Aims

This exercise aims to get you to apply the design patterns you have learned in Chapter 3 in MapReduce programming.

Background

If you have not finished solving the 3 problems in Lab 3, please keep working on them first 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).

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. Compute a Symmetric Term Co-occurrence Matrix Using the "Pair" Approach

In this problem, the co-occurrence of (w, u) is defined as: both w and u appears in one line of a document. By this definition, (w, u) and (u, w) are treated as the same. Use the "pair" approach again to solve this problem.

Create a new class CoTermSynPair.java in package "comp9313.lab3".

Hints:

1. How to modify CoTermNSPair.java slightly to solve this problem?
2. Do you need to change the reducer and combiner?

The generated pairs should be fewer than the nonsymmetric version. The count of (w, u) and (u, w) are merged in the symmetric problem. For example, you will see "a mad 22" and "mad a 21" in the result of Problem 2 in Lab3, and in this problem you will only see "a mad 43" in the output.

If your code is correct, you should output 978,615 pairs.

Problem 2. Compute a Symmetric Term Co-occurrence Matrix Using the "Stripe" Approach

The problem is the same as defined in Problem 1. The task is to use the "stripe" approach to solve it.

Input is in format of (line number, line). Output is in format of ((w, u), co-occurrence count).

Create a new class CoTermSynStripe.java in package "comp9313.lab3".

Hints:

1. You only need to modify the map() function in CoTermNSStripe.java. When you get a pair of terms w and u, you need to consider the alphabetical order of w and u. If w < u, you write the information to the MapWritable object for w; otherwise, you need to emit a key-value pair for u, i.e., (u, (w, 1)).

   For example, give a line (b, c, d, a), for term b, we create a MapWritable object bMap. Next we process each term appearing after b in the same line.
   For (b, c), we put (c, 1) to bMap.
   For (b, d), we put (d, 1) to bMap.
   For (b, a), b > a (alphabetical order), thus, we create a MapWritable object aMap, and put (b, 1) to aMap, and emit (a, aMap) to the reducer.
   All terms after b are processed, and we emit (b, bMap) to the reducer.
2. Do you need to change the combiner and reducer?

If your codes are correct, the results obtained should be the same as obtained in Problem 1.

Problem 3. 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. Refer to slides 49-51 of Chapter 3.
2. 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)?
3. How to write the codes for the reducer to compute the relative frequency of a pair?
4. Can you use the reducer as the combiner in this problem? How to write a combiner to aggregate partial results?
5. 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 4. Computer Relative Frequency Using Order Inversion (version 2)

You are required to customize a WritableComparable class to solve the problem defined in Problem 3, 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 reducer 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. Refer to slides 41 and 56 of Chapter 3. Try both methods.
3. How to configure the main function (configure your partitioner class)?

Your results should be the same as that of Problem 3 (stored in two output files because two reducers are specified).

Solutions of the Four Problems

I hope that you are able to finish all problems by yourself, since the hints are already given.

All the source codes will be published in the course homepage on Friday next week.