Please upload your solutions to the designated Moodle assignment on or before the due date as a single zip file using your group id as the file name. Include some brief instructions on how to run your solution to each problem in a file called `Problem_X.txt`. *All solutions may be submitted in groups of up to 3 students.*

Inverted Index & Query Processing in MapReduce

Problem 1. **12 Points**

(a) Implement a basic form of an *Inverted Index* (based on the pseudo code provided on Slide #64 of Chapter 3 from the lectures slides) in the Java APIs of Apache Hadoop. Consider the following points:

- Use the `WordCount2.java` example as a template for your inverted-index implementation.
- Take over the `-skippatterns` and `-casesensitive` command-line options from `WordCount2.java`.
- Enable the `job.setCombinerClass(...)` option in the main method to enable a local aggregation.
- Consider the new version of `Wikipedia-En-41784-Articles-1lineperdoc.tar.gz` from Moodle as input to your inverted index. You may assume that each line in the `wiki_*.1lineperdoc` files corresponds to exactly one Wikipedia article, where the `<doc id="X"...>` opening tag denotes an article with id `X` and the substring between the `<doc ...>` opening and `</doc>` closing tags contains the entire text content of the article.
- Split the content of each Wikipedia article into reasonable word tokens by using the regular expression `[\w]+` as in a previous exercise. Turn all tokens into lower cases if the `-casesensitive` flag is not given, and also remove stopwords if the `-skippatterns` option is enabled.
- The output of your inverted index should be emitted by the reducers into one or more `part-r-XXXXX` files in your Hadoop output directory. The format of these files should be as follows:

  `term<tab>docid<tab>score`

  where `term` is a word token extracted by your tokenizer, `docid` is the id of the Wikipedia article in which the term occurs, and `score` is the frequency of the term in that article.

5 Points

(b) Implement a simple form of a search engine in the Java APIs of Apache Hadoop by implementing a *Reduce-Side Join* over the `part-r-XXXXX` files you implemented in (a). Also here, consider the following points:

- Use the `WordCount2.java` example to see how command-line parameters can be passed to the `Mappper` and `Reducer` classes, respectively.
- Specifically, pass a user-provided list of keywords (i.e., the "search strings") from the command-line to the `Mapper` class.
- Implement a `map` method which reads one line from each `part-r-XXXXX` file at a time and extracts the three fields `term`, `docid`, `score` from each such line.
- For each line that contains a `term` which occurs in the provided list of keywords (stored in the `Mapper` class), emit the current `score` of that line as a value by using the current `docid` as a key emitted by the `map` method.
- At the `reduce` method, make sure that you then sum up all the `score` values received under a same `docid` as key, and finally emit this `docid` together with its summed up `score` values as one result of your join operation.

5 Points

(c) Repeat steps (a) and (b) of this exercise by using the *Secondary Sorting* optimization (based on the pseudo code provided on Slide #65 of Chapter 3 from the lectures slides) in the Java APIs of Apache Hadoop. How does this optimization affect the usage of a `Combiner` class? **2 Points**

Analytical Queries in MapReduce

Problem 2. **12 Points**

Consider again the TSV files we downloaded from IMDB for the first exercise sheet. This time, implement the first two analytical queries from Problem 3 (a)–(b) of Exercise Sheet #1 in MapReduce:

(a) Select the top 20 movies with the highest ratings. **6 Points**

(b) Select how many movies each director has directed, stop after the top 20 directors with the most movies. **6 Points**

Note that you need to implement a suitable function to parse the various fields from each line of a TSV file into separate strings inside your `map` methods in Java. Splitting each line by `[\t]+` should suffice for our purpose. Also make sure to sort the output of your `reduce` methods in a proper way.

Analytical Queries in Apache Pig

Problem 3. **12 Points**

Also here, consider the TSV files we downloaded from IMDB for the first exercise sheet. This time, implement the first three analytical queries from Problem 3 (a)–(c) of Exercise Sheet #1 in Apache Pig by using the Pig Latin operators presented in the lecture slides:

(a) Select the top 20 movies with the highest ratings. **2 Points**

(b) Select how many movies each director has directed, stop after the top 20 directors with the most movies. **4 Points**

(c) Select the top 20 *pairs* of actors and directors that occur together in a same movie, ordered by the number of movies in which they co-occur. **6 Points**