**Mahesh Reddy**

**Data Mining Assignment -2**

1. Show the closed cuboids for the following dimensions for the automobiles dataset attached along with this assignment: (make), (make, type), (make, type, year), and (make, type, year, color). The total price is the measure to be accumulated. A closed cube contains only the closed cells in all its cuboids. See page 161 of the chapter on data cubes available on Blackboard site.

**Answer:**

**4-D cuboid (make, type, year, color):**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [**price**](http://localhost/phpMyAdmin/sql.php?db=csv_db&table=cubetestdata&sql_query=select+sum%28price%29%2Cmake%2Ctype%2Cyear+from+cubetestdata+group+by+make%2Ctype%2Cyear%0AORDER+BY+sum%28price%29+ASC&session_max_rows=25&is_browse_distinct=0&token=a865f2fba33a48a51fb5378bfcb1ab57) | [**make**](http://localhost/phpMyAdmin/sql.php?db=csv_db&table=cubetestdata&sql_query=select+sum%28price%29%2Cmake%2Ctype%2Cyear+from+cubetestdata+group+by+make%2Ctype%2Cyear%0AORDER+BY+%60cubetestdata%60.%60make%60+ASC&session_max_rows=25&is_browse_distinct=0&token=a865f2fba33a48a51fb5378bfcb1ab57) | [**type**](http://localhost/phpMyAdmin/sql.php?db=csv_db&table=cubetestdata&sql_query=select+sum%28price%29%2Cmake%2Ctype%2Cyear+from+cubetestdata+group+by+make%2Ctype%2Cyear%0AORDER+BY+%60cubetestdata%60.%60type%60+ASC&session_max_rows=25&is_browse_distinct=0&token=a865f2fba33a48a51fb5378bfcb1ab57) | [**year**](http://localhost/phpMyAdmin/sql.php?db=csv_db&table=cubetestdata&sql_query=select+sum%28price%29%2Cmake%2Ctype%2Cyear+from+cubetestdata+group+by+make%2Ctype%2Cyear%0AORDER+BY+%60cubetestdata%60.%60year%60+ASC&session_max_rows=25&is_browse_distinct=0&token=a865f2fba33a48a51fb5378bfcb1ab57) | **color** |
| 50000 | Ford | Truck | 2008 | blue |
| 29000 | Ford | Truck | 2009 | black |
| 43000 | Ford | Van | 2008 | red |
| 24000 | Ford | Van | 2009 | black |
| 24000 | Ford | Van | 2011 | black |
| 42000 | GM | Car | 2012 | blue |
| 22000 | GM | Car | 2012 | red |
| 24000 | GM | Car | 2012 | white |
| 22000 | GM | Truck | 2010 | red |
| 12000 | Honda | Car | 2008 | red |
| 24500 | Honda | Car | 2014 | red |
| 32000 | Honda | SUV | 2009 | red |
| 68000 | Honda | Van | 2010 | green |
| 29000 | Honda | Van | 2010 | white |
| 42000 | Honda | Van | 2011 | red |
| 41000 | Honda | Van | 2015 | black |
| 32000 | Mazda | car | 2011 | black |
| 25000 | Mazda | Car | 2012 | blue |
| 28000 | Mazda | Car | 2015 | black |
| 28000 | Toyota | Car | 2011 | black |
| 29000 | Toyota | SUV | 2009 | black |
|  |  |  |  |  |

**3-D cuboid (make, type, year):**

**Identified the closed cuboids for 3-D from the above 4-D cuboid. Closed cuboids are not highlighted, rest all are highlighted with red.**

|  |  |  |  |
| --- | --- | --- | --- |
| [**price**](http://localhost/phpMyAdmin/sql.php?db=csv_db&table=cubetestdata&sql_query=select+sum%28price%29%2Cmake%2Ctype%2Cyear+from+cubetestdata+group+by+make%2Ctype%2Cyear%0AORDER+BY+sum%28price%29+ASC&session_max_rows=25&is_browse_distinct=0&token=a865f2fba33a48a51fb5378bfcb1ab57) | [**make**](http://localhost/phpMyAdmin/sql.php?db=csv_db&table=cubetestdata&sql_query=select+sum%28price%29%2Cmake%2Ctype%2Cyear+from+cubetestdata+group+by+make%2Ctype%2Cyear%0AORDER+BY+%60cubetestdata%60.%60make%60+ASC&session_max_rows=25&is_browse_distinct=0&token=a865f2fba33a48a51fb5378bfcb1ab57) | [**type**](http://localhost/phpMyAdmin/sql.php?db=csv_db&table=cubetestdata&sql_query=select+sum%28price%29%2Cmake%2Ctype%2Cyear+from+cubetestdata+group+by+make%2Ctype%2Cyear%0AORDER+BY+%60cubetestdata%60.%60type%60+ASC&session_max_rows=25&is_browse_distinct=0&token=a865f2fba33a48a51fb5378bfcb1ab57) | [**year**](http://localhost/phpMyAdmin/sql.php?db=csv_db&table=cubetestdata&sql_query=select+sum%28price%29%2Cmake%2Ctype%2Cyear+from+cubetestdata+group+by+make%2Ctype%2Cyear%0AORDER+BY+%60cubetestdata%60.%60year%60+ASC&session_max_rows=25&is_browse_distinct=0&token=a865f2fba33a48a51fb5378bfcb1ab57) |
| 50000 | Ford | Truck | 2008 |
| 29000 | Ford | Truck | 2009 |
| 43000 | Ford | Van | 2008 |
| 24000 | Ford | Van | 2009 |
| 24000 | Ford | Van | 2011 |
| 88000 | GM | Car | 2012 |
| 22000 | GM | Truck | 2010 |
| 12000 | Honda | Car | 2008 |
| 24500 | Honda | Car | 2014 |
| 32000 | Honda | SUV | 2009 |
| 97000 | Honda | Van | 2010 |
| 42000 | Honda | Van | 2011 |
| 41000 | Honda | Van | 2015 |
| 32000 | Mazda | car | 2011 |
| 25000 | Mazda | Car | 2012 |
| 28000 | Mazda | Car | 2015 |
| 28000 | Toyota | Car | 2011 |
| 29000 | Toyota | SUV | 2009 |

**2-D cuboid (make, type):**

**Identified the closed cuboids for 2-D from the above 3-D cuboid. Closed cuboids are not highlighted, rest all are highlighted with red.**

|  |  |  |
| --- | --- | --- |
| [**price**](http://localhost/phpMyAdmin/sql.php?db=csv_db&table=cubetestdata&sql_query=select+sum%28price%29%2Cmake%2Ctype+from+cubetestdata+group+by+make%2Ctype%0AORDER+BY+sum%28price%29+ASC&session_max_rows=25&is_browse_distinct=0&token=a865f2fba33a48a51fb5378bfcb1ab57) | [**make**](http://localhost/phpMyAdmin/sql.php?db=csv_db&table=cubetestdata&sql_query=select+sum%28price%29%2Cmake%2Ctype+from+cubetestdata+group+by+make%2Ctype%0AORDER+BY+%60cubetestdata%60.%60make%60+ASC&session_max_rows=25&is_browse_distinct=0&token=a865f2fba33a48a51fb5378bfcb1ab57) | [**type**](http://localhost/phpMyAdmin/sql.php?db=csv_db&table=cubetestdata&sql_query=select+sum%28price%29%2Cmake%2Ctype+from+cubetestdata+group+by+make%2Ctype%0AORDER+BY+%60cubetestdata%60.%60type%60+ASC&session_max_rows=25&is_browse_distinct=0&token=a865f2fba33a48a51fb5378bfcb1ab57) |
| 79000 | Ford | Truck |
| 91000 | Ford | Van |
| 88000 | GM | Car |
| 22000 | GM | Truck |
| 36500 | Honda | Car |
| 32000 | Honda | SUV |
| 180000 | Honda | Van |
| 85000 | Mazda | Car |
| 28000 | Toyota | Car |
| 29000 | Toyota | SUV |

**1-D cuboid (make):**

**Identified the closed cuboids for 1-D from the above 2-D cuboid. Closed cuboids are not highlighted, rest all are highlighted with red.**

|  |  |
| --- | --- |
| [**price**](http://localhost/phpMyAdmin/sql.php?db=csv_db&table=cubetestdata&sql_query=select+sum%28price%29%2Cmake%2Ctype+from+cubetestdata+group+by+make%2Ctype%0AORDER+BY+sum%28price%29+ASC&session_max_rows=25&is_browse_distinct=0&token=a865f2fba33a48a51fb5378bfcb1ab57) | [**make**](http://localhost/phpMyAdmin/sql.php?db=csv_db&table=cubetestdata&sql_query=select+sum%28price%29%2Cmake%2Ctype+from+cubetestdata+group+by+make%2Ctype%0AORDER+BY+%60cubetestdata%60.%60make%60+ASC&session_max_rows=25&is_browse_distinct=0&token=a865f2fba33a48a51fb5378bfcb1ab57) |
| 57000 | Toyota |
| 85000 | Mazda |
| 110000 | GM |
| 170000 | Ford |
| 248500 | Honda |

1. Take the graph program examples in MapReduce paradigm from the attached zipped file. Run them to make sure that you are able to execute them on the Hadoop system using your account. Then submit your program and output for the following tasks:
   1. The input file contains the source node id, the target node id, and the amount of flow from the source to the target. Your program must output the total outflow from each source node by listing <key, value> pairs where the key is the source node id and the value is the amount of outflow from the node.

**Code:**

package org.test;

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;

import org.test.WordCount.IntSumReducer;

public class NodeDegree {

public static class Map extends Mapper<LongWritable, Text, IntWritable, IntWritable> {

private IntWritable outKey = new IntWritable();

private IntWritable outValue = new IntWritable();

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

String line = value.toString();

if (line.length() > 1) {

String[] inLine = line.split("\t");

outKey.set(Integer.parseInt(inLine[0]));

outValue.set(Integer.parseInt(inLine[2]));

context.write(outKey, outValue);

}

}

}

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

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

private IntWritable result = new IntWritable();

public void reduce(IntWritable key, Iterable<IntWritable> values, Context context)

throws IOException, InterruptedException {

int cntr = 0;

for (IntWritable val : values) {

cntr += val.get();

}

result.set(cntr);

context.write(key, result);

}

}

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

Configuration conf = new Configuration();

Job job = new Job(conf, "nodeDegree");

job.setMapOutputKeyClass(IntWritable.class);

job.setMapOutputValueClass(IntWritable.class);

job.setJarByClass(NodeDegree.class);

job.setMapperClass(Map.class);

job.setCombinerClass(Reduce.class);

job.setReducerClass(Reduce.class);

//to increasing the number of reducer uncomment the line below

//job.setNumReduceTasks(5);

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

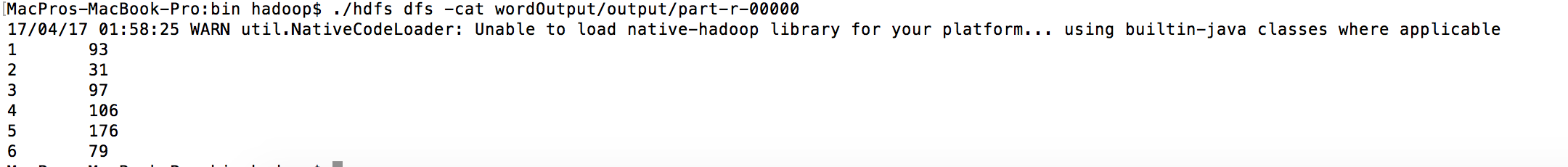
FileOutputFormat.setOutputPath(job, new Path(args[1]));

job.waitForCompletion(true);

}

}

Output:



* 1. The input file is the same as for part (a) above. The output file must contain <key value> pairs such that the key is the node id and the value is a pair of wo numbers, the first is the number of incoming edges (indegree) and the second number is the number of outflowing edges (outdegree) of the node.

A larger data file will soon be made available to run your program on. For each of these two tasks, submit your program and the output files generated for each of the tasks with the larger test file.

**Code:**

package org.test;

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;

import org.test.WordCount.IntSumReducer;

public class NodeDegreeB {

public static class Map extends Mapper<LongWritable, Text, IntWritable, Text> {

private IntWritable outKey = new IntWritable();

private Text outValue = new Text();

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

int count = 0;

String line = value.toString();

if (line.length() > 1) {

String[] inLine = line.split("\t");

for (String st : inLine) {

outKey.set(Integer.parseInt(st));

if (count == 0) {

outValue.set("0 1");

count = 1;

} else {

outValue.set("1 0");

}

context.write(outKey, outValue);

}

}

}

}

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

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

private Text result = new Text();

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

throws IOException, InterruptedException {

int cntr = 0;

int inBoundCount = 0;

int outBoundCount = 0;

for (Text val : values) {

String inOutValues[] = val.toString().split("\t");

inBoundCount = inBoundCount + Integer.parseInt(inOutValues[0]);

outBoundCount = outBoundCount + Integer.parseInt(inOutValues[1]);

}

String finalValues = inBoundCount + "\t" + outBoundCount;

result.set(finalValues);

context.write(key, result);

}

}

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

Configuration conf = new Configuration();

Job job = new Job(conf, "nodeDegree");

job.setMapOutputKeyClass(IntWritable.class);

job.setMapOutputValueClass(Text.class);

job.setJarByClass(NodeDegreeB.class);

job.setMapperClass(Map.class);

job.setCombinerClass(Reduce.class);

job.setReducerClass(Reduce.class);

//to increasing the number of reducer uncomment the line below

//job.setNumReduceTasks(5);

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

FileInputFormat.addInputPath(job, new Path(args[0]));

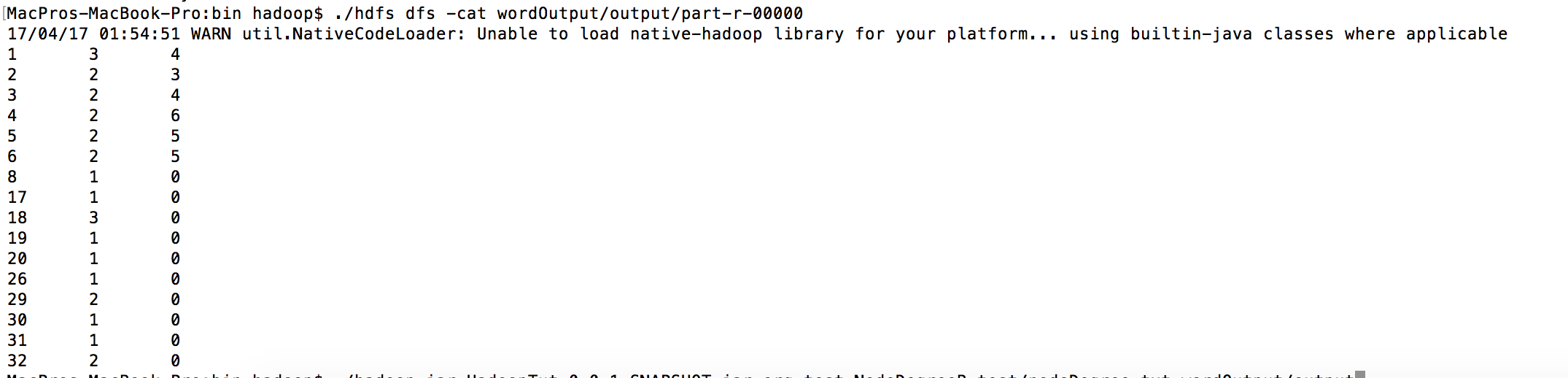
FileOutputFormat.setOutputPath(job, new Path(args[1]));

job.waitForCompletion(true);

}

}

**Output:**



1. **An input data file looks as shown below and we want an output from a map reduce program that is also shown below.**

**INPUT:**

003 Alicia student

001 John student

001 Tim student

002 Gary student

002 Jason student

003 Ohio State

001 Indiana State

003 Phil student

002 Kentucky State

001 Mary student

002 Carla student

003 Pete student

**Output:**

001 John Indiana

001 Tim Indiana

001 Mary Indiana

002 Gary Kentucky

002 Jason Kentucky

002 Carla Kentucky

003 Alicia Ohio

003 Phil Ohio

003 Pete Ohio

**Sol:**

We get every line as input to the Mapper function. Once I receive that, I will return ID, Category, Value as output of Mapper function.

**Mapper will return:**

**Key** **Values**

003 {student, Alicia}

001 {student, John}

001 {student, Tim}

002 {student, Gary}

002 {student, Jason}

003 {State, Ohio}

001 {State, Indiana}

003 {student, Phil}

002 {State, Kentucky}

001 {student, Mary}

002 {student, Carla}

003 {student, Pete}

**Reducer:** we perform shuffle sort on the Mapper output. The output after that looks like this,

**Shuffle Sort Output:**

**[001 [**student, John**][** student, Tim**][** State, Indiana**][** student, Mary**]]**

**[002 [**student, Gary**][** student, Jason**][** State, Kentucky**][** student, Carla**]]**

**[003 [**student, Alicia**][** State, Ohio**][** student, Phil**][** student, Pete**]]**

After observing the output, I want to reapply another sorting on the values of the Shuffle Sort Output, So that I can get State in the first place and other student data as remaining fields. The output will look like this

**[001 [** State, Indiana**] [**student, John**][** student, Tim**] [** student, Mary**]]**

**[002 [** State, Kentucky**] [**student, Gary**][** student, Jason**] [** student, Carla**]]**

**[003 [** State, Ohio**] [**student, Alicia**] [** student, Phil**][** student, Pete**]]**

**Final Reducer Output:**

001 John Indiana

001 Tim Indiana

001 Mary Indiana

002 Gary Kentucky

002 Jason Kentucky

002 Carla Kentucky

003 Alicia Ohio

003 Phil Ohio

003 Pete Ohio