# Assignment 1

07.11.23

Maria Bruno (3636989), Francesca Carlon (3664579), Linnet Moxon (3387157)

**Task 1**

**a)**

water -> 1 2 3 4 7

nice -> 1 5 6

sun -> 5 6 7 8 9

is -> 6

**b)**

water -> 1 2 3 4 7

nice -> 1 5 6

sun -> 5 6 7 8 9

is -> 6

intersection of *water* and *nice* without skip pointer:

start with *nice*, because the postings-list is shorter. Compare 1 from *nice* and 1 from *water*. There is a match. Increase the number in both lists. Compare 5 from *nice* and 2 from *water*. No match. Compare 5 from *nice* and 3 from *water*. No match. Compare 5 from *nice* and 4 from *water*. No match. 4 in *water* is the lower number, increase it. Compare 7 from *water* with 5 from *nice*. No match. Increase *nice*. Compare 6 from *nice* to 7 from *water*. No match.

**Total: 1 Match**

Intersection of *water* and *nice* with skip pointer:

If we go from 1 to 4 in *water*, we skip the comparison of 5 from *nice* to 2 and 3 in *water* and therefore save some steps.

Skip pointer in sun:

In theory you could add a skip pointer here, although it's not necessary, because except from *sun* no postings-list goes beyond 7 and therefore the intersection ends after 7 anyways.

**Task 2**

```
def tokenize(text):

    string = text

    str_w_spaces = " "

    for character in string():

        if character is not letter and is not number:

            character = white_space  + character + white_space

        else:

            character = character

        str_w_spaces =+ character

    list_of_tokens = str_w_spaces.SplitAtWhiteSpace()       # as the function split() in python, that will return a list of tokens, split at white space

    return list_of_tokens
```

The function takes "O'Really isn't in S." as input string. For every character in the string, that is not a letter or a number (here: apostrophe and dot) it adds a white space around that character. All the characters – punctuation padded with white space – are then added to a new string. This new string is split into a list of tokens at white space and saved as list_of_tokens. This list is returned in the end.

If you give the function the sentence "O'Really isn't in S." as input string, it will return

["O", "'", "Really", "isn", "'", "t", "in", "S", "."]

The algorithm has a linear runtime in the number of input characters n (O(n)) because, in the for-loop, it goes through each element of the string just once.

**Task 3**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | v1 (j) | v2 | v3 | v4 | v5 |
| | | | **M** | **O** | **U** | **S** | **E** |
| | | **0** | **1** | **2** | **3** | **4** | **5** |
| u1 (i) | **M** | **1** | 0 | 1 | 2 | 3 | 4 |
| u2 | **U** | **2** | **1** | **1** | **1** | **2** | **3** |
| u3 | **O** | **3** | **2** | **1** | 1* | **2** | **3** |
| u4 | **S** | **4** | **3** | **2** | **2** | **1** | **3** |
| u5 | **E** | **5** | **4** | **3** | **3** | **3** | **1** |

*because we look above two levels -> u1v1 = 0 => u3v3 = 0+1 = 1

When filling the matrix according to the Damerau-Levenshtein algorithm rules the cell u3v3 fits the condition $u_i = v_{j-1}$ and $u_{i-1} = v_j$. Therefore to fill it, we look at all cells two levels above, two levels to the left, and two levels diagonally. This means, for u3v3 we consider u1v1, u1v2, u1v3, u2v1, u3v1, according to $D_{i-2,\ j-2}$. We add +1 for the operation "transpose" to each one of the cells. The minimum value is 1.

All the other cells are filled according to the other rules of the Damerau-Levenshtein algorithm (insert, delete, replace).

**Programming Task 1**

```
"""
    This file stores all terms from the tweets.csv-file as
keys in a dictionary.
    The values are the number of documents that term occurs
in.
    The postings-lists and the method query() are to be added.
"""


"""
    prepare_file:   opens file and extracts relevant infor-
mation
                    tokenizes and normalizes the text of the
tweets
                    :return a list of the form ['ID', ['term',
'term', ...], 'ID', ...]
"""


def prepare_file(filename):
    # create list to store relevant information
    tweets = []
    # create list to store ID and tokenized and normalized
text
    new_content = []
    # read data and store it in a list (line by line)
    with open(filename, encoding="utf8") as f:
        for lines in f:
            line = lines.split('\t')
            line = [line[1], line[4]]
            tweets.append(line)
    # call tokenize_and_normalize to prepare the tweets
    for lst in tweets:
        new_content.append(lst[0])
        new_content.append(tokenize_and_normalize(lst[1]))
    # return list with IDs and tokenized, normalized tweets
    return new_content


"""
    helper function for tokenization and normalization
    future normalization to be added: clean for emojis, stop
words, punctuations, web-links
    :return list of tokens in lower case
"""


def tokenize_and_normalize(text):
    normalized_token = []
    # split text by whitespace
```

```
    tokens = text.split()
    # normalize all tokens with lower case
    for word in tokens:
        normalized_token.append(word.lower())
    return normalized_token


"""
    loop through list to store every term as key once
    loop through list to count number of documents the term 
appears in
    :return dictionary of the form {'term': number of docu-
ments the term occurs in}
"""


def create_dict(filename):
    content = prepare_file(filename)
    term_dict = {}
    for item in content:
        for word in item:
            if word not in term_dict.keys():
                term_dict.setdefault(word)
    return term_dict

# at this point the function returns a dictionary with the 
terms as keys, and no values
# the problem here is the form: ['tweet-ID', [list of normal-
ized tokens], 'tweet-ID', [list of normalized tokens], ...]
# where no mapping (pointing) between tweet-ID and text is 
happening.
# therefore we cannot loop through the list and count in which 
documents the term occurs in
# idea for the future: in prepare_file store an initial_dic-
tionary with the tweet-ID as key, and the list of normalized 
tokens as
# value: {ID1: [list of normalized tokens], ID2: [list of nor-
malized tokens], ...]
# then create new_dictionary of form {'term': size, 'term': 
size, ...} by looping over the values of
# the initial_dictionary and extract the terms.
# next step: for every term (= key in new_dictionary) loop 
over all values in initial_dictionary and
# increase value for term in new_dictionary by one, if term is 
found in one document.
```