html 文件检索系统开发文档二

精 82 张翀 2018010625

1. 增量开发说明

a) 第一次大作业中已经完成了第二次大作业的基本项：输入关键字，输出系统检索到的文档集；检索模型为布尔模型。且已制作了 GUI。

b) 本次开发完成加分项：搜索词自动纠错、搜索词自动补全、GUI 友好化、即时检索（也就是所有要求中的加分项）。

c) 本次文档的内容组成：检索功能的简单复述、自动纠错的实现、自动补全的实现、GUI 的说明、即时检索的实现、检索速度和准确度分析、代码结构。

2. 实验环境与类库

a) 实验环境：python3.5

b) 新增功能用到的第三方类库：无

c) 新增功能用到的 python 标准库：re、tkinter

d) 数据来源：cacm

3. 检索功能复述

a) 功能综述

基于布尔模型，在分词之后建立带有位置信息的倒排索引和 bigram 索引。利用 bigram 索引实现通配符的单词查询，用倒排索引实现布尔运算。用布尔运算和位置信息相结合，实现短语查询。

在 GUI 界面，核心的两个功能为：1）可带通配符的短语查询，2）可带通配符的布尔运算式查询。检索出的结果在母窗口显示文件名和论文题目，可以通过子窗口浏览具体内容。

注：本部分的内容基本复制自上一次大作业文档，并略作删减。

b) 短语查询

本项目的短语查询主要基于位置索引。

假设短语 p 由单词 p[0],p[1],...,p[n-1]组成，那么首先利用多元 AND queries 得到同时包含这些单词的文档 d[0],d[1],...,d[k-1]。对于其中任意一个文档 d[i]，各单词有不同的出现位置列表，将这些列表作笛卡尔积，即可得到所有这些词的出现位置的组合。在这些组合中，只要能找到一个组合是公差为 1 的等差数列，那么短语 p 就在文档 d[i]中

出现。由此就实现了短语查询。

c) 通配符查询

本项目支持任意多次将单词中的任意长的一段替换为"*"进行查询，主要基于bigram_index。

首先考虑单个星号的情况。以 a*bc 为例，就只需要利用 bigram_index 进行对['$a', 'bc','c$']的 and query 即可，最后的得到的词再进行检查。

接下来考虑多个星号的情况。定义第一个星号左边部分为<left>，最后一个星号右边部分为<right>。以下分四种情况考虑：

1) <left>和<right>的长度都不为 0，则将他们都拆分成 bigram。以 left**right 为例，则左边拆分为['$l','le','ef','ft']，右边拆分为['ri','ig','gh','ht','t$']，对这两个集合的并集作 and query，得到词集 Q。Q 中可能有不满足通配符表达式的词，但是 Q 中词的总数不会太多，因此只要用表达式进行一次筛选即可。之前说到的简单情况是可以并入这种情况的。

2) <left>和<right>长度都为 0。分以下几种情况：

2a) 字符串总长度大于等于 4

如果去掉左右的星号后，里面没有星号，那么将里面的部分拆分成不带开头和结尾的 bigram 进行 and query。如果里面还有星号，那么可能匹配结果较多，先取所有词集，再进行检查筛选。

2b) 字符串总长度为 3

那么其必然是"***"或是"*a*"，前者返回所有词集，后者返回所有词集再作筛选即可。

2c) 字符串长度为 1 或 2

那么必然是"**"或"*"，返回所有词集即可。

3) <left>或<right>中的一个长度为零。取不为零的那个，展开成 bigram，进行 and query 搜索后，对得到的词集进行筛选。

得到可取词之后，可用通过倒排索引直接得到对应的文档集，这些文档集求并即可。

d) 带通配符的短语查询

对于短语中每一个词，其根据之前的方法得到多个可能对应的词，这些词组成一个列表。将这些列表作笛卡尔积，即可得到可能对应的短语。对每个这样的短语作短语查询，最后的文档序号集求并集即可。

e) 任意个数单词的布尔表达式查询

此功能需要人工先转化为析取范式，然后程序识别 &、|、!三个字符串来判断目标，

进行运算。先算出&式的结果，再计算|的结果。如 r|(p&!q)就是先计算 p&!q，再计算 r。对于每一个&式，程序自动会把 NOT 表达式放在最后以便和 AND 连用，即自动将 p&!q&r 转化为 p&r&!q（无需人工调整）。因为是析取范式，所以在输入框中不用加括号和空格，识别程序也更加简单。由于有之前的支持，此处可以支持带有通配符。

4. 功能优化与实现

a) 自动纠错

考虑到布尔表达式查询从使用者的角度来看很难出错，且无法判断是拼写错误还是忘了加运算符（如 taptop 可能是 laptop 的误拼写，也可能是 tap|top 漏了符号），故此处仅实现短语查询的自动纠错。

纠错的逻辑为：如果进行短语查询没有得到结果，那么将原来短语中的某一段替换成星号进行带通配符的短语查询，不停地更换替换的位置直到某种情况下找到了结果。所谓的“某一段”，即单个字符或者两个相连的字符，其中不能包含空格。

以’an app a’为例，假设该短语没有检索到对应的文档，那么使用星号进行局部替换。如果只替换一个字符，那么就有如下候选项：’an app *’, ‘an ap* a’, ‘an a*p a’, ‘an *pp a’, ‘a* app a’, ‘*n app a’. 如果替换两个字符，那么就有如下候选项：’an a* a’, ‘an *pp a’, ‘* app a’. 在这些候选项中，只要有一个能得到搜索结果，那么就将其视作修正的搜索内容，并输出对应结果。如果没有任何一个可以找到搜索结果，那么就放弃修正。

事实上，对于日常使用中的拼写错误或是漏写通配符的错误，这种修正方式已经基本可以涵盖所有情况。在修正失败的情况下耗时量可能较大，但修正失败也基本意味着想搜的短语确实是不出现在任何文档中的。

b) 自动补全

和自动纠错的逻辑类似，自动补全也仅仅针对短语查询而言，即自主推断短语查询中最后一个词的完整形式。

本功能也实现了即时化，即以每 0.5s 一次的频率进行补全提示。为了能在 tkinter 中周期性地触发它，使用了 tkinter.after 定时器。在启动主窗口之前，定时 0.5s 后运行补全提示。在补全提示函数中，也定时了 0.5s 后再次运行。从而可以周期性运行该函数。

由于进行了即时化，为保证效率，如果短语中有通配符，那么只根据最后一个单词进行补全，如果没有通配符，则根据整个短语进行补全。

所谓根据最后一个单词进行补全，乃是将短语的最后一个单词取出，然后利用 bigram_index 查找相匹配的单词。换言之，其只是进行了一次通配符查询中的寻找可用词。之后，用找到的第一个可用词替换原来短语中的最后一个词，然后进行短语查询即可。如果最后一个词本身带了星号或是找不到可用词，那么补全提示即去掉最后一个词（即走向更加通配的查询）。

所谓根据整个短语进行补全，乃是在最后一个单词的候选可用词中，选出和整个短

语一起查询都有结果的某一个作为补全提示。如果都没有结果，那自然也是去掉最后一个词。

c) GUI 实现

GUI 使用 tkinter 制作。利用控件等功能，增强交互性，界面友好。**为避免影响阅读流畅性，具体说明放至另一篇文档中。**

d) 即时检索的实现

首先，出于运行效率和 CPU 占用考虑，设定了其频率是 1 秒 1 次，且不对有通配符的短语或布尔表达式检索。支持普通的短语查询和布尔表达式，其他情况则显示之前的结果。如果是空字符串或是空格，同样显示之前的结果。

其即时化的实现方式和之前的自动补全是一样的，即利用 tkinter.after 定时器。

5. 实验结果与分析

a) 检索的准确度

就检索本身的准确度而言，由于基于布尔模型，准确度只依赖于之前建立的倒排索引，而倒排索引的准确度则主要基于分词。

就自动纠错的准确度而言，自然是相较于不纠错时提高了，而且考虑了用通配符替代两个字符的情形，必然比替代一个字符要高，但相对而言必然会带来用时的提升。同时，用通配符替代的字符如果偏多，很可能会纠正至并非原意的短语查询。而且根据日常经验来看，拼写出错往往是单字母打错或遗漏、两字母换位这种错误，且两个词同时出错概率很低；因此，替代上限为 2 是一个合理的值，且实际纠错时应当先考虑替代一个字符的情况。

就自动补全的准确度而言，如果是只根据最后一个单词补全，而不利用整个短语的信息，则不可避免地较低，因为我在项目中仅给出了一个备选项。如果根据短语补全，那么用时和空间则不可避免地上升，但准确度毫无疑问较高。但无论如何，自动补全是只能补全最后一个单词的，且该单词已出现的部分不能出错。尽管如此，其还可以通过不断地迭代来实现可进行检索的补全。例如，“have axn problrm”因为出错，会被补全为”have axn”，进而再补全为”have”，此时虽然不是想搜的东西，但不断跟从提示也可以得到检索的结果。更何况，对有错语句的修正不应该靠自动补全来完成。

尽管自动补全和自动纠错的准确度都有限，两者可以通过互相结合互补短处。自动纠错可以处理前面单词的错误，而自动补全可以让最后一个单词更加精确。以“an proble”为例，自动纠错是无法处理的，自动补全则可以先把它变成”an problem”，然后自动纠错再把它变成”a problem”.

b) 检索的速度

就检索本身而言，数据量较小，不带有通配符的查询也较快。但是如果带上通配符，

查询时间就不确定了——尽管 and queries 和检查的单次运算时间并不多，但是大量的通配符可能带来大量的可能选择，这些可能选择以乘法方式（笛卡尔积）扩大着运算量。如“l**t”用时 0.006s,“* *”则是在跑了两分钟后被我强制停止程序。因此，对于带有独立“*”的搜素应当尽量避免，而算法在处理的时候也可以考虑遍历文档而非单词。

就自动纠错而言，因为不断尝试带有通配符的选择，可能会占用大量时间。如果最后找不到，那便是尝试了可能十几次的带通配符的短语查询，且通配符可能不止一个（即原来就有通配符，再加上了通配符）。即使一次出结果，通配符的引入也可能带来极大时间消耗。以“one of a”为例，因为找不到结果（从英语语法来看肯定是错的，用时也不会长），于是搜索了“one of *”，有 one 的文档很多，有 of 的文档很多，“*”更是代表了所有的词，搜索的量一下便很多了，用时 8s。相比之下，搜索“one of”却只用了 0.003s；搜索“one of a*”因为有答案无需纠正，只用了 0.6s。这也体现出，通配符带来了时间上的极不稳定。

就自动补全而言，如果仅根据最后一个单词，用时极短——只需要进行 bigram 的 and query 和检查即可。将通配符的不稳定性限制在一个单词内，不和其他单词产生倍数相乘的影响。如果根据短语，因为限制了不允许通配符出现，其用时也很短，只不过是加上了多次无通配符短语查询的时间。

6. 重要函数与变量说明

源代码包括 ini_and_func.py 和 run.py 两个文件，前者为项目的实现，后者则是针对本项目制作的 GUI，直接在命令行中运行即可，需等待约 10s 以完成初始化，文件同目录内需有名为”cacm”、内含 html 文件的文件夹。

以下是 ini_and_func.py 中的函数和变量说明

a) 函数

and_query_skip(a,b):即对两个词或列表的使用 skip pointer 的 and query

and_query(a,b):即对两个词或列表的不使用 skip pointer 的 and query

multi_and_query_skip(t):对多个词的使用 skip pointer 的 and query（词为列表格式）

multi_and_query(t):同上，不使用 skip pointer

run_time(func,epoch):测试某语句 func 运行 epoch 次的时间

inverted_untuple_desartes(c):将带括号的笛卡尔积转化为不带括号的列表形式，但是顺序会颠倒。

strict_desc(list):判断一个 list 中的值是否严格递减

judge_adaj(positions):有一系列的词，每个词在文档中有一系列的位置，从而形成了二维列表 positions。用该函数判断是否存在一种位置组合，使得这些词按照顺序出现在文档里（即用于短语查询）。

phrase_search(s):对短语字符串 s 进行查询

generate_bigram(w):将一个词转化为bigram列表的形式，包括开头的$?和结尾的?$

search_two_bigram(b1,b2):即相当于两个 bigram（如'ab'和'cd'）或是词列表的查询词的 and query

search_list_bigram(l):即对一个列表的 bigram 进行 and query

check(pattern,w):检查 w 是否符合某模式串

search_star(s):返回通配符单词 s 出现的文档

single_word_wild_card_search(s):返回单词 s 出现的文档，无论是否有通配符

and_query_wild(a,b):类似于之前的 and_query，但是允许对象带有通配符

whole_wild_search(s):进行可带通配符的短语查询，并执行最后的去重，返回文档编号集

num2doc(ans):将文档编号集转化为文档名集合

final_search(s):对大小写进行兼容的可带通配符的短语查询，返回文档名集合

OR_2(a,b):对两个可能含有通配符的单词/列表作 OR 运算

NOT_1(a):对列表/可能含有通配符的单词作 NOT 运算

AND_NOT(a,b):进行 a and(not b)的运算，a 可以是可能含有通配符的单词，也可以是列表，b 只能是可能含有通配符的单词

split_or(s):提取析取范式中的每一个&表达式

dealwithand(s):计算一个&表达式(如 a&!b&c，即 a AND(NOT b) AND c)的检索结果

multi_OR(l):对析取范式中每一个&表达式的结果进行合并

bool_calculate(s):对析取范式字符串进行检索

find_like_phrase(p):将字符串中连续的 1~2 个字符替换为*，不替换空格，返回一个列表即所有位置的替换结果，用于表示和字符串 p 长得像的带通配符的字符串

correct_wild_phrase_search(p):支持通配符和自动纠正的短语查询，返回二元组，如果进行了纠正第一个元素是纠正的字符串，否则是空字符串；第二个元素是查询的文档集合，如果进行了纠正就对应纠正后的，如果没进行纠正就对应纠正前的。

candidates(w):对于一个未打完的单词，给出其的候选补全词。

auto_complete(p):返回字符串 p 自动补全的结果。

b) 变量

path:即 cacm 文件夹的名字

punctuations:即需从分词结果中去掉的英文符号

word_doc_dict:即无位置信息的倒排索引

positional_index:即有位置信息的倒排索引，形式为{词:{文档：[位置]}

bigram_word:即 bigram 的倒排索引

以下是 run.py 中的函数和变量说明

c) 函数

dynamic_search():即时搜索

search():根据搜索框内容进行搜索

sarch2():根据补全的建议内容进行搜索

view_article():建立浏览文档内容的子窗口

get_art():获取文档内容

refresh_art_pre()/refresh_art_next():获取检索结果上一篇/下一篇文档内容

complete_tip():即时获取补全提示

d) 变量

window:母窗口

mayaim:自动补全的提示结果

current_articles:检索结果中的文档序号

rank_art:在子窗口中显示内容的文档序号

e:输入框

tcomplete:补全提示框

tcorrent:纠错提示框

tnum:搜索结果数、用时提示框

tres:输出正式检索结果的文本框

tres2:输出即时检索结果的文本框

tart:输出文档内容的文本框

c:搜索按键

c2:搜索补全建议的按键

c3:弹出文档内容子窗口的按键

c4:查看前一篇文档内容按键

c5:查看后一篇文档内容按键

7. GUI 使用说明

见另一篇 PDF 文档《GUI 使用说明》。