

Lab2 倒排索引 实验报告

10152130122 钱庭涵

inverted_index.py

主函数中，创建需要用到的文件夹。

```
56 if (not (os.path.exists('./index_file'))):  
57     os.mkdir('./index_file')
```

创建 path 列表，添加所有 lab1 中 disk12 经过预处理后得到的结果文件所在的文件夹路径。

```
59 path = []  
60 path.append('../lab1/code/Answer/document/AP')  
61 path.append('../lab1/code/Answer/document/DOE')  
62 path.append('../lab1/code/Answer/document/FR')  
63 path.append('../lab1/code/Answer/document/WSJ')  
64 path.append('../lab1/code/Answer/document/ZIFF')
```

给所有文件做 ID 标记，从 0 开始标记，创建文件名和文件 ID 映射的 map，存储到 map.json 文件中。

```
66 file_num = 0  
67 document_map = {}  
68 for count in range(len(path)):  
69     for a in os.walk(path[count]):  
70         document_files = a[2]  
71         for f in document_files:  
72             document_map[f] = file_num  
73             file_num += 1  
74 file0 = open('./map.json', 'w')  
75 json.dump(document_map, file0)  
76 file0.close()  
77 print('map succeed')
```

全局变量，block_num 是将所有文件分块后的块数，total_word 是所有文件的 text 中的 token 总数，在之后计算总词频时会用到。

```
79 block_num = 0  
80 total_word = 0
```

遍历所有文件，按每 10000 个文件为一块进行分块处理，每个文件夹中最后不足 10000 个的算作一块，调用 SPIMI_Invert 函数，传递参数 document_files[start:end] 为当前块中所有文件名列表，count 为文件夹路径编号，total_word 用来统计总词数，block_num 统计块数。

```
82 for count in range(len(path)):
83     for a in os.walk(path[count]):
84         document_files = a[2]
85         start = 0
86         while start < len(document_files):
87             end = start + 10000
88             if end > len(document_files):
89                 end = len(document_files)
90             total_word = SPIMI_Invert(document_files[start:end], count, total_word)
91             start += 10000
92             block_num += 1
```

跳转到 SPIMI_Invert 函数，dictionary 字典用于存储该块中所有索引，file_len 用于存储每个文件 ID 对应的文件的词数。

调用 xml.etree.ElementTree 模块，用 xml 文件的方式打开文件，根据文件名称，从 document_map 中获取 file_ID，token_list 为当前文件中所有 token 的列表，total_word 统计总词数，file_len 存储文件词数。

```
14 def SPIMI_Invert(file_stream, path_count, total_word):
15     dictionary = {}
16     file_len = {}
17     for file in file_stream:
18         tree = ET.parse(path[path_count] + '/' + file)
19         file_ID = document_map[file]
20         root = tree.getroot()
21         token_list = re.split(' ', root.find('TEXT').text)
22         total_word += len(token_list)
23         file_len[file_ID] = len(token_list)
```

dictionary 字典结构如下：（在未计算出词频时，词频位置先记录词数）

`{'token': {'rate': 总词频, 'pos': {'fileID': {'rate': 词频, 'pos': [pos1, pos2, ...]}, ...}}, ...}`

遍历当前文件中所有 token，如果 token 不在 dictionary 字典中，就将其添加到字典中，rate 初始化为 1，并记录 file_ID 和 token 在文件中的位置。

如果 token 在 dictionary 字典中，将对应的总词数加 1，判断 file_ID 是否在该 token 对应的字典中。若不在，初始化并添加 file_ID 对应的字典；若在，对应的词数加 1，并将此时的位置添加到位置信息中。

```
24 for i in range(len(token_list)):
25     if token_list[i] not in dictionary.keys():
26         dictionary[token_list[i]] = {'rate': 1, 'pos': {file_ID: {'rate': 1, 'pos': [i]}}}
27     else:
28         dictionary[token_list[i]]['rate'] += 1
29         if file_ID not in dictionary[token_list[i]]['pos'].keys():
30             dictionary[token_list[i]]['pos'][file_ID] = {'rate': 1, 'pos': [i]}
31         else:
32             dictionary[token_list[i]]['pos'][file_ID]['rate'] += 1
33             dictionary[token_list[i]]['pos'][file_ID]['pos'].append(i)
```

接下来要做的是：把该块的索引分成 27 个索引，分别为 26 个英文字母开头的 token 和其他 token 对应的索引，即每个块生成 27 个索引文件。

```
35 token_dict_tmp = {'a': {}, 'b': {}, 'c': {}, 'd': {}, 'e': {}, 'f': {}, 'g': {},
36                  'h': {}, 'i': {}, 'j': {}, 'k': {}, 'l': {}, 'm': {}, 'n': {},
37                  'o': {}, 'p': {}, 'q': {}, 'r': {}, 's': {}, 't': {},
38                  'u': {}, 'v': {}, 'w': {}, 'x': {}, 'y': {}, 'z': {}, '_other': {}}
```

遍历 dictionary 字典中所有 token，对于每个 token 对应的各个文件中的词数，将其替换成词频。

如果当前 token 为空字符串，则添加到 token_dict_tmp['_other'] 对应的字典中。

否则，判断 token 的第一个字符为哪个英文字母，就添加到哪个英文字母对应的字典中去，若 token 的第一个字符不为英文字母，则添加到 '_other' 对应的字典中。

```
39 for token in dictionary.keys():
40     for name in dictionary[token]['pos'].keys():
41         dictionary[token]['pos'][name]['rate'] = dictionary[token]['pos'][name]['rate'] / file_len[name] * 1.0
42     if token == '':
43         token_dict_tmp['_other'][token] = dictionary[token]
44     elif token[0] in token_dict_tmp.keys():
45         token_dict_tmp[token[0]][token] = dictionary[token]
46     else:
47         token_dict_tmp['_other'][token] = dictionary[token]
```

此时 27 个索引的字典已经建立完毕，遍历这 27 个字典，对每个字典按照 key 进行升序排序，创建对应的 json 文件，使用 json.dump 将 27 个字典分别写入对应的文件中，并 print 成功信息，最后返回所有文件总词数 total_word。

```
48 for key in token_dict_tmp.keys():
49     token_dict_tmp[key] = sort_dictionary(token_dict_tmp[key])
50     file1 = open('./index_file/index_' + key + '_' + str(block_num) + '.json', 'w')
51     json.dump(token_dict_tmp[key], file1)
52     file1.close()
53     print('index_' + key + '_' + str(block_num) + ' succeed')
54 return total_word
```

字典排序函数 **sort_dictionary** 如下，先对字典中的 key 值进行升序排序，再创建一个新的字典，将未排序的字典按照排序后的 key 值，对应复制到新字典中，最后返回新字典。

```
7 def sort_dictionary(unsort_dict):
8     keys = sorted(unsort_dict.keys())
9     sort_dict = {}
10    for key in keys:
11        sort_dict[key] = unsort_dict[key]
12    return sort_dict
```

跳转到主函数，输出最后得到的 block_num 和 total_word。

```
94 print('block_num = ' + str(block_num)) # block_num = 77
95 print('total_word = ' + str(total_word)) # total_word = 190163909
```

现在得到的索引文件为 77*27 个，接下来要做的是，把每块得到的相同字母开头（或其他符号）的索引合并到一起，即每 77 个索引合并成 1 个索引。

index_dict 字典用于合并索引时存储。

```
97 index_dict = {}
```

遍历生成的所有索引，因为索引在文件夹中按名称排序，所以只要将 document_files 中每 77 个索引合并到一起即可。

count 变量用于帮助统计读入的索引是否达到 77 个。

对于 index_file 文件夹中每个文件，使用 json.load() 读入文件中的字典，若 count 是 block_num 的倍数，则将读入的字典拷贝到 index_dict 中。

否则，对于读入的字典中的每个 key，若 key 不在 index_dict 的 keys 中，则将其添加到 index_dict 中；否则，计算更新 index_dict 中该 key 对应的总词数，并更新位置信息。

print 当前索引文件加载成功信息，再将其删除，减少不必要的存储空间。

```
99     for a in os.walk('./index_file'):
100         document_files = a[2]
101         count = len(document_files)
102         for file_name in document_files:
103             file2 = open('./index_file/' + file_name, 'r')
104             dict_tmp = json.load(file2)
105             file2.close()
106             if count % block_num == 0:
107                 index_dict = dict_tmp.copy()
108                 count -= 1
109             else:
110                 for word in dict_tmp.keys():
111                     if word not in index_dict.keys():
112                         index_dict[word] = dict_tmp[word]
113                     else:
114                         index_dict[word]['rate'] += dict_tmp[word]['rate']
115                         index_dict[word]['pos'].update(dict_tmp[word]['pos'])
116                 count -= 1
117             print(file_name[:-5] + ' load succeed')
118             os.remove('./index_file/' + file_name)
```



若 count 是 block_num 的倍数，则说明合并的索引文件已达 77 个，遍历 index_dict 字典中所有 word，对于每个 word 对应的总词数，将其替换成总词频。

对 index_dict 字典按照 key 进行升序排序，创建对应的 json 文件，使用 json.dump 将 index_dict 字典写入对应的文件中，并 print 成功信息。




























```
120     if count % block_num == 0:
121         for word in index_dict.keys():
122             index_dict[word]['rate'] = index_dict[word]['rate'] / total_word * 1.0
123         index_dict = sort_dictionary(index_dict)
124         file3 = open('./index_file/' + file_name[:-7] + '.json', 'w')
125         json.dump(index_dict, file3)
126         file3.close()
127         print(file_name[:-7] + ' succeed')
```

Result :

最后得到一个 map 索引表和一个文件夹 index_file，文件夹中存放有 27 个 token 索引文件。

 map	2018/4/10 22:02	JSON File	20,814 KB
 index_file	2018/4/11 0:39	文件夹	

脑 > 本地磁盘 (E:) > TTde > 学习 > 大三下 > 信息检索 > lab2 > index_file

名称	修改日期	类型	大小
 index_other	2018/4/11 0:43	JSON File	587,855 KB
 index_a	2018/4/10 23:56	JSON File	402,937 KB
 index_b	2018/4/10 23:58	JSON File	216,945 KB
 index_c	2018/4/11 0:04	JSON File	515,084 KB
 index_d	2018/4/11 0:07	JSON File	282,625 KB
 index_e	2018/4/11 0:09	JSON File	248,738 KB
 index_f	2018/4/11 0:10	JSON File	247,135 KB
 index_g	2018/4/11 0:11	JSON File	132,074 KB
 index_h	2018/4/11 0:12	JSON File	139,258 KB
 index_i	2018/4/11 0:14	JSON File	223,161 KB
 index_j	2018/4/11 0:14	JSON File	49,614 KB
 index_k	2018/4/11 0:14	JSON File	38,914 KB
 index_l	2018/4/11 0:16	JSON File	206,818 KB
 index_m	2018/4/11 0:18	JSON File	332,005 KB
 index_n	2018/4/11 0:19	JSON File	153,989 KB
 index_o	2018/4/11 0:21	JSON File	155,940 KB
 index_p	2018/4/11 0:24	JSON File	439,144 KB
 index_q	2018/4/11 0:25	JSON File	20,620 KB
 index_r	2018/4/11 0:27	JSON File	308,018 KB
 index_s	2018/4/11 0:33	JSON File	584,371 KB
 index_t	2018/4/11 0:35	JSON File	269,340 KB
 index_u	2018/4/11 0:36	JSON File	96,477 KB
 index_v	2018/4/11 0:36	JSON File	65,020 KB
 index_w	2018/4/11 0:37	JSON File	165,997 KB
 index_x	2018/4/11 0:37	JSON File	3,886 KB
 index_y	2018/4/11 0:38	JSON File	35,668 KB
 index_z	2018/4/11 0:38	JSON File	4,661 KB

压缩 (γ 编码) :

index_compress.py :

使用 γ 编码对创建好的 27 个索引进行压缩，但并不懂为什么能够进行压缩，VB 编码也是，看 ppt，似乎 VB 编码和 γ 编码都是以 01 形式存储，但是转换成 01 的话不是位数更多了吗，并不太懂，只是按照自己的理解写了一个 γ 编码，转换成 01 的编码形式，再存储到字典的 key 中，但这种方法并不能达到压缩效果，索引文件反而更大了。

至于 VB 编码，因为感觉跟上述 γ 编码一样，也是位数更多，因此没有尝试写。