**Problem Statement**

We have a dataset of sales of different TV sets across different locations.

Records look like:

Samsung|Optima|14|Madhya Pradesh|132401|14200

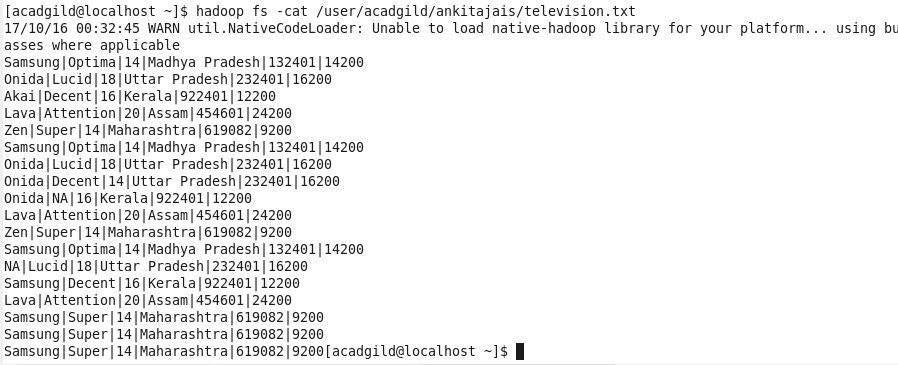
The fields are arranged like:

Company Name|Product Name|Size in inches|State|Pin Code|Price

There are some invalid records which contain 'NA' in either Company Name or Product Name.

**Solution:**

**Television.txt input**

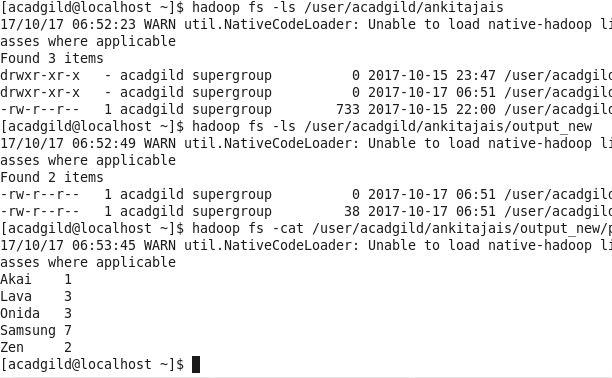


**2. Write a Map Reduce program to calculate the total units sold for each Company.**

**Solution:**

|  |
| --- |
| Package  com.MRtotalunitsoldCompany; |
|  |  |
|  | import java.io.IOException; |
|  | import java.util.\*; |
|  | import org.apache.hadoop.fs.Path; |
|  | import org.apache.hadoop.conf.\*; |
|  | import org.apache.hadoop.io.IntWritable; |
|  | import org.apache.hadoop.io.LongWritable; |
|  | import org.apache.hadoop.io.Text; |
|  | import org.apache.hadoop.mapreduce.\*; |
|  | import org.apache.hadoop.mapreduce.Job; |
|  | 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 MRtotalSoldCompany |
|  | { |
|  | public static class SoldCompanyMap 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 |
|  | { |
|  | //loop through each line of the television text file |
|  | String[] linesArray = value.toString().split(" "); |
|  | //print each line of the file to output -- for debugging |
|  | /\* for(String s: linesArray) |
|  | \* { |
|  | System.out.println(s); |
|  | } |
|  | \*/ |
|  | //loop through each column in the line to find the invalid company and product names |
|  | for(String line : linesArray) |
|  | { |
|  | String[] lineArray = line.split("\\|"); |
|  | /\*for(String s: lineArray){ |
|  | System.out.println(s);} -- for debug in purpose \*/ |
|  | Text company = new Text(lineArray[0]); |
|  | Text product = new Text(lineArray[1]); |
|  | // Remove lines which have company or product as "NA" |
|  | if(!company.equals(new Text("NA"))) |
|  | { |
|  | if(!product.equals(new Text("NA"))) |
|  | { |
|  | context.write(company,one); |
|  |  |
|  | } |
|  | } |
|  | } |
|  | /\* String[] lineArray = value.toString().split("|"); |
|  | StringTokenizer company = new StringTokenizer(lineArray[0]); |
|  | while(company.hasMoreTokens()) |
|  | { |
|  | value.set(company.nextToken()); |
|  | context.write(value, new IntWritable(1)); |
|  | }\*/ |
|  |  |
|  | } |
|  | } |
|  | public static class SoldCompanyReduce extends Reducer<Text, IntWritable, Text, IntWritable> |
|  | { |
|  |  |
|  | public void reduce(Text key,Iterable<IntWritable> values, Context context)throws IOException, InterruptedException |
|  | { |
|  |  |
|  | int sum = 0; |
|  | //sum total units sold for each company |
|  | for (IntWritable x : values) |
|  | { |
|  | sum += x.get(); |
|  |  |
|  | } |
|  |  |
|  |  |
|  | context.write(key,new IntWritable(sum)); |
|  |  |
|  |  |
|  |  |
|  | } |
|  |  |
|  | } |
|  |  |
|  | @SuppressWarnings("deprecation") |
|  | public static void main(String[] args)throws Exception |
|  | { |
|  | // TODO Auto-generated method stub |
|  | Configuration conf = new Configuration(); |
|  | Job job = Job.getInstance(conf, "TotalSoldPriceCompany"); |
|  | job.setJarByClass(MRtotalSoldCompany.class); |
|  | job.setMapperClass(SoldCompanyMap.class); |
|  | job.setReducerClass(SoldCompanyReduce.class); |
|  | job.setMapOutputKeyClass(Text.class); |
|  | job.setMapOutputValueClass(IntWritable.class); |
|  |  |
|  | job.setOutputKeyClass(Text.class); |
|  | job.setOutputValueClass(IntWritable.class); |
|  |  |
|  |  |
|  | job.setInputFormatClass(TextInputFormat.class); |
|  | job.setOutputFormatClass(TextOutputFormat.class); |
|  |  |
|  | FileInputFormat.addInputPath(job, new Path(args[0])); |
|  | FileOutputFormat.setOutputPath(job,new Path(args[1])); |
|  |  |
|  |  |
|  | Path out=new Path(args[1]); |
|  | out.getFileSystem(conf).delete(out); |
|  |  |
|  |  |
|  | System.exit(job.waitForCompletion(true) ? 0:1); |
|  | } |
|  |  |
|  | } |

**Output:**

****

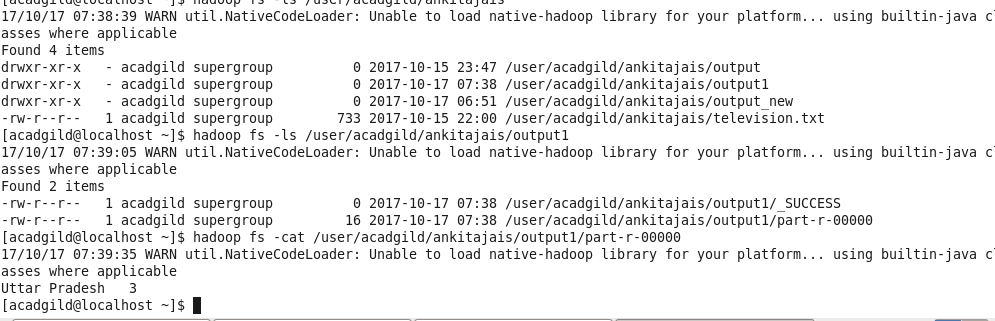
**3. Write a Map Reduce program to calculate the total units sold in each state for Onida**

**company.**

**Solution:**

|  |
| --- |
| package com.MRtotalunitsoldState; |
|  |  |
|  | import java.io.IOException; |
|  | import org.apache.hadoop.fs.Path; |
|  | import org.apache.hadoop.conf.\*; |
|  | import org.apache.hadoop.io.IntWritable; |
|  | import org.apache.hadoop.io.LongWritable; |
|  | import org.apache.hadoop.io.Text; |
|  | 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 MRtotalSoldState |
|  | { |
|  | public static class SoldStateMap 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 |
|  | { |
|  | //loop through each line of the television text file |
|  | String[] linesArray = value.toString().split(" "); |
|  |  |
|  | //print each line of the file to output -- for debugging |
|  | /\* for(String s: linesArray){ |
|  | System.out.println(s);}\*/ |
|  | //loop through each column in the line to find the invalid company and product names |
|  | for(String line : linesArray) |
|  | { |
|  | String[] lineArray = line.split("\\|"); |
|  | /\*for(String s: lineArray){ |
|  | System.out.println(s);} -- for debugg in purpose \*/ |
|  | Text company = new Text(lineArray[0]); |
|  | Text product = new Text(lineArray[1]); |
|  | Text state = new Text(lineArray[3]); |
|  | // Remove lines which have company or product as "NA" |
|  | if(!company.equals(new Text("NA"))) |
|  | { |
|  | if(!product.equals(new Text("NA"))) |
|  | { |
|  | if(company.equals(new Text("Onida"))) |
|  | { |
|  | context.write(state, one); |
|  | } |
|  | } |
|  | } |
|  | } |
|  | } |
|  | } |
|  | public static class SoldStateReduce extends Reducer<Text, IntWritable, Text, IntWritable> |
|  | { |
|  |  |
|  | public void reduce(Text key,Iterable<IntWritable> values, Context context)throws IOException, InterruptedException |
|  | { |
|  |  |
|  | int sum = 0; |
|  | //sum total units sold for each company |
|  | for (IntWritable x : values) |
|  | { |
|  | sum += x.get(); |
|  |  |
|  | } |
|  |  |
|  |  |
|  | context.write(key,new IntWritable(sum)); |
|  |  |
|  |  |
|  |  |
|  | } |
|  |  |
|  | } |
|  |  |
|  | @SuppressWarnings("deprecation") |
|  | public static void main(String[] args)throws Exception |
|  | { |
|  | // TODO Auto-generated method stub |
|  | Configuration conf = new Configuration(); |
|  | Job job = Job.getInstance(conf, "TotalSoldUnitsStateCompany"); |
|  | job.setJarByClass(MRtotalSoldState.class); |
|  | job.setMapperClass(SoldStateMap.class); |
|  | job.setReducerClass(SoldStateReduce.class); |
|  | job.setMapOutputKeyClass(Text.class); |
|  | job.setMapOutputValueClass(IntWritable.class); |
|  |  |
|  | job.setOutputKeyClass(Text.class); |
|  | job.setOutputValueClass(IntWritable.class); |
|  |  |
|  |  |
|  | job.setInputFormatClass(TextInputFormat.class); |
|  | job.setOutputFormatClass(TextOutputFormat.class); |
|  |  |
|  | FileInputFormat.addInputPath(job, new Path(args[0])); |
|  | FileOutputFormat.setOutputPath(job,new Path(args[1])); |
|  |  |
|  |  |
|  | Path out=new Path(args[1]); |
|  | out.getFileSystem(conf).delete(out); |
|  |  |
|  |  |
|  | System.exit(job.waitForCompletion(true) ? 0:1); |
|  | } |
|  |  |
|  |  |
|  | } |

**Output:**

****