Aims:

This exercise aims to get you to:

1. Practice with the use of Hadoop partitioners
2. Run the example WordCount (with partitioners) using Hadoop server
3. (Optional) Install Hadoop in your own computer/laptop

Adding a Partitioner to the WordCount example

Partitioners are very useful when the outputs of our MapReduce computations need to be grouped together in different partitions (files) according to a given criterion. In this exercise, we will ask you to create a partitioner for the WordCount example and use three partitions. The first partition must contain all (and only) the words starting with the letter ‘a’. The second partition must contain all (and only) the words starting with the letter ‘b’. The third and last partition will contain all the remaining words.

```
public static class WordPartitioner extends Partitioner <Text, IntWritable> {

        @Override
        public int getPartition(Text key, IntWritable value, int numReduceTasks) {
               char keyChar0 = key.toString().charAt(0);

               if (numReduceTasks == 0) {
                      return 0;
               }

               if (keyChar0 == 'a') {
                      return 0;
               } else if (keyChar0 == 'b') {
                      return 1 % numReduceTasks;
               } else {
                      return 2 %  numReduceTasks;
               }
        }
 }
```

The class `WordPartitioner` above contains the logic for partitioning the <key,value> pairs processed by the reducer. Notice that the `WordPartitioner` class extends the class Partitioner, which handles pairs of the type `<Text, IntWritable>`.

Our `WordPartitioner` class must override the method `getPartition` as shown above. Notice that this method gets in input three parameters: key, value and `numReduceTasks`. You are by now familiar with the first two parameters. The last parameters is the total number of reduce tasks defined in the driver (we will get back to this later). The value returned by this method correspond to the partition/reducer where each <key,value> will be sent.

Configure the Driver

Once you have created the class `WordPartitioner`, you now need to set up the partitioner in the driver. To do so, you need to add the following lines to the `main` method of your `WordCount` class.

```
 public static void main(String[] args) throws Exception {
   Configuration conf = new Configuration();
```

```
    Job job = Job.getInstance(conf, "word count");
    job.setNumReduceTasks(3);
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setReducerClass(IntSumReducer.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setPartitionerClass(WordPartitioner.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
```

Notice the line with `job.setNumReduceTasks(3)` above. This line tells Hadoop to create 3 reducers to process the reduction phase. In addition, you need to tell the driver where to find the partitioner's implementation. This is done with the line `job.setPartitionerClass(WordPartitioner.class)`.

you also need to add the following line to the list of your import classes, to import the Partitioner class :

```
import org.apache.hadoop.mapreduce.Partitioner;
```

Running your code

You can now run your code, e.g., from Eclipse. You will notice that the output/ directory still contains a single partition. The reason for this is that Eclipse is running your code using the development libraries instead of an actual Hadoop server. In the following sections, we will show you how to generate the Jar file of your program and run it on Hadoop server.

Generating your program's Jar

The next step consists in generating the Jar file of the WordCount project. If you use Eclipse, you can just right click on the projects pom.xml file and then in the menu select Run As -> Maven Install.

If the building process is successful, you should see in Eclipse's console an output similar to:

```
...
...
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2.936 s
[INFO] Finished at: 2019-06-25T13:53:54+10:00
[INFO] Final Memory: 20M/214M
[INFO] ------------------------------------------------------------------------
```

This building process produces as a result a Jar file that can be found in your project's "target" directory (in Eclipse, you can typically find the "target" folder on left menu where the project's artefacts are listed). This is the Jar file that will be used by Hadoop next.

Executing your Jar with Hadoop server

The final step consists in executing the Jar file of your WordCount program using Hadoop. We have set up a Hadoop server in the lab computers (VLab). In order to do this, first you need to open a terminal in VLab and run the following program:

```
$ 9313
```

This program will setup the environment needed to run your program on Hadoop. You should see an output like the following:

```
$ 9313
Welcome to COMP9313!
newclass starting new subshell for class COMP9313...
```

Then, in order to execute your Jar file generated in the previous section, in the terminal you need to change to the directory containing your Jar file:

```
$ cd PATH_TO_DIRECTORY_CONTAINING_YOUR_JAR_FILE
```

Next, you need to execute your program in Hadoop using the following command:

```
$ hadoop jar YOU_JAR_FILE.jar YOUR_MAIN_CLASS YOUR_INPUT_DIRECTORY YOUR_OUTPUT_DIRECTORY
```

Here, `YOU_JAR_FILE.jar` is the name of you Jar file, `YOUR_MAIN_CLASS` is the class name of the driver in your program, `YOUR_INPUT_DIRECTORY` is the full path of your input directory, and `YOUR_OUTPUT_DIRECTORY` is the full path of your output directory.

The `YOUR_MAIN_CLASS` is in fact your package-name.class-name; which in the case of this exercise would be: au.edu.unsw.cse.bdmc.wordcount.WordCount

If the program is executed successfully, you will end up with the output directory containing the three partition files as specified in the `WordPartitioner` class. For example:

```
YOUR_OUTPUT_DIRECTORY/
        part-r-00000
        part-r-00001
        part-r-00002
```

Each file should contain the words as partitioned in the `WordPartitioner` class.

Using a different logic for partitioning

You can play with the code above to change the number of reducers/partitions as well as the partitioning logic. For example, you can try to partition words based on their length.

Optional: Installing Hadoop in your own computer/laptop

The instructions presented here must be done on your own computer/laptop. Assuming you have Java SDK installed in your computer/laptop, you can download Apache Hadoop from the following link:

http://hadoop.apache.org/releases.html

Once you downloaded Apache Hadoop (usually a .zip or .tar.gz file), extract the file and remember the location of the resulting directory.

Open the terminal (command line tool) and go to the directory above. For example:

```
$ cd HADOOP_HOME_DIRECTORY
```

(replace HADOOP_HOME_DIRECTORY with the name of the Hadoop directory in your computer/laptop).

Note1: You may need to set up some environment variables (which depends on the OS you use). More details can be found here: https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

You can try to run Hadoop by issuing the following command:

```
$ bin/hadoop
```

Note1: In Windows systems, the command is typically called `hadoop.cmd`

The output should be something like the following:

```
Usage: hadoop [OPTIONS] SUBCOMMAND [SUBCOMMAND OPTIONS]
 or    hadoop [OPTIONS] CLASSNAME [CLASSNAME OPTIONS]
  where CLASSNAME is a user-provided Java class

  OPTIONS is none or any of:

--config dir                     Hadoop config directory
--debug                          turn on shell script debug mode
--help                           usage information
...
```

If you manage to see the above result, you should be ready for running your MapReduce program.