

Pen and Paper Task 1

Subtask A

- sun \rightarrow 1, 2, 3, 4, 7
- nice \rightarrow 1, 5, 6
- water \rightarrow 5, 6, 8, 9
- is \rightarrow 6
- beer \rightarrow 10

Subtask B

- sun \rightarrow 1, 2, 3, 4, 7
- nice \rightarrow 1, 5, 6
- water \rightarrow 5, 6, 8, 9
- is \rightarrow 6
- beer \rightarrow 10

Example query: *nice AND is*

1. Comparisons without skip pointers: 1 & 6, 5 & 6, 6 & 6 \Rightarrow 3 comparisons

2. Comparisons with skip pointers: 1 & 6, 6 & 6 \Rightarrow 2 comparisons

Without skip pointers we must compare the terms step by step, although 5 in **nice** is still smaller than 6 in **is**. With skip pointers we can skip the 5 in **nice** and directly go to the 6 in **nice**.

Pen and Paper Task 2

```
1  tokenize(text: string):  
2      token = ''  
3      list_of_tokens = []  
4  
5      for char in text:  
6          if char == ' ': #whitespace  
7              list_of_tokens.add(token)  
8              token = '' #empty string  
9          if char is a symbol:  
10             list_of_tokens.add(token)  
11             list_of_token.add(char)  
12             token = '' #empty string  
13         else:  
14             token += char  
15  
16     return list_of_tokens
```

Pen and Paper Task 3

Query: *Gates /2 Microsoft*

(Gates, 4): $\{1:[3], 2:[6], 3:[2, 17], 4:[1]\}$

(Microsoft, 4): $\{1:[1], 2:[1, 21], 3:[3], 5:[16, 22, 51]\}$

cross product = $\{ 1:[(3,1)], 2:[(6,1), (6,21)], 3:[(2,3), (17,3)], 4:[(1,16), (1,22), (1,51)] \}$

From all tuples in the cross product, the tuples (3,1) and (2,3) fulfill the query's condition $\text{abs}(\text{tuple}[1] - \text{tuple}[0]) \leq 2$. So the answer is: document 1, document 3.

Programming Task 1

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

```
54
55         #update postings list
56         postings_lists[-1].append(doc_id)
57
58     return dictionary, postings_lists
59
60
61 def query(term_1: str, term_2: str = ''):
62     if term_2 == '':
63         #only term_1
64         pass
65     else:
66         #term_1 AND term_2
67         pass
68     pass
69
70
71 if __name__ == "__main__":
72     index()
```

code/script.py