Assignment 2

1. **Describe the design you used for Parts 1, 2, 3. For Part 3 this should include the keys and values you used.**

   All 3 parts have been coded in JAVA 1.7 (as supported by VM environment). I have used 1 jar file (hadoop-core-1.2.1) in my code.

   Part 1 and Part 2:

   4 classes have been coded.
   Index.java: This class implements the Inverted Index. The Inverted Index can store both the document id and the word position. This class reads all the .txt files present in the current directory and indexes all the words.
   The index is implemented using a `HashMap< String, ArrayList<Location> >`. The key is of type String. Key is the word being stored. The value is of type Location. Location is a User defined Class storing the document id and Word position as integer values. After HashMap is created, it is saved in a text file named "Index.txt".

   Location.java: Helper class. Objects of this class are stored as value in the inverted index. It just stores the doc id and word position as integer values.

   InvertedIndex.java: This is the User Interface. This class first reads the data stored in file "Index.txt". Then, prompts the user to proceed with Boolean Query operations. This class contains the main() method.

   Query.java: This class is used to process the Boolean queries. Control flows from InvertedIndex.java to Query.java when search is performed. This class first reads the contents of file Index.txt. This file contains the inverted index.
   Then it creates a HashMap of type (`String, ArrayList<Location> ).` Then search operation is performed on the HashMap created. The result of Boolean search is stored in an `ArrayList<Location>` and printed on screen.

   Part 3:

   4 classes have been coded.
   InvertedIndex.java: This class contains the main() method. This class controls the execution of the whole process (MapReduce). This class contains the classes IndexMap.java and IndexReduce.java inside it (in the form of sub classes).

IndexMap.java: This class implements the Map function by extending the class org.apache.hadoop.mapreduce.Mapper. It reads all the .txt files present in the input directory and indexes all the words present in all the files.
The key-value pair created is of the form (Word, file_name : position). Both key and value are strings. Here,
(word) = key
(file_name : position) = value
The file_name is obtained from FilePath. The position value of each word is calculated using Regular Expression (Matcher class). In Java, implementation of Hadoop API, these key-value pairs are stored as objects of class org.apache.hadoop.io.Text.

IndexReduce.java: This class implements the Reduce function by extending the class org.apache.hadoop.mapreduce.Reducer. It reads the key-value pairs created by Mapper class (Text objects) and populates the data into a List<String>. Then this list is sorted alphabetically according to the keys. Then the list is written in file part-r-00000.

Test.java: This file implements the Boolean search operation on the Inverted Index produced from MapReduce. It first reads the file part-r-00000, then populates each record in `Map<String , ArrayList<String>>.` Then search operation is performed on the values in the map. The result is stored in `ArrayList<String>` and displayed.

2. **For parts 1 and 2, what else would you have done in the inverted index implementation, given more time, energy, resources, etc.**

Don't think anything more can be done in terms of design and implementation for Part 1 and part 2. Though, I must admit that Part 2 that I have implemented has many bugs (more on that in readme doc). So, given more time and resource, I would like to make the query bug free.

For part 1, index is storing both document id and word position (for both part 1 and part 3). This has taken a lot of effort and time. If my index only recorded the doc id, I could have dedicated more time to the Boolean Query part.

Also, I would like to code the assignment in Python.

Also, I have hard coded the stop words. I would like to implement a feature, where the user can select the stopwords.

3. **For parts 1,2 how difficult was it to implement the inverted index? How difficult would it be to implement another task, given this experience? What would be straightforward? What would take more time?**

   Implementing the Inverted index in part 1 was straightforward. The Boolean query implementation in Part 2 and Part 3 was a very difficult and tricky.

   Implementing another similar task should be easier, but in programming you never know!!!

   Setting up the environment (VM, Hadoop, JAR files) would be straightforward.

   I would certainly spend much more time in the Boolean Query part.

4. **For part 3, how many passes of MapReduce did you use?**

   MapReduce in part 3 has 1 MapReduce Pass. Thus, there is only 1 mapper, and 1 reducer.

   Mapper is reading a word and converting it into a (key, value) pair of

   (Word, file_name : position). Both key and value are strings.

   Word = key

   file_name : position = value