

## Task 1

- Tkey-value pairs with (termID, docID).
- The parser writes the output to the segment files.
- The location of the relevant segment files.
- The collected values (docIDs) for a given key (termID).

### Discssion:

The right amount of parsers depend on the size of the input and how it is splitted. Therefore it depends on how well the input can be splitted in splits with a reasonable size. Each parser should have a similar amount of data to work on. If we look for example at the English alphabetic characters, we could decide to split 26 characters into equal parts, e.g. divide 26 by 4 or by 6. For a well-balanced working speed each splitted input should be of the same size and the absolute number of splits shouldn't be too large for better efficiency. Therefore one parser for each term would be inefficient while one parser overall wouldn't be useful because there is no need for splitting data.

The parsers write the segment files, where one file exists for each term partition. We need an inverter for each term partition, because the master gives for each term another inverter.

## Task 2

Break the update document into several splits and assign the splits to idle computer which parses these splits and outputs partitions of terms. Inverter uses these partitions to write new postings lists and merge them with the existing.

Only use the idle computer so that the entire system will not slow down too much.

## Task 3

### Subtask 3.1

$$k = 10, b \approx 0,5$$

### Subtask 3.2

$$M = 100000$$

## Task 4

$$216_{10} = 11011000_2$$

**variable byte code:** 00000001 11011000

**gamma code:**

Offset: 1011000

$$\text{lenght } 7_{10} = 11111110_2$$

$\gamma$ -code: 111111101011000

## Task 5

**Sequence:** 1111011000100110000

## Programming Task 2

### Subtask 2

```
1 import csv, re, nltk
2
3 class Search:
4
5     def __init__(self, filename: str):
6         self.filename = filename
7         self.index = self.getIndex()
8         self.dictionary, self.postings_lists = self.getIndex()
9         self.bigrams_index = self.getBigramIndex()
10        self.bigrams_dictionary, self.bigrams-postings_lists = self.bigrams.index
11
12    def getIndex(self):
13        dictionary = {}
14        postings_lists = []
15
16        tokenizer = nltk.RegexpTokenizer(r"\w+")
17
18        with open(self.filename, 'r') as file:
19            reader = csv.reader(file, delimiter = '\t')
20            postings = []
21
22            #iterate through each row of the table
23            for row in reader:
24                #(doc_id, url, pub_date, title, news_text) = row
25                docID = row[0]
26                news_text = row[-1]
27
28                #tokenize and normalize news text
29                #this procedure will remove symbols like !?() etc.
30                #the set data structure will remove all duplicates
31                news_text_norm = set(tokenizer.tokenize(news_text.lower()))
32
33                #generate postings
34                #iterate through each term
35                for term in news_text_norm:
36                    postings.append((term, docID))
37
38            #sort postings
39            postings = sorted(postings[1:], key = lambda tup: tup[0])
40
41
42        post_id = 0
43        post_size = 0
44        #iterate through postings
45        for posting in postings:
46            term, doc_id = posting
47
48            if term not in dictionary:
49                #update the dictionary with the new term
50                #initialize the postings size
51                #save the postings id,
```

```
52         #witch is the position of the postings list
53         #into the postings lists
54         dictionary.update({term: [post_size+1, post_id]})
55
56         #initialize a new postings list
57         postings_lists.append([ doc_id ])
58
59         #update postings id
60         post_id +=1
61     else:
62         #update size of posting
63         dictionary[term][0] += 1
64
65         #update postings list
66         postings_lists[-1].append(doc_id)
67
68     return dictionary , postings_lists
69
70
71 def getPostingList(self , postings_listID: int) -> list:
72     """Will return a list with the postings given the postings list ID.
73
74     Args:
75         postings_listID (int): The ID of the postings list.
76
77     Returns:
78         list: return the list with the postings.
79     """
80     return self.postings_lists[postings_listID]
81
82 def query(self , term1: str , term2: str = '') -> list:
83     """Search if one or two terms are contained in the same document.
84     Then returns the document ID and the news text.
85
86     Args:
87         term1 (str): A term
88         term2 (str, optional): A term or nothings. Defaults to ''.
89
90     Returns:
91         list: A list of results
92     """
93
94     #dictionary , postings_lists = self.index
95     out_list = []
96
97     #CASE 1: only one term
98     if term2 == '':
99         postID = self.bigrams_dictionary[term1][1]
100         postings_list = self.getPostingList(postID)
101
102         #retrive text
103         with open(filename , 'r') as file:
104             reader = csv.reader(file , delimiter = '\t')
105
106             #iterate through each row of the table
107             for row in reader:
```

```
108         docID = row[0]
109         news_text = row[-1]
110
111         if docID in postings_list:
112             out_list.append((docID, news_text))
113
114     #CASE 2: two terms
115     else:
116         intersection_list = []
117
118         term1_postID = self.bigrams_dictionary[term1][1]
119         term2_postID = self.bigrams_dictionary[term2][1]
120
121         term1_postings_list = self.getPostingList(term1_postID)
122         term2_postings_list = self.getPostingList(term2_postID)
123
124         #intersection algorithm
125         for term1_docID in term1_postings_list:
126             for term2_docID in term2_postings_list:
127                 if term1_docID == term2_docID :
128                     intersection_list.append(term1_docID )
129
130         #retrive text
131         with open(filename, 'r') as file:
132             reader = csv.reader(file, delimiter = '\t')
133
134             #iterate through each row of the table
135             for row in reader:
136                 #(docID, url, pub_date, title, news_text) = row
137                 docID = row[0]
138                 news_text = row[-1]
139
140                 if docID in intersection_list:
141                     out_list.append((docID, news_text))
142                     #out_list.append((news_text))
143
144     return out_list
145
146 def getTermBigrams(self, term: str):
147     """Returns the bigrams of a given term.
148
149     Args:
150         term (str): A term.
151
152     Returns:
153         tuple: The bigrams of a term.
154     """
155
156     # solve some issues
157     if not term:
158         tuple_bigrams = ()
159
160     # bigrams for wildcards on the left side
161     elif term[0] == '*':
162         tuple_bigrams = tuple(list(nltk.bigrams(term)) + [(term[-1], '$')])[1:]
163
```

```
164     # bigrams for wildcards on the right side
165     elif term[-1] == '*':
166         tuple_bigrams = tuple([( '$', term[0])] + list(nltk.bigrams(term))[:-1])
167
168     # bigrams
169     else:
170         tuple_bigrams = tuple([( '$', term[0])] + list(nltk.bigrams(term)) + [(
171             term[-1], '$')])
172
173     bigrams = []
174     for bigram in tuple_bigrams:
175         #join the bigram tuple into one string
176         bigrams.append(''.join([char for char in bigram])).strip()
177
178     return tuple(bigrams)
179
180 def getBigramIndex(self):
181     """Generate a Bigram Index from an other Index"""
182
183     #generate a new dictionary witch contains
184     #bigrams of the terms as the key
185     bigrams_dictionary = {}
186     for term in self.dictionary:
187         bigrams_dictionary.update({self.getTermBigrams(term): self.dictionary[
188             term]})
189
190     return bigrams_dictionary, self.postings_lists
191
192 def getWildcardTerms(self, term: str) -> list:
193     """Retuns a list of terms for a given term with a wildcard.
194     The terms will be returned in the form of bigrams.
195
196     Args:
197         term (str): A term or a part of it.
198
199     Returns:
200         [list]: A list of term's bigrams
201     """
202     out_list = []
203     bigrams_term_wildcard = self.getTermBigrams(term)
204
205     if '*' not in term:
206         out_list.append(bigrams_term_wildcard)
207
208     # wildcard on the right side
209     elif term[0] == '$' and term[-1] != '$':
210         for bigrams_term_dictionary in self.bigrams_dictionary:
211             if bigrams_term_dictionary[0:len(bigrams_term_wildcard)] ==
212                 bigrams_term_wildcard:
213                 out_list.append(bigrams_term_dictionary)
214
215     #wildcard on the left side
216     elif term[0] != '$' and term[-1] == '$':
217         for bigrams_term_dictionary in self.bigrams_dictionary:
```

```
217         if bigrams_term_dictionary[:, -1][0:len(bigrams_term_wildcard)] ==
218             bigrams_term_wildcard[:, -1]:
219             out_list.append(bigrams_term_dictionary)
220
221     # wildcard in the of the term or no wildcard
222     else:
223         term_splits = term.split('*')
224         term_split_1 = self.getTermBigrams(term_splits[0])[:, -1]
225         term_split_2 = self.getTermBigrams(term_splits[1])[1:]
226
227         for bigrams_term_dictionary in self.bigrams_dictionary:
228             if bigrams_term_dictionary[:, len(term_split_1)] == term_split_1:
229                 if bigrams_term_dictionary[:, -1][len(term_split_2)] ==
230                     term_split_2[:, -1]:
231                     out_list.append(bigrams_term_dictionary)
232
233     return out_list
234
235 def queryWildcards(self, term1: str, term2: str) -> list:
236     """Returns the resoult of a query with wildcards implementation.
237     A query for every term in the list of terms found for a given wildcard.
238
239     Args:
240         term1 (str): A term.
241         term2 (str): A term.
242
243     Returns:
244         list: A list with the results of all the queries.
245     """
246     out_list = []
247     bigrams_list_term1 = self.getWildcardTerms(term1)
248     bigrams_list_term2 = self.getWildcardTerms(term2)
249
250     for bigrams_term1 in bigrams_list_term1:
251         for bigrams_term2 in bigrams_list_term2:
252             out_list.append(self.query(bigrams_term1, bigrams_term2))
253
254     return out_list
255
256
257 if __name__ == "__main__":
258     filename = 'assignment1/code/postillon.csv'
259     search = Search(filename=filename)
260
261     #print(search.query(search.getTermBigrams('wei  '), search.getTermBigrams('ma  e '))
262     #print(search.query(search.getTermBigrams('weiss '), search.getTermBigrams('ma  e
263     #print(search.query(search.getTermBigrams('wei  '), search.getTermBigrams('masse
264     #print(search.query(search.getTermBigrams('weiss '), search.getTermBigrams('masse
265
266     #wildcards
```

```
267 #print(search.queryWildcards('wei  ', 'ma  e'))  
268 #print(search.queryWildcards('weiss', '*a  e'))  
269 print(search.queryWildcards('wei*', '*asse'))  
270 #print(search.queryWildcards('wei*s', 'm*sse'))
```

code/script.py