**Author: Long Pan**

# 1. Basic Idea

I have experience on building a text-based search engine for a course project, so I believe some techniques can also be applied for this assignment. Here is my basic idea:

Step1: Extract all the words from the web page by parsing HTML with the help of Jsoup.

Step2: Scan the document and store all possible phrases along with their frequency, the number of occurrence, which would be used for scoring. Here I use a Trie data structure to do so.

Step3: Return k phrases which are most relevant to the content of the web page, based on their frequency. This is done by partial sorting.

# 2. Stop words

Stop Words are words which do not contain important significance to be used in Search Queries, such as *the*, *a*, *of* and *to* etc. The same is also true with this assignment. I simply eliminated all the stop words while scanning.

# 3. Stemming

For grammatical reasons, documents are going to use different forms of a word, such as *organize*, *organizes*, and *organizing*. In many situations, it is useful for a search for considering different forms of a word as a same one. That’s the reason I feel stemming is a must. Fortunately, I was able to find a existing java class for doing stemming, and I did a little encapsulation on it for my own purpose.

# 2. Trie

# I implemented a trie structure for storing all the phrases. The difference between this one and a common trie structure is that the value of each node is a string, and the path from root to any node forms a phrase, the frequency of which is stored inside that node. The biggest advantage of using trie is that it saves a lot space since many phrase share the same prefixes.

# 3. Partial Sorting

Total sorting is the problem of returning a list of items such that its elements all appear in order, while partial sorting is returning a list of the k smallest (or k largest) elements in order. Heaps admit a simple single-pass partial sort when k is fixed: insert the first k elements of the input into a max-heap. Then make one pass over the remaining elements, add each to the heap in turn, and remove the largest element. Each insertion operation takes O(log k) time, resulting in O(n log k) time overall.

# 4. Sample output

# 

