Aims

This exercise aims to get you to apply the order inversion and value-to-key conversion design patterns for MapReduce programming you have learned in Chapter 3.2.

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

Create a project "Lab4" in Eclipse and create a package "comp9313.lab4" 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).

Download the pg100.txt file and put it to HDFS by:

```
$ hdfs dfs -mkdir input
$ hdfs dfs -put ~/pg100.txt input
```

Problem 1. Computer Relative Frequency Using Order Inversion

The problem is to compute the relative frequency $f(w_j|w_i)$, i.e.,

$$f(w_j|w_i) = \frac{N(w_i, w_j)}{\sum_{w'} N(w_i, w')}$$

Here, N(., .) indicates the number of times a particular co-occurring term pair is observed in the corpus. We need the count of the term co-occurrence, divided by the marginal (the sum of the counts of the conditioning variable co-occurring with anything else).

In this problem, we consider the nonsymmetric co-occurrence. That is, w_i and w_j co-occur if w_j appears after w_i in the same line, which is defined the same as in Problem 2 of Lab 3.

Create a new class "NSRelativeFreq.java" in the package "comp9313.lab4" and solve this problem. Your output should be in format of $(w_i, w_j, f(w_j|w_i))$, and you need to use DoubleWritable to serialize the value $f(w_j|w_i)$.

Hints:

1. The mapper only needs to be modified a little bit. Just emit one more special key for a pair of terms.
   The problem is, what is the type of the output key of map()? How to guarantee that the order is correct, and thus the special key can be sent to the reducer first (You can still use Text to wrap a pair of terms)?
2. How to write the codes for the reducer to compute the relative frequency of a pair?
3. Can you use the reducer as the combiner in this problem? How to write a combiner to aggregate partial results?
4. How to specify the configuration in the main function? (Be careful if your map output value is different from the reduce output value!)

You can use the results of Problem 2 in Lab 3 to verify the correctness of your output (e.g., check the pairs where the first term is "zounds").

Question: What will happen if you set the number of reducers more than 1?

Problem 2. Computer Relative Frequency Using Order Inversion (version 2)

You are required to customize a WritableComparable class to solve the problem defined in Problem 1, which is the output key type of the map () function. This is more complicated than using Text to wrap the pair of terms. In order to see the effects of the partitioner, set the number of reducers to 2 by adding the following line in the main function:

```
job.setNumReduceTasks(2);
```

Create a new class "NSRelativeFreq2.java" in the package "comp9313.lab4" and solve this problem. Your output should be in format of (w_i, w_j, $f(w_j|w_i)$), and you need to use DoubleWritable to serialize the value $f(w_j|w_i)$.

Hints:

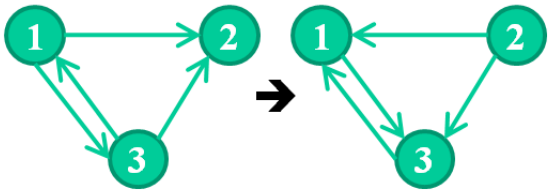
1. Please refer to the StringPair class provided.
2. Remember that you need to either override hashCode() in your WritableComparable class or implement a Partitioner to guarantee the correctness. Try both methods if you have time.
3. How to configure the main function (configure your partitioner class if there is one)?

Your results should be the same as that of Problem 1 (stored in two output files because two reducers are specified).

Problem 3. Reverse Graph Edge Directions

The problem is to reverse graph edge directions of the input directed graph and sort the output adjacency list in ascending node order. You can refer to Slide 61 of Chapter 3.2.

Create a new class “ReverseGraphEdge.java” in the package “comp9313.lab4” and solve this problem.



Input files:

In the input file, each line is in format of: “FromNodeId\tToNodeId$_1$ ToNodeId$_2$ ToNodeId$_3$ … ToNodeId$_n$”. In the above example, the input file is:

```
3\t1 2
1\t2 3
```

Output:

The format of the output file is the same as that of the input file. Moreover, the lines are required to be sorted accendingly according to their FromNodeIds. In each line, the ToNodeIds are also required to be sorted in the accending order.

```
1\t3
2\t1 3
3\t1
```

Hints:

1. Because of the requirement of ascending order of ToNodeIds, you are required to use secondary sort to solve this problem. Please refer to the slides 65-68 of Chapter 3.2 about how to do partitioning, how to use the grouping comparator, and how to define the order for keys.

Solutions of the Problems

I hope that you can finish all problems by yourself, since the hints are already given. All the source codes will be published in the course homepage on Sunday in the same week.