Here 6 links are used for reading from the web using jsoup. The paragraphs from the webpages are read and the words are extracted and stored in a triedata structure.

Now, to find the word entered by the user, trie data structure would be used to search as follow:



Hashmap is used to find the count of occurrences of a word from each webpage. The hasmap would display their word count respectively and if the word is not present in one of the document the count is shown as zero.

To remove the stop words, as we read each word from the webpage’s paragraph pass that word to the stopword function containing a list of all stop words (stopwords.txt). If a match is found that word is not inserted into a trie datastructure as well the hashmap.

Also, to display the total of all the words from all the documents I maintained a separate map which displays the words as well their total occurrences found from all the documents.

For ranking, there is ranked based on number of occurrences of the word on a page.   
For ranking, there is sorted using mergesort of it to be efficient and fast.

Used in here:

pages.**sort**(new PageComparator());

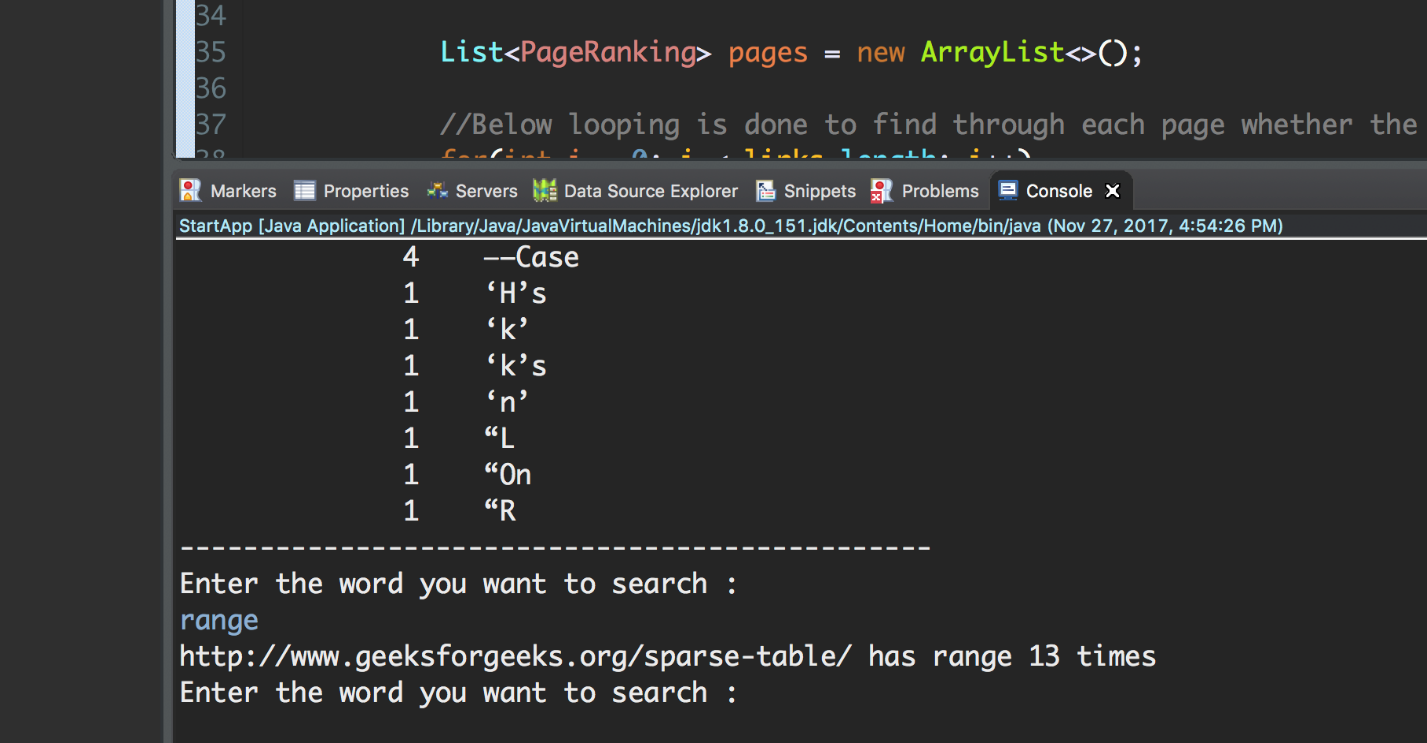
for (PageRanking page : pages) {

System.out.println(page.getName() + " has " + word + " " + page.getRank() + " times");

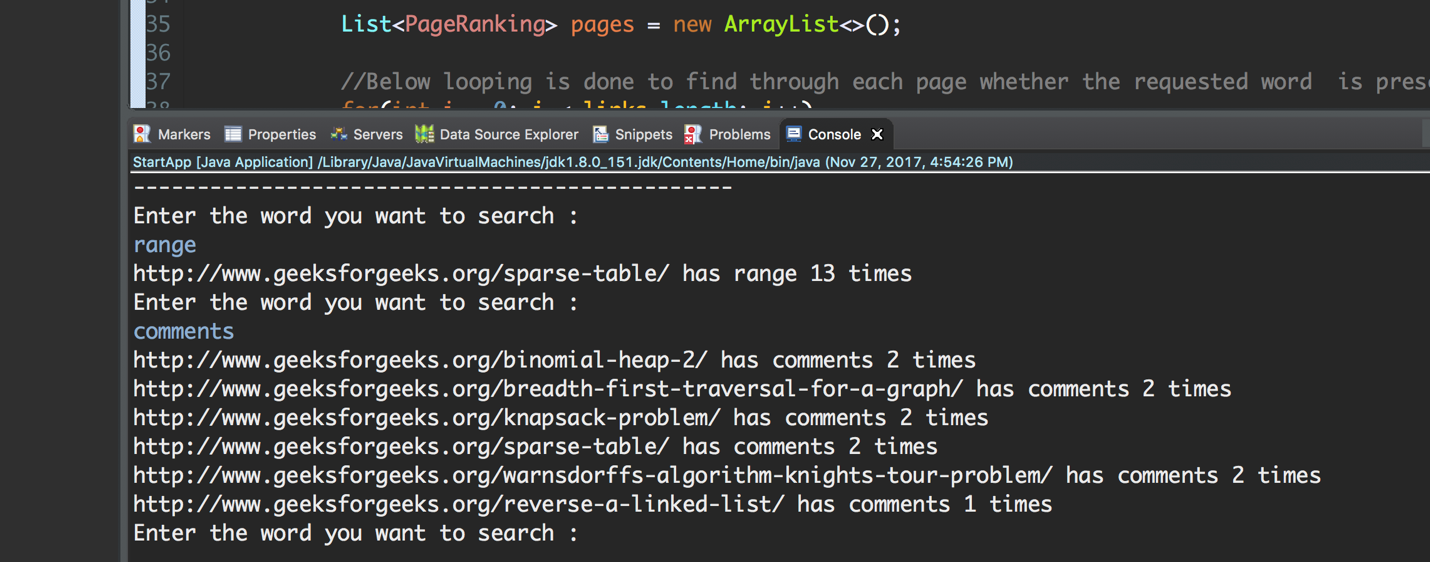
}

Sample Output:

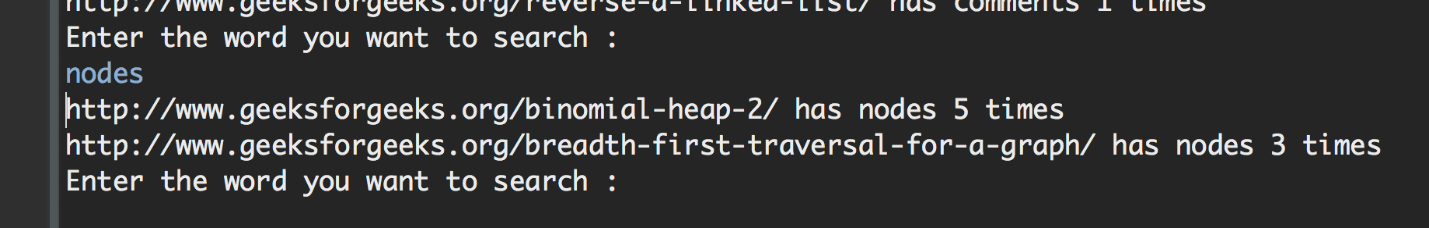
1] Since the word “range” is only available in one of the links of sparse table it only shows that link.



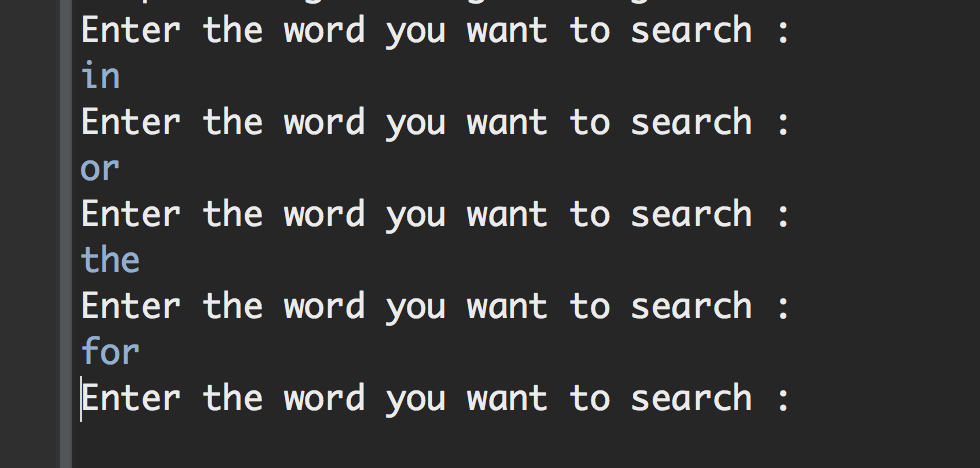
2] Word: Comments



3] Word: nodes



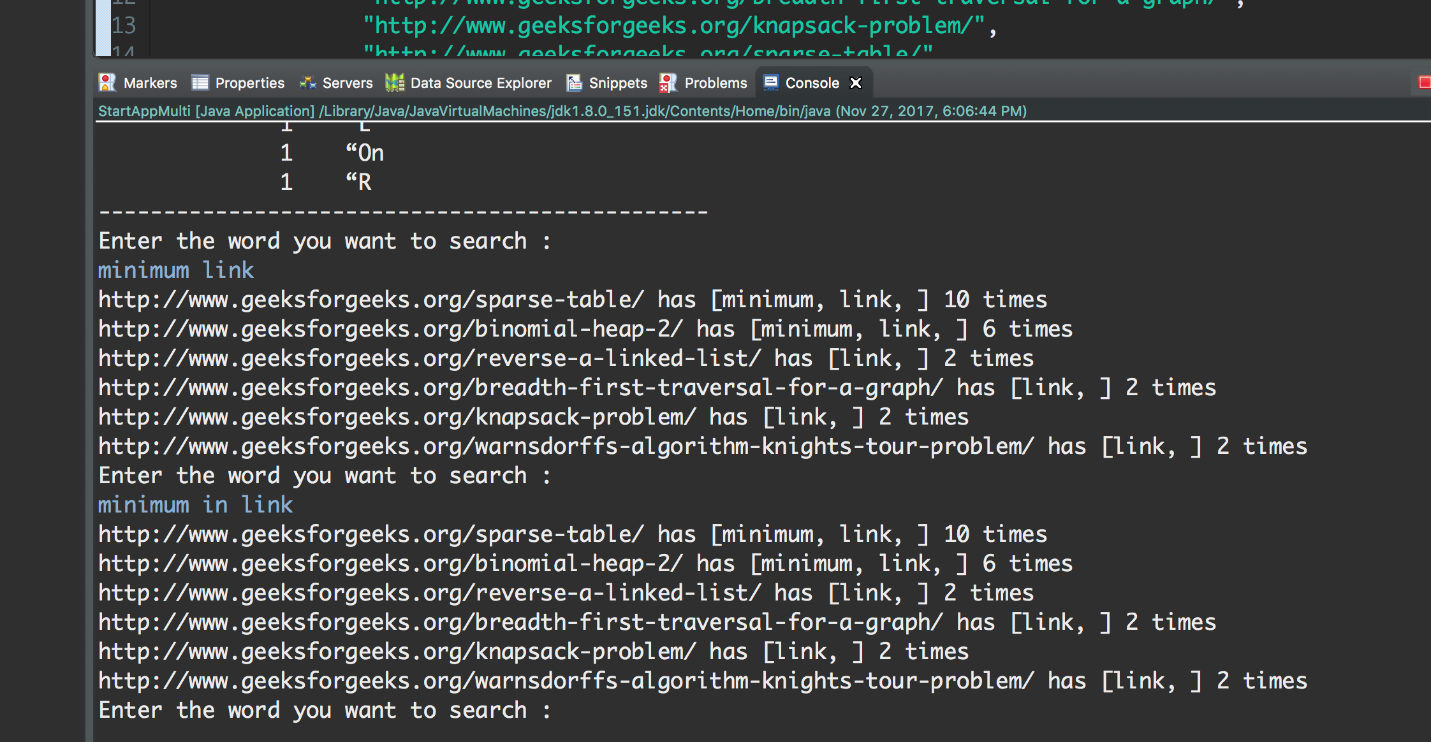
4] Stop Words being eradicated in output results (Stored in stopwords.txt):



FOR MULTIPLE KEYWORDS

For more than one keywords, we search the pages containing the word by trie and sort them first based on number of keywords out of search query and then in buckets by number of their individual occurrences.

Start with: **StartAppMulti.java**

****

When we put “Minimum in Link” … “in” gets disbarred from the search results since it’s a stop word.