基于文本内容的新闻推荐与检索（2）实验报告

牛浩宇 软件 72 2017010729

1 实验目标与环境

本实验的目标是利用实验一所提取的。我在开发时使用的操作系统是 macOS Mojave 10.14.2，使用的 IDE 是 Clion 2018.3，编程语言使用 C++11。经更换环境验证，程序在 Windows 10 + Visual Studio 环境下也可以正常运行。

2 抽象数据结构说明

本实验中，我主要实现了 docList, AVL 树 BalTree，docHash, wordlist 几种数据结构。

2.1 docList 类

此类即为所要求实现的文档链表结构。

一个本类对象包含两个 docListNode *_front/_node 用于存储链表头尾，一个 CharString term 用于记录这个文档链表存储的是哪个单词的数据，int termID 用于存储单词 ID, int docNumber 和 int occur 分别存储单词出现在多少篇文章中（即文档链表的长度）和单词在所有文档中总共的出现次数。

本类主要包含的函数有构造函数以及实验所要求各函数。对于 add，要求传入新插入文档的 docID 和该文档中本单词的出现次数_times；对于 search，要求传入所查询文档的 docID，查找成功则返回对应的 docListNode*，失败则返回空指针 nullptr；对于 edit，要求传入所修改的文档的 oldID，修改后的 newID，以及修改后文档中单词的出现次数 newTimes；对于 remove，要求传入需要删除的文档的 docID，如果找不到对应文档就不删除。

2.2 BalTree 类

此类即为所要求实现的平衡二叉树结构，具体采用 AVL 树实现。同时，每个 BalTreeNode 节点中都包含一个 Queue<docList>，即文档链表（单词）的链表，从而实现了倒排文档的构建。

一个本类对象仅包含一个 BalTreeNode *root 用于存储树的根节点。更进一层地，一个 BalTreeNode 类对象，包含三个 BalTreeNode *parent/leftChild/rightChild 分别存储双亲/左孩子/右孩子节点，int termID 用于存储当前节点的 ID，int nodeHeight 用于存储当前节点的高度（规定最底层高度为 1，向上递增），以及一个 Queue<docList> words 用于存储同一 ID 的不同单词，这里的 Queue 是在实验一中我自主完成的数据结构。注意，由于单词的 ID 可能重复，因此同一节点只能保证 ID 唯一，其内部可能还有若干不同单词，类似于开链法构建的 HashTable，因此采用了 Queue<docList>。

对于 insert，要求传入单词 CharString &word 和首次出现的文档 docID；对于 search，要求传入开始搜索的根节点 node 和需要查找的节点 id；对于 searchWord，要求传入需要查找的单词内容 target（从根节点 root 开始搜索）；对于 adjust，要求传入刚刚修改过的节点，函数将通过旋转平衡自动维持 AVL 树平衡，并更新各节点高度；对于 importFromDoc，要求传入文档名字 fileName，函数将打开 output 目录下对应文档，并将其中所有单词都加入倒排索引。

2.3 docHash 类

在上述的 AVL 树以外，我自主完成了词典索引机制的选做部分，即使用哈希表实现了倒排文档索引，其功能与 AVL 树完全相同，但结构上更为简单。

本类哈希表由开链法构建，即每个__docHash_node 内部都包含有 Queue<docList> words（与 AVL 树节点相同），从而使得哈希表的装载系数可以大于一。注意哈希函数针对的是 GB2312 编码的汉字，因此输入文件需要符合该编码，否则可能导致哈希表存储效率降低。

对于 importFromDoc，要求要求传入文档名字 fileName，函数将打开 output 目录下对应文档，并将其中所有单词都加入倒排索引（同 AVL 树）。对于 contained，要求传入需要查找的单词内容 strIn，并返回对应的文档链表指针 docList*，找不到则返回空指针。

在 main 函数中构建倒排文档时，我提供了 extractInfo 函数的两种重载，分别基于 AVL 树和 docHash 哈希表实现。但基于代码连贯性的考虑，仅有 AVL 树构建倒排文档的方法被调用。

2.4 wordList 类

本类与 docList 类相似，但更为简单，主要用于实现存储每篇文章中出现的不同单词。

具体数据成员的含义和函数成员的用法请参见 2.1 docList 类的说明。

2.5 relatedFileNode 类

本类主要在查询和推荐功能中使用。

一个本类对象包含 int docID 存储文档编号，int wordIncluded 存储该文档中出现的不同所给关键词的个数，int totalTimes 存储不同关键词在该文档中出现的总次数。进一步地，我们将一个文档的权重定义为$\mathrm{wordIncluded} \times \mathrm{totalTimes}$，从而实现同时出现多个关键词的文档排序更靠前的效果。

3 简要算法说明

在实验一的基础上，程序扫描并读取读取分词结果文件，并基于 AVL 树构建倒排索引文档。由于 ID 生成函数仅对首字符敏感，使不同的单词可能得到同一 ID，故而 AVL 树的每一个节点都包含 Queue<docList>，即文档链表的链表。每一个单词对应于一个 docList，同一 ID 的单词串在一起，构成了一个节点。

倒排文档构建完成后，首先实现批量查询功能。程序逐行扫描请求文件，并对每一行的每一个关键词进行统计，该行的结果存储在一个 relatedFileNode 类的对象 answers 中。对于每个关键词，程序在上面构建好的 AVL 树中找到对应的文档链表，并按照对应信息更新 answers 状态。最后，程序对 answers 中的不同文档按权重降序排列，完成输出。

然后实现推荐功能。程序逐行扫描请求文件，并在解析后的网页文件中寻找对应的新闻标题。找到以后，利用 wordList 类统计该新闻中出现各单词的词频。然后类比批量查询功能，将该新闻中出现的单词作为查询输入，将所得不同文档按权重降序排列，输出权重最高的前 5 篇文档（如果有 5 篇）。这里，为了平衡推荐效果和运行效率，每篇文档的权重采取类似卷积的方式进行计算，其定义如下：

$$weight_{doc} = \sum_{keyword} (Frequency_{query} \times Frequency_{target})$$

其中右端第一项是关键词在所给新闻中出现的频率，第二项是关键词在被查找网页中出现的频率，权重即为对所有关键词进行求和。可见，对于所给新闻中出现频率越高的单词，如果另一篇文章中这个单词的出现频率也较高，且两篇文章有更多的相同关键词，则权重越大。出于实际需要的考虑，我在输出结果中屏蔽了输入的文章，以获得更好的推荐效果。

此外，出于不重复操作和节约助教评审时间的考虑，我在 main 函数中注释掉了实验一的部分，并将解析出的新闻和分词后的文件放置在 ./output 路径下，从而可在上述步骤中直接调用。（经粗略测试，对所给 781 个文件完成解析分词所需时间在 60s 数量级。）

4 实验流程、操作说明与试验结果

按实验文档中要求，直接执行 query.exe 或 gui.exe 即可得到试验结果。注意，执行之前需要在路径下准备好所需的 input 和 output 文件夹，以及两个查询文件 query1.txt 和 query2.txt，都放在可执行文件的同级目录下。

对于 qurey.exe，其运行结果直接生成到 result1.txt 和 result2.txt 中。对于 gui.exe，其运行结果直接在用户界面中展示，双击每一项新闻可直接通过默认浏览器打开。用户交互界面中每一个按钮的功能都已标明，助教直接使用即可。但请注意：通过关键词进行搜索时，关键词之间要求通过空格区分；通过新闻标题进行搜索时，要求新闻标题与已有标题全文精确匹配。

5 功能亮点

在倒排文档的构建上，我通过 AVL 树和哈希表两种方式完成了构建。

面向对象思想得到了较为彻底的贯彻：建立对象，将可能会重复用到的代码封装为函数接口。

我的推荐算法在保证一定推荐准确度的基础上，良好地地利用了已有的倒排文档，从而提高了搜索速率。

最后，我还实现了功能可选择的 GUI 用户交互界面。用户在界面中直接输入要查询的关键词组或新闻标题（要求全文匹配）即可直接得到所需结果。双击 GUI 界面中返回的新闻标题，还可直接用本地默认浏览器打开对应网页，排版/图文一应俱全。

6 实验体会

本次试验中，我吸取了实验一的教训，一边编写代码一边同时在 macOS 和 win10 双平台测试，避免了最终迁移代码时所可能遇到的困难。但由于我之前没有在 Qt 平台上构建如此复杂的项目的经历，使得我在对自己的程序图形化时举步维艰，遇到了许多大大小小的 bug，例如中文在 Qt 下的编码问题，不同 IDE(Clion, Visual Studio, Qt Creator)对应不同编译器可能对同一段代码返回不同的结果等。（但是，GUI 界面和数据结构课有什么关系呢？）

此外，上课时听起来感觉还算简单的 AVL 树，实现起来也比较复杂：尤其是调整平衡时的旋转操作，变量名繁多，指针指来指去。我由于编写程序时粗心，使一个指针指向错误的对象，最终花费了半天（约 6 小时）的有效工作时间才解决这个 bug。可见，听起来简单的东西，实现起来却可能很难，正如 Linus 所说：Talk is cheap. Show me the code.

由于上述我遇到的未曾预料的困难，严重耽误了我完成大作业的进度，差点没能赶上 ddl。这也是我这次实验最重要的一点收获：不要总是卡 DDL，一定要给自己预留充足的时间。

最后，谢谢张力老师和助教的细心指导！

*按惯例：作业好简单啊，不够达到训练的目标，求 加量，求加难度