XI'AN JIAOTONG-LIVERPOOL UNIVERSITY

西 交 利 物 浦 大 学

YEAR 4

COURSE WORK SUBMISSION

Name	Osmond	Wilbert
ID Number	*1926308*	
Programme	*Exchange (non-UoL)*	
Module Title	*Big Data Analytics*	
Module Code	*CSE313*	
Assignment Title	*Lab 2*	
Submission Deadline	*(Mitigating circumstances)*	
Lecturer Responsible	*Gangmin Li*	

I certify that:

- I have read and understood the University's definitions of COLLUSION and PLAGIARISM (available in the Student Handbook of Xi'an Jiaotong-Liverpool University).

With reference to these definitions, I certify that:

- I have not colluded with any other student in the preparation and production of this work;
- this document has been written solely by me and in my own words except where I have clearly indicated and acknowledged that I have quoted or used figures from published or unpublished sources (including the web);
- where appropriate, I have provided an honest statement of the contributions made to my work by other people including technical and other support staff.

I understand that unauthorised collusion and the incorporation of material from other works without acknowledgement (plagiarism) are serious disciplinary offences.

Signature  Date ……4th January 2020……

For Academic Office use:	Date Received	Days Late	Penalty

1 My result files

In both tasks, I have decided to prioritize the content of the words, instead of how they were written. Consequently, the words will not be case-sensitive in the program, e.g. 'This violence" is considered to be equal to "this violence". The reason for this is that they ultimately represent the same bigram of words "this" and "violence".

1.1 Task 1

The format of task 1 output is as such: the left column is the bigram which contains two words and the right column contains the frequency of the bigram's appearance in the dataset. The top 10 most common bigrams, ordered descendingly, are listed in the following table:

Bigram	Frequency of appearance
i am	1855
of the	1760
i ll	1745
my lord	1666
in the	1662
i have	1620
i will	1566
to the	1452
it is	1080
to be	975

Figure 1: The top 10 most common bigrams and their frequency of appearance.

1.2 Task 2

The format of task 2 output is as such: the left column represents the line number and the right column is the line content. An important thing to note is that only instances of the verb-stem-format "torture" are included. Meaning, any verb variations forms, for instance "tortured", is not accepted. The lines that contains the word "torture" in the dataset, are listed in the following table:

Line number	Line content
647	Do in consent shake hands to torture me,
2428	Is't not enough to torture me alone,
3664	With vilest torture let my life be ended.
9406	All length is torture. Since the torch is out,
22663	By a sharp torture.
22729	Drawn on with torture.
23407	Which is our honour, bitter torture shall
23414	IACHIMO. Thou'lt torture me to leave unspoken that
23415	Which to be spoke would torture thee.
42200	That so her torture may be shortened.

43465	You go about to torture me in vain.
44577	And torture him with grievous ling'ring death.
44736	From thee to die were torture more than death.
44769	O, torture me no more! I will confess.
55229	Turning dispiteous torture out of door!
55808	Let hell want pains enough to torture me!
65580	That same Berowne I'll torture ere I go.
67932	Than on the torture of the mind to lie
74098	SHYLOCK. I am very glad of it; I'll plague him, I'll torture him; I
77114	then torture my wife, pluck the borrowed veil of modesty
83140	Refuse me, hate me, torture me to death!
86189	OTHELLO. If thou dost slander her and torture me,
87934	The time, the place, the torture. O, enforce it!
94565	To torture thee the more, being what thou art.
97732	This torture should be roar'd in dismal hell.

Figure 2: All lines that include the word “torture” in the dataset, with its line number.

2 My code with comments

I hereby state that there is a partial contribution to my code by a teacher assistant in CSE313 Big Data Analytics, Linlin Du.

2.1 Task 1

```
package edu.xjtlu.cse313.assignment;

import java.io.IOException;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
```

```
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WordCount extends Configured implements Tool {
    public static void main(String[] args) throws Exception {
        System.out.printIn(Arrays.toString(args));
        Int res = ToolRunner.run(new Configuration(), new WordCount(), args);
        System.exit(res);
    }

    @Override
    public int run(String[] args) throws Exception {
        System.out.printIn(Arrays.toString(args));
        // create the job
        Job job = new Job(getConf(), “WordCount”);
        job.setJarByClass(WordCount.class);

        // mapper will produce Text - Int pairs (bigram - count)
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        job.setMapperClass(Mapper.class);
        job.setReducerClass(Reduce.class);

        // both input and output are plain text files
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // start the job
        job.waitForCompletion(true);

        return 0;
    }

public static class Mapper extends Mapper<LongWritable, Text, Text, IntWritable>{
    private final IntWritable ONE = new IntWritable(1);
        private final Text bigram = new Text();

        /**
         * The map() method emits every bigram in each line
         * @param key
```

```
         * @param value
         * @param context
         * @throws IOException
         * @throws InterruptedException
         */
        @Override
        public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
         // reading line by line, all pairs of word in that line
         // are added to the key,value set with frequency 1.
         // The last word of the previous line is also added!
         String[] words = value.toString().toLowerCase().split("\\s+");
         String previous = null;
         for (String word: words) {
           if (word.length() > 0) {
            if (previous != null) {
              bigram.set(previous + " " + word);
              context.write(bigram, ONE);
            }

            previous = word;
          }
         }
        }

    }

public static class Reduce extends Reduce<Text, IntWritable, Text, IntWritable>{
    Map<String,Integer> map=new HashMap<String, Integer>();
    protected void reduce(Text key, Iterable<IntWritable> iter, Context context)
                        throws IOException, InterruptedException {
        // get the global frequency of each pair of words
        int count=0;
        for (IntWritable wordCount : iter) {
            count+=wordCount.get();
        }
        String name = key.toString();
        map.put(name, count);
    }

    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException
{
        List<Map.Entry<String, Integer>> list=new
    LinkedList<Map.Entry<String,Integer>>(map.entrySet());

        Collections.sort(list, new Comparator<Map.Entry<String,Integer>> {
            @Override
```

```
            public int compare(Entry<String, Integer> arg0, Entry<String, Integer>
        arg1) {
             return (int) (arg1.getValue() – arg0.getValue());}
            });

            for(int i=0, i<10; i++){
             context.write(new Text(list.get(i).getKey()),new
        IntWritable(list.get(i).getValue()));
            }
        }
    }

}
```

2.2 Task 2

```
package edu.xjtlu.cse313.assignment;

import java.io.IOException;
import java.util.Arrays;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class Torture extends Configured implements Tool {
    @Override
    public int run(String[] args) throws Exception {
        // create the job
        Job job = new Job(getConf(), “Torture”);
        job.setJarByClass(Torture.class);

        job.setMapperClass(Map.class);
        job.setOutputKeyClass(IntWritable.class);
        job.setOutputValueClass(Text.class);
```

```
        // Map-only job, since the reducer has nothing to do
        job.setNumReduceTasks(0);

        // input and output are both plain text files
        job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        // start the job
        return job.waitForCompletion(true) ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(), new Torture(), args);
        System.exit(res);
    }

    public static class Map extends Mapper<LongWritable, Text, IntWritable, Text> {
        /**
        * The map() method emits lines that contain the word “torture”
        * @param key
        * @param value
        * @param context
        * @throws IOException
        * @throws InterruptedException
        */

        // save the result as <LINE_NO, LINE_TEXT>
        private static int line_number = 0;
        private String line_content;

        @Override
        public void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {
            // update every time a line is read
            line_number++;
            line_content = value.toString().trim();

            // if the line contains 'torture', add it as a key
            if (line_content.toLowerCase().matches(".*\\storture[\\.\\s].*"))
                context.write(new IntWritable(line_number), new Text(line_content));
        }
    }
}
```

3 Hadoop command list and order

First, I changed the working directory to the Desktop, which contains two files: *pg100.txt* and my program, *executer.jar*.

```
cd ~/Desktop
```

I subsequently added the file pg100.txt to Hadoop's HDFS

```
hadoop fs -put pg100.txt
```

Now, the working environment is ready to execute the jobs.

3.1 Task 1

Run the program specifying the class to find bigrams.

```
hadoop jar executer.jar edu.xjtlu.cse313.assignment.WordCount pg100.txt bigrams
```

Obtain the output file generated in the output folder that is specified in the previous command (i.e. *bigrams*).

```
hadoop fs -ls bigrams
```

This shows a folder consisting a file labelled *part-r-00000*, which comprises all bigrams in the dataset along with their frequency of appearance. To get the top 10, the file initially needs to be ordered according to the third key. The third key is the frequency of appearance of the bigram, since the format of each line is "*word1 word2 frequency"*. Subsequently, its 10-items head is printed.

```
hadoop fs -cat bigrams/part-r-00000 | sort -nrk 3 | head -10
```

The output is then printed to the console.

3.2 Task 2

Run the program specifying the class to find lines containing the word 'torture'.

```
hadoop jar executer.jar edu.xjtlu.cse313.assignment.torture pg100.txt lines
```

Obtain the output file generated in the output folder that is specified in the previous command (i.e., *lines*).

```
hadoop fs -ls lines
```

This shows a folder containing a file named *part-m-00000*, which comprises all lines in the dataset which has the word "torture" in it. Every key-value pair in this file incorporates a line number and the line content. All that is required to print them is only displaying the file contents.

```
hadoop fs -cat lines/part-m-00000
```

The output is then printed to the console.