基于文本内容的电影检索与推荐（第二部分）实验报告

吴佳龙 2018013418

一、 实验目标

本学期实验的最终目标是实现一个基于文本内容的电影检索与推荐系统，可以对电影网页进行信息提取和分词，并以此为基础建立倒排文档，实现电影查询及简单的推荐功能。

本实验的第二部分将通过第一部分实现的接口，对电影数据库进行信息提取和分词，并依据以上结果建立倒排文档，最终实现给定关键词的电影检索功能和给定电影名的电影推荐功能。

二、 实验环境

开发环境：64 位 Win10 系统，Visual Studio Community 2017，C++ 语言

三、 抽象数据结构说明

本次实验实现的数据结构如下：

1、 二叉平衡树 SplayTree

SplayTree 是一个模板类，模板参数 key_t, data_t 分别表示键和值的类型。

该模板类依据二叉平衡树的一种实现方式：伸展树（Splay Tree）算法 [1] 实现，能够在 $O(\log n)$ 的时间复杂度完成二叉搜索树的插入、删除、查找操作。

SplayTree 模板类实现的成员函数（运算符）有：

- 修改：adjust（调整二叉树使其平衡）, insert, remove
- 查询（索引）：search, find, at, operator [], list

2、 文档链表 DocumentList

DocumentList 类的链表节点存储了文档相关的信息：序号 id 以及文档在链表中的排序依据 rating 。在进行添加、修改和删除等操作之后始终保持链表依据 rating 降序排列。在本次实验中，rating 表示单词在文档中的出现次数。

DocumentList 类实现的成员函数（运算符）有：

- 修改：add, remove, edit（修改 id 对应的 rating，并维持有序）
- 查询：search（查找给定 id 对应的 rating）, empty
- 遍历：begin, end（返回迭代器类型，同时还实现了迭代器的 operator ++ 和 next 等操作）

3、 倒排文档 InvertedIndex

倒排文档由索引和文档链表组成，能够实现根据内容查询文档的快速操作。

InvertedIndex 是一个模板类，它的模板参数是索引数据结构类型 index_t。在本次实验中，index_t 可取 SplayTree 或 HashMap。

在 InvertedIndex 中 index_t 的 data_t=TermInfo，这一结构存储了单词，单词 id，单词出现文档数，单词出现总次数，以及对应的文档链表。

`InvertedIndex` 模板类实现的成员函数有：

- `inc`：在文档链表 `word` 中找到文档 `docId`，并将出现次数加 1
- `add`：在文档链表 `word` 中插入文档 (`docId`, `rating`)
- `search`：查找 `word` 对应的 `TermInfo`

四、 算法说明

1、 电影检索

建立倒排文档：读取电影文件，解析信息，并将电影名和剧情简介分词（结果输出至数据库，以便下次直接读取）。依据分词结果，通过索引数据结构，将该文档快速加入逐个单词对应的文档链表中。

检索：首先将用户给定的关键词进一步分词，然后依次查询分词后的各个关键词的文档链表。合并以上链表后，将检索得到的电影根据出现的关键词个数为第一关键字，总出现次数为第二关键字进行排序并输出。

2、 电影推荐

推荐：推荐依据电影信息加权算出，具体地：

$$\text{score} = \text{评分}/2 + 5 \times \text{电影类型 IoU} + \text{导演交集大小} + \text{top5 主演交集大小} + \text{标签交集大小} + \text{制片地区交集大小}$$

采用以下算法：根据电影的类型再建立一倒排文档，文档链表中的排序依据为电影的评分；推荐时，逐个遍历给定的电影的类型的文档链表，取出其中前 $\text{topK} \times 1.5$ 个节点，取并集后依据 score 排序给出前 topK 个作为推荐。

五、 输入输出及操作相关说明

在可执行文件同目录下应有配置文件 FilmContentSystem.config，否则将采用默认配置。

默认的 config 文件如右。与实验一相同的配置项的说明和要求详见实验一文档。其中 query1.txt , query2.txt 请以 UTF-8 编码存储，否则将不保证能正常运行。

只有配置示例的键值是有效的，请保证 config 文件的正确性，以及 config 文件应存储为 UTF-8 编码，否则将不保证能正常运行。

在 INPUT_DIR 下已经放置了数据库文件（电影文件、解析和分词结果等），请不要修改。

```
DICT_PATH = "dict/dict.txt"
HMM_PATH = "dict/HMM.txt"
STOP_PATH = "dict/stopwords.txt"
USE_HMM = true
USE_STOP = true
INPUT_DIR = "input"
OUTPUT_DIR = "input/info"
RETRIEVAL_INPUT = "query1.txt"
RETRIEVAL_OUTPUT = "result1.txt"
RECOMMEND_INPUT = "query2.txt"
RECOMMEND_OUTPUT = "result2.txt"
```

1、批量检索和批量推荐 query.exe

在以上都确保无误的情况下，点击 query.exe 运行即可。在屏幕上可能会打印一些日志信息，这是正常的。

query1.txt 为一行一个查询，关键词之间使用空格分开；result1.txt 为每行多个用空格隔开的 (urlID,occurTimes)，分别表示电影序号和关键字出现总次数。

query2.txt 为一行一个查询，为电影名；result2.txt 为一行 5 个空格隔开的推荐 (docID，newsName)，分别表示电影序号和标题。

注：检索的结果依赖于分词结果，可能与直接字符匹配的结果不同。比如，“泰坦尼克”不作为一个单词，可能不能检索到较合理的结果，而“泰坦尼克号”则可以检索到合理的结果。

2、 图形用户界面 gui.exe

图形界面采用标签页的形式，共有三种标签页：

- 主标签页：由图标和输入框组成，若输入与数据库中电影标题完全匹配，则直接跳转至对应的电影标签页；否则跳转到检索结果。
- 检索结果标签页：点击电影标题可跳转至电影标签页。
- 电影标签页：点击推荐的电影可跳转至对应的电影标签页。

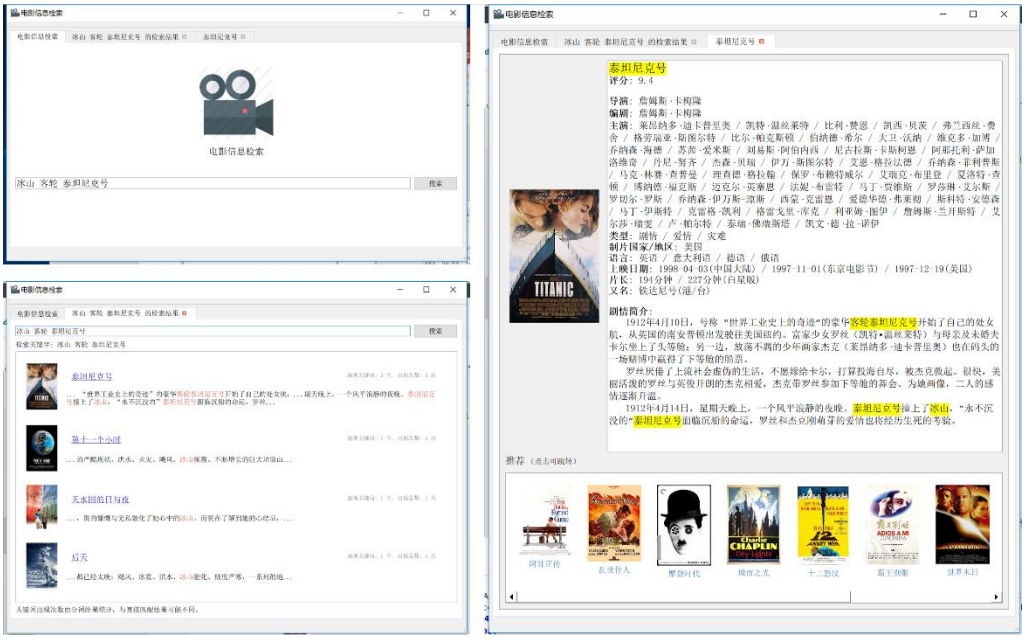

图 1 图形用户界面

六、 流程概述

query.exe 的执行流程由 `FilmContentSystemApplication` 的 `run` 函数实现，具体如下：

- 载入配置 `loadConfig` -> 载入字典 `initDictionary` -> 载入数据库 `loadDatabase` -> 倒排文档 `buildIndex` -> 批量检索 `doRetrieve` -> 批量推荐 `doRecommend`

gui.exe 的执行流程如图：

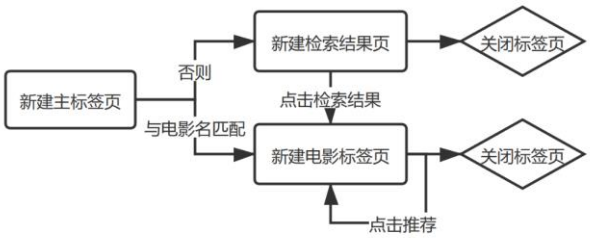

图 2 GUI 执行流程

七、 实验结果

1、 批量检索和批量推荐 query.exe 和用户图形界面 gui.exe

批量检索和批量推荐 query.exe 的输入输出运行结果如下，结果符合预期。

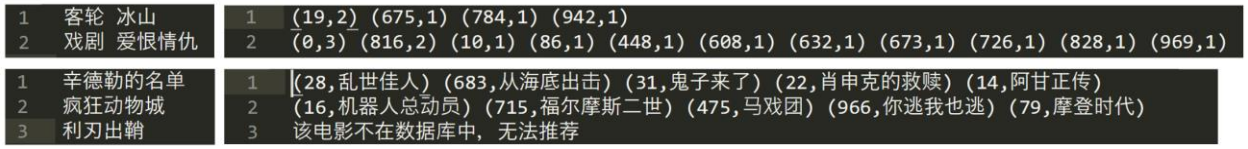

图 3 左上到右下依次为 query1.txt result1.txt query2.txt result2.txt

用户图形界面 gui.exe 的运行结果见图 1，结果符合预期。

2、 比较平衡二叉树和哈希表的效率差异

随机生成 100000 行的 query1.txt 和 query2.txt，比较两种索引结构在建立索引、查询效率等方面的差异，进行 5 次实验，结果如下：

实验编号	SplayTree			HashMap		
	建立索引	批量检索	批量推荐	建立索引	批量检索	批量推荐
1	0.146	6.631	4.347	0.095	6.623	4.34
2	0.162	6.813	4.303	0.099	6.641	4.355
3	0.164	6.79	4.335	0.084	6.653	4.392
4	0.146	6.71	4.329	0.096	6.652	4.381
5	0.155	6.715	4.354	0.095	6.716	4.409
平均	0.1546	6.7318	4.3336	0.0938	6.657	4.3754

结论：

- 在建立索引时，HashMap 的效率稳定地高于 SplayTree；
- 在批量检索、批量推荐时，由于索引并不是算法效率的瓶颈（主要是排序和值拷贝），两者的时间差异并不显著。

八、 功能亮点说明

1、比较平衡二叉树和哈希表在建立索引、查询效率等方面的差异；
2、友好的用户界面。

九、 实验体会

本次实验是自己第一次运用面向对象的编程思想，从最简单的数据类型逐步实现数据结构，最终形成一个用户友好的应用；这一过程加深了对 C++语言的理解，巩固了对面向对象技术的掌握，从而对以后编写同一级别的应用更具信心。

参考资料

[1] Splay tree – Wikipedia, https://en.wikipedia.org/wiki/Splay_tree.