Assignment1 Report

Name: Ke Wang NetID: kw427

I implemented this project by hash map on the Java platform.

1. Data Structure

In this assignment, I mainly used 2 Hash Maps (below) to implement the invert index.

HashMap<String, LinkedList<Integer>>

a). The first hash map "term_freq" stores the term and its occurrences in a document: HashMap<String term, LinkedList<Integer> sorted locations of the occurrences>, the key represents the distinct terms in one document and the value is a linked list which stores the locations of term occurrences in ascending order, so the size of the linked list is the term frequency, i.e. the number of times that the term occurs in the document.

I used an ArrayList of this hash map, in which the index represents each document.

b). The second hash map "doc_freq" stores the term and ids of documents that contains the term. HashMap<String term, LinkedList<Integer> sorted documents id>, the key represents all the distinct terms in the corpus, and the value is a linked list which stores the sorted id of the documents which contains the term, so the size of the linked list is the document frequency, i.e., the number of documents that contain the term. This hash map is generated by merging the ArrayList of "term_freq" hash maps.

2. Algorithm

a). The framework of the project:

-Invert.java -Node.java -ReadFile.java -Test.java

b). **ReadFile.java** contains the following methods:

***readStopList*:** Read the stop list and return an array list storing the words in the stop list.

***readEachDocument*:** Read test data in each document from course webpage, create a hash map "term_freq" for each document. I use regex to remove all the non-letter characters.

readFile: read the document and store the words.

c). **Node.java**: Structure to store id of the documents and sum of the tf.idf indexes.

d). **Invert.java**

This is the main class of the project, with the global variables and major functions as follows:

```
ArrayList<String> stoplist;
ArrayList<HashMap<String, LinkedList<Integer>>> list; //get all documents term-location hash map
HashMap<String, LinkedList<Integer>> doc_freq;//get term-doc hash map
ArrayList<String> keyList;//the whole list of terms
```

getAllTermFreq: Call *readEachDocument()* iteratively to get term_freq for each document. The index is doc_id.

getDocFreq: Iterate the hash maps of the term frequency for all documents, merge them and finally get the hash map for all distinct terms and the document frequency.

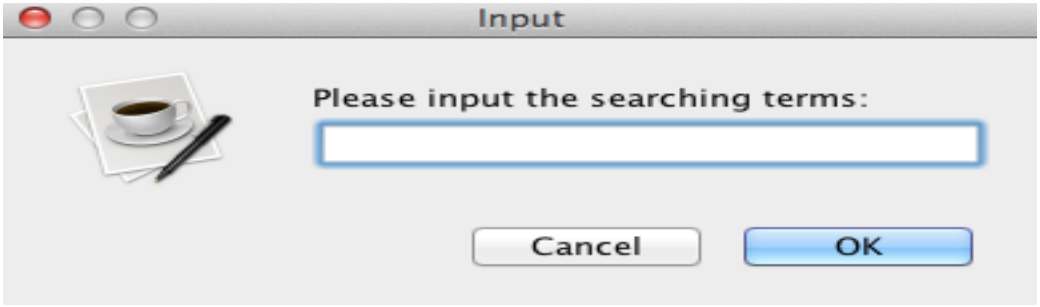
sortKey: Sort terms in alphabetic order in keyList, which can be used for sequential processing.

getPostings: Call getPostingForTerm() iteratively to get postings of necessary information.

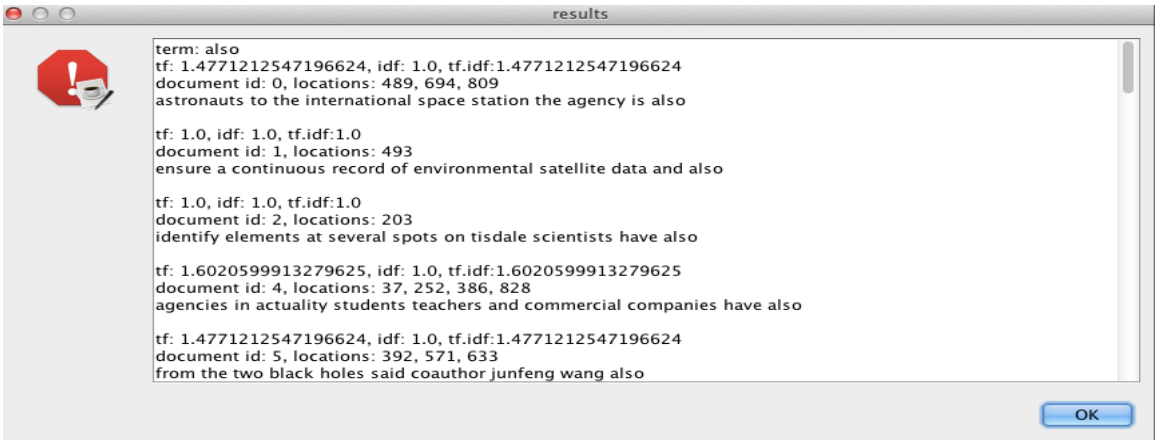
querySingle: Query single term, get the tf,idf,tf.idf, doc_id, locations and ten words in the context.
queryMultiple: Get intersection of terms' doc_list, and sort sums of tf.idf in descending order.
e). **Test.java:** This the test file including some simple GUI.
Using term as key or sequentially access the terms in the "keyList" and then search the doc_freq hash map, the index data structure supports both random look-up and sequential processing.

3. How to run

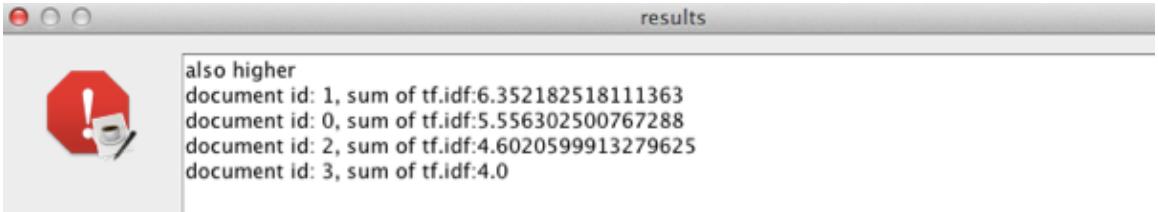
Run the Test.java, then you will get the user input:



Input the terms, e.g "also", result as follows: For each posting, it includes three lines of the term weightings, document ids and locations, and the ten words in the context within the documents.



"also higher": Display the list of the id of the documents that contain all the terms, one document per line together with the sum of the tf.idf scores for the input terms.



If the corpus doesn't contain the word(or intersection of multiple words), it will display an message like "Sorry, we cannot find the terms in the documents".
The program will loop until the term ZZZ is input or nothing is input or we click "cancel" button.