WORDCOUNT

```
import java.io.IOException;
import java.util.StringTokenizer;
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 org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.fs.Path;

public class WordCount
{
public static class Map extends Mapper<LongWritable,Text,Text,IntWritable> {
public void map(LongWritable key, Text value,Context context) throws
IOException,InterruptedException{
String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
while (tokenizer.hasMoreTokens()) {
value.set(tokenizer.nextToken());
context.write(value, new IntWritable(1));
}
}
```

```
}
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 x: values)
{
sum+=x.get();
}
context.write(key, new IntWritable(sum));
}
}

public static void main(String[] args) throws Exception {

Configuration conf= new Configuration();
Job job = new Job(conf,"My Word Count Program");
job.setJarByClass(WordCount.class);
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
Path outputPath = new Path(args[1]);
//Configuring the input/output path from the filesystem into the job
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
//deleting the output path automatically from hdfs so that we don't have to delete it
explicitly
```

```
outputPath.getFileSystem(conf).delete(outputPath);
//exiting the job only if the flag value becomes false
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
}
```

MINMAX TEMPERATURE

```
import java.io.IOException;
import java.util.StringTokenizer;
import java.text.DecimalFormat;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.output.MultipleOutputs;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class MinMaxTemperature {
public static String calOutputName = "California";
public static String nyOutputName = "Newyork";
public static String njOutputName = "Newjersy";
public static String ausOutputName = "Austin";
```

```
public static String bosOutputName = "Boston";
public static String balOutputName = "Baltimore";

public static class WhetherForcastMapper extends Mapper<Object, Text, Text, Text> {
 public void map(Object keyOffset, Text dayReport, Context con) throws IOException,
InterruptedException {
 //StringTokenizer strTokens = new StringTokenizer(dayReport.toString(), "\t");
 StringTokenizer strTokens = new StringTokenizer(dayReport.toString(),"\t");
 int counter = 0;
 Float currnetTemp = null;
 Float minTemp = Float.MAX_VALUE;
 Float maxTemp = Float.MIN_VALUE;
 String date = null;
 String currentTime = null;
 String minTempANDTime = null;
 String maxTempANDTime = null;

 while (strTokens.hasMoreElements()) {
  if (counter == 0) {
   date = strTokens.nextToken();
  } else {
   if (counter % 2 == 1) {
    currentTime = strTokens.nextToken();
   }
   else {

    currnetTemp = Float.parseFloat(strTokens.nextToken());
  /*  DecimalFormat df = new DecimalFormat();
   currnetTemp = df.parse(strTokens.nextToken()).floatValue(); */
    if (minTemp > currnetTemp) {
```

```
     minTemp = currnetTemp;
     minTempANDTime = minTemp + "AND" + currentTime;
    }
    if (maxTemp < currnetTemp) {
     maxTemp = currnetTemp;
     maxTempANDTime = maxTemp + "AND" + currentTime;
    }
   }
  }
  counter++;
 }
 // Write to context - MinTemp, MaxTemp and corresponding time
 Text temp = new Text();
 temp.set(maxTempANDTime);
 Text dateText = new Text();
 dateText.set(date);
 try {
  con.write(dateText, temp);
 } catch (Exception e) {
  e.printStackTrace();
 }

 temp.set(minTempANDTime);
 dateText.set(date);
 con.write(dateText, temp);
 }
}

public static class WhetherForcastReducer extends Reducer<Text, Text, Text, Text> {
```

```
MultipleOutputs<Text, Text> mos;

public void setup(Context context) {
 mos = new MultipleOutputs<Text, Text>(context);
}

public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
InterruptedException {
 int counter = 0;
 String reducerInputStr[] = null;
 String f1Time = "";
 String f2Time = "";
 String f1 = "", f2 = "";
 Text result = new Text();
 for (Text value : values) {

  if (counter == 0) {
   reducerInputStr = value.toString().split("AND");
   f1 = reducerInputStr[0];
   f1Time = reducerInputStr[1];
  }

  else {
   reducerInputStr = value.toString().split("AND");
   f2 = reducerInputStr[0];
   f2Time = reducerInputStr[1];
  }

  counter = counter + 1;
 }
```

```
 if (Float.parseFloat(f1) > Float.parseFloat(f2)) {


  result = new Text("Time: " + f2Time + " MinTemp: " + f2 + "\t" + "Time: " + f1Time + "
MaxTemp: " + f1);
 } else {
 result = new Text("Time: " + f1Time + " MinTemp: " + f1 + "\t" + "Time: " + f2Time + "
MaxTemp: " + f2);
 }
 String fileName = "";
 if (key.toString().substring(0, 2).equals("CA")) {
  fileName = MinMaxTemperature.calOutputName;
 } else if (key.toString().substring(0, 2).equals("NY")) {
  fileName = MinMaxTemperature.nyOutputName;
 } else if (key.toString().substring(0, 2).equals("NJ")) {
  fileName = MinMaxTemperature.njOutputName;
 } else if (key.toString().substring(0, 3).equals("AUS")) {
  fileName = MinMaxTemperature.ausOutputName;
 } else if (key.toString().substring(0, 3).equals("BOS")) {
  fileName = MinMaxTemperature.bosOutputName;
 } else if (key.toString().substring(0, 3).equals("BAL")) {
  fileName = MinMaxTemperature.balOutputName;
 }
 String strArr[] = key.toString().split("_");
 key.set(strArr[1]); //Key is date value
 mos.write(fileName, key, result);
}


@Override
public void cleanup(Context context) throws IOException,InterruptedException {
 mos.close();
```

```
 }
}

public static void main(String[] args) throws IOException,
 ClassNotFoundException, InterruptedException {
 Configuration conf = new Configuration();
 Job job = Job.getInstance(conf, "Wheather Statistics of USA");
 job.setJarByClass(MinMaxTemperature.class);

 job.setMapperClass(WhetherForcastMapper.class);
 job.setReducerClass(WhetherForcastReducer.class);

 job.setMapOutputKeyClass(Text.class);
 job.setMapOutputValueClass(Text.class);

 job.setOutputKeyClass(Text.class);
 job.setOutputValueClass(Text.class);

 MultipleOutputs.addNamedOutput(job, calOutputName,TextOutputFormat.class,
Text.class, Text.class);
 MultipleOutputs.addNamedOutput(job, nyOutputName,TextOutputFormat.class,
Text.class, Text.class);
 MultipleOutputs.addNamedOutput(job, njOutputName,TextOutputFormat.class, Text.class,
Text.class);
 MultipleOutputs.addNamedOutput(job, bosOutputName,TextOutputFormat.class,
Text.class, Text.class);
 MultipleOutputs.addNamedOutput(job, ausOutputName,TextOutputFormat.class,
Text.class, Text.class);
 MultipleOutputs.addNamedOutput(job, balOutputName,TextOutputFormat.class,
Text.class, Text.class);
```

```
 // FileInputFormat.addInputPath(job, new Path(args[0]));
 // FileOutputFormat.setOutputPath(job, new Path(args[1]));
Path outputPath = new Path(args[1]);
//Configuring the input/output path from the filesystem into the job
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
//deleting the output path automatically from hdfs so that we don't have to
//delete it explicitly
outputPath.getFileSystem(conf).delete(outputPath);
//exiting the job only if the flag value becomes false
 try {
  System.exit(job.waitForCompletion(true) ? 0 : 1);
 } catch (Exception e) {
  // TODO Auto-generated catch block
  e.printStackTrace();
 }
}
}
```

MATRIXMULTIPLICATION

```
import java.io.IOException;
import java.util.*;
import java.util.AbstractMap.SimpleEntry;
import java.util.Map.Entry;

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 MatrixMultiplication {

 public static class Map
 extends org.apache.hadoop.mapreduce.Mapper<LongWritable, Text, Text, Text> {
        @Override
        public void map(LongWritable key, Text value, Context context)
                        throws IOException, InterruptedException {
                Configuration conf = context.getConfiguration();
                int m = Integer.parseInt(conf.get("m"));
                int p = Integer.parseInt(conf.get("p"));
                String line = value.toString();
                // (M, i, j, Mij);
                String[] indicesAndValue = line.split(",");
                Text outputKey = new Text();
                Text outputValue = new Text();
                if (indicesAndValue[0].equals("M")) {
                        for (int k = 0; k < p; k++) {
                                outputKey.set(indicesAndValue[1] + "," + k);
                                // outputKey.set(i,k);
                                outputValue.set(indicesAndValue[0] + "," + indicesAndValue[2]
                                                + "," + indicesAndValue[3]);
                                // outputValue.set(M,j,Mij);
                                context.write(outputKey, outputValue);
```

```
                }
            } else {
                // (N, j, k, Njk);
                for (int i = 0; i < m; i++) {
                    outputKey.set(i + "," + indicesAndValue[2]);
                    outputValue.set("N," + indicesAndValue[1] + ","
                            + indicesAndValue[3]);
                    context.write(outputKey, outputValue);
                }
            }
        }
}

  public static class Reduce
  extends org.apache.hadoop.mapreduce.Reducer<Text, Text, Text, Text> {
        @Override
        public void reduce(Text key, Iterable<Text> values, Context context)
                throws IOException, InterruptedException {
            String[] value;
            //key=(i,k),
            //Values = [(M/N,j,V/W),..]
            HashMap<Integer, Float> hashA = new HashMap<Integer, Float>();
            HashMap<Integer, Float> hashB = new HashMap<Integer, Float>();
            for (Text val : values) {
                value = val.toString().split(",");
                if (value[0].equals("M")) {
                    hashA.put(Integer.parseInt(value[1]),
Float.parseFloat(value[2]));
                } else {
```

```
                                hashB.put(Integer.parseInt(value[1]),
Float.parseFloat(value[2]));
                        }
                }
                int n = Integer.parseInt(context.getConfiguration().get("n"));
                float result = 0.0f;
                float m_ij;
                float n_jk;
                for (int j = 0; j < n; j++) {
                        m_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f;
                        n_jk = hashB.containsKey(j) ? hashB.get(j) : 0.0f;
                        result += m_ij * n_jk;
                }
                if (result != 0.0f) {
                        context.write(null,
                                        new Text(key.toString() + "," + Float.toString(result)));
                }
        }
}

  public static void main(String[] args) throws Exception {
          if (args.length != 2) {
         System.err.println("Usage: MatrixMultiplication <in_dir> <out_dir>");
         System.exit(2);
      }
          Configuration conf = new Configuration();
      // M is an m-by-n matrix; N is an n-by-p matrix.
      conf.set("m", "1000");
      conf.set("n", "100");
      conf.set("p", "1000");
```

```
        @SuppressWarnings("deprecation")
                Job job = new Job(conf, "MatrixMultiplication");
        job.setJarByClass(MatrixMultiplication.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        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);
    }
}
```