Hanjiao Qiu
CSCIE63 Big Data Analytics
Assignment 03

Assignment 03 Solution

Homework manifest

Problem	Source file	Output directory
	Original file: WordCount.java Inverter.java	output output7
1	WordCount2.java	output2
2	WordCount3.java	ouput3
3	WordCount4.java	output4
4	WordCount5.java WordCount6.java	output5, output6 output9, output 10
5	Inverter2.java	output8

Only source files (exclude original files) and very small sample output files (top 50 lines) will be uploaded. The data from each output file will be displayed after each problem.

Problem 1) Modify attached class WordCount.java so that its result excludes the following stop words:

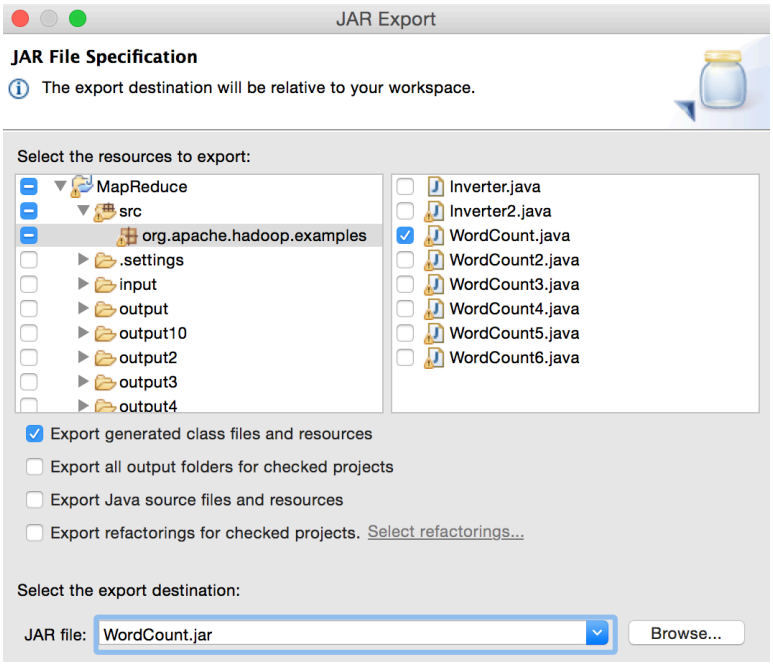
I		
a	in	
about	is	who
an	it	will
are	of	with
as	on	the
at	or	www
be	that	
by	the	
com	this	
for	to	
from	was	
how	what	
	when	
	where	

as well as special characters (dashes, parentases, etc). Stop word lists could be much lomger than this. You do not have to be extremely thoroughful. You would like to get a more or less clean list of ordinary words with the numbers of their occurances. Do not fret. Be reasonable. Perform analysis on te text of James Joyce's Ulysis.

Solution:

1. Compile the sample file WordCount.java to a jar file through Eclipse. Download the Apache Hadoop tarball CDH5.5.1and add external JAR files to Eclipse compile environment following Marina's notes.

2. Transfer the JAR file to the VM through shared folder. Login into the Ubuntu VMware machine through ssh. Run the JAR file on VM.




```
hqiu@bos-mp9cx>> ssh cloudera@192.168.80.141
cloudera@192.168.80.141's password:
Last login: Fri Feb 19 06:41:17 2016 from 192.168.80.1
[cloudera@localhost ~]$ ls
core.10053  Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
[cloudera@localhost ~]$ cd Documents/
[cloudera@localhost Documents]$ cp -r /mnt/hgfs/VM_shared/HW03/ .
[cloudera@localhost Documents]$ cd HW03/
[cloudera@localhost HW03]$ ls
input          Inverter.jar    WordCount3.jar  WordCount5.jar
Inverter2.jar  WordCount2.jar  WordCount4.jar  WordCount.jar
```

Set up the environment variable in .bash_profile in advance to run Map Reduce jobs.

```
[cloudera@localhost HW03]$ vi ~/.bash_profile
[cloudera@localhost HW03]$ source ~/.bash_profile
```

HW03_MapReduce — cloudera@localhost:~/Documents/HW03 -

cloudera@localhost:~/Documents/HW03

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

JAVA_HOME=/usr/java/jdk1.8.0_60
export JAVA_HOME

HADOOP_CLASSPATH=${JAVA_HOME}/lib/tools.jar
export HADOOP_CLASSPATH

PATH=$PATH:$HOME/bin:${JAVA_HOME}/bin
export PATH

export HADOOP_MAPRED_HOME=/usr/lib/hadoop-mapreduce
~
```

```
[cloudera@localhost HW03]$ hadoop fs -ls ulysses
Found 1 items
-rw-r--r--   1 cloudera supergroup    1573079 2016-02-11 12:21 ulysses/4300.txt
```

The input file is already put into the HDFS. Run the jar file using the following command:

```
[cloudera@localhost HW03]$ hadoop jar WordCount.jar org.apache.hadoop.examples.WordCount ulysses output
```

```
[cloudera@localhost HW03]$ hadoop jar WordCount.jar org.apache.hadoop.examples.WordCount ulysses output
16/02/19 07:00:51 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/02/19 07:00:52 INFO input.FileInputFormat: Total input paths to process : 1
16/02/19 07:00:52 INFO mapreduce.JobSubmitter: number of splits:1
16/02/19 07:00:53 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1455215283455_0007
16/02/19 07:00:53 INFO impl.YarnClientImpl: Submitted application application_1455215283455_0007
16/02/19 07:00:53 INFO mapreduce.Job: The url to track the job: http://localhost:8088/proxy/application_1455215283455_
0007/
16/02/19 07:00:53 INFO mapreduce.Job: Running job: job_1455215283455_0007
16/02/19 07:01:00 INFO mapreduce.Job: Job job_1455215283455_0007 running in uber mode : false
16/02/19 07:01:00 INFO mapreduce.Job:  map 0% reduce 0%
16/02/19 07:01:08 INFO mapreduce.Job:  map 100% reduce 0%
16/02/19 07:01:15 INFO mapreduce.Job:  map 100% reduce 100%
```

3. Check the output file. We can see that the parsed words in the first columns have digits and special characters. Next step is to remove the special characters, digits and stop words.

```
[cloudera@localhost HW03]$ hadoop fs -ls output
Found 2 items
-rw-r--r--   1 cloudera supergroup          0 2016-02-19 07:01 output/_SUCCESS
-rw-r--r--   1 cloudera supergroup     527547 2016-02-19 07:01 output/part-r-00000
[cloudera@localhost HW03]$ hadoop fs -cat output/part-r-00000 | head -20
"Come   1
"Defects,"      1
"I      1
"Information    1
"J"     1
"Plain  2
"Project        5
"Right  1
"Viator"        1
#4300]  1
$5,000) 1
%       2
&c,     2
&c.     1
'46.    1
'92     1
'AS-IS' 1
'Slife, 1
'TWAS   1
'Tis    8
```

```
[cloudera@localhost HW03]$ hadoop fs -cat output/part-r-00000 | tail -20
zebra   1
zenith  3
zephyrs,        1
zero    1
zest.   1
zigzag  2
zigzagging      1
zigzags,        1
zivio,  1
zmellz  1
zodiac  1
zodiac. 1
zodiacal        2
zoe)_   1
zones:  1
zoo.    1
zoological      1
zouave's        1
zrads,  2
zrads.  1
```

4. This is the class to solve this problem. In class WordCount2 line 58, I use a function processWord() to remove the special characters, digits in the words, and transform them into lower cases. Moreover, it will mark out the stop words and exclude them from the final output.

```
37 public class WordCount2 {
38
39     static HashSet<String> stopwords = new HashSet<String>(Arrays.asList("i", "a", "an", "and", "or", "of", "to",
40             "about", "above", "after", "all", "are", "be", "but", "he", "she", "by", "can't", "for", "do", "i",
41             "has", "don't", "her", "is", "in", "our", "his", "with", "that", "you", "it", "was", "on", "him",
42             "as", "at", "com", "from", "how", "on", "the", "this", "what", "when", "where", "who", "will", "www",
43             "not", "only", "other", "off", "me", "your", "my", "we", "so", "no", "up", "until", "hasn't", "which"));
44
45     public static class TokenizerMapper
46     extends Mapper<Object, Text, Text, IntWritable>{
47
48         private final static IntWritable one = new IntWritable(1);
49         private Text word = new Text();
50
51         public void map(Object key, Text value, Context context
52                 ) throws IOException, InterruptedException {
53             StringTokenizer itr = new StringTokenizer(value.toString());
54             while (itr.hasMoreTokens()) {
55                 word.set(itr.nextToken());
56
57                 // Exclude the stop words and special characters
58                 word = processWord(word);
59
60                 // Don't write "stopwords" and "unknown" into results.
61                 if (word.toString().equals("stopwords") || word.toString().equals("unknown")) {
62                     continue;
63                 }
64
65                 context.write(word, one);
66             }
67         }
68     }
```

```
public static Text processWord(Text word) {
    Text newWord;

    String str = word.toString().toLowerCase();

    String delimiters = "[.,;:$%&'+/\\*\\#\\-\\?!\\\"\\[\\]\\(\\)\\{\\}\\d_]+";
    String[] args = str.split(delimiters);

    String newStr = null;
    if (args.length > 0) {
        if (! args[0].isEmpty()) {
            // Get the first valid word
            newStr = args[0];
        } else {
            // If a word starts with a special character, the first word is empty
            // after parsing. We should choose the second word.
            newStr = args[1];
        }
        // System.out.println(str + "\t" + newStr);

        // Exclude the stop words.
        if (! stopwords.contains(newStr)) {
            newWord = new Text(newStr);
        } else {
            newWord = new Text("stopwords");
        }
    } else {
        newWord = new Text("unknown");
    }

    return newWord;
}
```

The delimiters contains most of the special characters and digits. It will extract the real word from the string. Be careful that if a word starts with a special character, the first word after split is an empty word and we should choose the second one. After parsing, if the word is null, it will be marked as “unknown”. If the word is in the stop words list, it will be marked as “stopwords”. In the Mapper function, both the “unknown” and “stopwords” will be skipped and won’t be written into the final output. The stop words are chose based on the references online and adjusted based on the test results. The stop words with lower frequency might not be in the list.

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length < 2) {
        System.err.println("Usage: wordcount2 <in> [<in>...] <out>");
        System.exit(2);
    }
    Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount2.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    for (int i = 0; i < otherArgs.length - 1; ++i) {
        FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
    }
    FileOutputFormat.setOutputPath(job,
            new Path(otherArgs[otherArgs.length - 1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

```
[cloudera@localhost HW03]$ hadoop jar WordCount2.jar org.apache.hadoop.examples.WordCount2 ulysses output2
16/02/19 07:05:34 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/02/19 07:05:35 INFO input.FileInputFormat: Total input paths to process : 1
16/02/19 07:05:35 INFO mapreduce.JobSubmitter: number of splits:1
16/02/19 07:05:35 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1455215283455_0008
16/02/19 07:05:35 INFO impl.YarnClientImpl: Submitted application application_1455215283455_0008
16/02/19 07:05:35 INFO mapreduce.Job: The url to track the job: http://localhost:8088/proxy/application_1455215283455_
0008/
16/02/19 07:05:35 INFO mapreduce.Job: Running job: job_1455215283455_0008
16/02/19 07:05:43 INFO mapreduce.Job: Job job_1455215283455_0008 running in uber mode : false
16/02/19 07:05:43 INFO mapreduce.Job:  map 0% reduce 0%
16/02/19 07:05:51 INFO mapreduce.Job:  map 100% reduce 0%
16/02/19 07:05:58 INFO mapreduce.Job:  map 100% reduce 100%
16/02/19 07:05:58 INFO mapreduce.Job: Job job_1455215283455_0008 completed successfully
```

```
[cloudera@localhost HW03]$ hadoop fs -ls output2
Found 2 items
-rw-r--r--   1 cloudera supergroup          0 2016-02-19 07:05 output2/_SUCCESS
-rw-r--r--   1 cloudera supergroup     308488 2016-02-19 07:05 output2/part-r-00000
[cloudera@localhost HW03]$ hadoop fs -cat output2/part-r-00000 | tail -20
zigzag  2
zigzagging      1
zigzags 1
zinfandel       9
zingari 1
zion    5
zip     1
zivio   1
zmellz  1
zodiac  2
zodiacal        2
zoe     106
zones   1
zoo     2
zoological      1
zouave  2
zrads   4
zulu    1
zulus   1
zut     1
```

```
[cloudera@localhost HW03]$ hadoop fs -cat output2/part-r-00000 | head -20
aaron   2
aback   1
abaft   1
abandon 1
abandoned       7
abandoning      1
abandonment     1
abasement       2
abatement       1
abattoir        1
abba    1
abbas   2
abbess  1
abbey   13
abbot   3
abbots  1
abbreviation    1
abdomen 2
abdominal       2
abe     1
```

5. Since we haven't change the sequence of (key, value) pair, no need to modify the main function. Export the Jar file. Run the Jar file on VM and check the results. No special characters and stop words are presented. Copy the result from HDFS to local directory. The file in output2 will be the input of later problems.

```
[cloudera@localhost HW03]$ hadoop fs -copyToLocal output2 /mnt/hgfs/VM_shared/HW03/
```

Problem 2) Write another MapReduce program which would read the "word count" output of the previous job and order the results by the declaining number of occurances.

Solution:

1. The main idea is first changing the sequence of (key, value) pair. Make the occurrences as the key and the word as the value. The key will be automatically ordered in increasing order. Then override the comparator to make them sort the key in decreasing order.

2. Since the WritableComparator requires the LongWritable type, I will use LongWritable type instead of the IntWritable here. In the mapper function, the StringTokenizer() will parse the input file by line and space/tab. So everytime I read in two string and make them as a (occurrences, word) pair. Change the sequence of output data type in Mapper and input/output data type of Reducer.

```
public class WordCount3 {

    public static class TokenizerMapper
    extends Mapper<Object, Text, LongWritable, Text>{

        private Text word = new Text();
        private Text count = new Text();

        public void map(Object key, Text value, Context context
                ) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                count.set(itr.nextToken());
                int intCount = Integer.parseInt(count.toString());
                context.write(new LongWritable(intCount), word);
            }
        }
    }
```

The same to the IntSumReducer class. The intermediate values are an occurrence number with a corresponding list of words. So for each word in the word list, we have to output it with the occurrence once.

```
public static class IntSumReducer
extends Reducer<LongWritable, Text, LongWritable, Text> {
    private Text result = new Text();

    public void reduce(LongWritable key, Iterable<Text> values,
            Context context
            ) throws IOException, InterruptedException {
        for (Text val : values) {
            result = val;
            context.write(key, result);
        }
    }
}
```

Define a decreasing order in key comparator.

```
// Custom descending sort comparator
public static class keyComparator extends WritableComparator {
    public keyComparator() {
        super(LongWritable.class, true);
    }

    @Override
    public int compare(WritableComparable wc1, WritableComparable wc2) {
        LongWritable key1 = (LongWritable) wc1;
        LongWritable key2 = (LongWritable) wc2;
        return -1 * key1.compareTo(key2);
    }
}
```

Be carefully that, we have to update the output key class and output value class to match the definition above.

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length < 2) {
        System.err.println("Usage: wordcount3 <in> [<in>...] <out>");
        System.exit(2);
    }
    Job job = new Job(conf, "word count");
    job.setJarByClass(WordCount3.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(LongWritable.class);
    job.setOutputValueClass(Text.class);
    job.setSortComparatorClass(keyComparator.class);

    for (int i = 0; i < otherArgs.length - 1; ++i) {
        FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
    }
    FileOutputFormat.setOutputPath(job,
            new Path(otherArgs[otherArgs.length - 1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

```
[cloudera@localhost HW03]$ hadoop jar WordCount3.jar org.apache.hadoop.examples.WordCount3 output2 output3
16/02/19 07:11:52 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/02/19 07:11:53 INFO input.FileInputFormat: Total input paths to process : 1
16/02/19 07:11:53 INFO mapreduce.JobSubmitter: number of splits:1
16/02/19 07:11:53 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1455215283455_0009
16/02/19 07:11:54 INFO impl.YarnClientImpl: Submitted application application_1455215283455_0009
16/02/19 07:11:54 INFO mapreduce.Job: The url to track the job: http://localhost:8088/proxy/application_1455215283455_
0009/
16/02/19 07:11:54 INFO mapreduce.Job: Running job: job_1455215283455_0009
16/02/19 07:12:01 INFO mapreduce.Job: Job job_1455215283455_0009 running in uber mode : false
16/02/19 07:12:01 INFO mapreduce.Job:  map 0% reduce 0%
16/02/19 07:12:07 INFO mapreduce.Job:  map 100% reduce 0%
16/02/19 07:12:14 INFO mapreduce.Job:  map 100% reduce 100%
16/02/19 07:12:14 INFO mapreduce.Job: Job job_1455215283455_0009 completed successfully
```

We can see that the stop words defined in the list are not in the results below. Of course we can exclude more stop words as an optimization.

```
[cloudera@localhost HW03]$ hadoop fs -ls output3
Found 2 items
-rw-r--r--   1 cloudera supergroup          0 2016-02-19 07:12 output3/_SUCCESS
-rw-r--r--   1 cloudera supergroup     308488 2016-02-19 07:12 output3/part-r-00000
[cloudera@localhost HW03]$ hadoop fs -cat output3/part-r-00000 | head -20
1208    said
1053    they
998     bloom
899     out
814     had
793     there
731     like
720     their
719     mr
706     one
699     have
672     them
586     o
579     then
571     stephen
564     if
510     were
491     old
473     says
452     down
```

Problem 3) Create a program that will find out how many words appear only once, how many twice, three times, four times and so on in James Joyce's Ulysis

Solution:

1. We already know the occurrence and the corresponding word list in problem 2. In this problem, we only need to count the number of words in the list. So the Mapper function remains the same as Problem 2.

```
43    public static class TokenizerMapper
44    extends Mapper<Object, Text, IntWritable, Text>{
45
46        private Text word = new Text();
47        private Text count = new Text();
48
49        public void map(Object key, Text value, Context context
50                ) throws IOException, InterruptedException {
51            StringTokenizer itr = new StringTokenizer(value.toString());
52            while (itr.hasMoreTokens()) {
53                word.set(itr.nextToken());
54                count.set(itr.nextToken());
55                int intCount = Integer.parseInt(count.toString());
56                context.write(new IntWritable(intCount), word);
57            }
58        }
59    }
```

2. In the Reducer class, calculate the number of words in the word list. The output pair is (occurrence, number of words), so the output data type is (IntWritable, IntWritable).

```
61    public static class IntSumReducer
62    extends Reducer<IntWritable, Text, IntWritable, IntWritable> {
63        private IntWritable result = new IntWritable();
64
65        public void reduce(IntWritable key, Iterable<Text> values,
66                Context context
67                ) throws IOException, InterruptedException {
68            int sum = 0;
69            for (Text val : values) {
70                sum++;
71            }
72            result.set(sum);
73            context.write(key, result);
74        }
75    }
```

3. Note that since the Mapper class and the Reducer class have different output key class and output value class, we should define them separately!

4. Also, we don't want combiner class here. Otherwise, the reduce function will be called twice and cause error.

```
77    public static void main(String[] args) throws Exception {
78        Configuration conf = new Configuration();
79        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
80        if (otherArgs.length < 2) {
81            System.err.println("Usage: wordcount4 <in> [<in>...] <out>");
82            System.exit(2);
83        }
84        Job job = new Job(conf, "word count");
85        job.setJarByClass(WordCount4.class);
86        job.setMapperClass(TokenizerMapper.class);
87        job.setReducerClass(IntSumReducer.class);
88        job.setMapOutputKeyClass(IntWritable.class);
89        job.setMapOutputValueClass(Text.class);
90        job.setOutputKeyClass(IntWritable.class);
91        job.setOutputValueClass(IntWritable.class);
92
93        for (int i = 0; i < otherArgs.length - 1; ++i) {
94            FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
95        }
96        FileOutputFormat.setOutputPath(job,
97                new Path(otherArgs[otherArgs.length - 1]));
98        System.exit(job.waitForCompletion(true) ? 0 : 1);
99    }
```

```
[cloudera@localhost HW03]$ hadoop jar WordCount4.jar org.apache.hadoop.examples.WordCount4 output2 output4
16/02/19 07:17:07 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/02/19 07:17:08 INFO input.FileInputFormat: Total input paths to process : 1
16/02/19 07:17:08 INFO mapreduce.JobSubmitter: number of splits:1
16/02/19 07:17:08 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1455215283455_0010
16/02/19 07:17:09 INFO impl.YarnClientImpl: Submitted application application_1455215283455_0010
16/02/19 07:17:09 INFO mapreduce.Job: The url to track the job: http://localhost:8088/proxy/application_1455215283455_
0010/
16/02/19 07:17:09 INFO mapreduce.Job: Running job: job_1455215283455_0010
```

```
[cloudera@localhost HW03]$ hadoop fs -ls output4
Found 2 items
-rw-r--r--   1 cloudera supergroup          0 2016-02-19 07:17 output4/_SUCCESS
-rw-r--r--   1 cloudera supergroup       1424 2016-02-19 07:17 output4/part-r-00000
[cloudera@localhost HW03]$ hadoop fs -cat output4/part-r-00000 | head -20
1       15656
2       4668
3       2199
4       1265
5       891
6       610
7       501
8       349
9       308
10      241
11      195
12      166
13      149
14      116
15      96
16      92
17      91
18      66
19      57
20      68
```

Problem 4) Combine operations of two MapReduce programs in Problems 1 and 3 above into a single program with chained MapReduce jobs.

Solution:

1. Chain Map Reduce jobs. In WordCount5.java, I use the old API and use the “ToolRunner”.

```
158    public static class ChainJobs extends Configured implements Tool {
159        public int run(String[] args) throws Exception {
160            Configuration conf = getConf();
161
162            // Job 1
163            Job job1 = new Job(conf, "word count 1");
164            job1.setJarByClass(WordCount5.class);
165            job1.setMapperClass(TokenizerMapper1.class);
166            job1.setCombinerClass(IntSumReducer1.class);
167            job1.setReducerClass(IntSumReducer1.class);
168            job1.setOutputKeyClass(Text.class);
169            job1.setOutputValueClass(IntWritable.class);
170
171            FileInputFormat.addInputPath(job1, new Path(args[0]));
172            FileOutputFormat.setOutputPath(job1, new Path(args[1]));
173
174            job1.waitForCompletion(true);
175
176            // Job 2
177            Job job2 = new Job(conf, "word count 2");
178            job2.setJarByClass(WordCount5.class);
179            job2.setMapperClass(TokenizerMapper2.class);
180            job2.setReducerClass(IntSumReducer2.class);
181            job2.setMapOutputKeyClass(IntWritable.class);
182            job2.setMapOutputValueClass(Text.class);
183            job2.setOutputKeyClass(IntWritable.class);
184            job2.setOutputValueClass(IntWritable.class);
185
186            FileInputFormat.addInputPath(job2, new Path(args[2]));
187            FileOutputFormat.setOutputPath(job2, new Path(args[3]));
188
189            return job2.waitForCompletion(true) ? 0 : 1;
190        }
191    }
```

```
193    public static void main(String[] args) throws Exception {
194        Configuration conf = new Configuration();
195        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
196        if (otherArgs.length < 4) {
197            System.err.println("Usage: wordcount5 <in> [<in>...] <out>");
198            System.exit(2);
199        }
200
201        ToolRunner.run(conf, new ChainJobs(), otherArgs);
202    }
```

In WordCount6.java, I directly chain the two jobs. In both files, the output of the first job will become the input of the second job. We use FileInputFormat.addInputPath() and FileOutputFormat.setOutputPath() to set the output of the first job to the input of the second job.

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length < 3) {
        System.err.println("Usage: wordcount6 <in> [<in>...] <out>");
        System.exit(2);
    }

    // Job 1
    Job job1 = new Job(conf, "word count 1");
    job1.setJarByClass(WordCount6.class);
    job1.setOutputKeyClass(Text.class);
    job1.setOutputValueClass(IntWritable.class);

    job1.setMapperClass(TokenizerMapper1.class);
    job1.setCombinerClass(IntSumReducer1.class);
    job1.setReducerClass(IntSumReducer1.class);

    FileInputFormat.addInputPath(job1, new Path(args[0]));
    FileOutputFormat.setOutputPath(job1, new Path(args[1]));

    job1.waitForCompletion(true);

    // Job 2
    Job job2 = new Job(conf, "word count 2");
    job2.setJarByClass(WordCount6.class);

    job2.setMapperClass(TokenizerMapper2.class);
    job2.setReducerClass(IntSumReducer2.class);
    job2.setMapOutputKeyClass(IntWritable.class);

    job2.setMapOutputValueClass(Text.class);
    job2.setOutputKeyClass(IntWritable.class);
    job2.setOutputValueClass(IntWritable.class);

    FileInputFormat.addInputPath(job2, new Path(args[1]));
    FileOutputFormat.setOutputPath(job2, new Path(args[2]));

    System.exit(job2.waitForCompletion(true) ? 0 : 1);
}
```

```
[cloudera@localhost HW03]$ hadoop jar WordCount5.jar org.apache.hadoop.examples.WordCount5 ulysses output5 output5
output6
16/02/19 07:22:44 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/02/19 07:22:45 INFO input.FileInputFormat: Total input paths to process : 1
16/02/19 07:22:46 INFO mapreduce.JobSubmitter: number of splits:1
16/02/19 07:22:46 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1455215283455_0013
16/02/19 07:22:46 INFO impl.YarnClientImpl: Submitted application application_1455215283455_0013
16/02/19 07:22:46 INFO mapreduce.Job: The url to track the job: http://localhost:8088/proxy/application_14552152834
55_0013/
16/02/19 07:22:46 INFO mapreduce.Job: Running job: job_1455215283455_0013
16/02/19 07:22:52 INFO mapreduce.Job: Job job_1455215283455_0013 running in uber mode : false
16/02/19 07:22:52 INFO mapreduce.Job:  map 0% reduce 0%
16/02/19 07:23:00 INFO mapreduce.Job:  map 100% reduce 0%
16/02/19 07:23:06 INFO mapreduce.Job:  map 100% reduce 100%
16/02/19 07:23:06 INFO mapreduce.Job: Job job_1455215283455_0013 completed successfully
```

We can see that the output of the first job is the same as the result of Problem 1.

```
[cloudera@localhost HW03]$ hadoop fs -cat output5/part-r-00000 | head -20
aaron   2
aback   1
abaft   1
abandon 1
abandoned       7
abandoning      1
abandonment     1
abasement       2
abatement       1
abattoir        1
abba    1
abbas   2
abbess  1
abbey   13
abbot   3
abbots  1
abbreviation    1
abdomen 2
abdominal       2
abe     1
```

The result of the second job is the same as the result of Problem 3.

```
[cloudera@localhost HW03]$ hadoop fs -cat output6/part-r-00000 | head -20
1       15656
2       4668
3       2199
4       1265
5       891
6       610
7       501
8       349
9       308
10      241
11      195
12      166
13      149
14      116
15      96
16      92
17      91
18      66
19      57
20      68
```

```
[cloudera@localhost HW03]$ diff output4/part-r-00000 output6/part-r-00000
[cloudera@localhost HW03]$ diff output2/part-r-00000 output5/part-r-00000
```

```
[cloudera@localhost HW03]$ hadoop jar WordCount6.jar org.apache.hadoop.examples.WordCount6 ulysses output9 output10
16/02/19 09:06:58 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/02/19 09:06:59 INFO input.FileInputFormat: Total input paths to process : 1
16/02/19 09:06:59 INFO mapreduce.JobSubmitter: number of splits:1
16/02/19 09:07:00 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1455215283455_0019
16/02/19 09:07:00 INFO impl.YarnClientImpl: Submitted application application_1455215283455_0019
16/02/19 09:07:00 INFO mapreduce.Job: The url to track the job: http://localhost:8088/proxy/application_1455215283455_
0019/
```

```
[cloudera@localhost HW03]$ hadoop fs -cat output10/part-r-00000 | head -20
1	15656
2	4668
3	2199
4	1265
5	891
6	610
7	501
8	349
9	308
10	241
11	195
12	166
13	149
14	116
15	96
16	92
17	91
18	66
19	57
20	68
```

No diff results mean two files are identical.

```
[cloudera@localhost HW03]$ diff output5/part-r-00000 output9/part-r-00000
[cloudera@localhost HW03]$ diff output6/part-r-00000 output10/part-r-00000
```

Problem 5) Move attached Inverter.java class from old MapReduce API to the new API. Demonstrate that new and old clas produce the same result. Use patent data set to demonstrate your work.

Solution:

1. Re-write the Inverter class using new API. The (key, value) pairs are (Text, Text). Change the sequence of (citing, cited) in the Mapper function.

```
16 public class Inverter2 {
17
18     public static class TokenizerMapper
19     extends Mapper<Object, Text, Text, Text> {
20
21         private Text str = new Text();
22         private Text citing = new Text();
23         private Text cited = new Text();
24
25         public void map(Object key, Text value, Context context
26                 ) throws IOException, InterruptedException {
27             StringTokenizer itr = new StringTokenizer(value.toString());
28             while (itr.hasMoreTokens()) {
29                 str.set(itr.nextToken());
30                 String[] args = str.toString().split(",");
31                 citing.set(args[0]);
32                 cited.set(args[1]);
33                 context.write(cited, citing);
34             }
35         }
36     }
```

```
38      public static class IntSumReducer
39      extends Reducer<Text, Text, Text, Text> {
40
41          public void reduce(Text key, Iterable<Text> values,
42                  Context context
43                  ) throws IOException, InterruptedException {
44              String csv = "";
45              for (Text value : values) {
46                  if (csv.length() > 0) csv += ",";
47                  csv += value.toString();
48              }
49              context.write(key, new Text(csv));
50          }
51      }
```

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length < 2) {
      System.err.println("Usage: Inverter2 <in> [<in>...] <out>");
      System.exit(2);
    }
    Job job = new Job(conf, "Inverter 2");
    job.setJarByClass(Inverter2.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);
    for (int i = 0; i < otherArgs.length - 1; ++i) {
      FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
    }
    FileOutputFormat.setOutputPath(job,
      new Path(otherArgs[otherArgs.length - 1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

```
[cloudera@localhost HW03]$ hadoop fs -mkdir patents
[cloudera@localhost HW03]$ hadoop fs -copyFromLocal input/cite75_99.txt patents
[cloudera@localhost HW03]$ hadoop fs -ls patents
Found 1 items
-rw-r--r--   1 cloudera supergroup  264075431 2016-02-19 07:27 patents/cite75_99.txt
```

Run the old Inverter class and the new one.

```
[cloudera@localhost HW03]$ hadoop jar Inverter.jar org.apache.hadoop.examples.Inverter patents output7
16/02/19 07:28:56 INFO Configuration.deprecation: key.value.separator.in.input.line is deprecated. Instead, use mapred
uce.input.keyvaluelinerecordreader.key.value.separator
16/02/19 07:28:56 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/02/19 07:28:57 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/02/19 07:28:57 INFO mapred.FileInputFormat: Total input paths to process : 1
16/02/19 07:28:57 INFO net.NetworkTopology: Adding a new node: /default-rack/127.0.0.1:50010
16/02/19 07:28:57 INFO mapreduce.JobSubmitter: number of splits:2
16/02/19 07:28:58 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1455215283455_0015
16/02/19 07:28:58 INFO impl.YarnClientImpl: Submitted application application_1455215283455_0015
16/02/19 07:28:58 INFO mapreduce.Job: The url to track the job: http://localhost:8088/proxy/application_1455215283455_
0015/
16/02/19 07:28:58 INFO mapreduce.Job: Running job: job_1455215283455_0015
16/02/19 07:29:05 INFO mapreduce.Job: Job job_1455215283455_0015 running in uber mode : false
16/02/19 07:29:05 INFO mapreduce.Job:  map 0% reduce 0%
16/02/19 07:29:20 INFO mapreduce.Job:  map 13% reduce 0%
```

```
[cloudera@localhost HW03]$ hadoop jar Inverter2.jar org.apache.hadoop.examples.Inverter2 patents output8
16/02/19 07:31:07 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
16/02/19 07:31:08 INFO input.FileInputFormat: Total input paths to process : 1
16/02/19 07:31:08 INFO mapreduce.JobSubmitter: number of splits:2
16/02/19 07:31:08 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1455215283455_0016
16/02/19 07:31:08 INFO impl.YarnClientImpl: Submitted application application_1455215283455_0016
16/02/19 07:31:08 INFO mapreduce.Job: The url to track the job: http://localhost:8088/proxy/application_1455215283455_
0016/
16/02/19 07:31:08 INFO mapreduce.Job: Running job: job_1455215283455_0016
16/02/19 07:31:16 INFO mapreduce.Job: Job job_1455215283455_0016 running in uber mode : false
16/02/19 07:31:16 INFO mapreduce.Job:  map 0% reduce 0%
16/02/19 07:31:31 INFO mapreduce.Job:  map 8% reduce 0%
```

```
[cloudera@localhost HW03]$ hadoop fs -cat output7/part-00000 | head -20
"CITED" "CITING"
1       3964859,4647229
10000   4539112
100000  5031388
1000006 4714284
1000007 4766693
1000011 5033339
1000017 3908629
1000026 4043055
1000033 4190903,4975983
1000043 4091523
1000044 4082383,4055371
1000045 4290571
1000046 5525001,5918892
1000049 5996916
1000051 4541310
1000054 4946631
1000065 4748968
1000067 5071294,4944640,5312208
1000070 4928425,5009029
```

```
[cloudera@localhost HW03]$ hadoop fs -cat output8/part-r-00000 | head -20
"CITED" "CITING"
1       3964859,4647229
10000   4539112
100000  5031388
1000006 4714284
1000007 4766693
1000011 5033339
1000017 3908629
1000026 4043055
1000033 4190903,4975983
1000043 4091523
1000044 4082383,4055371
1000045 4290571
1000046 5525001,5918892
1000049 5996916
1000051 4541310
1000054 4946631
1000065 4748968
1000067 5071294,4944640,5312208
1000070 4928425,5009029
```

The result files are of the same size.

```
[cloudera@localhost HW03]$ hadoop fs -ls output7
Found 2 items
-rw-r--r--   1 cloudera supergroup          0 2016-02-19 07:30 output7/_SUCCESS
-rw-r--r--   1 cloudera supergroup  158078539 2016-02-19 07:30 output7/part-00000
[cloudera@localhost HW03]$ hadoop fs -ls output8
Found 2 items
-rw-r--r--   1 cloudera supergroup          0 2016-02-19 07:32 output8/_SUCCESS
-rw-r--r--   1 cloudera supergroup  158078539 2016-02-19 07:32 output8/part-r-00000
```

```
[cloudera@localhost HW03]$ hadoop fs -cat output8/part-r-00000 | tail -20
99993   4327613
999930  5609215
999932  4605176
999936  5014973,5642878
999940  3876070,4022472
999941  5447466
999945  5207231,5569166
999949  5640640
999951  5316622,5374468
999957  5755359
999961  5782495,5738381,5878901,4171117,4262874,5048788,4871140,4832301,4437639
999965  5052613
999968  3916735
999971  3965843
999972  4038129
999973  4900344,5427610
999974  5464105,4560073,4728158
999977  4092587
999978  3915443
999983  5143114,5394715,5806555
```

```
[cloudera@localhost HW03]$ hadoop fs -cat output7/part-00000 | tail -20
99993   4327613
999930  5609215
999932  4605176
999936  5014973,5642878
999940  3876070,4022472
999941  5447466
999945  5207231,5569166
999949  5640640
999951  5316622,5374468
999957  5755359
999961  5878901,5738381,5782495,4171117,4262874,5048788,4871140,4832301,4437639
999965  5052613
999968  3916735
999971  3965843
999972  4038129
999973  4900344,5427610
999974  5464105,4560073,4728158
999977  4092587
999978  3915443
999983  5143114,5394715,5806555
```

The results are the same. On Linux VM, the sequence of the second column might differ. But the contents are the same. The results are identical when running on my Mac locally.

```
hqiu@bos-mp9cx>> cat output8/part-r-00000 | tail -20
99993   4327613
999930  5609215
999932  4605176
999936  5642878,5014973
999940  3876070,4022472
999941  5447466
999945  5569166,5207231
999949  5640640
999951  5374468,5316622
999957  5755359
999961  5782495,5738381,4437639,5878901,4871140,4832301,4262874,4171117,5048788
999965  5052613
999968  3916735
999971  3965843
999972  4038129
999973  5427610,4900344
999974  4560073,4728158,5464105
999977  4092587
999978  3915443
999983  5806555,5394715,5143114
hqiu@bos-mp9cx>> diff output7/part-00000 output8/part-r-00000
hqiu@bos-mp9cx>>
```