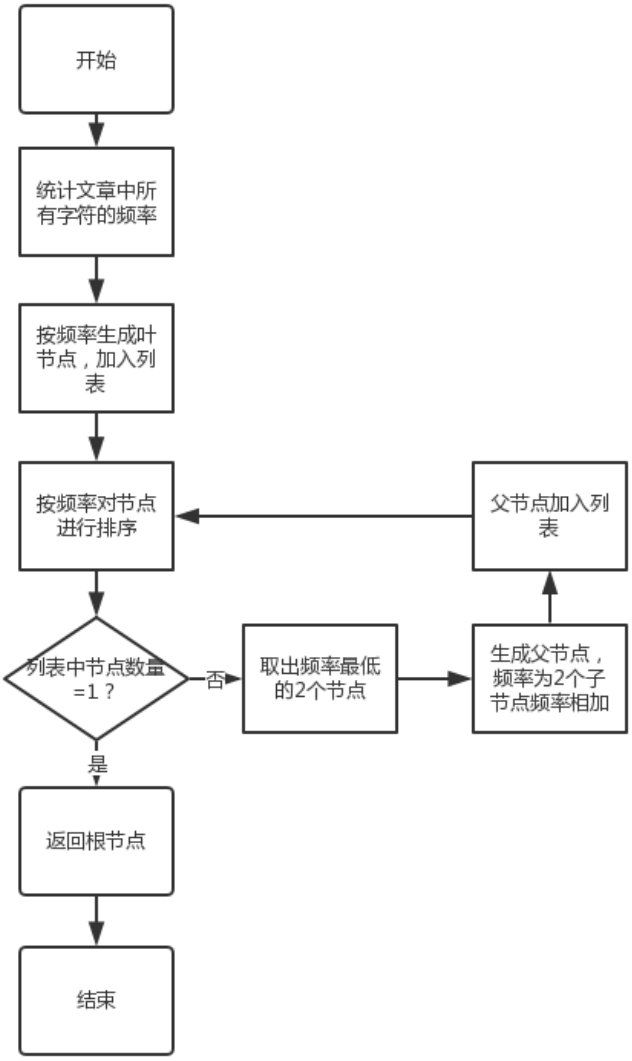
|-def count(self)：计数。

|-def substitute(self)：替换当前高亮的一个结果。

|-def substitute_all(self)：替换所有符合条件的结果。

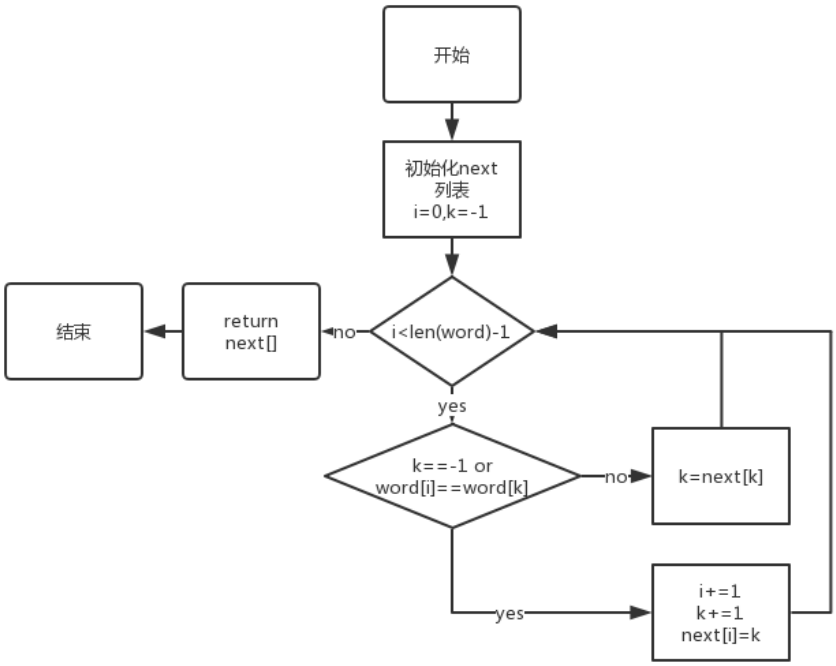
3　详细设计

3.1 构造哈夫曼树

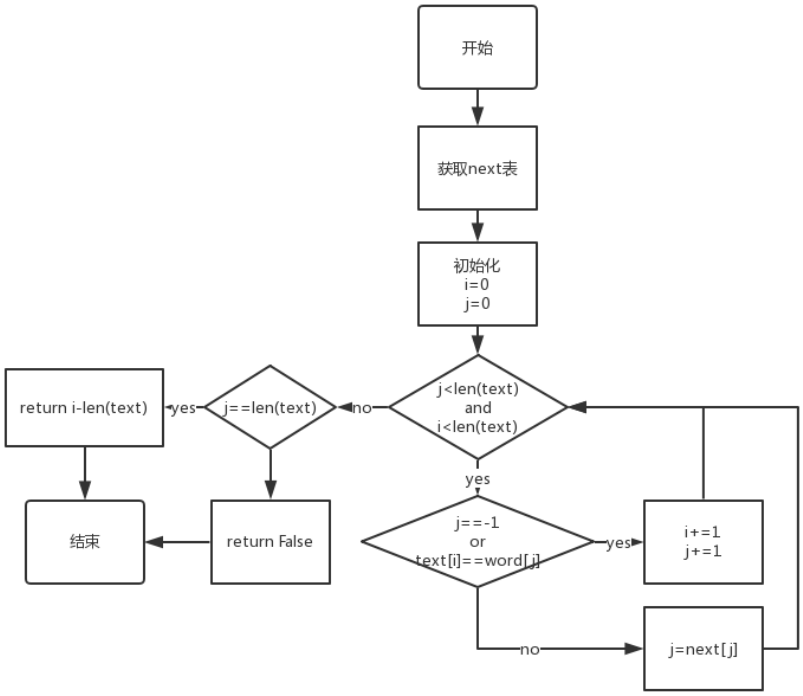


3.2 KMP 算法

3.2.1 获取 next 表



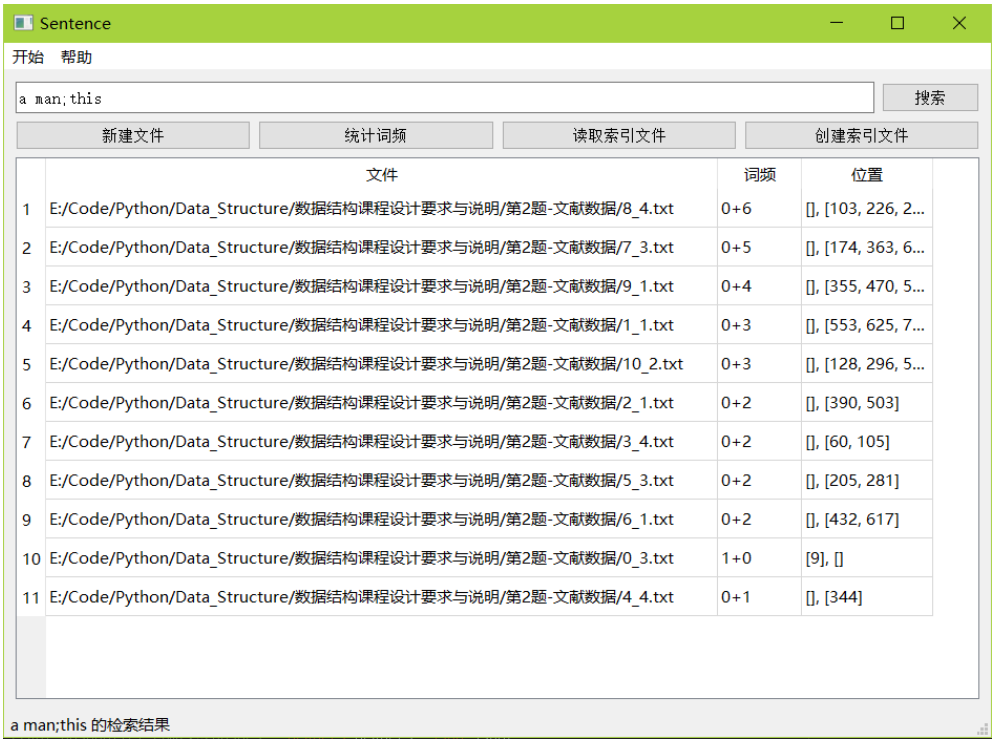
3.2.2 文本匹配



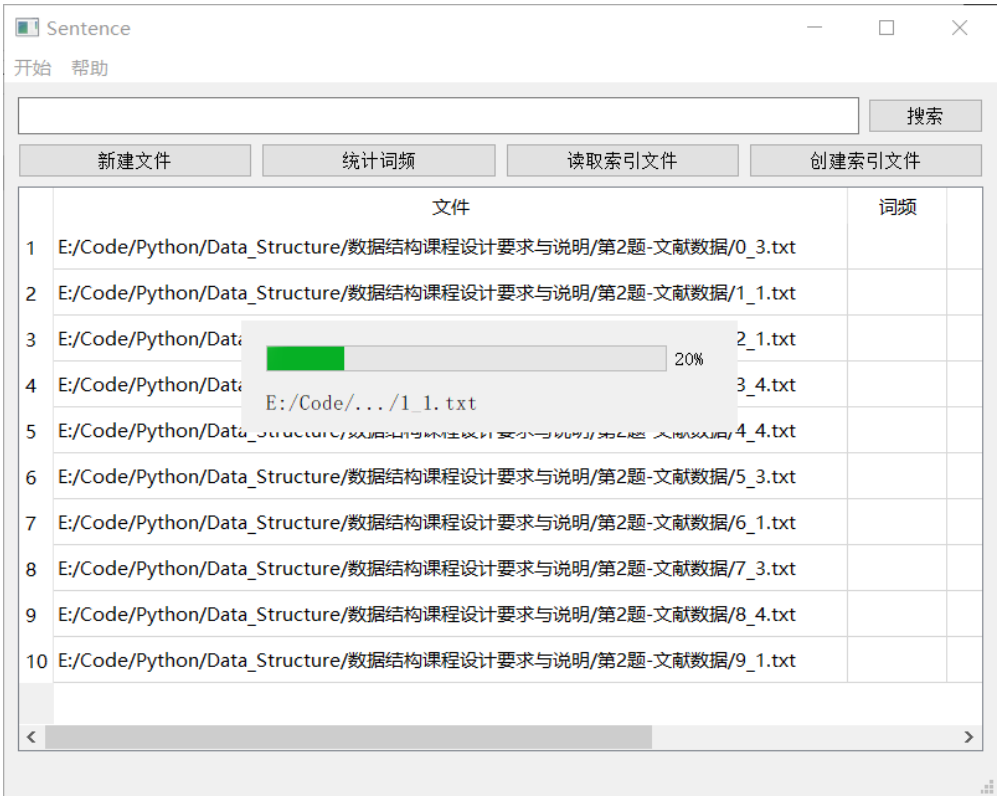
3.3 UI 设计

3.3.1 主界面

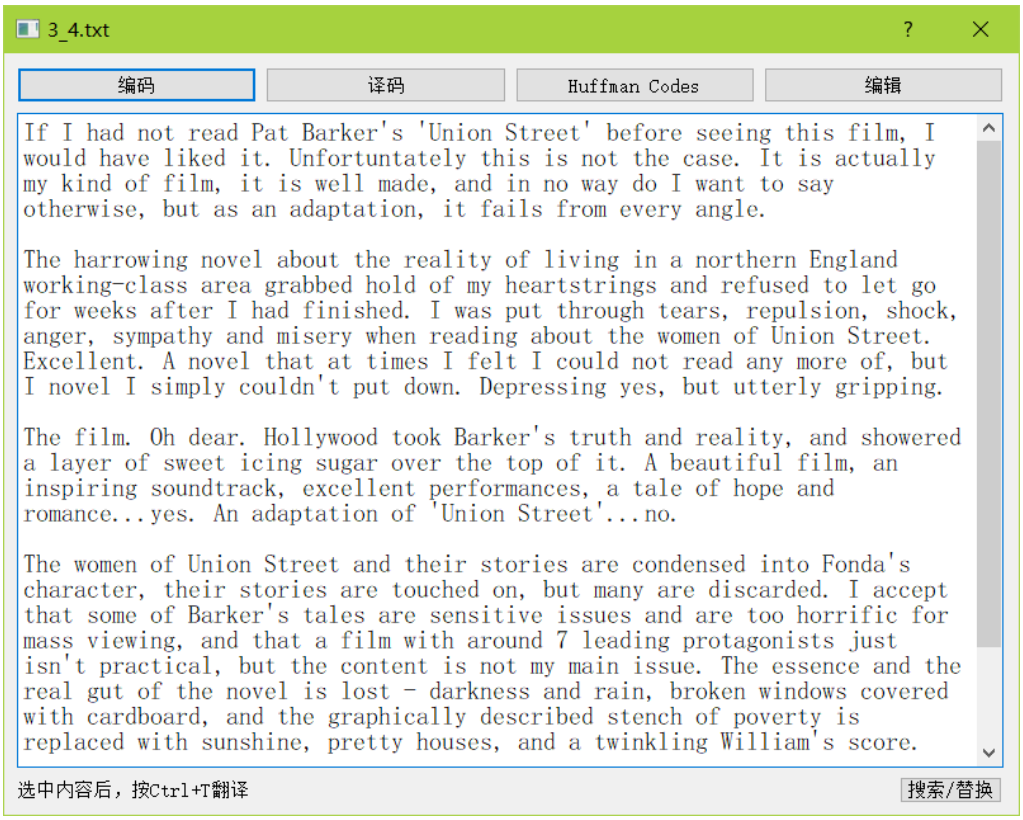
搜索多个单词和词组



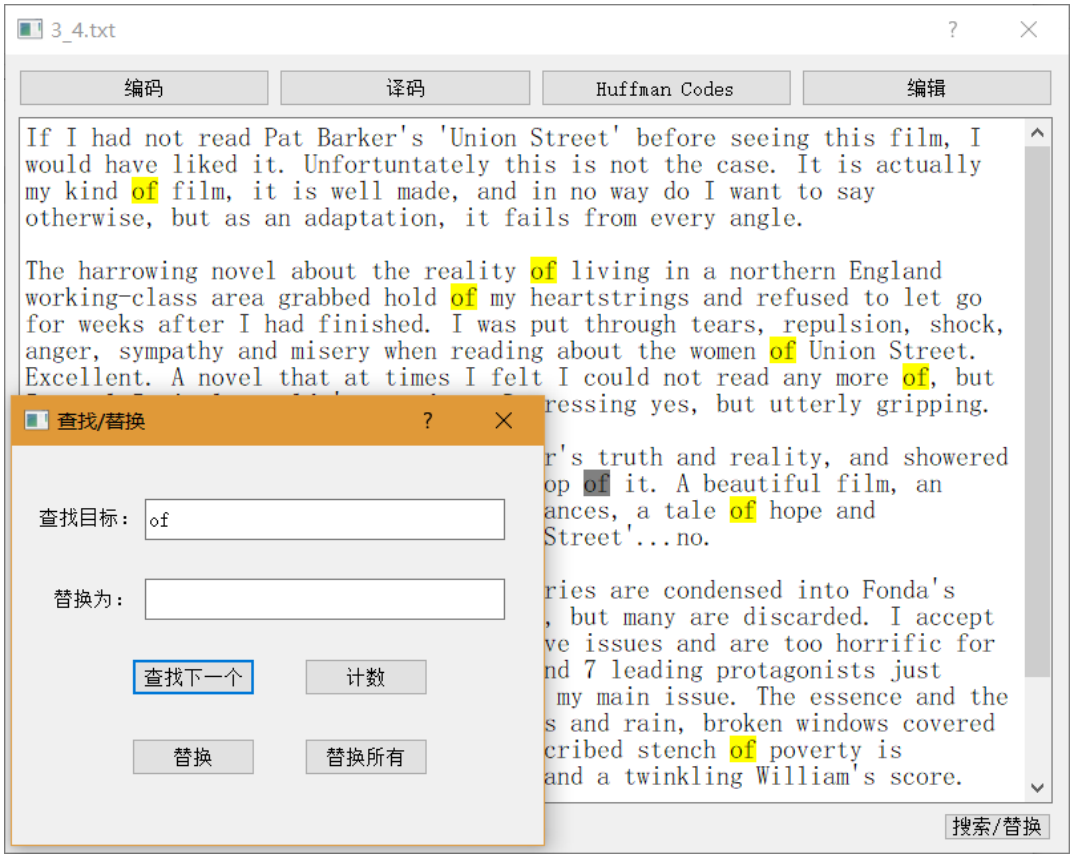
创建索引



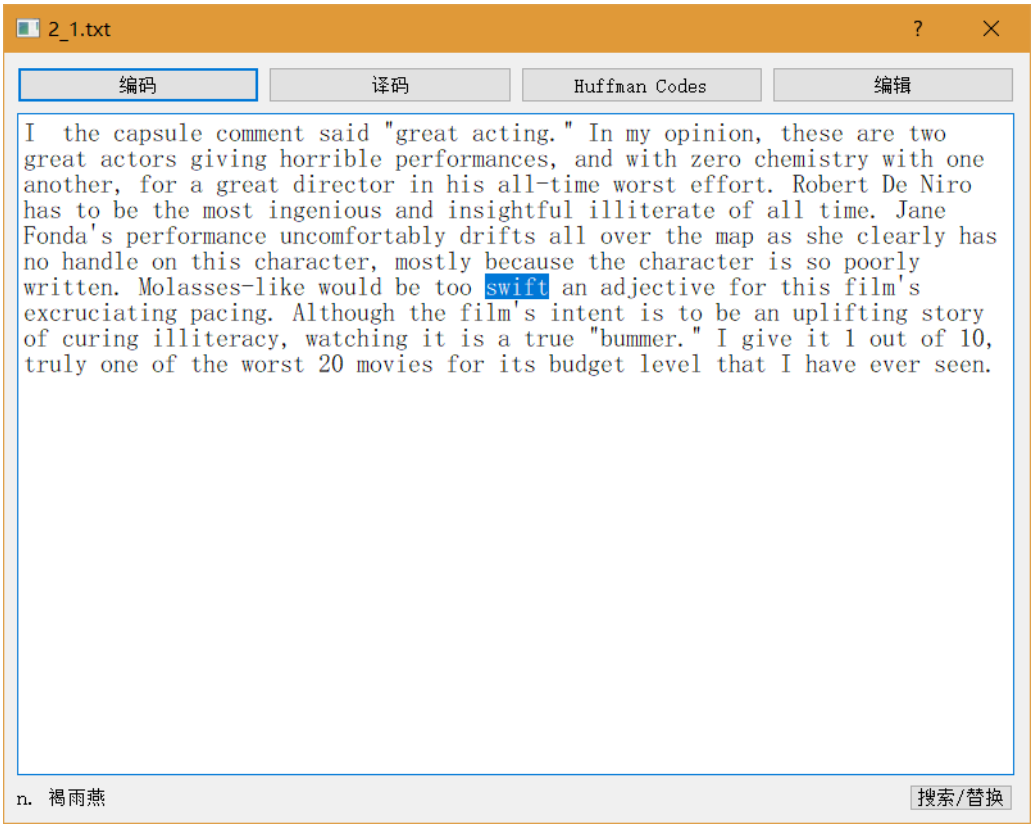
3.3.2 item 窗口



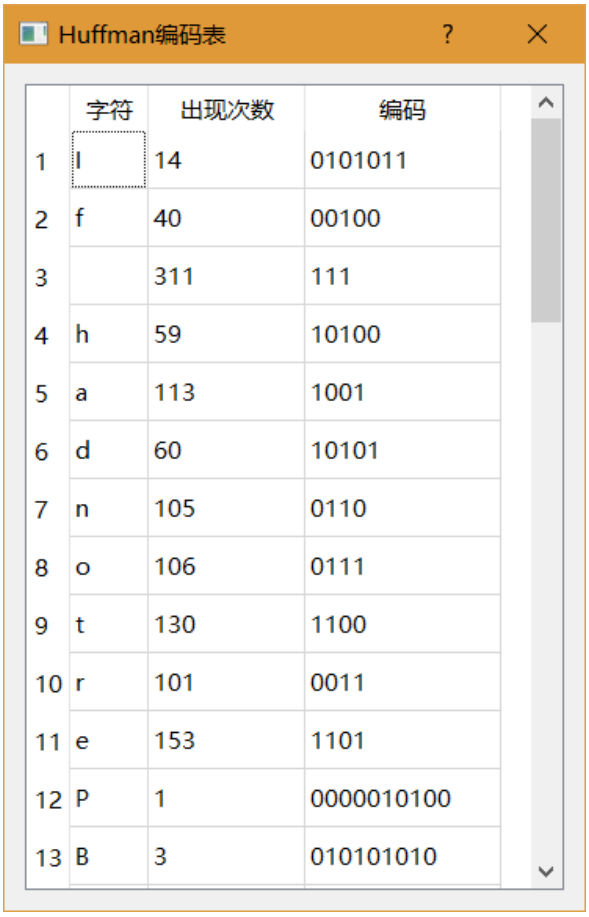
查找下一个



翻译功能



显示哈夫曼编码表



	字符	出现次数	编码
1	l	14	0101011
2	f	40	00100
3		311	111
4	h	59	10100
5	a	113	1001
6	d	60	10101
7	n	105	0110
8	o	106	0111
9	t	130	1100
10	r	101	0011
11	e	153	1101
12	P	1	0000010100
13	B	3	010101010

4 测试

4.1 测试用例

4.1.1 所给的多个英文文本文档

测试功能：文档的导入、列表清空、单词及词组查找、词频统计、建立和读取索引。

4.1.2 所给的单个英文文本文档

测试功能：哈夫曼树和和哈夫曼表的创建、哈夫曼编码、译码、编辑和保存、文本高亮、文本搜索、文本替换、翻译。

4.1.3 随机输入的字符串

测试功能：单词及词组查找、文本搜索、文本编辑、文本替换、文本高亮。
特殊用例：

a. 不输入字符串，直接运行。
b. 输入“
”等标签。

4.1.4 随机选择的字符串

测试功能：翻译。
特殊用例：

a. 不联网的情况下，程序是否会报错。
b. 不选择任何字符或选择空格等特殊字符。

4.2 测试步骤

a. 导入若干个文档，然后在当前目录下建立索引，清空表单。
b. 读取刚才建立的索引，观察文件名字及数量是否与之前相符。
c. 点击“统计词频”，观察是否正确。
d. 在搜索栏输入多个单词和词组，用“;”连接，观察搜索结果是否正确，排序是否正确。
e. 双击一行，进入该文档浏览窗口，依次点击 Huffman Codes、编码、译码检查哈夫曼编码功能是否正常。
f. 选中一个单词，用快捷键“Ctrl+T”进行翻译。
g. 点击右下角的“搜索/替换”或使用快捷键“Ctrl+F”，并在新出现的窗口中进行操作，检查功能是否正常运行。
h. 点击“编辑”，更改文本，再点“保存”，在外部窗口中查看该文档是否做出了对正确修改的保存。

4.3 测试结果

a. 导入的文档正确显示，索引建立正常。清空表单后，表格为空。
b. 文档数量及路径与之前相同。

c. 词频从大到小排列，显示正常。
d. 输入“apple;this man”，结果按大小排列，显示正常。
e. 正确打开窗口并显示文本，同时之前搜索的单词高亮处理正常。哈夫曼编码功能正确。
f. 选中单词，在联网的情况下翻译正确；不联网的情况下，准确显示未联网的提醒。
g. 窗口显示正常，“查找下一个”、“计数”、“替换”、“替换下一个”功能全部正常。
h. 修改文本，在程序外检查，已成功保存。

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——需要自己完成哈夫曼编码（包括统计词频，建立哈夫曼树，根据哈夫曼树计算对应的编码），KMP 算法（之前只是听说过，完全不清楚原理及算法），而且这一切都需要用图形用户界面来展现出来！于是只能硬着头皮上了。

最开始时，我决定从数据结构和算法入手，先不管图形界面。于是开始在网上学习 KMP 算法的原理及步骤。在和同学讨论的过程中，我们慢慢地结合着资料把代码写了出来，并测试了与传统算法的速度对比，很有成就感。

然后是哈夫曼树。这部分相对简单，因为其原理我们已经在上学期的课程中接触过，所以主要的时间都用来写代码。过程很顺利，达到了理想结果。

到了最麻烦的图形界面部分，开始时我本想使用 wxpython 库来实现，但在简单的了解过后，我发现这个库的更新情况不是很理想，尤其是它的 designer 和库的版本不同步，designer 生成的代码甚至直接运行会报错，于是我放弃了，转向更为成熟的、跨平台的 PyQt5。早就听说过 Qt 的大名，终于有机会使用下了。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，已达到高亮的效果。

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做通过自然语言识别来对文本进行一些情感分析或者全文概括，但在查阅部分资料后发现很难实现。首先，文本本身多为记叙文，很长且情感并不明显，其次需要的类似的数据集几乎找不到，所以最终放弃了这个想法。

最后是将整个程序封装成 windows 下运行的 exe 应用程序，使用了 pyinstaller 库。过程很简单，只需要在命令行进行一些操作就好了。

在整个编写课设的过程中，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。复习巩固了哈夫曼树的创建。学习了 KMP 这种字符串匹配的算法。同时我对于 Python 的使用也更加熟练。

对于自己完成课设情况的评价呢，我觉得我完成的还是不错的（哈哈哈）。虽然从美观方面来讲很一般，但是功能的实现以及代码的结构还是可以的，基本功能和扩展功能基本都做到了，后期的 Bug 调试也基本保证了正常使用过程中程序不会报错崩溃。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，各方面都有所提升，也很有成就感。能看着整个程序跑起来还是非常爽的！