Alberto Saponaro - saponaroalberto97@gmail.com
Walter Väth - walter.vaeth@gmail.com
Chong Shen - st143575@stud.uni-stuttgart.de
Xin Pang - Email

Task 1

- The master assign a split to the parser
- The parser emits (term, docID)-pairs
- The master gives the postings to the inverter
- The inverter sorts the postings and write them in the posting list

Task 2

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
lenght $7_{10} = 11111110_1$
γ-code: 111111101011000

Task 5

Sequence: 1111011000100110000

Alberto Saponaro - saponaroalberto97@gmail.com
Walter Väth - walter.vaeth@gmail.com
Chong Shen - st143575@stud.uni-stuttgart.de
Xin Pang - Email

Programming Task 2

Subtask 2

```
import csv, re, nltk

class Search:

    def __init__(self, filename: str):
        self.filename = filename
        self.index = self.getIndex()
        self.dictionary, self.postings_lists = self.getIndex()
        self.bigrams_index = self.getBigramIndex()
        self.bigrams_dictionary, self.bigrams_postings_lists = self.bigrams_index

    def getIndex(self):
        dictionary = {}
        postings_lists = []

        tokenizer = nltk.RegexpTokenizer(r"\w+")

        with open(self.filename, 'r') as file:
            reader = csv.reader(file, delimiter = '\t')
            postings = []

            #iterate through each row of the table
            for row in reader:
                #(doc_id, url, pub_date, title, news_text) = row
                docID = row[0]
                news_text = row[-1]

                #tokenize and normalize news text
                #this procedure will remove symbols like !?() etc.
                #the set data structure will remove all duplicates
                news_text_norm = set(tokenizer.tokenize(news_text.lower()))

                #generate postings
                #iterate through each term
                for term in news_text_norm:
                    postings.append((term, docID))

            #sort postings
            postings = sorted(postings[1:], key = lambda tup: tup[0])


            post_id = 0
            post_size = 0
            #iterate through postings
            for posting in postings:
                term, doc_id = posting

                if term not in dictionary:
                    #upate the dictionary with the new term
                    #initialize the postings size
                    #save the postings id,
```

Alberto Saponaro - saponaroalberto97@gmail.com
Walter Väth - walter.vaeth@gmail.com
Chong Shen - st143575@stud.uni-stuttgart.de
Xin Pang - Email

```
                    #witch is the position of the postings list
                    #into the postings lists
                    dictionary.update({term: [post_size+1, post_id]})

                    #initialize a new postings list
                    postings_lists.append([doc_id])

                    #update postings id
                    post_id +=1
                else:
                    #update size of posting
                    dictionary[term][0] += 1

                    #update postings list
                    postings_lists[-1].append(doc_id)

        return dictionary, postings_lists


    def getPostingList(self, postings_listID: int) -> list:
        """Will return a list with the postings given the postings list ID.

        Args:
            postings_listID (int): The ID of the postings list.

        Returns:
            list: return the list with the postings.
        """
        return self.postings_lists[postings_listID]

    def query(self, term1: str, term2: str = '') -> list:
        """Search if one or two terms are contained in the same document.
        Then returns the document ID and the news text.

        Args:
            term1 (str): A term
            term2 (str, optional): A term or nothings. Defaults to ''.

        Returns:
            list: A list of results
        """

        #dictionary, postings_lists = self.index
        out_list = []

        #CASE 1: only one term
        if term2 == '':
            postID = self.bigrams_dictionary[term1][1]
            postings_list = self.getPostingList(postID)

            #retrive text
            with open(filename, 'r') as file:
                reader = csv.reader(file, delimiter = '\t')

                #iterate through each row of the table
                for row in reader:
```

Alberto Saponaro - saponaroalberto97@gmail.com
Walter Väth - walter.vaeth@gmail.com
Chong Shen - st143575@stud.uni-stuttgart.de
Xin Pang - Email

```
                    docID = row[0]
                    news_text = row[-1]

                    if docID in postings_list:
                        out_list.append((docID, news_text))

        #CASE 2: two terms
        else:
            intersection_list = []

            term1_postID = self.bigrams_dictionary[term1][1]
            term2_postID = self.bigrams_dictionary[term2][1]

            term1_postings_list = self.getPostingList(term1_postID)
            term2_postings_list = self.getPostingList(term2_postID)

            #intersection algorithm
            for term1_docID in term1_postings_list:
                for term2_docID  in term2_postings_list:
                    if term1_docID  == term2_docID :
                        intersection_list.append(term1_docID )

            #retrive text
            with open(filename, 'r') as file:
                reader = csv.reader(file, delimiter = '\t')

                #iterate through each row of the table
                for row in reader:
                    #(docID, url, pub_date, title, news_text) = row
                    docID = row[0]
                    news_text = row[-1]

                    if docID in intersection_list:
                        out_list.append((docID, news_text))
                        #out_list.append((news_text))

        return out_list

    def getTermBigrams(self, term: str):
        """Returns the bigrams of a given term.

        Args:
            term (str): A term.

        Returns:
            tuple: The bigrams of a term.
        """

        # solve some issues
        if not term:
            tuple_bigrams = ()

        # bigrams for wildcards on the left side
        elif term[0] == '*':
            tuple_bigrams = tuple(list(nltk.bigrams(term)) + [(term[-1], '$')])[1:]

```

Alberto Saponaro - saponaroalberto97@gmail.com
Walter Väth - walter.vaeth@gmail.com
Chong Shen - st143575@stud.uni-stuttgart.de
Xin Pang - Email

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

Alberto Saponaro - saponaroalberto97@gmail.com
Walter Väth - walter.vaeth@gmail.com
Chong Shen - st143575@stud.uni-stuttgart.de
Xin Pang - Email

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

Alberto Saponaro - saponaroalberto97@gmail.com
Walter Väth - walter.vaeth@gmail.com
Chong Shen - st143575@stud.uni-stuttgart.de
Xin Pang - Email

```
267     #print(search.queryWildcards('weiß', 'maße'))
268     #print(search.queryWildcards('weiss', '*aße'))
269     print(search.queryWildcards('wei*', '*asse'))
270     #print(search.queryWildcards('wei*s', 'm*sse'))
```

code/script.py