**Name: Afifa Dalal**

**Enrolment No: 202001619010083**

**Subject: Hadoop _Assignment**

**Q1 Develop a MapReduce job that will split each line of the input text file in to tokens and then count the occurrences of each unique token. The input text file should be available on the HDFS. The output will be in the form of: Hadoop 5 Hotspot 2**

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
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.output.FileOutputFormat;

public class WordCount {
  // Mapper class
  public static class TokenizerMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();
    public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
      // Split each line into tokens
      String[] tokens = value.toString().split("\\s+");
      for (String token: tokens) {
        word.set(token);
        context.write(word, one); // Emit each word with a count of 1
```

```
        }
      }
    }

   // Reducer class
    public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
      private IntWritable result = new IntWritable();
      public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
          sum += val.get();
        }
        result.set(sum);
        context.write(key, result); // Emit the word and its total count
      }
    }
    // Main method
    public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
      Job job = Job.getInstance(conf, "word count");
      job.setJarByClass(WordCount.class);
      job.setMapperClass(TokenizerMapper.class);
      job.setCombinerClass(IntSumReducer.class);
      job.setReducerClass(IntSumReducer.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);
    }
```

```
}
```

**Q2 Develop a MapReduce job that will analyze the minimum temperature for each year. The input text file should be available on HDFS. The output will be in the form of: 2014 -1 2015 20**

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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.output.FileOutputFormat;

public class MinTemperature {
  // Mapper class
  public static class TemperatureMapper extends Mapper<Object, Text, Text, IntWritable> {
    private Text year = new Text();
    private IntWritable temperature = new IntWritable();
    public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
      String[] fields = value.toString().split(",");
      if (fields.length == 2) {
        year.set(fields[0].trim()); // Extract the year
        temperature.set(Integer.parseInt(fields[1].trim())); // Extract the temperature
        context.write(year, temperature); // Emit year and temperature as key-value pairs
      }
    }
  }
```

```
  // Reducer class
  public static class MinTemperatureReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable minTemperature = new IntWritable();
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
      int minTemp = Integer.MAX_VALUE;
      // Find the minimum temperature for the current year
      for (IntWritable value : values) {
        minTemp = Math.min(minTemp, value.get());
      }
      minTemperature.set(minTemp);
      context.write(key, minTemperature);
    }
  }
  // Main method
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "min temperature");
    job.setJarByClass(MinTemperature.class);
    job.setMapperClass(TemperatureMapper.class);
    job.setReducerClass(MinTemperatureReducer.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);
  }
}
```

**Q3 Develop a MapReduce job that will split each line of the input text file in to tokens and then find the average count of all the tokens in the input file. The input text file should be available on the HDFS. The output will be in the form of: Hadoop 5 Hotspot 2 Average Count = 3.5**

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;


public class TokenCountJob {
  // Mapper class
public class TokenCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();
  public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
    String[] tokens = value.toString().split("\\s+");
    for (String token : tokens) {
      word.set(token);
      context.write(word, one); // Emit each token with a count of 1
    }
  }
}
  // Reducer class
```

```
public class TokenCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
   private int totalTokens = 0; // Total count of all tokens
   private int uniqueTokens = 0; // Total number of unique tokens
   public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
         sum += val.get(); // Sum the occurrences of each token
      }
      context.write(key, new IntWritable(sum)); // Emit the token and its total count
      totalTokens += sum; // Update total tokens
      uniqueTokens++; // Update unique token count
   }
   @Override
   protected void cleanup(Context context) throws IOException, InterruptedException {
      double average = (double) totalTokens / uniqueTokens;
      context.write(new Text("Average count"), new IntWritable((int) average)); // Emit the average
count
   }
}
   // Main method
   public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
      Job job = Job.getInstance(conf, "token count with average");
      job.setJarByClass(TokenCountJob.class);
      job.setMapperClass(TokenCountMapper.class);
      job.setReducerClass(TokenCountReducer.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);
   }
```

}

**Q4 Develop a MapReduce job that will take text file input and will produce the output: (1) Count of the total token having length greater than or equal to four characters. Total count for token = 5**

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class TokenLengthJob {
  // Mapper class
public class TokenLengthMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text token = new Text();
  public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
    // Split the line into tokens (words)
    String[] tokens = value.toString().split("\\s+");
    // Iterate over tokens and check length
    for (String word : tokens) {
```

```
        if (word.length() >= 4) {
            token.set(word); // Emit only tokens with length >= 4
            context.write(token, one);
        }
      }
   }
}
   // Reducer class
public class TokenLengthReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
   private int totalCount = 0;
   public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
      int sum = 0;
      // Count occurrences of each token
      for (IntWritable val : values) {
         sum += val.get();
      }
      // Emit the token and its count
      context.write(key, new IntWritable(sum));
      // Update the total count of tokens with length >= 4
      totalCount += sum;
   }
   @Override
   protected void cleanup(Context context) throws IOException, InterruptedException {
      // Emit the total count in cleanup after all reduce tasks are done
      context.write(new Text("Total count for tokens with length >= 4"), new
IntWritable(totalCount));
   }
}
   // Main method
   public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
      Job job = Job.getInstance(conf, "token length count");
```

```
        job.setJarByClass(TokenLengthJob.class);
        job.setMapperClass(TokenLengthMapper.class);
        job.setReducerClass(TokenLengthReducer.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);
    }
}
```

**Q5 Develop a MapReduce job that will count the total number of females voters in the record file Voters.txt. The input text file should be available on the HDFS. The file must have fields: ID, NAME, GENDER, AGE. The output will be in the form of: No. of female voters are: 5**

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class FemaleVoterJob {
```

```
   // Mapper class
public class FemaleVoterMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
   private final static IntWritable one = new IntWritable(1);
   private Text female = new Text("Female Voter");
   public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
      // Split the line by commas (CSV format)
      String[] fields = value.toString().split(",");
      // Check if the record has 4 fields and the third field is 'F' (gender)
      if (fields.length == 4 && fields[2].trim().equalsIgnoreCase("F")) {
         context.write(female, one); // Emit "Female Voter" with a count of 1
      }
   }
}
// Reducer class
public class FemaleVoterReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
   public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
      int totalFemales = 0;
      // Sum all the occurrences of female voters
      for (IntWritable val : values) {
         totalFemales += val.get();
      }
      // Emit the total count of female voters
      context.write(new Text("No. of female voters are : "), new IntWritable(totalFemales));
   }
}
   // Main method
   public static void main(String[] args) throws Exception {
      Configuration conf = new Configuration();
      Job job = Job.getInstance(conf, "female voter count");
      job.setJarByClass(FemaleVoterJob.class);
      job.setMapperClass(FemaleVoterMapper.class);
```

```
        job.setReducerClass(FemaleVoterReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(args[0]));  // Input path from HDFS
        FileOutputFormat.setOutputPath(job, new Path(args[1])); // Output directory in HDFS
        System.exit(job.waitForCompletion(true) ? 0:1);
    }
}
```

**Q6 Develop a MapReduce job that will count the number of reviews given by each unique user for the Musical Instruments. The file should be available on HDFS. Its available on Moodle (Musical_instruments_reviews.csv). The output should be in the form of A00625243BI8W1SSZNLMD  3 A10044ECXDUVKS 2**

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class ReviewCountJob {
    // Mapper class
public class ReviewCountMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
```

```
    private final static IntWritable one = new IntWritable(1);
    private Text reviewerID = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        // Split the line by commas
        String[] fields = value.toString().split(",");

        // Check if there are enough fields and extract the reviewerID
        if (fields.length > 0) {
            reviewerID.set(fields[0].trim()); // First field is reviewerID
            context.write(reviewerID, one);   // Emit the reviewerID with a count of 1
        }
    }
}
// Reducer class
public class ReviewCountReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
        int totalReviews = 0;

        // Sum all the occurrences of reviews for each user
        for (IntWritable val: values) {
            totalReviews += val.get();
        }

        // Emit the reviewerID and the total number of reviews
        context.write(key, new IntWritable(totalReviews));
    }
}
    // Main method
    public static void main(String[] args) throws Exception {
```

```
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "review count");
        job.setJarByClass(ReviewCountJob.class);
        job.setMapperClass(ReviewCountMapper.class);
        job.setReducerClass(ReviewCountReducer.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);
    }
}
```