Week 2:
Text Transformation

- This assignment is due on **28th October, 2015 (13:30)**
- You can discuss the problems with other groups of this course or browse the Internet to get help. However, copy and paste is cheating.
- There are 13 weekly exercises in total. In each one of them, all assignments sum up to 20 points. You need to achieve at least 80% of all assignments during the course in order to participate in the final exam. Hence, you need to achieve at least 208 points in total (*13*20*0.8=208*).
- Submission at
  `https://www.dcl.hpi.uni-potsdam.de/submit/`
  - only pdf files
  - one file per group per week (week2.pdf)
  - put your names and matriculation numbers on *each* sheet in the pdf file

Assignment 1: Text Statistics

Zipf's law and Heaps' law belong to a class called power laws.

a) Name two other relations that follow a power law apart from term frequencies in languages. **2 P**

b) What are the 3 most occuring words in English and in German? **1 P**

Assignment 2: Text Processing

Raw text documents are usually pre-processed before an index is build.

a) In most cases, it is not trivial how to interpret a patent claim. The patent language usually contains legal terms and expressions, or new definitions of existing words. In addition, when considering two synonyms, it is not obvious that they have the same meaning when used in a patent text. Give some examples of such words. **1 P**

b) What are the pros and cons of using a stopword filter? **3 P**

c) Are there any specific stopwords for patent data retrieval used in the literature? **1 P**

d) What are the pros and cons of stemming and lemmatization? **3 P**

e) When does stemming take place, at indexing or query time? Exlain your answer. **2 P**

f) Can stemming lower precision or recall in a simple keyword retrieval system? Exlain your answer. **3 P**

Assignment 3: (Programming) Indexing

In last week's assignment you learnt how to read data from an XML stream and you downloaded the patent zip files. In this week, we will use the testData.xml file to construct our index as follows:

- You need to override `abstract void index(String directory)`. This method is initially defined in the *SearchEngine* abstract class. It is also included in the *SearchEngineTemplate* class, which extends *SearchEngine*. You should leave the latter unchanged and replace the name of the *SearchEngineTemplate* class according to your team name, for example *SearchEngineMyTeamName*. This will be the file where you build your implementation.

- In the index method you need to implement an XML parser and generate the index. Design your index in a way that allows fast searching later on. The time for index creation is not so critical and we can speed it up later by using multi-threading.
  - First you need to pre-process the raw input. Decide on how to tokenize, whether to use a stopword list and/or stemming. For these steps you can use existing code — you don't need to come up with a stopword list or implement a new stemmer! There are many well-used stopword lists, stemming algorithms etc. in the literature — although you are welcome to evaluate them and choose the most effective ones on patent data.
  - For each patent you should only keep the text of the *abstract* element. This will be considered the text of each document (each patent) and the document ID will be the *doc-number* in the *publication-reference*.
  - We want to be able to answer phrase queries at the end. One possibility to this end is to store the position of a term in a document in the posting lists.
  - Considering the size of the entire dataset in our future scaling, you need to take into account the memory limitations during your implementation. Thus, you need to find clever ways to generate the index in parts. Afterwards you should merge the index parts back into one huge index file. (hint: Java offers the class `RandomAccessFile` to read (and write) in a big file without loading all its content into main memory.)
  - Considering again the memory contraints when you scale up your implementation, it is important to generate a *seek list* (an index for your index) which maps terms to file-offsets in the huge index file and store it together with the index.
  - Note that storing each String term and Integer offset would need too much space. Although the seek list may fit into the main memory when using the small testing file, it will not fit when you use the overall dataset.
- Once you have generated your index you need to prepare for answering queries. For this purpose you need to override
  `abstract boolean loadIndex(String directory)` in
  your *SearchEngineMyTeamName* class to load the seek list and to start your search engine.
- Last but not least, we want to implement a *simple keyword search* this week. Therefore you need to override `abstract ArrayList<String> search(String query, int topK, int prf).` Ignore 'topK' and 'prf' for now. The return value should be a list of patent invention titles of the documents that contain the query term. (hint: don't forget to pre-process the query)

a) Print the list of the patent invention titles in the development set that match the queries:

- "selection" **1 P**
- "device" **1 P**
- "justify" **1 P**
- "write" **1 P**

The ordering of the titles does not matter at this point.

b) Write down your implementation choices so far. Namely, the source where you downloaded your stopword list and your stemming algorithm etc.