Aims

This exercise aims to get you to apply the design patterns you have learned in Chapter 2.2 in MapReduce programming.

If you have not finished solving the problems in Lab 2, please keep working on them first and then move to Lab 3.

Create a folder "Lab3". Put all your codes written in this week's lab in this folder and keep a copy for yourself after you have finished all problems. Download the input file "pg100.txt" from the following link:

https://webcms3.cse.unsw.edu.au/COMP9313/21T3/resources/66890

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

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

Problem 1. Compute a Nonsymmetric Term Co-occurrence Matrix Using the "Pair" Approach

The problem is to compute the number of co-occurrence for each pair of terms (w, u) in the document. In this problem, the co-occurrence of (w, u) is defined as: u appears after w in a line of document. This means that, the co-occurrence counts of (w, u) and (u, w) are **different!** The task is to use the "pair" approach to solve this problem.

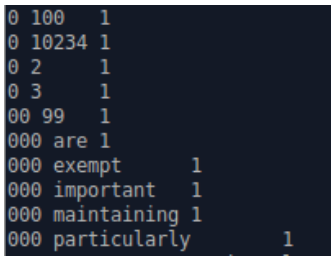
Input is in format of (line number, line). Output is in format of ((w, u), co-occurrence count).

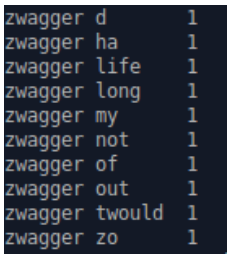
Create a new class CoTermNSPair.java in package "comp9313.lab3".

Hints:

1. Refer to the pseudo-code in slide 27 of Chapter 2.2. Note that the condition u is in "NEIGHBORS(w)" means that u appears after w in the same line in this problem.
   For example, give a line "a, b, c, d", for term "a", you should generate ((a, b), 1), ((a, c), 1), and ((a, d), 1).
2. What is the data type of the map output key? How to store the pair of terms? (A simple method is to concatenate the two terms as a string)
3. How to write the reducer?
4. How about the combiner?
5. How to configure the job in main function?

The head and the tail of the result are like:





If your code is correct, you should output 1,161,210 pairs.

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

The problem is the same as Problem 2. 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 CoTermNSStripe.java in package “comp9313.lab3”.

Hints:

1. Refer to the pseudo-code in slide 30 of Chapter 2.2.
2. You need to use MapWritable as the map output value. MapWritable is a wrapper for java Map in Hadoop.
   When processing a line in the map() function, in the first for loop, you need to create a MapWritable object, and for all terms appearing after the current term w, you need to use the MapWritable object to store the information.

   More usage about MapWritable please refer to:
   https://hadoop.apache.org/docs/r3.3.1/api/org/apache/hadoop/io/MapWritable.html
   Some MapWritable examples can be found at:
   http://www.programcreek.com/java-api-examples/index.php?api=org.apache.hadoop.io.MapWritable

3. In the reducer, you will receive a list of MapWritable objects for the same term w. You need to aggregate them and generate a final “stripe”, and then output the key-value pairs in format of ((w, u), co-occurrence count). You can use the following code to iterate the key-value pairs in a MapWritable object:

```
for (MapWritable val : values) {
        Set<Entry<Writable, Writable> > sets = val.entrySet();
```

```
        for(Entry<Writable, Writable> entry: sets){
                //entry.getKey() to get the key
                //entry.getValue() to get the value
                ……
        }
}
```

4. The mapper output is <Text, MapWritable>, while the reducer output is <Text, IntWritable>, and the output value types are different. Therefore, you need to specify this in the main function, by adding the following line:

   ```
   job.setMapOutputValueClass(MapWritable.class)
   ```

5. How to write the combiner? Can you use the reducer as the combiner in this problem? Remember, the input of the combiner is the output of the mapper, and the output of the combiner is the input of the reducer. If you need to write a combiner class, define it as:

   ```
   public static class YOURCOMBINERCLASS extends Reducer<Text,
   MapWritable, Text, MapWritable>
   ```

   Then, in this class, you need to override the function reduce(), which is similar as in the reducer class.

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

Problem 3. 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 appear 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, 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 4. Compute a Symmetric Term Co-occurrence Matrix Using the "Stripe" Approach

The problem is the same as defined in Problem 3. 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 3.

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 Friday next week.