package hadoop.mumbai;  
  
import java.io.IOException;  
  
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.mapreduce.Mapper;  
import org.apache.hadoop.mapreduce.Reducer;  
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.apache.hadoop.util.GenericOptionsParser;  
  
public class StockVolumeSumAvg {  
  
 public static class MyMapper extends Mapper<LongWritable, Text, Text, LongWritable>{  
 Text kword = new Text();  
 LongWritable vword = new LongWritable();  
 public void map(LongWritable key, Text value, Context context) throws InterruptedException, IOException  
 {  
 String line = value.toString();  
 String[] parts = line.split("\\t");  
 if(parts.length == 9)  
 {  
 String stockName = parts[1];  
 long volume = Long.valueOf(parts[7]);  
 kword.set(stockName);  
 vword.set(volume);  
 context.write(kword, vword);  
 }  
 }  
 }  
   
 public static class MyReducer extends Reducer<Text, LongWritable, Text, Text>  
 {  
 Text vword = new Text();  
 public void reduce(Text key, Iterable<LongWritable> values, Context context) throws IOException, InterruptedException  
 {  
 long sum = 0;  
 double avg = 0.0;  
 int counter = 0;  
 for(LongWritable value : values)  
 {  
 sum = sum + value.get();  
 counter++;  
 }  
 avg = (double) sum / counter;  
 vword.set("sum: " + sum + "\tAverage: " + avg);  
 context.write(key, vword);  
 }  
 }   
   
 public static void main(String args[]) throws IOException, InterruptedException, ClassNotFoundException  
 {  
 Configuration conf = new Configuration();  
 String otherArgs[] = new GenericOptionsParser(conf, args).getRemainingArgs();  
 if(otherArgs.length != 2)  
 {  
 System.out.println("Usage is: hadoop jar jarfile MainClass input output");  
 System.exit(1);  
 }  
 Job job = new Job(conf, "Finding sum of the the stock Volume");  
 job.setJarByClass(StockVolumeSumAvg.class);  
   
 job.setMapperClass(MyMapper.class);  
 job.setReducerClass(MyReducer.class);  
   
 job.setMapOutputKeyClass(Text.class);  
 job.setMapOutputValueClass(LongWritable.class);  
 job.setOutputKeyClass(Text.class);  
 job.setOutputValueClass(Text.class);  
   
 job.setInputFormatClass(TextInputFormat.class);  
 job.setOutputFormatClass(TextOutputFormat.class);  
   
 FileInputFormat.addInputPath(job, new Path(otherArgs[0]));  
 FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));  
   
 System.exit(job.waitForCompletion(true) ? 0 : 1);  
 }  
}

package hadoop.mumbai;  
  
import java.io.IOException;  
import java.util.HashMap;  
import java.util.Map;  
  
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.hive.serde2.io.DoubleWritable;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.mapreduce.Mapper;  
import org.apache.hadoop.mapreduce.Reducer;  
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.apache.hadoop.util.GenericOptionsParser;  
  
public class StocksMinMaxOHLC {  
  
 /\*\*  
 \* @param args  
 \* @throws IOException   
 \* @throws ClassNotFoundException   
 \* @throws InterruptedException   
 \*/  
 public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {  
 Configuration conf = new Configuration();  
 String otherArgs[] = new GenericOptionsParser(conf, args).getRemainingArgs();  
 if(otherArgs.length != 3)  
 {  
 System.out.println("Usage is: hadoop jar jarfile MainClass input output parameter[open | high | low | close ]");  
 System.exit(1);  
 }  
 Map<String, Integer> parameters= new HashMap<String, Integer>();  
 parameters.put("open", 3);  
 parameters.put("high", 4);  
 parameters.put("low", 5);  
 parameters.put("close", 6);  
 String parameter = otherArgs[2];  
   
 conf.setInt("name", parameters.get(parameter));  
 Job job = new Job(conf, "Finding min and max of open close high low of every stock");  
 job.setJarByClass(StocksMinMaxOHLC.class);  
   
 job.setMapperClass(MyMapper.class);  
 job.setReducerClass(MyReducer.class);  
   
 job.setMapOutputKeyClass(Text.class);  
 job.setMapOutputValueClass(DoubleWritable.class);  
   
 job.setOutputKeyClass(Text.class);  
 job.setOutputValueClass(Text.class);  
   
 job.setInputFormatClass(TextInputFormat.class);  
 job.setOutputFormatClass(TextOutputFormat.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 MyMapper extends Mapper<LongWritable, Text, Text, DoubleWritable>{  
 Text kword = new Text();  
 DoubleWritable vword = new DoubleWritable();  
 int index = 0;  
 public void setup(Context context)  
 {  
 Configuration conf = context.getConfiguration();  
 index = Integer.valueOf(conf.get("name"));  
 }  
 public void map(LongWritable key, Text value, Context context) throws InterruptedException, IOException  
 {  
 String line = value.toString();  
 String[] parts = line.split("\\t");  
 if(parts.length == 9)  
 {  
 String stockName = parts[1];  
 double trade = Double.valueOf(parts[index]);  
 kword.set(stockName);  
 vword.set(trade);  
 context.write(kword, vword);  
 }  
 }  
 }  
   
 public static class MyReducer extends Reducer<Text, DoubleWritable, Text, Text>  
 {  
 Text vword = new Text();  
 public void reduce(Text key, Iterable<DoubleWritable> values, Context context) throws IOException, InterruptedException  
 {  
 double min = Double.MAX\_VALUE;  
 double max = 0.0;  
 for(DoubleWritable value : values)  
 {  
 double current = value.get();  
 max = (max>current)?max:current;  
 min = (min<current)?min:current;  
 }  
 vword.set("Min: " + min + "\tMax: " + max);  
 context.write(key, vword);  
 }  
 }   
}

package hadoop.mumbai;  
  
import java.io.IOException;  
  
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.mapreduce.Mapper;  
import org.apache.hadoop.mapreduce.Reducer;  
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.apache.hadoop.util.GenericOptionsParser;  
  
public class StocksCount {  
  
 /\*\*  
 \* @param args  
 \* @throws IOException   
 \* @throws ClassNotFoundException   
 \* @throws InterruptedException   
 \*/  
 //This is the method for defining the MapReduce Driver  
 public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {  
 Configuration conf = new Configuration();  
 String otherArgs[] = new GenericOptionsParser(conf, args).getRemainingArgs();  
   
 if(otherArgs.length != 2)  
 {  
 System.out.println("Usage is: hadoop jar jarfile MainClass input output");  
 System.exit(1);  
 }  
   
 Job job = new Job(conf, "Stocks Counting");  
   
 job.setJarByClass(StocksCount.class);  
   
 //To set Mapper and Reduce classes  
 job.setMapperClass(MyMapper.class);  
 job.setReducerClass(MyReducer.class);  
   
 //Output Key-Value data types Type  
 job.setMapOutputKeyClass(Text.class);  
 job.setMapOutputValueClass(LongWritable.class);  
   
 job.setOutputKeyClass(Text.class);  
 job.setOutputValueClass(LongWritable.class);  
   
 //To inform input output Formats to MapReduce Program   
 job.setInputFormatClass(TextInputFormat.class);  
 job.setOutputFormatClass(TextOutputFormat.class);  
   
 //Inform input and output File or Directory locations  
 FileInputFormat.addInputPath(job, new Path(otherArgs[0]));  
 FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));  
   
 //Inform the job termination criteria  
 System.exit(job.waitForCompletion(true) ? 0 : 1);   
 }  
   
 public static class MyMapper extends Mapper<LongWritable, Text, Text, LongWritable>  
 {  
 Text kword = new Text();  
 LongWritable vword = new LongWritable();  
 public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException  
 {  
 String line = value.toString();  
 String[] parts = line.split("\\t");  
 if(parts.length == 9)  
 {  
 kword.set(parts[1]);  
 vword.set(1);  
 context.write(kword, vword);  
 }  
 }  
 }  
   
 public static class MyReducer extends Reducer<Text, LongWritable, Text, LongWritable>  
 {  
 public void reduce(Text key, Iterable<LongWritable> values, Context context) throws IOException, InterruptedException  
 {  
 long sum = 0;  
 for(LongWritable value : values)  
 {  
 sum = sum + value.get();  
 }  
 context.write(key, new LongWritable(sum));  
 }  
 }  
}

package hadoop.mumbai;  
  
import java.io.IOException;  
  
import org.apache.hadoop.conf.Configuration;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.hive.serde2.io.DoubleWritable;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.mapreduce.Mapper;  
import org.apache.hadoop.mapreduce.Reducer;  
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.apache.hadoop.util.GenericOptionsParser;  
  
public class StockChange {  
  
 /\*\*  
 \* @param args  
 \* @throws IOException   
 \* @throws ClassNotFoundException   
 \* @throws InterruptedException   
 \*/  
 //This is the method for defining the MapReduce Driver  
 public static void main(String[] args) throws IOException, InterruptedException, ClassNotFoundException {  
 Configuration conf = new Configuration();  
 String otherArgs[] = new GenericOptionsParser(conf, args).getRemainingArgs();  
   
 if(otherArgs.length != 2)  
 {  
 System.out.println("Usage is: hadoop jar jarfile MainClass input output");  
 System.exit(1);  
 }  
   
 Job job = new Job(conf, "Stocks change per day");  
   
 job.setJarByClass(StockChange.class);  
   
 //To set Mapper and Reduce classes  
 job.setMapperClass(MyMapper.class);  
 job.setReducerClass(MyReducer.class);  
   
 //Output Key-Value data types Type  
 job.setMapOutputKeyClass(Text.class);  
 job.setMapOutputValueClass(Text.class);  
   
 job.setOutputKeyClass(Text.class);  
 job.setOutputValueClass(DoubleWritable.class);  
   
 //To inform input output Formats to MapReduce Program   
 job.setInputFormatClass(TextInputFormat.class);  
 job.setOutputFormatClass(TextOutputFormat.class);  
   
 //Inform input and output File or Directory locations  
 FileInputFormat.addInputPath(job, new Path(otherArgs[0]));  
 FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));  
   
 //Inform the job termination criteria  
 System.exit(job.waitForCompletion(true) ? 0 : 1);   
 }  
   
 public static class MyMapper extends Mapper<LongWritable, Text, Text, Text>  
 {  
 Text kword = new Text();  
 Text vword = new Text();  
 public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException  
 {  
 String line = value.toString();  
 String[] parts = line.split("\\t");  
 if(parts.length == 9)  
 {  
 kword.set(parts[1] + "\t" + parts[2]);  
 vword.set(parts[3] + ":" + parts[6]);  
 context.write(kword, vword);  
 }  
 }  
 }  
   
 public static class MyReducer extends Reducer<Text, Text, Text, DoubleWritable>  
 {  
 DoubleWritable vword = new DoubleWritable();  
 public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException  
 {  
 double change = 0.0;  
 for(Text value : values)  
 {  
 String[] parts = value.toString().split("\\:");  
 if(parts.length == 2)  
 {  
 change = Double.valueOf(parts[1]) - Double.valueOf(parts[0]);  
 }  
 }  
 vword.set(change);  
 context.write(key, vword);  
 }  
 }  
}

package hadoop.mumbai;  
  
import java.io.IOException;  
import java.util.Iterator;  
  
import org.apache.hadoop.conf.Configuration;  
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.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.mapred.TextInputFormat;  
import org.apache.hadoop.mapred.lib.MultipleTextOutputFormat;  
  
public class StockMultipleFiles {  
  
 /\*\*  
 \* @param args  
 \* @throws IOException  
 \* @throws InterruptedException  
 \* @throws ClassNotFoundException  
 \*/  
 public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {  
 Configuration conf = new Configuration();  
 JobConf job = new JobConf(conf);  
  
 job.setJobName("Finding Avg and Sum of Stock Volume");  
  
 job.setJarByClass(StockMultipleFiles.class);  
 //Mapper and Reducer classes  
 job.setMapperClass(MyMapper.class);  
 job.setReducerClass(MyReducer.class);  
  
 //Output Key-Value Data types  
 job.setMapOutputKeyClass(Text.class);  
 job.setMapOutputValueClass(LongWritable.class);  
  
 job.setOutputKeyClass(Text.class);  
 job.setOutputValueClass(Text.class);  
  
 //Inform Input/Output Formats  
 job.setInputFormat(TextInputFormat.class);  
 job.setOutputFormat(MyMultipleOutputFileFormat.class);  
  
 //Inform file or Directory locations  
 FileInputFormat.addInputPath(job, new Path(args[0]));  
 FileOutputFormat.setOutputPath(job, new Path(args[1]));  
  
 //Inform termination criteria  
 JobClient.runJob(job);  
  
 }  
  
 //This is my Mapper Class  
 public static class MyMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, LongWritable>  
 {  
 public void map(LongWritable key, Text value,  
 OutputCollector<Text, LongWritable> collect, Reporter reporter)  
 throws IOException {  
 String line = value.toString();  
 String parts[] = line.trim().split("\\t");  
 if(parts.length == 9)  
 {  
 String stcokName = parts[1];  
 long volume = Long.valueOf(parts[7]);  
 collect.collect(new Text(stcokName), new LongWritable(volume));  
 }  
 }  
 }  
  
 //THis is my Reducer class  
 public static class MyReducer extends MapReduceBase implements Reducer<Text, LongWritable, Text, Text>  
 {  
 public void reduce(Text key, Iterator<LongWritable> values,  
 OutputCollector<Text, Text> collect, Reporter reporter)  
 throws IOException {  
 long sum = 0;  
 int counter = 0;  
 while(values.hasNext())  
 {  
 sum = sum + values.next().get();  
 counter++;  
 }  
 float avg = (float) sum/counter;  
 String emitValue = sum + "\t" + avg;  
 collect.collect(key, new Text(emitValue));  
 }  
 }  
 public static class MyMultipleOutputFileFormat extends MultipleTextOutputFormat<Text, Text>  
 {  
 public String generateFileNameForKeyValue(Text key, Text value, String name)  
 {  
 return new Path(key.toString(), value.toString()).toString();  
 }  
 }  
}

package hadoop.mumbai;  
  
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.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.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.mapred.TextInputFormat;  
import org.apache.hadoop.mapred.TextOutputFormat;  
import org.apache.hadoop.mapred.lib.ChainMapper;  
import org.apache.hadoop.util.GenericOptionsParser;  
  
public class MyChainMapper {  
  
 /\*\*  
 \* @param args  
 \* @throws IOException  
 \*/  
 public static void main(String[] args) throws IOException {  
 Configuration conf = new Configuration();  
 String otherArgs[] = new GenericOptionsParser(conf, args).getRemainingArgs();  
  
 JobConf job = new JobConf(conf);  
 job.setJobName("Chaining MapReduce");  
 job.setJarByClass(MyChainMapper.class);  
  
 job.setInputFormat(TextInputFormat.class);  
 job.setOutputFormat(TextOutputFormat.class);  
  
 FileInputFormat.setInputPaths(job, new Path(otherArgs[0]));  
 FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));  
  
 Configuration mapperOneConf = new Configuration(false);  
 JobConf mapJob1 = new JobConf(mapperOneConf);  
 ChainMapper.addMapper(job, MyMapperOne.class, LongWritable.class, Text.class, Text.class, LongWritable.class, true, mapJob1);  
  
 Configuration mapperSecConf = new Configuration(false);  
 JobConf mapJob2 = new JobConf(mapperSecConf);  
 ChainMapper.addMapper(job, MyMapperSecond.class, Text.class, LongWritable.class, Text.class, LongWritable.class, true, mapJob2);  
  
 job.setReducerClass(MyReducer.class);  
  
 Configuration mapperThreeConf = new Configuration(false);  
 JobConf mapJob3 = new JobConf(mapperThreeConf);  
 ChainMapper.addMapper(job, MyMapperThree.class, Text.class, LongWritable.class, Text.class, LongWritable.class, true, mapJob3);  
  
 JobClient.runJob(job);  
 }  
  
 public static class MyMapperOne extends MapReduceBase implements Mapper<LongWritable, Text, Text, LongWritable>  
 {  
 public void map(LongWritable key, Text value,  
 OutputCollector<Text, LongWritable> collect, Reporter reporter)  
 throws IOException {  
 String line = value.toString();  
 StringTokenizer st = new StringTokenizer(line);  
 while(st.hasMoreTokens())  
 {  
 collect.collect(new Text(st.nextToken()), new LongWritable(1));  
 }  
 }  
 }  
  
 public static class MyMapperSecond extends MapReduceBase implements Mapper<Text, LongWritable, Text, LongWritable>  
 {  
 List<String> stopwords = null;  
 public void configure(JobConf conf)  
 {  
 conf = new JobConf();  
 stopwords = new ArrayList<String>();  
 stopwords.add("and");  
 stopwords.add("is");  
 stopwords.add("am");  
 stopwords.add("at");  
 stopwords.add("in");  
 stopwords.add("after");  
 stopwords.add("did");  
 stopwords.add("will");  
 }  
 public void map(Text key, LongWritable value,  
 OutputCollector<Text, LongWritable> collect, Reporter reporter)  
 throws IOException {  
 String myKey = key.toString();  
 if(!stopwords.contains(myKey))  
 {  
 collect.collect(key, value);  
 }  
 }  
 }  
  
 public static class MyReducer extends MapReduceBase implements Reducer<Text, LongWritable, Text, LongWritable>  
 {  
 public void reduce(Text key, Iterator<LongWritable> values,  
 OutputCollector<Text, LongWritable> collect, Reporter reporter)  
 throws IOException {  
 long sum = 0;  
 while(values.hasNext())  
 {  
 sum = sum + values.next().get();  
 }  
 collect.collect(key, new LongWritable(sum));  
 }  
 }  
 public static class MyMapperThree extends MapReduceBase implements Mapper<Text, LongWritable, Text, LongWritable>  
 {  
 public void map(Text key, LongWritable value,  
 OutputCollector<Text, LongWritable> collect, Reporter reporter)  
 throws IOException {  
 long val = value.get() \* 10;  
 collect.collect(key, new LongWritable(val));  
 }  
 }  
}