1. Wordcount Program

Mapper Code: You have to copy paste this program into the WCMapper Java Class file.

// Importing libraries

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.MapReduceBase;

import org.apache.hadoop.mapred.Mapper;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reporter;

public class WCMapper extends MapReduceBase implements Mapper<LongWritable,

Text, Text, IntWritable> {

// Map function

public void map(LongWritable key, Text value, OutputCollector<Text,

IntWritable> output, Reporter rep) throws IOException

{

String line = value.toString();

// Splitting the line on spaces

for (String word : line.split(" "))

{

if (word.length() > 0)

{

output.collect(new Text(word), new IntWritable(1));

} } } }

Reducer Code: You have to copy paste this program into the WCReducer Java Class file

// Importing libraries

import java.io.IOException;

import java.util.Iterator;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reducer;

import org.apache.hadoop.mapred.Reporter;

public class WCReducer extends MapReduceBase implements Reducer<Text,

IntWritable, Text, IntWritable> {

// Reduce function

public void reduce(Text key, Iterator<IntWritable> value,

OutputCollector<Text, IntWritable> output,

Reporter rep) throws IOException

{

int count = 0;

// Counting the frequency of each words

while (value.hasNext())

{

IntWritable i = value.next();

count += i.get();

}

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

} }

Driver Code: You have to copy paste this program into the WCDriver Java Class file.

// Importing libraries

import java.io.IOException;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

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.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class WCDriver extends Configured implements Tool {

public int run(String args[]) throws IOException

{

if (args.length < 2)

{

System.out.println("Please give valid inputs");

return -1;

}

JobConf conf = new JobConf(WCDriver.class);

FileInputFormat.setInputPaths(conf, new Path(args[0]));

FileOutputFormat.setOutputPath(conf, new Path(args[1]));

conf.setMapperClass(WCMapper.class);

conf.setReducerClass(WCReducer.class);

conf.setMapOutputKeyClass(Text.class);

conf.setMapOutputValueClass(IntWritable.class);

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

JobClient.runJob(conf);

return 0;

}

// Main Method

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

{

int exitCode = ToolRunner.run(new WCDriver(), args);

System.out.println(exitCode);

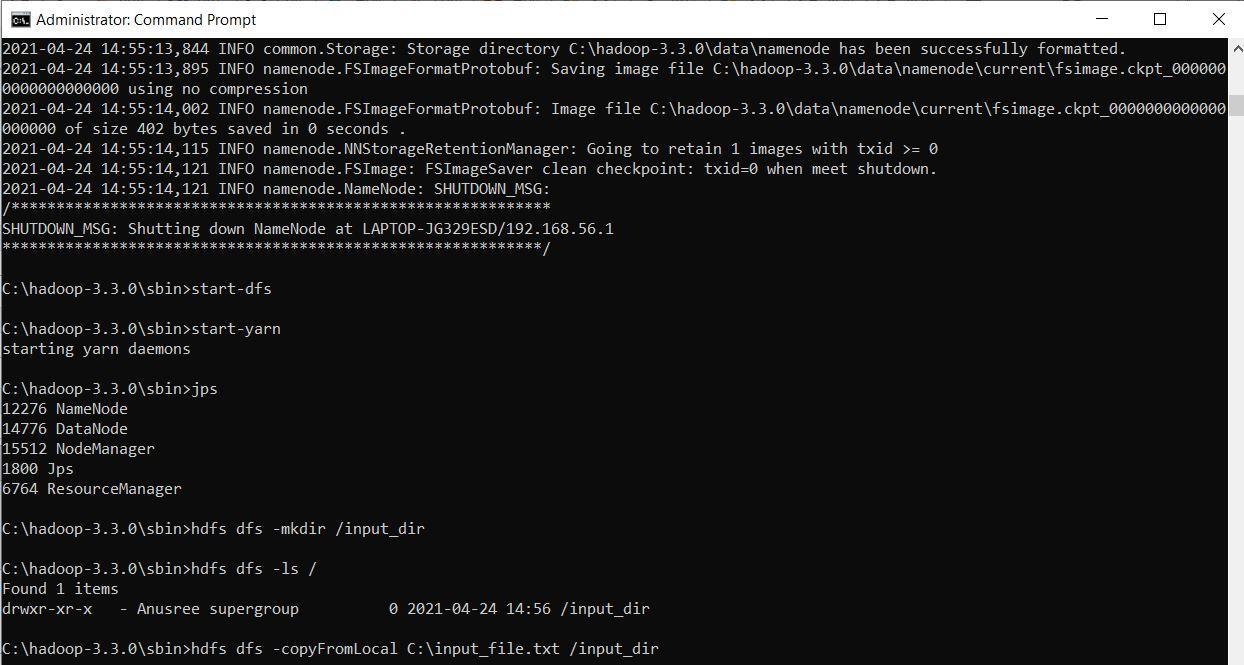
}

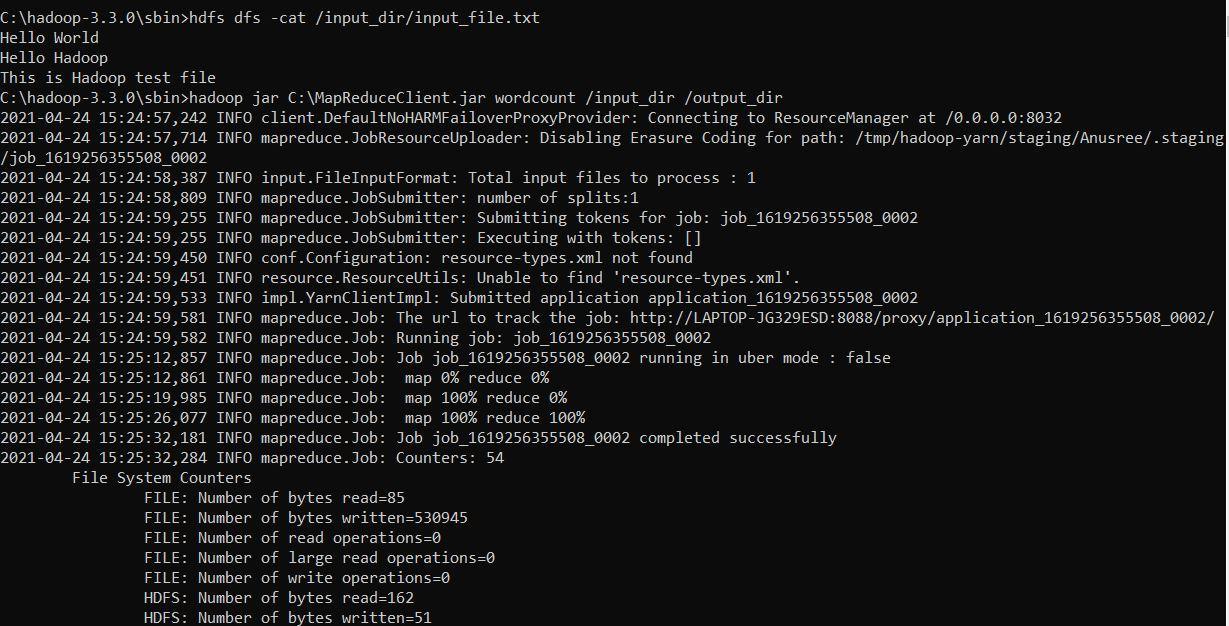
}

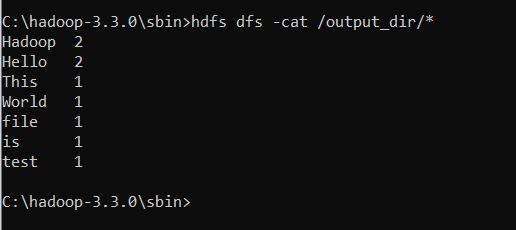
1.hadoop fs -copyFromLocal /home/hduser/Desktop/file1.txt /rgs/test.txt

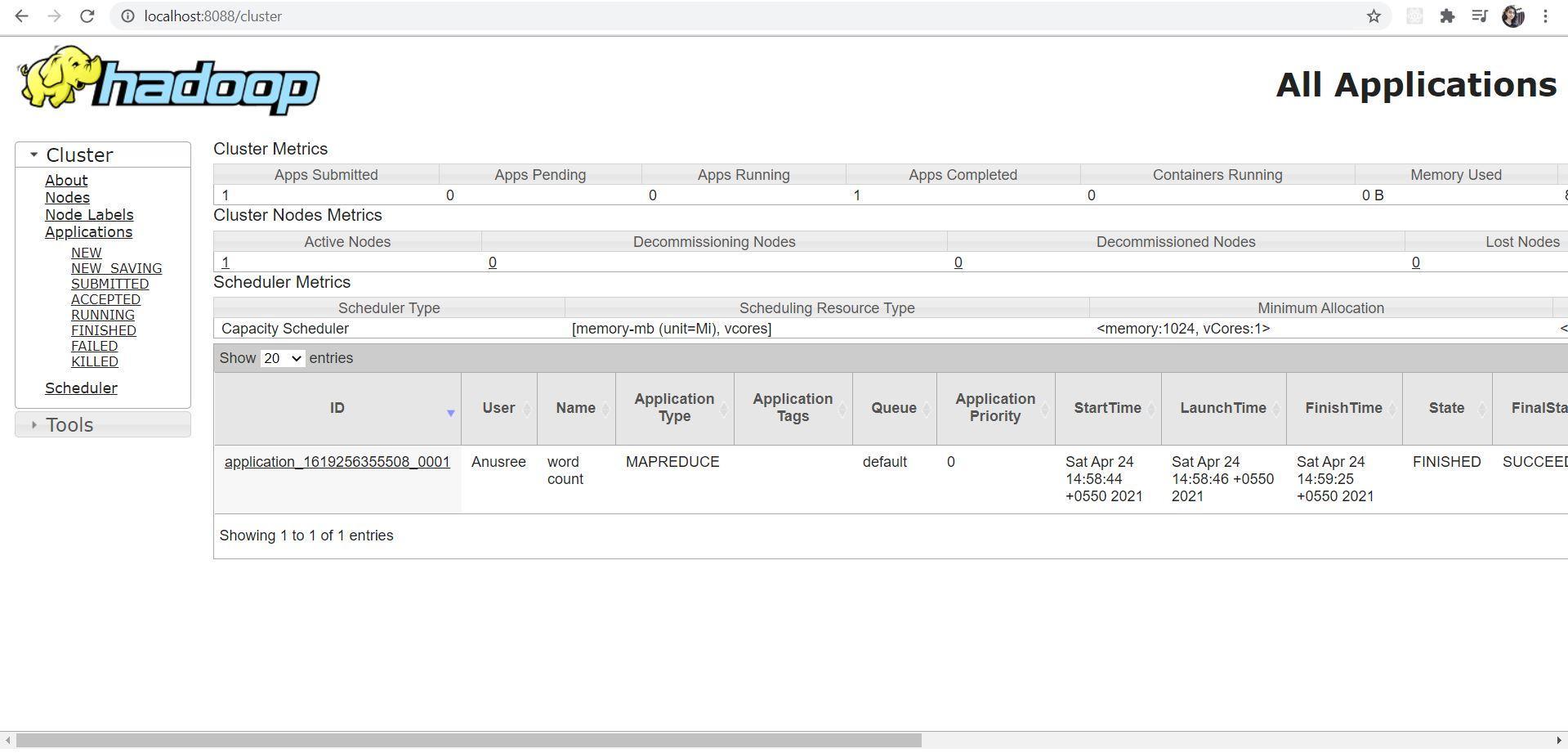
2.hadoop jar /home/hduser/Desktop/MapReduceClient.jar WCDriver /pgb/test.txt /output/

3.hdfs dfs -cat /output/\*









. For a given Text file, create a Map Reduce program to sort the content in an alphabetic order

listing only top ‘n’ maximum occurrence of words.

Driver-TopN.class

**package** samples.topn;  
  
**import** java.io.IOException;  
**import** java.util.StringTokenizer;  
**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.lib.input.FileInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
**import** org.apache.hadoop.util.GenericOptionsParser;  
  
**public** **class** TopN {  
 **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: TopN <in> <out>");  
 System.exit(2);  
 }   
 Job job = Job.getInstance(conf);  
 job.setJobName("Top N");  
 job.setJarByClass(TopN.**class**);  
 job.setMapperClass(TopNMapper.**class**);  
 job.setReducerClass(TopNReducer.**class**);  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(IntWritable.**class**);  
 FileInputFormat.addInputPath(job, **new** Path(otherArgs[0]));  
 FileOutputFormat.setOutputPath(job, **new** Path(otherArgs[1]));  
 System.exit(job.waitForCompletion(**true**) ? 0 : 1);  
 }  
   
 **public** **static** **class** TopNMapper **extends** Mapper<Object, Text, Text, IntWritable> {  
 **private** **static** **final** IntWritable one = **new** IntWritable(1);  
   
 **private** Text word = **new** Text();  
   
 **private** String tokens = "[\_|$#<>\\^=\\[\\]\\\*/\\\\,;,.\\-:()?!\"']";  
   
 **public** **void** map(Object key, Text value, Mapper<Object, Text, Text, IntWritable>.Context context) **throws** IOException, InterruptedException {  
 String cleanLine = value.toString().toLowerCase().replaceAll(**this**.tokens, " ");  
 StringTokenizer itr = **new** StringTokenizer(cleanLine);  
 **while** (itr.hasMoreTokens()) {  
 **this**.word.set(itr.nextToken().trim());  
 context.write(**this**.word, one);  
 }   
 }  
 }  
}

TopNCombiner.class

**package** samples.topn;  
  
**import** java.io.IOException;  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**public** **class** TopNCombiner **extends** Reducer<Text, IntWritable, Text, IntWritable> {  
 **public** **void** reduce(Text key, Iterable<IntWritable> values, Reducer<Text, IntWritable, Text, IntWritable>.Context context) **throws** IOException, InterruptedException {  
 **int** sum = 0;  
 **for** (IntWritable val : values)  
 sum += val.get();   
 context.write(key, **new** IntWritable(sum));  
 }  
}

TopNMapper.class

**package** samples.topn;  
  
**import** java.io.IOException;  
**import** java.util.StringTokenizer;  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Mapper;  
  
**public** **class** TopNMapper **extends** Mapper<Object, Text, Text, IntWritable> {  
 **private** **static** **final** IntWritable one = **new** IntWritable(1);  
   
 **private** Text word = **new** Text();  
   
 **private** String tokens = "[\_|$#<>\\^=\\[\\]\\\*/\\\\,;,.\\-:()?!\"']";  
   
 **public** **void** map(Object key, Text value, Mapper<Object, Text, Text, IntWritable>.Context context) **throws** IOException, InterruptedException {  
 String cleanLine = value.toString().toLowerCase().replaceAll(**this**.tokens, " ");  
 StringTokenizer itr = **new** StringTokenizer(cleanLine);  
 **while** (itr.hasMoreTokens()) {  
 **this**.word.set(itr.nextToken().trim());  
 context.write(**this**.word, one);  
 }   
 }  
}

TopNReducer.class

**package** samples.topn;  
  
**import** java.io.IOException;  
**import** java.util.HashMap;  
**import** java.util.Map;  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
**import** utils.MiscUtils;  
  
**public** **class** TopNReducer **extends** Reducer<Text, IntWritable, Text, IntWritable> {  
 **private** Map<Text, IntWritable> countMap = **new** HashMap<>();  
   
 **public** **void** reduce(Text key, Iterable<IntWritable> values, Reducer<Text, IntWritable, Text, IntWritable>.Context context) **throws** IOException, InterruptedException {  
 **int** sum = 0;  
 **for** (IntWritable val : values)  
 sum += val.get();   
 **this**.countMap.put(**new** Text(key), **new** IntWritable(sum));  
 }  
   
 **protected** **void** cleanup(Reducer<Text, IntWritable, Text, IntWritable>.Context context) **throws** IOException, InterruptedException {  
 Map<Text, IntWritable> sortedMap = MiscUtils.sortByValues(**this**.countMap);  
 **int** counter = 0;  
 **for** (Text key : sortedMap.keySet()) {  
 **if** (counter++ == 20)  
 **break**;   
 context.write(key, sortedMap.get(key));  
 }   
 }  
}

miscutils.java

package utils;

import java.util.\*;

public class MiscUtils {

/\*\*

\* sorts the map by values. Taken from:

\* <http://javarevisited.blogspot.it/2012/12/how-to-sort-hashmap-java-by-key-and-value.html>

\*/

public static <K extends Comparable, V extends Comparable> Map<K, V> sortByValues(Map<K, V> map) {

List<Map.Entry<K, V>> entries = new LinkedList<Map.Entry<K, V>>(map.entrySet());

Collections.sort(entries, new Comparator<Map.Entry<K, V>>() {

@Override

public int compare(Map.Entry<K, V> o1, Map.Entry<K, V> o2) {

return o2.getValue().compareTo(o1.getValue());

}

});

//LinkedHashMap will keep the keys in the order they are inserted

//which is currently sorted on natural ordering

Map<K, V> sortedMap = new LinkedHashMap<K, V>();

for (Map.Entry<K, V> entry : entries) {

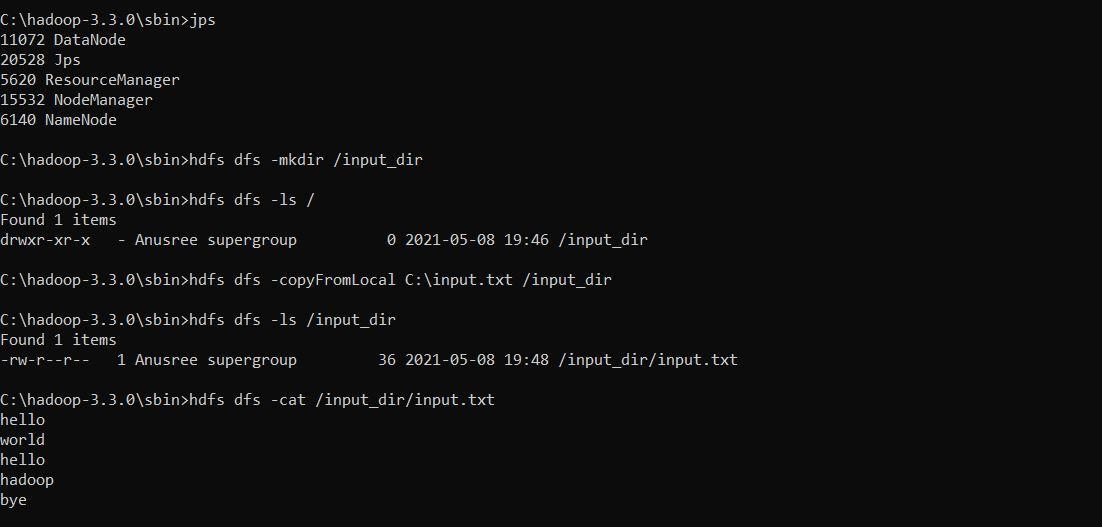
sortedMap.put(entry.getKey(), entry.getValue());

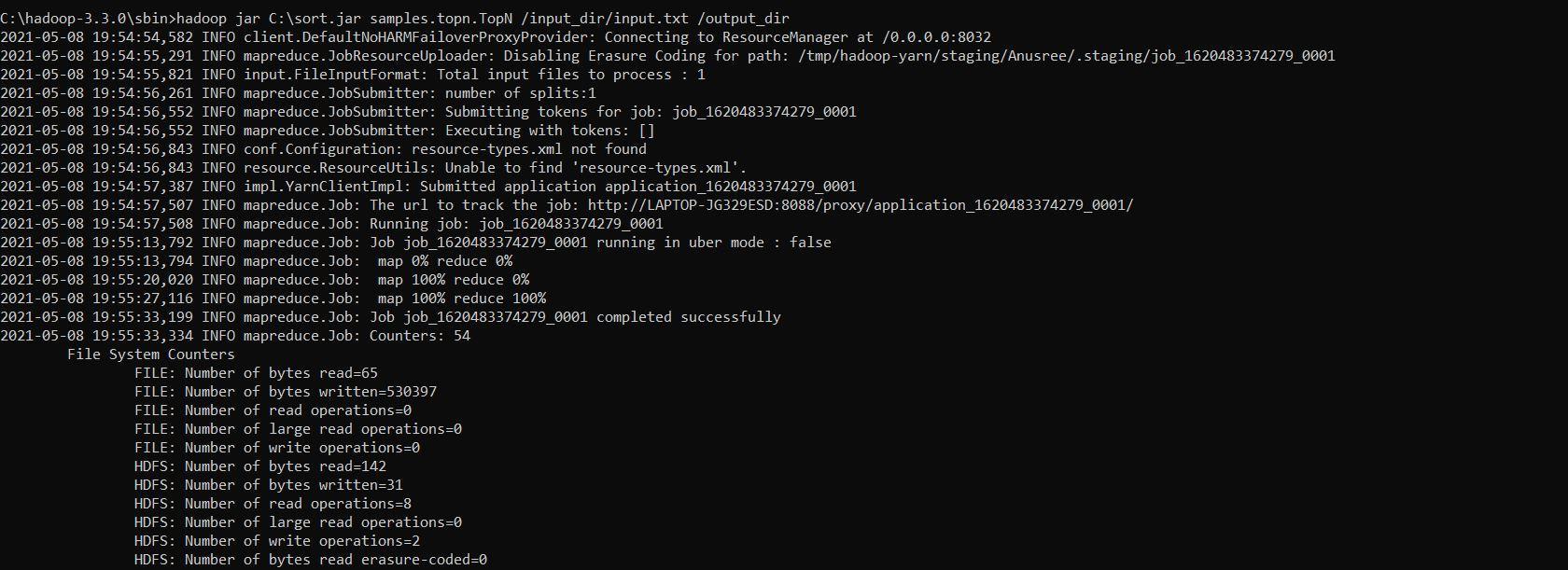
}

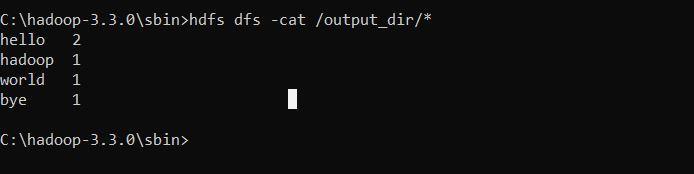
return sortedMap;

}

}







3. From the following link extract the weather data  
  
<https://github.com/tomwhite/hadoop-book/tree/master/input/ncdc/all>

Create a Map Reduce program to  
  
a) find average temperature for each year from NCDC data set.

AverageDriver

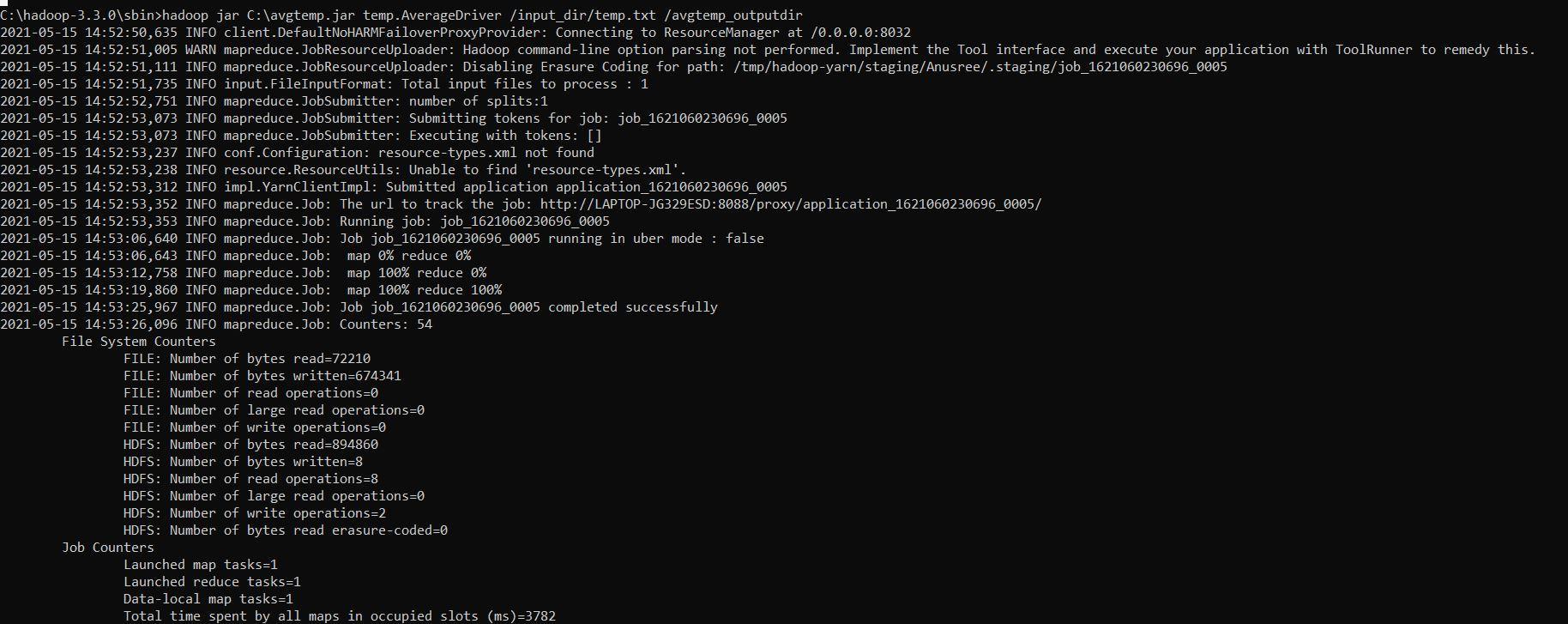
**package** temp;  
  
**import** org.apache.hadoop.fs.Path;  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Job;  
**import** org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
  
**public** **class** AverageDriver {  
 **public** **static** **void** main(String[] args) **throws** Exception {  
 **if** (args.length != 2) {  
 System.err.println("Please Enter the input and output parameters");  
 System.exit(-1);  
 }   
 Job job = **new** Job();  
 job.setJarByClass(AverageDriver.**class**);  
 job.setJobName("Max temperature");  
 FileInputFormat.addInputPath(job, **new** Path(args[0]));  
 FileOutputFormat.setOutputPath(job, **new** Path(args[1]));  
 job.setMapperClass(AverageMapper.**class**);  
 job.setReducerClass(AverageReducer.**class**);  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(IntWritable.**class**);  
 System.exit(job.waitForCompletion(**true**) ? 0 : 1);  
 }  
}

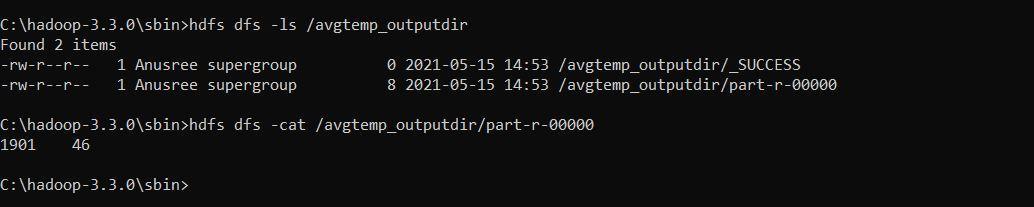
AverageMapper

**package** temp;  
  
**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;  
  
**public** **class** AverageMapper **extends** Mapper<LongWritable, Text, Text, IntWritable> {  
 **public** **static** **final** **int** MISSING = 9999;  
   
 **public** **void** map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, IntWritable>.Context context) **throws** IOException, InterruptedException {  
 **int** temperature;  
 String line = value.toString();  
 String year = line.substring(15, 19);  
 **if** (line.charAt(87) == '+') {  
 temperature = Integer.parseInt(line.substring(88, 92));  
 } **else** {  
 temperature = Integer.parseInt(line.substring(87, 92));  
 }   
 String quality = line.substring(92, 93);  
 **if** (temperature != 9999 && quality.matches("[01459]"))  
 context.write(**new** Text(year), **new** IntWritable(temperature));   
 }  
}

AverageReducer

**package** temp;  
  
**import** java.io.IOException;  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**public** **class** AverageReducer **extends** Reducer<Text, IntWritable, Text, IntWritable> {  
 **public** **void** reduce(Text key, Iterable<IntWritable> values, Reducer<Text, IntWritable, Text, IntWritable>.Context context) **throws** IOException, InterruptedException {  
 **int** max\_temp = 0;  
 **int** count = 0;  
 **for** (IntWritable value : values) {  
 max\_temp += value.get();  
 count++;  
 }   
 context.write(key, **new** IntWritable(max\_temp / count));  
 }  
}





b) find the mean max temperature for every month

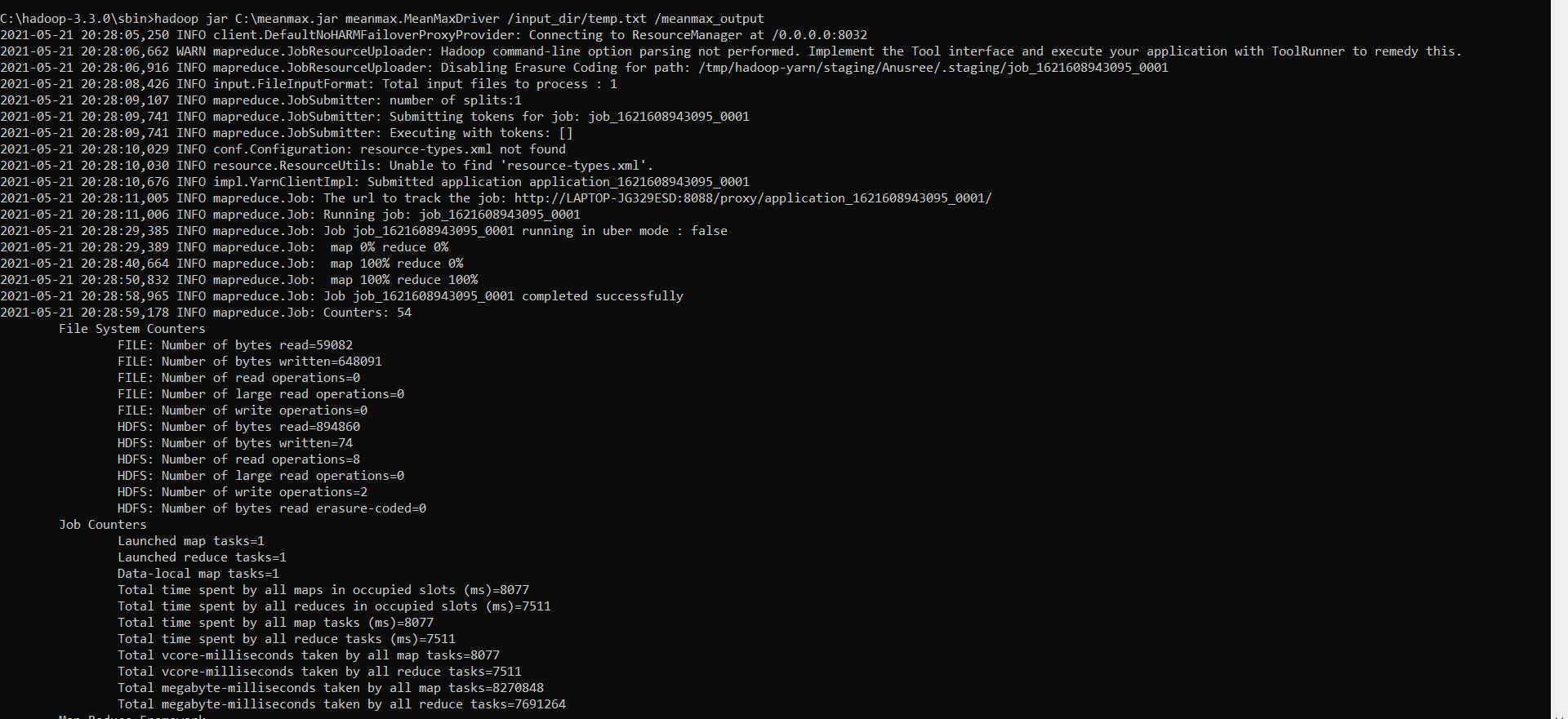
MeanMax

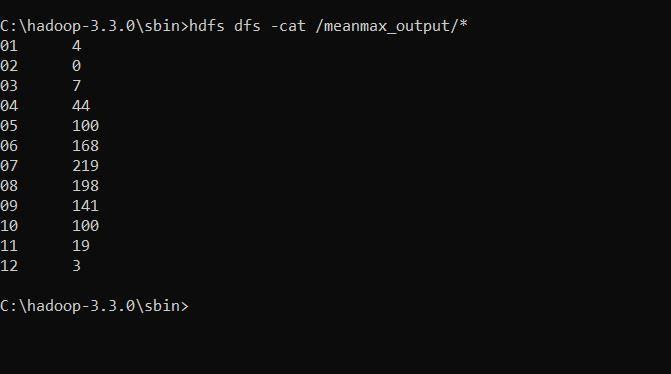
MeanMaxDriver.class

**package** meanmax;  
  
**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;  
  
**public** **class** MeanMaxDriver {  
 **public** **static** **void** main(String[] args) **throws** Exception {  
 **if** (args.length != 2) {  
 System.err.println("Please Enter the input and output parameters");  
 System.exit(-1);  
 }   
 Job job = **new** Job();  
 job.setJarByClass(MeanMaxDriver.**class**);  
 job.setJobName("Max temperature");  
 FileInputFormat.addInputPath(job, **new** Path(args[0]));  
 FileOutputFormat.setOutputPath(job, **new** Path(args[1]));  
 job.setMapperClass(MeanMaxMapper.**class**);  
 job.setReducerClass(MeanMaxReducer.**class**);  
 job.setOutputKeyClass(Text.**class**);  
 job.setOutputValueClass(IntWritable.**class**);  
 System.exit(job.waitForCompletion(**true**) ? 0 : 1);  
 }  
}  
  
MeanMaxMapper.class

**package** meanmax;  
  
**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;  
  
**public** **class** MeanMaxMapper **extends** Mapper<LongWritable, Text, Text, IntWritable> {  
 **public** **static** **final** **int** MISSING = 9999;  
   
 **public** **void** map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, IntWritable>.Context context) **throws** IOException, InterruptedException {  
 **int** temperature;  
 String line = value.toString();  
 String month = line.substring(19, 21);  
 **if** (line.charAt(87) == '+') {  
 temperature = Integer.parseInt(line.substring(88, 92));  
 } **else** {  
 temperature = Integer.parseInt(line.substring(87, 92));  
 }   
 String quality = line.substring(92, 93);  
 **if** (temperature != 9999 && quality.matches("[01459]"))  
 context.write(**new** Text(month), **new** IntWritable(temperature));   
 }  
}  
  
MeanMaxReducer.class

**package** meanmax;  
  
**import** java.io.IOException;  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**public** **class** MeanMaxReducer **extends** Reducer<Text, IntWritable, Text, IntWritable> {  
 **public** **void** reduce(Text key, Iterable<IntWritable> values, Reducer<Text, IntWritable, Text, IntWritable>.Context context) **throws** IOException, InterruptedException {  
 **int** max\_temp = 0;  
 **int** total\_temp = 0;  
 **int** count = 0;  
 **int** days = 0;  
 **for** (IntWritable value : values) {  
 **int** temp = value.get();  
 **if** (temp > max\_temp)  
 max\_temp = temp;   
 count++;  
 **if** (count == 3) {  
 total\_temp += max\_temp;  
 max\_temp = 0;  
 count = 0;  
 days++;  
 }   
 }   
 context.write(key, **new** IntWritable(total\_temp / days));  
 }  
}





4. Create a Hadoop Map Reduce program to combine information from the users file along with

Information from the posts file by using the concept of join and display user\_id, Reputation and

Score.

Create a Hadoop Map Reduce program to combine information from the users file along with

Information from the posts file by using the concept of join and display user\_id, Reputation and

Score.

// JoinDriver.java

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.\*;

import org.apache.hadoop.mapred.lib.MultipleInputs;

import org.apache.hadoop.util.\*;

public class JoinDriver extends Configured implements Tool {

public static class KeyPartitioner implements Partitioner<TextPair, Text> {

@Override

public void configure(JobConf job) {}

@Override

public int getPartition(TextPair key, Text value, int numPartitions) {

return (key.getFirst().hashCode() & Integer.MAX\_VALUE) %

numPartitions;

}

}

@Override

public int run(String[] args) throws Exception {

if (args.length != 3) {

System.out.println("Usage: <Department Emp Strength input>

<Department Name input> <output>");

return -1;

}

JobConf conf = new JobConf(getConf(), getClass());

conf.setJobName("Join 'Department Emp Strength input' with 'Department Name

input'");

Path AInputPath = new Path(args[0]);

Path BInputPath = new Path(args[1]);

Path outputPath = new Path(args[2]);

MultipleInputs.addInputPath(conf, AInputPath, TextInputFormat.class,

Posts.class);

MultipleInputs.addInputPath(conf, BInputPath, TextInputFormat.class,

User.class);

FileOutputFormat.setOutputPath(conf, outputPath);

conf.setPartitionerClass(KeyPartitioner.class);

conf.setOutputValueGroupingComparator(TextPair.FirstComparator.class);

conf.setMapOutputKeyClass(TextPair.class);

conf.setReducerClass(JoinReducer.class);

conf.setOutputKeyClass(Text.class);

JobClient.runJob(conf);

return 0;

}

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

int exitCode = ToolRunner.run(new JoinDriver(), args);

System.exit(exitCode);

}

}

// JoinReducer.java

import java.io.IOException;

import java.util.Iterator;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.\*;

public class JoinReducer extends MapReduceBase implements Reducer<TextPair, Text, Text,

Text> {

@Override

public void reduce (TextPair key, Iterator<Text> values, OutputCollector<Text, Text>

output, Reporter reporter)

throws IOException

{

Text nodeId = new Text(values.next());

while (values.hasNext()) {

Text node = values.next();

Text outValue = new Text(nodeId.toString() + "\t\t" + node.toString());

output.collect(key.getFirst(), outValue);

}

}

}

// User.java

import java.io.IOException;

import java.util.Iterator;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FSDataInputStream;

import org.apache.hadoop.fs.FSDataOutputStream;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.\*;

import org.apache.hadoop.io.IntWritable;

public class User extends MapReduceBase implements Mapper<LongWritable, Text, TextPair,

Text> {

@Override

public void map(LongWritable key, Text value, OutputCollector<TextPair, Text> output,

Reporter reporter)

throws IOException

{

String valueString = value.toString();

String[] SingleNodeData = valueString.split("\t");

output.collect(new TextPair(SingleNodeData[0], "1"), new

Text(SingleNodeData[1]));

}

}

//Posts.java

import java.io.IOException;

import org.apache.hadoop.io.\*;

import org.apache.hadoop.mapred.\*;

public class Posts extends MapReduceBase implements Mapper<LongWritable, Text, TextPair,

Text> {

@Override

public void map(LongWritable key, Text value, OutputCollector<TextPair, Text> output,

Reporter reporter)

throws IOException

{

String valueString = value.toString();

String[] SingleNodeData = valueString.split("\t");

output.collect(new TextPair(SingleNodeData[3], "0"), new

Text(SingleNodeData[9]));

}

}

// TextPair.java

import java.io.\*;

import org.apache.hadoop.io.\*;

public class TextPair implements WritableComparable<TextPair> {

private Text first;

private Text second;

public TextPair() {

set(new Text(), new Text());

}

public TextPair(String first, String second) {

set(new Text(first), new Text(second));

}

public TextPair(Text first, Text second) {

set(first, second);

}

public void set(Text first, Text second) {

this.first = first;

this.second = second;

}

public Text getFirst() {

return first;

}

public Text getSecond() {

return second;

}

@Override

public void write(DataOutput out) throws IOException {

first.write(out);

second.write(out);

}

@Override

public void readFields(DataInput in) throws IOException {

first.readFields(in);

second.readFields(in);

}

@Override

public int hashCode() {

return first.hashCode() \* 163 + second.hashCode();

}

@Override

public boolean equals(Object o) {

if (o instanceof TextPair) {

TextPair tp = (TextPair) o;

return first.equals(tp.first) && second.equals(tp.second);

}

return false;

}

@Override

public String toString() {

return first + "\t" + second;

}

@Override

public int compareTo(TextPair tp) {

int cmp = first.compareTo(tp.first);

if (cmp != 0) {

return cmp;

}

return second.compareTo(tp.second);

}

// ^^ TextPair

// vv TextPairComparator

public static class Comparator extends WritableComparator {

private static final Text.Comparator TEXT\_COMPARATOR = new Text.Comparator();

public Comparator() {

super(TextPair.class);

}

@Override

public int compare(byte[] b1, int s1, int l1,

byte[] b2, int s2, int l2) {

try {

int firstL1 = WritableUtils.decodeVIntSize(b1[s1]) + readVInt(b1, s1);

int firstL2 = WritableUtils.decodeVIntSize(b2[s2]) + readVInt(b2, s2);

int cmp = TEXT\_COMPARATOR.compare(b1, s1, firstL1, b2, s2, firstL2);

if (cmp != 0) {

return cmp;

}

return TEXT\_COMPARATOR.compare(b1, s1 + firstL1, l1 - firstL1,

b2, s2 + firstL2, l2 - firstL2);

} catch (IOException e) {

throw new IllegalArgumentException(e);

}

}

}

static {

WritableComparator.define(TextPair.class, new Comparator());

}

public static class FirstComparator extends WritableComparator {

private static final Text.Comparator TEXT\_COMPARATOR = new Text.Comparator();

public FirstComparator() {

super(TextPair.class);

}

@Override

public int compare(byte[] b1, int s1, int l1,

byte[] b2, int s2, int l2) {

try {

int firstL1 = WritableUtils.decodeVIntSize(b1[s1]) + readVInt(b1, s1);

int firstL2 = WritableUtils.decodeVIntSize(b2[s2]) + readVInt(b2, s2);

return TEXT\_COMPARATOR.compare(b1, s1, firstL1, b2, s2, firstL2);

} catch (IOException e) {

throw new IllegalArgumentException(e);

}

}

@Override

public int compare(WritableComparable a, WritableComparable b) {

if (a instanceof TextPair && b instanceof TextPair) {

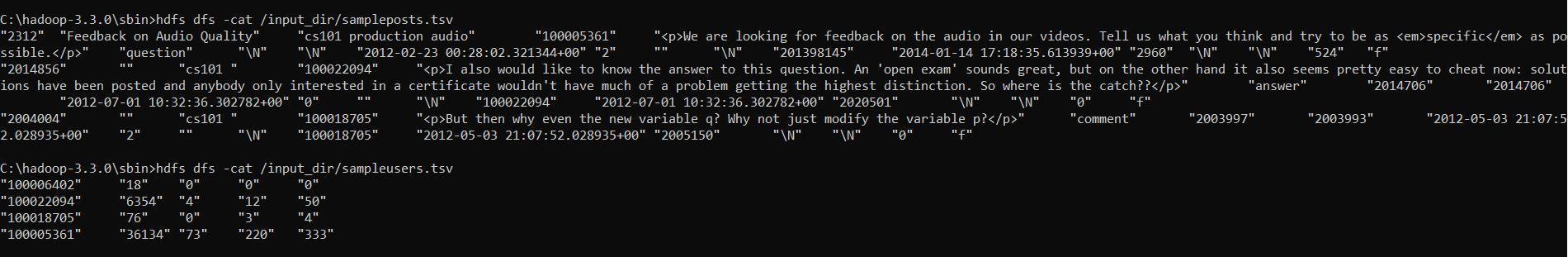
return ((TextPair) a).first.compareTo(((TextPair) b).first);

}

return super.compare(a, b);

}

} }



LAB8/Department\_Employee\_join\_example/DeptName.txt

