DSBDA Group B Assignments

Group B : 01

Program :

```
package org.myorg;

import

java.io.IOException;

import java.util.*;

import

org.apache.hadoop.fs.Path;

import

org.apache.hadoop.conf.*;

import org.apache.hadoop.io.*;

import org.apache.hadoop.mapreduce.*;

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;
```

```
public class WordCount

{

        public static class Map 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 line = value.toString();

                  StringTokenizer tokenizer = new

                  StringTokenizer(line); while

                  (tokenizer.hasMoreTokens())

                  {

                       word.set(tokenizer.nextToken(

                       )); context.write(word, one);

                  }

             }

}


public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>

{

       public void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException

       {

            int sum = 0;
            for (IntWritable val : values)
           {
                sum += val.get();
           }
          context.write(key, new IntWritable(sum));
     }
}
public static void main(String[] args) throws Exception
{
         Configuration conf = new
         Configuration(); Job job = new Job(conf,
         "wordcount");
         job.setOutputKeyClass(Text.class);
         job.setOutputValueClass(IntWritable.cla
         ss); job.setMapperClass(Map.class);
         job.setReducerClass(Reduce.class);
         job.setInputFormatClass(TextInputFormat.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        FileInputFormat.addInputPath(job, new
        Path(args[0])); FileOutputFormat.setOutputPath(job,
        new Path(args[1])); job.waitForCompletion(true);
}
```

Output :

Compile `WordCount.java` and create a jar:

```
$ bin/hadoop com.sun.tools.javac.Main WordCount.java

$ jar cf wc.jar WordCount*.class
```

Sample text-files as input:

```
$ bin/hadoop fs -ls /user/wordcount/input/

/user/wordcount/input/file01

/user/wordcount/input/file02


$ bin/hadoop fs -cat /user/wordcount/input/file01

Hello World Bye World


$ bin/hadoop fs -cat /user/wordcount/input/file02

Hello Hadoop Goodbye Hadoop
```

Output:

```
$ bin/hadoop jar wc.jar WordCount /user/wordcount/input
/user/wordcount/output

Output:

$ bin/hadoop fs -cat /user/wordcount/output/part-r-00000

Bye 1

Goodbye 1

Hadoop 2

Hello 2

World 2
```

Group B : 02

SalesCountry.java

```
package
SalesCountry;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import
org.apache.hadoop.mapred.*;
public class SalesCountryDriver
{
	public static void main(String[] args)
	{
		JobClient my_client = new JobClient();
		// Create a configuration object for the job
		JobConf job_conf = new JobConf(SalesCountryDriver.class);
		// Set a name of the Job
		job_conf.setJobName("SalePerCountry");
		// Specify data type of output key and value
		job_conf.setOutputKeyClass(Text.class);
```

```
                job_conf.setOutputValueClass(IntWritable.class);
                // Specify names of Mapper and Reducer Class
                job_conf.setMapperClass(SalesCountry.SalesMapper.class);
                job_conf.setReducerClass(SalesCountry.SalesCountryReducer.class);
                // Specify formats of the data type of Input and output
                job_conf.setInputFormat(TextInputFormat.class);
                job_conf.setOutputFormat(TextOutputFormat.class);
                // Set input and output directories using command line arguments,
                //arg[0] = name of input directory on HDFS, and arg[1] = name of output
                //directory to be created to store the output file.
                FileInputFormat.setInputPaths(job_conf, new Path(args[0]));
                FileOutputFormat.setOutputPath(job_conf, new Path(args[1]));
                my_client.setConf(job_conf);
                try
                 {
                        // Run the job
                        JobClient.runJob(job_conf);
                } catch (Exception e) { e.printStackTrace(); }
        }
}
```

SalesCountryReducer.jav

a package SalesCountry;

import

java.io.IOException;

```
import java.util.*;

import

org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.*;

public class SalesCountryReducer extends MapReduceBase implements
Reducer<Text, IntWritable, Text, IntWritable>

 {

        public void reduce(Text t_key, Iterator<IntWritable> values, r<Text,IntWritable>
output, Reporter reporter) throws IOException

        {

                Text key = t_key; int frequencyForCountry

                = 0; while (values.hasNext())

                {

                        // replace type of value with the actual type of our
                                                    value

                        IntWritable value = (IntWritable) values.next(); frequencyForCountry
                += value.get();

                }

                output.collect(key, new IntWritable(frequencyForCountry));

        }

}
```

SalesMapper.java

```
package

SalesCountry;

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.mapred.*;

public class SalesMapper extends MapReduceBase implements
Mapper<LongWritable, Text, Text, IntWritable>

{

        private final static IntWritable one = new IntWritable(1);

        public void map(LongWritable key, Text value, OutputCollector<Text,
IntWritable> output, Reporter reporter) throws IOException

        {

                String valueString = value.toString();

                String[] SingleCountryData = valueString.split(",");

                output.collect(new Text(SingleCountryData[7]),

                one);

        }

}
```

Group B : 03

```
import

java.io.IOException;

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.util.StringTokenizer;

import

org.apache.hadoop.conf.Configuration;

import

org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import

org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import

org.apache.hadoop.mapred.FileInputFormat;

import

org.apache.hadoop.mapred.FileOutputFormat;

import org.apache.hadoop.mapred.JobClient;

import org.apache.hadoop.mapred.JobConf;
```

```
import

org.apache.hadoop.mapred.KeyValueTextInputFormat;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.Mapper;

import

org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reducer;

import org.apache.hadoop.mapred.Reporter;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;


public class Weather extends Configured implements Tool

{

        final long DEFAULT_SPLIT_SIZE = 128 * 1024 * 1024;

        public static class MapClass extends MapReduceBase implements
Mapper<LongWritable, Text, Text, Text>

        {

                private Text word = new

                Text(); private Text values =

                new Text();

                public void map(LongWritable key, Text value, OutputCollector<Text,
        Text> output, Reporter reporter) throws IOException

                {

                        String line = value.toString();

                        StringTokenizer itr = new

                        StringTokenizer(line); int counter = 0;

                        String key_out = null;

                        String value_str = null;
```

```
boolean skip = false;

loop:while (itr.hasMoreTokens() && counter<13)

{

        String str = itr.nextToken();

        switch (counter)

         {

                case 0:

                key_out = str;

                if(str.contains("STN"))

                {

                        //Ignoring rows where station id is

                        all 9 skip = true;

                        break loop;

                }

                else

                { break;}

                case 2:

                int hour
        =Integer.valueOf(str.substring(str.lastIndexOf("_")+1,
        str.length()));

                str =

                str.substring(4,str.lastIndexOf("_")-2);

                if(hour>4 && hour<=10)

                { str = str.concat("_section1");

                } else if(hour>10 &&

                hour<=16)

                { str = str.concat("_section2"); }
```

```
else if(hour>16 && hour<=22)

{ str = str.concat("_section3"); }

else{ str =

str.concat("_section4"); }

key_out = key_out.concat("_").concat(str);

break;

case 3:

if(str.equals("9999.9"))

{

                skip =

                 true;

                break

                 loop;

}

Else

{ value_str = str.concat(" "); break; }

case 4:

if(str.equals("9999.9"))

{

            skip = true;

            break loop;

}

else{ value_str = value_str.concat(str).concat(" "); break;

} case 12:

 if(str.equals("999.9"))

{

            skip = true; break loop;
```

```
                                        }
                                        else{ value_str = value_str.concat(str).concat(" "); break;
                                        } default: break;
                                  } counter++;
                            }
                            if(!skip)
                            {
                                  word.set(key_out);
                                  values.set(value_str);
                                  output.collect(word, values);
                            }
                      }
}
public static class MapClassForJob2 extends MapReduceBase implements
Mapper<Text, Text, Text, Text>
 {
      private Text key_text = new Text();
      private Text value_text = new Text();
      public void map(Text key, Text value, OutputCollector<Text, Text> output, Reporter
reporter) throws IOException
      {
            String str = key.toString();
            String station = str.substring(str.lastIndexOf("_")+1,
            str.length()); str = str.substring(0,str.lastIndexOf("_"));
            key_text.set(str);
```

```
                StringTokenizer itr = new

                StringTokenizer(value.toString()); String str_out =

                station.concat("<");

                while (itr.hasMoreTokens())

                {

                     String nextToken = itr.nextToken("

                    "); str_out =

                    str_out.concat(nextToken);

                    str_out = ((itr.hasMoreTokens()) ? str_out.concat(",") :
                str_out.concat(">"));

                }

                value_text.set(str_out); output.collect(key_text,value_text);

        }

}

public static class Reduce extends MapReduceBase implements Reducer<Text, Text,
Text, Text>
{

        private Text value_out_text = new Text();

        public void reduce(Text key, Iterator<Text> values, OutputCollector<Text, Text>
output, Reporter reporter) throws IOException

        {

                 double sum_temp =

                0; double sum_dew

                = 0; double

                sum_wind = 0; int

                count = 0;

                while (values.hasNext())

                {

                    String str = values.next().toString();
```

```
          StringTokenizer itr = new
          StringTokenizer(str); int count_vector = 0;
          while (itr.hasMoreTokens())
          {
               String nextToken = itr.nextToken("
               "); if(count_vector==0)
               {
                    sum_temp += Double.valueOf(nextToken);
               }
               if(count_vector==1)
               {
                    sum_dew += Double.valueOf(nextToken);
               }
               if(count_vector==2)
               {
                    sum_wind += Double.valueOf(nextToken);
               }
               count_vector++;
          } count++;
     }
     double avg_tmp = sum_temp /
     count; double avg_dew = sum_dew
     / count; double avg_wind =
     sum_wind / count;
```

```
            System.out.println(key.toString()+" count is "+count+" sum of
        temp is "+sum_temp+" sum of dew is "+sum_dew+" sum of wind is
        "+sum_wind+"\n");

            String value_out =
        String.valueOf(avg_tmp).concat("").concat(String.valueOf(avg_dew)).concat("
        ").concat(String.valueOf(avg_wind));

             value_out_text.set(value_out);

            output.collect(key,

            value_out_text);

        }

    }

    public static class ReduceForJob2 extends MapReduceBase implements
    Reducer<Text, Text, Text, Text>

     {

        private Text value_out_text = new Text();

        public void reduce(Text key, Iterator<Text> values, OutputCollector<Text, Text>
    output, Reporter reporter) throws IOException

        {

             String value_out = "";

            while

            (values.hasNext())

            {

                value_out = value_out.concat(values.next().toString()).concat(" ");

            }

             value_out_text.set(value_out);

            output.collect(key, value_out_text);

        }

    }

    static int printUsage()
```

```
{
          System.out.println("weather [-m <maps>] [-r <reduces>] <job_1 input> <job_1
          output>
<job_2 output>");
          ToolRunner.printGenericCommandUsage(System.out);
          return -1;
}
public int run(String[] args) throws Exception
{
          Configuration config = getConf();
          JobConf conf = new JobConf(config, Weather.class);
          conf.setJobName("Weather Job1");
          conf.setOutputKeyClass(Text.class);
          conf.setOutputValueClass(Text.class);
          conf.setMapOutputKeyClass(Text.class);
          conf.setMapOutputValueClass(Text.class);
          conf.setMapperClass(MapClass.class);
          conf.setReducerClass(Reduce.class);
          List<String> other_args = new
          ArrayList<String>(); for(int i=0; i < args.length;
          ++i)
          {
                    try
                    {
                              if ("-m".equals(args[i]))
                              {
```

```
                    conf.setNumMapTasks(Integer.parseInt(args[++i]));

                }

                else if ("-r".equals(args[i]))

                {

                    conf.setNumReduceTasks(Integer.parseInt(args[++i]));

                }

                else

                {

                    other_args.add(args[i]);

                }

            }

            catch (NumberFormatException except)

            {

                System.out.println("ERROR: Integer expected instead of " +
                args[i]); return printUsage();

            }

            catch (ArrayIndexOutOfBoundsException except)

            {

                 System.out.println("ERROR: Required parameter missing from " +
                 args[i-
            1]);

                return printUsage();

            }

        }

        FileInputFormat.setInputPaths(conf, other_args.get(0));
        FileOutputFormat.setOutputPath(conf, new
        Path(other_args.get(1)));
        JobClient.runJob(conf);
        JobConf conf2 = new JobConf(config, Weather.class);
        conf2.setJobName("Weather Job 2");
        conf2.setOutputKeyClass(Text.class);
        conf2.setOutputValueClass(Text.class);
        conf2.setInputFormat(KeyValueTextInputFormat.class);
        conf2.setMapOutputKeyClass(Text.class);
        conf2.setMapOutputValueClass(Text.class);
        conf2.setMapperClass(MapClassForJob2.class);
        conf2.setReducerClass(ReduceForJob2.class);
        FileInputFormat.setInputPaths(conf2, new
        Path(other_args.get(1))); FileOutputFormat.setOutputPath(conf2,
        new Path(other_args.get(2))); JobClient.runJob(conf2);
        return 0;
    }
    public static void main(String[] args) throws Exception
    {
         int res = ToolRunner.run(new Configuration(), new Weather(), args);
        System.exit(res);
    }}
```