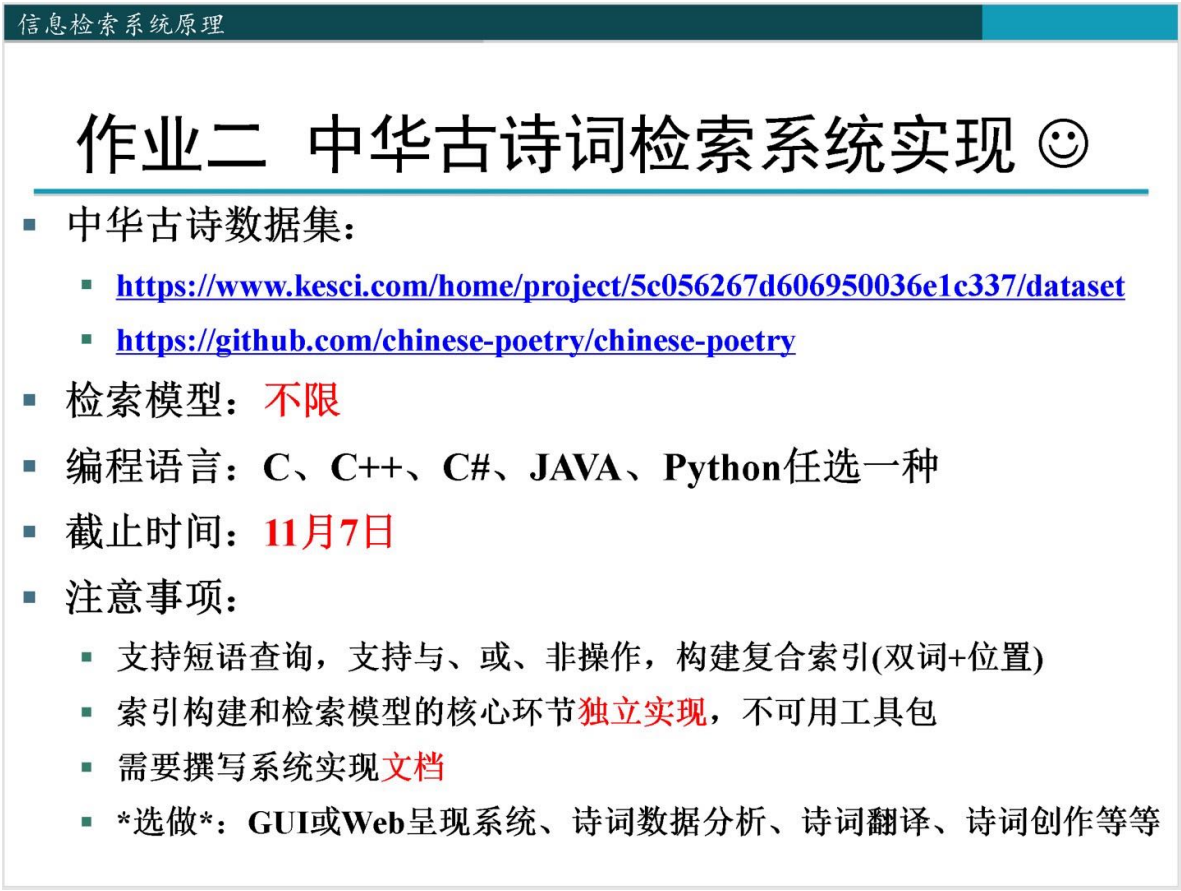
中华诗词检索系统——Poetry Google

> 李伟 1711350 jack-lio'github最新修改于2019.11.7 系统实现功能：实现一个支持短语查询、与或非查询的系统，需要构建复合索引（双词+位置），能够对诗文进行基本的检索功能。在此基础上本系统添加了对与不同域的查询功能，能够支持对作者，标题和正文的分类检索。



> ***系统启动与文件说明***: 系统构建不同功能实现模块在不同的文件中，`data_import_process`文件实现了对与繁体中文的简体化功能，以及对于一些索引和诗文数据的导入函数，`index_build`文件中编写了各种难索引的构建函数，按照参数可以构建相应的索引文件，存储路径已经预先规定，`query_op`文件编写了对于查询的相关操作实现，包括基于作者，标题，正文的查询，并实现了对于短语和与或非的查询支持，以及将编号序列转为诗文数据的转换函数，`backend`文件实现了后端的响应支持，`index.html`文件编写了前端的HTML实现。`create`文件下存储了生成的各种索引文件以及转换为简体中文之后的诗文数据文件。在建立好索引文件之后，运行backend文件程序，然后打开index.html文件（注意要联网），之后即可进行系统的使用了。

目录

0. 实验环境

编程语言:python、HTML 调用的包:flask、`elementUI`的样式文件、`snowNLP`系统环境：window10专业版，8GB内存 编程IDE：`pycharm` 前端浏览工具：google浏览器

1. 前期准备

本次实验的数据集来自和鲸社区的中华古诗词数据集，在开始构建索引之前先查看一下数据集的数据情况。数据集中包含了250000首宋诗与57000首唐诗，以及唐朝宋朝的诗人信息数据。

1.1 数据集解析

首先查看诗文数据，如下所示：

```
  {
    "strains": [
      "平平仄仄仄，平仄仄平平。",
      "仄仄平平仄，平平仄仄平。",
      "平平平仄仄，平仄仄平平。",
      "仄仄平平仄，平平仄仄平。"
    ],
    "author": "薛能",
    "paragraphs": [
      "長安那不住，西笑又東行。",
      "若以貧無計，何因事有成。",
      "雲峰天外出，江色草中明。",
      "謾忝相於分，吾言世甚輕。"
    ],
    "title": "送進士許棠下第東歸"
  },
```

以看到，诗文以`json`格式存储，每一首诗文中保存四个信息段，包含作何信息，标题，正文，以及平仄韵律信息，在本系统中只取用作者、标题、正文三个域的信息。同时可以看 到，存储的信息是繁体格式，考虑到检索的方便性，可以将文字转为简体格式。

再看作者信息，如下所示：

```
{
        "name": "宋太祖",
        "desc": "宋太祖赵匡胤（九二七～九七六），涿州（今属河北）人。生于洛阳。后汉乾祐元年（九四八），枢密使郭威讨李守真，应募为部属。后周显德中，从征淮南、寿春，积功至殿前都指挥使、义成军节度使，改忠武军节度使。显德六年（九五九），升检校太傅、殿前都点检。恭帝即位，改检校太尉、归德军节度使。七年春，发动陈桥兵变，即帝位，国号宋，改元建隆（九六o）。继而平定李筠、李重进叛乱。乾德元年（九六三），平荆湖；三年，平后蜀。开宝二年（九六九），亲征北汉；四年，平南汉；八年，平南唐。在位十七年。开宝九年冬十月卒，年五十。太宗太平兴国二年（九七七），葬永昌陵。事见《宋史》卷一至三《太祖纪》。"
    },
```

作者信息主要包含作者的名，以及生平介绍，同样是以繁体字体 呈现，上面为转为简体字后的效果,调用`snowNLP`的繁体转换实现转为简体字的功能。

1.2 数据处理和导入

通过上述的数据解析，可以看出，对于数据的处理主要包含两个部分，即文本的简体字化和数据的读取。 由于是 json 格式，数据的读取采用 json.load() 函数实现，将繁体字转为简体字，调用 snowNLP 自然语言处理包中的转简体字功能。具体的实现代码如下所示：

```
# 给诗文和作者信息转为简体字，同时将生成的结果存入新的文件目录下
# 依次处理的为诗人信息（唐朝、宋朝）、宋诗、唐诗
# 作者信息格式为： 作者 描述
# 诗文信息格式为： 作者 诗名 诗文
# 转换使用的工具保卫SnowNLP
def convert_poets_lang():
    # 唐朝诗人信息
    load_f_t = open(authors_tang_path, 'r', encoding="UTF-8")
    t_dump = open("./create/tang/authors.json",'w',encoding="UTF-8")
    authors_t = json.load(load_f_t)
    info_list_t = []
    for item in authors_t:
        info_list_t.append({"name":zhconv.convert(item["name"], 'zh-
cn'),"desc":zhconv.convert(item["desc"], 'zh-cn')})
    json.dump(info_list_t,t_dump,indent=4,ensure_ascii=False)
    t_dump.close()
    print("--------》》》》唐朝诗人信息处理完毕")
    # 宋朝诗人信息
    load_f_s = open(authors_song_path, 'r', encoding="UTF-8")
    s_dump = open("./create/song/authors.json", 'w', encoding="UTF-8")
    authors_s = json.load(load_f_s)
    info_list_s = []
    for item in authors_s:
        info_list_s.append({"name":zhconv.convert(item["name"], 'zh-
cn'),"desc":zhconv.convert(item["desc"], 'zh-cn')})
    json.dump(info_list_s, s_dump, indent=4,ensure_ascii=False)
    s_dump.close()
    print("--------》》》》宋朝诗人信息处理完毕")
    # 宋朝诗文
    for j in range(0,255):
        poets_list = []
        f = open("./chinesepoetry/poet.song.%d.json"%(j*1000),'r',encoding="UTF-
8")
        dump_f = open("./create/song/%d.json"%(j*1000),'w',encoding="UTF-8")
        temp = json.load(f)
        for k  in range(len(temp)):
            poets_text = ""
            for item in temp[k]["paragraphs"]:
                poets_text+=zhconv.convert(item,'zh-cn')
            poets_list.append({"author":zhconv.convert(temp[k]["author"],'zh-
cn'),"title":zhconv.convert(temp[k]["title"],'zh-cn'),
                               "paragraphs":poets_text})
        json.dump(poets_list, dump_f, indent=4,ensure_ascii=False)
        print(j)
        dump_f.close()
    print("--------》》》》唐朝诗文处理完毕")
    # 唐朝诗文
    for j in range(0,58):
        poets_list = []
        f = open("./chinesepoetry/poet.tang.%d.json"%(j*1000),'r',encoding="UTF-
8")
        dump_f = open("./create/tang/%d.json" % (j * 1000), 'w', encoding="UTF-
8")
```

```
        temp = json.load(f)
        f.close()
        for k  in range(len(temp)):
            poets_text = ""
            for item in temp[k]["paragraphs"]:
                poets_text += zhconv.convert(item, 'zh-cn')
            poets_list.append({"author": zhconv.convert(temp[k]["author"], 'zh-
cn'),
                               "title": zhconv.convert(temp[k]["title"], 'zh-
cn'),
                               "paragraphs": poets_text})
        json.dump(poets_list,dump_f,indent=4,ensure_ascii=False)
        dump_f.close()
        print(j)
    print("--------》》》》宋朝诗文处理完毕")

# 按照编号顺序导入诗文信息，包括诗文以及作者诗名等信息
# 返回值为诗文的总量count ,和诗文信息的列表 poets_info
# date: 2019.11.1
def import_poets_info():
    count = 0
    poets_info = []
    for j in range(0,255):    # 按照分块存储的格式重新读取宋诗
        f = open("./create/song/%d.json"%(j*1000),'r',encoding="UTF-8")
        temp = json.load(f)
        for k  in range(len(temp)):
            poets_info.append(temp[k])
            count = count + 1
    for j in range(0,58):     # 读取唐诗
        f = open("./create/tang/%d.json"%(j*1000),'r',encoding="UTF-8")
        temp = json.load(f)
        for k  in range(len(temp)):
            poets_info.append(temp[k])
            count = count + 1
    return count,poets_info
```

2. 索引构建

基于系统功能实现的需求，本次需要建立的索引包括文本的双字索引，文本的单字位置索引，作者的索引，标题的位置索引。单字位置索引将一行文本视为一个列表，逐一获取单个字符进行处理，双字索引的构建基于前后的两个位置标志的配合实现。作者索引采用作者名字作为词项，作者的诗文编号作为倒排索引的记录值，标题索引采用类似于正文位置索引的方式构建位置索引，具体的实现代码下所示***(构建标题的位置索引方式和正文的位置索引构建方式类似，在下面不做说明)***:

```
# 构建作者的索引，其后保存其所做诗文的编号，实现基于作者查询诗文的功能
# param: authtor_index_file 索引存储路径，poet_info 诗文作者信息列表
# return:  NULL
# date: 2019.11.3
def build_author_index(author_index_file,poet_info,count):
    # 建立字典
    index_dict = {}
    for i in range(count):
        author_name = poet_info[i]["author"]
        if author_name not in index_dict.keys():
            index_dict[author_name] = [i]
        else:
```

```
            index_dict[author_name].append(i)
    # 将索引存储
    f = open(author_index_file,'w',encoding="UTF-8")
    json.dump(index_dict,f, indent=0, ensure_ascii=False)
    f.close()
    print("->>>>>>>作者索引构建完成，共有诗人%d名"%(len(index_dict.keys())))

# 构建位置索引和双字索引
# 即以单字为词项，其后跟随存在该词项的诗文编号和出现该词项的偏移量
def build_p_d_index(position_index_file,double_index_file,poets):
    word_index_t = {}
    dword_index_t = {}
    for no in range(len(poets)):
        if(no%1000==0):
            print("->>>>>>",no)     # 输出日志信息到console
        word_list = [i for i in poets[no]] #逐一读取字符，将诗文的文本数据转为一个单字
的列表数据结构
        length = len(word_list)
        for i in range(length):
            # 构建双字索引表
            if (i > 0 and word_list[i] not in[', ', '。'] and word_list[i-1] not
in [', ', '。']):
# 双字索引的词项由前后两个单词组成，不停向后移动取词的起点即可
# --------|^^^|-----------
# --------<item>----------
# ---->>move right--------
# ----------|^-^|---------
                word_item = word_list[i-1]+word_list[i]
                #print(word_item)
                if word_item not in dword_index_t.keys():
                    dword_index_t[word_item] = [no]
                else:
                    if (no not in dword_index_t[word_item]):
                        dword_index_t[word_item].append(no)

            # 构建单字索引表
            if word_list[i] in [', ','。']:       # 构建单字索引跳过逗号和句号，但是在
索引的位置上逗号和句号的位置保留
                continue
            if(word_list[i] not in word_index_t.keys()):
                word_index_t[word_list[i]] = {  no:[i],}
            else:
                if(no not in word_index_t[word_list[i]].keys()):
                    word_index_t[word_list[i]][no]=[i]
                else:
                    word_index_t[word_list[i]][no].append(i)
    # 把字典转为列表格式，分块存储
    w_list = word_index_t.items()
    d_list = dword_index_t.items()

    temp = []         # 暂时存储的临时空间，实现将长字典分块存储在许多个不同的文件中
    count =  0
    for item in w_list:
        temp.append(item)
        if(len(temp)==500):
            dump_s = open(position_index_file + "%d.json" % (count*500), 'w',
encoding="UTF-8")
            json.dump(temp, dump_s, indent=0, ensure_ascii=False)
```

```
            dump_s.close()
            temp.clear()
            count=count+1
            print("pos<<<<",count)
    dump_d = open(position_index_file + "%d.json" % (count*500), 'w',
encoding="UTF-8")
    json.dump(temp, dump_d, indent=0, ensure_ascii=False)
    dump_d.close()

    temp.clear()    # 清除temp列表，方便后面的双字索引存储使用
    count = 0
    print("---->>>>>单字索引存储完成,有词项",len(w_list))
    for item in d_list:
        temp.append(item)
        if(len(temp)==10000):
            dump_d = open(double_index_file + "%d0000.json" % (count), 'w',
encoding="UTF-8")
            json.dump(temp, dump_d, indent=0, ensure_ascii=False)
            dump_d.close()
            temp.clear()
            count=count+1
            print("dou<<<<", count)
    dump_d = open(double_index_file + "%d0000.json" % (count), 'w',
encoding="UTF-8")
    json.dump(temp, dump_d, indent=0, ensure_ascii=False)
    dump_d.close()
    print("---->>>>>双字索引存储完成，有词项",len(d_list))
```

3. 查询实现

本系统要求实现基于与或非的查询功能，其中基础的查询功能包括短语查询，短语查询的实现有两种情况，如果是查询双字的话，直接采用双词查询返回结果，如果是查询单字或是多余两个字的短语查询，则采用位置索引进行查询功能的实现。对于查询文本中的括号和and ，or，not等操作符采用正则表达式进行匹配，然后将查询表达式转为后缀形式，再进行查询，最后返回结果，返回的结果为诗文的ID编号，需要经过convert函数获得相应的诗文内容（在这一步中可以添加对查询词项的替换工作，加上 `<b></b>` 的标签使得显示的诗文在后续的web端展示的时候能够突出显示查询词项，实现用户交互的友好性）。相关主要实现函数代码如下所示：

```
# 定义查询种类
query_type = {"title":1,"author":2,"paragraphs":3, "mix":4}

# 查询函数，传入参数为查询文本
# 函数实现功能为对输入的查询文本进行判断，如果是双字查询则使用双字索引
# 否则，使用位置索引实现基于短语的查询功能，具体的双字查询和短语查询均通过独立的函数实现
# 本函数中只进行调用和相关的处理
# 默认为混合查询模式，即自行进行判断查询获取结果，较慢，如果需要精准的快速查询选择相应类别的直接
查询模块
# param： query_sentence 查询文本
# return： 结果文本的编号列表
# date：2019.11.2
def query(query_sentence,type):
    # 处理查询文本,将查询文本分为一个单字的列表，短语查询忽略“，”和“。”
    if query_type[type] == 1:                    # 标题查询
        text_list = [i for i in query_sentence]
        return phrase_query(text_list,title_index)
    elif query_type[type] == 2:                    # 调用作者查询模块
```

```
        return  author_query(query_sentence)
    elif query_type[type] == 3:                         # 诗文文本查询模块，实现与或非的查询功能
        query_items = re.findall(r"\(|\)|[Aa][Nn][Dd]|[Oo][Rr]|[Nn][Oo][Tt]|[^a-zA-z\(\) ]*",query_sentence)      # 通过正则表达式进行分划区块
        query_items_postfix = convert2postfixexpr(query_items)
 # 转为后缀形式的查询
        result = []
        print(query_items_postfix)
        for item in query_items_postfix:
            print(result)
            if re.fullmatch(r"[Aa][Nn][Dd]",item):    #与， 将result的最上面两个结果合并
                second = result.pop()
                first = result.pop()
                print("and")
                print([i for i in first if i in second])
                result.append(list(set(first).intersection(set(second))))
            elif re.fullmatch(r"[Oo][Rr]",item):  # 或，合并后去重
                second = result.pop()
                first = result.pop()
                print("or")
                result.append(list(set(first).union(set(second))))
            elif re.fullmatch(r"[Nn][Oo][Tt]",item): # not ,通过count求出对于全局的补集
                first = result.pop()
                print("not")
                result.append([i for i in  range(0,count) if i not in first])
            else:                    # 对查询项进行查询操作，结果压入result
                text_list = [i for i in item]
                print(text_list)
                if (', ' not in text_list and '。' not in text_list\
                        and len(text_list) == 2):  # 调用处理双字查询的函数
                    result.append(dw_query(item))
                    print(result)
                else:
                    result.append(phrase_query(text_list, position_index))  # 调用基于位置查询的处理函数查询诗文正文
                    print(result)
        if len(result)==1:    # 返回查询结果，如果栈中剩下的项大于一个，说明表达式错误
            return result.pop()
        else:
            return []
#     elif query_type[type] == 4:  # 混合查询模式，即不指定特定的查询域，进行所有结果的查询，找到了所有的结果进行合并返回，功能比较鸡肋，最终没有实现
#         text_list = [i for i in query_sentence]
#         return []
    # 返回查询结果，如果没有找到，则返回空列表
    return []
```

本次实现与或非短语查询的核心在于表达式的**中缀转后缀语法分析功能**，实现函数为`convert2postfixexpr(expr_list)`函数，实现的方式是采用栈进行转换，具体的实现代码和详细的注释如下所示：

```
# 将获取的与或非表达式item列表转为后缀形式，之后进行与或非查询
# not 优先级最高，and 次之 ， or最低
# param: expr_list 为经过正则处理之后形成的表达式元素列表
```

```
# 返回一个list（栈类型），其中保存一个后缀形式的查询表达式
# date：2019.11.6
def convert2postfixexpr(expr_list):
    sym_stack  = list()          # 符号栈
    item_stack = list()          # 查询元素栈
    queue_dict = {")":4,"not":3,"and":2,"or":1,"(":0} # 定义与或非的优先级
    for item in expr_list :
        if re.fullmatch(r"\(|\)|[Aa][Nn][Dd]|[Oo][Rr]|[Nn][Oo][Tt]",item):   # 通过正则表达式匹配运算符
            while(len(sym_stack)!=0): #符号栈栈不为空，不停弹出栈顶进行比较
                top = sym_stack.pop()          # stack top
                if queue_dict[item.lower()] <= queue_dict[top.lower()]:  # 将入栈的操作符优先级比栈顶低，需要弹出栈顶元素
                    if top != ')':  # 非），弹出栈顶即可
                        item_stack.append(top)
                        continue
                    else:  # 如果栈顶为”）“，表示识别到括号表达式,则将括号以内的符号全部弹出
                        while (len(sym_stack) != 0):
                            t_top = sym_stack.pop()
                            if t_top == '(':
                                break
                            item_stack.append(t_top)
                else:           # 将入栈的操作符优先级更高，入栈退出循环
                    sym_stack.append(top)
                    sym_stack.append(item)
                    break
            if len(sym_stack) == 0:  # stack is empty , push stack
                sym_stack.append(item)
        elif item != "":     # 去除表达式元素列表中匹配出来的空符号
            item_stack.append(item)          # 为查询项则直接入栈
    # 最后如果符号栈不为空，则将符号栈中所有非括号内容全部放入item_stack
    while(len(sym_stack)!=0):
        top  = sym_stack.pop()
        if top not in ["(",")"]:
            item_stack.append(top)
    # 返回后缀形式的查询列表
    return item_stack
```

4. 前后端交互实现

本实验中系统实现了前后端的交互，前端界面使用`elementUI`实现，后端采用`flask`建立,自己设计了简单的LOGO如下图所示，当查询结果大于1000项的时候只返回前1000项，同时对查询到的结果进行突出显示，前端实现界面如下所示（因前端代码较为简单，可以查看源代码了解相关详情）：

5. 系统运行效果

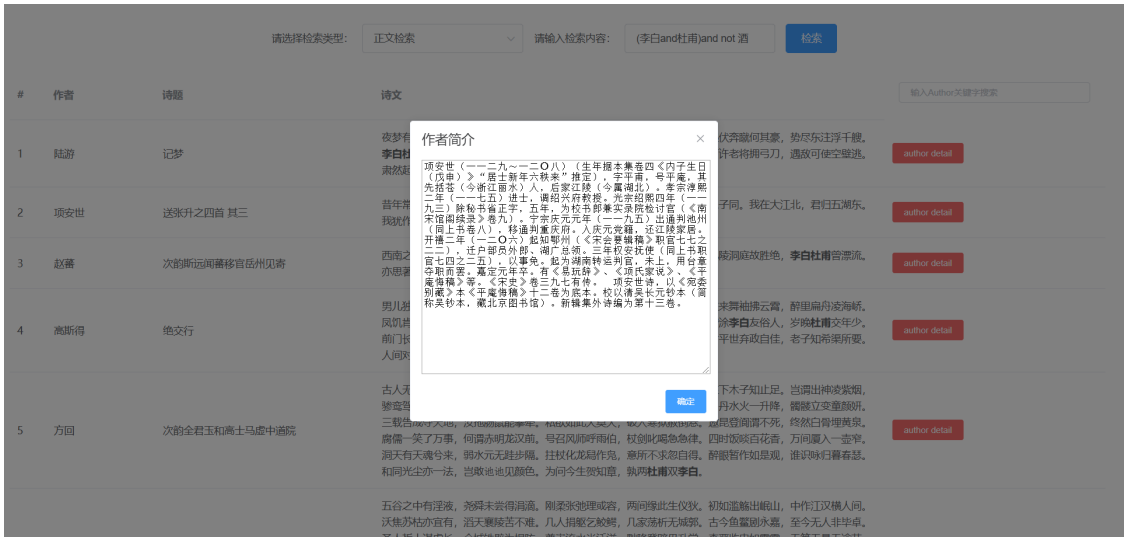
- 作者查询：

- 标题查询：

- 正文查询&短语查询支持：

- 与或非查询支持：

- 获取作者简介：



6. 总结

本次实现的系统功能比较简单，但是稳定性较好，通过通过这次的检索系统实现，熟悉了短语查询的实现方式，即双词索引或者位置索引方式，加深了对于课堂上学习的知识的理解，也锻炼了编程能力。同时在前端界面的实现上更加注重了用户的友好性，在前端实现上将***查询项突出显示***，并提供了***表达式方式的与或非查询模式***，有利于用户的查询支持，同时将***检索结果和诗人简介链接***，可以进一步了解作者的相关信息，达到了信息检索的***可拓展性***。