Exercise 1 – Basic Search System – Report

Running the program

```
Enter type of normalization for Vocabulary (s for stemming or f for case folding):
s
Enter type of indexing (bag for bag-of-words or bi for biword):
bag
Enter if document-length should be considered by scoring-algorithm (y for yes or n for no):
y
Enter path to newsgroups(data):
D:/Wolfi/workspace_intellij/gir-ex1/data/20_newsgroups_subset/alt.atheism
Indexing...
Enter path to topic-file:
D:/Wolfi/workspace_intellij/gir-ex1/data/topics/topic1
```

figure 1: inputs at the CLI

To start the program, start Main via Command Line.

At the start of the program choose between stemming and case folding. After that, you can choose between bi-word- and bag-of-words-index. Finally, you can choose if you want the document-length to be considered by the scoring-algorithm.

Then the program asks for the path to the newsgroups-folder. If a valid folder is given, the indexing with the desired index starts. When the indexing is done, a valid path to a topic file is needed. After that, the results are printed of the search are printed.

Technical Report

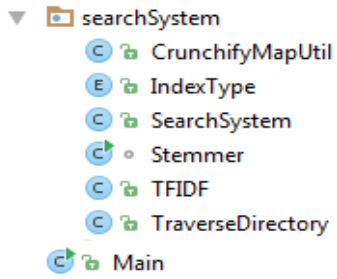

figure 2: structure

Tokenizing & Index Creation

The Main-Class is the class which is runnable. It is responsible for managing the CLI, and checking user inputs for validity. The directory with to be indexed is given to SearchSystem. There the folder is traversed (Class TraverseDirectory), a list of files is created (traverseDirectory()), and each of the files is then indexed (indexFile()).

Natasa Nikic, 1325974
Wolfgang Weintritt, 1327191

Before the words are filtered out, the header is removed from each file. Special characters like .,?!:;"'_ and some more are removed. Multiple spaces in a row are replaced by one space. Special Strings (email-addresses) are removed.

Now the rest of the file should be only normal words. The lines are connected and analysed via indexConcatLines(). There our two datastructures are filled.

```
// dictionary: word => hashmap <file, occurrence-list>
private HashMap<String, HashMap<String, ArrayList<Integer>>> dictionary;
// documents: file => hashmap <word, number of occurrences>
private HashMap<String, HashMap<String, Integer>> documents;
```

Every word (or bi-word) is stemmed (or case-folded) and then saved to those two lists. Stemming is done via the Class Stemmer, which was taken from http://www.tartarus.org/~martin/PorterStemmer. This class implements the Porter Stemming Algorithm for its functionality. A tolerant match is implemented, by the Class Soundex from the Apache Commons Codec 1.10 API.

We create two lists, because it is easier to create the TF-IDF-Score with this information.

Scoring

Then the TF-IDF-score is calculated (Class TFIDF). For each word in each directory, the TF-IDF-score is calculated, and saved in a HashMap.

```
// tfidf: word => hashmap <file, score>
private HashMap<String, HashMap<String, Double>> tfidf;
```

After the user enters a word, the file is also indexed via the same methods, but saved into another HashMap:

```
 // topicWords: word => log(occurrences)
private HashMap<String, Double> topicWords;
```

It uses logarithm, so that a word, which occurs x times more often than another word in a query, isn't x-times more important.

Then this list is given to the TFIDF-Class. The score of a document d for a query q is:

$$\text{Score}(q,d) = \sum_{t \in q \cap d} \text{tf.idf}_{t,d}$$

If the user chooses to consider the document-length, the following formula is used to calculate tf:

$$\text{tf}(t,d) = \frac{f(t,d)}{\max\{f(w,d) : w \in d\}}$$

Natasa Nikic, 1325974
Wolfgang Weintritt, 1327191

After the score of all documents is created, and saved in a HashMap, this HashMap has to be sorted by values. This is done with the class CrunchifyMapUtil. The code is from *http://crunchify.com/java-how-to-sort-a-map-on-the-values-the-map-interface-java-collections/*. After this step, the output is printed to the CLI, an output-txt-file is created, and another search can be started.

```
Enter path to topic-file:
D:/Wolfi/workspace_intellij/gir-ex1/data/topics/topic1

sortedCrunchifyMapValue:

topic1  Q0  alt.atheism\53146   1   0.6696610982669922  group2-exercise1
topic1  Q0  alt.atheism\53421   2   0.5529435490602153  group2-exercise1
topic1  Q0  alt.atheism\51144   3   0.5162967946174646  group2-exercise1
topic1  Q0  alt.atheism\53466   4   0.49420986528621047 group2-exercise1
topic1  Q0  alt.atheism\53057   5   0.48304507324835416 group2-exercise1
topic1  Q0  alt.atheism\53273   6   0.4631278813007677  group2-exercise1
```

figure 3: score-output