Aims

This exercise aims to get you to apply the value-to-key conversion design pattern you have learned in Chapter 4 in MapReduce programming.

Background

If you have not finished solving the 7 problems in Labs 3 and 4, please keep working on them, and then move to Lab 5.

Create a project “Lab5” in Eclipse, and create a package “comp9313.lab5” in this project. Put all your codes written in this week’s lab in this package, and keep a copy for yourself after you have finished all problems.

For all problems, using the following code to tokenize a line of document:

```
StringTokenizer itr = new StringTokenizer(value.toString(),
        " *$&#/\t\n\f\"'\\,.:;?![](){}<>~-_");
```

Convert all terms to lower case (by using toLowerCase() function).

Put the input file to HDFS by:

```
$ wget http://www.gutenberg.org/cache/epub/100/pg100.txt
$ $HADOOP_HOME/bin/hdfs dfs -mkdir input
$ $HADOOP_HOME/bin/hdfs dfs -put ~/pg100.txt input
```

Problem 1. Construct a Boolean Inverted Index Using Secondary Sort

The problem is to construct an inverted index for Boolean text retrieval. We first split the pg100.txt into 10 parts using the following command:

```
$ split -d -n 10 pg100.txt pg100_
```

The pg100.txt file will be split into 10 parts, each has the name pg100_[01-09]. You upload all these files into the “input” folder of HDFS.

```
$ hdfs dfs -put pg100_* input
```

Your output is an inverted list, which is used to indicate which file contains the specific word. Here we ignore the letter cases, and convert all terms to lower case.

For example, you have the following three input files:

```
Test_0: A b c
```

```
Test_1: B c d
Test_2: C a d
```

Your inverted list should be like:

```
a: Test_0, Test_2
b: Test_0, Test_1
c: Test_0, Test_1, Test_2
d: Test_1, Test_2
```

Create a new class BooleanInvertedList.java in package “comp9313.lab5”. You must also sort the filename according to the suffix id as well. For example, the above three files `Test_0, Test_1, Test_2` are sorted according to the suffix id 0, 1 and 2. Therefore, you need to use the “value-to-key conversion” approach to make Hadoop do the sorting.

Hints:

1. Refer to the example of Chapter 4. Note that you do not need to generate term frequency in this problem.
2. What should be done in the map() function? (For each term, emit a pair of ([term, fileName], fileName). In the map() function, you can use the following code to obtain the filename of the current input:

   ```
   ((FileSplit) context.getInputSplit()).getPath().getName();
   ```

3. What is the data type of the map output key? How to store the pair of terms? (You need to customize a WritableComparable class in this problem, to wrap two Text/String objects.)
4. How to guarantee that the key-value pairs relevant to the same term are sent to the same reducer? In order to test the correctness of your output, set the number of reducer to 2 (try both overriding hashCode() and implementing a Partitioner class).
5. How to write the grouping comparator? Refer to slide 18 of Chapter 4. (removing this class and run your code again to see what will happen)
6. How about the combiner? Is the combiner useful in this problem?
7. What is the data type of the reducer’s output value? (You can use a String object to store the list of file names)
8. How to configure the main function? (Configure the key-value output data type of the mapper and the reducer! Also configure the grouping comparator!)

You can download the code template at:

https://webcms3.cse.unsw.edu.au/COMP9313/18s1/resources/14997

Problem 2. Construct a Boolean Inverted Index Using Secondary Sort (version 2)

In Problem 1, we use a String object to store the list of filenames. Another way of doing this is to implement a Writable class by extending from ArrayWritable. Do some research on your own to learn how to write a StringArrayWritable, and try to construct the index by using this class.

The output should be the same as that of Problem 1.