Task 0

cd

mkdir Project5

cd Project5

mkdir Part\_1

cd Part\_1

mkdir Task0

cd Task0

cp /home/public/WordCount.java /home/student101/Project5/Part\_1/Task0/

javac -classpath $(hadoop classpath) WordCount.java

hadoop fs -copyFromLocal /home/public/words.txt /user/student101/words.txt

jar cf wordcount.jar .

hadoop jar wordcount.jar WordCount /user/student101/words.txt /user/student101/word\_output.

(this name should be different everytime)

hadoop dfs -cat /user/student101/word\_output/part-r-00000

hadoop dfs -cat /user/student101/word\_output/part-r-00001

hadoop dfs -cat /user/student101/word\_output/part-r-00002

hadoop fs -cat /user/student101/word\_output/part-r-000\* > /home/student101/Project5/Part\_1/Task0/Task0Output

// ======================= WordCount.java ==========================================

import java.io.IOException;

import java.util.StringTokenizer;

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

import org.apache.hadoop.fs.Path;

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

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.IntWritable;

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 WordCount extends Configured implements Tool {

public static class WordCountMap extends Mapper<LongWritable, Text, Text, IntWritable>

{

//initialize IntWritable and word

private final static IntWritable one = new IntWritable(1);

private Text word = new Text();

@Override

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

{

//converts text value to string

String line = value.toString();

//tokenize string

StringTokenizer tokenizer = new StringTokenizer(line);

//iterate tokens and get words with count 1

while(tokenizer.hasMoreTokens())

{

word.set(tokenizer.nextToken());

context.write(word, one);

}

}

}

public static class WordCountReducer extends Reducer<Text, IntWritable, Text, IntWritable>

{

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

{

//initialize sum

int sum = 0;

//iterate through value and get their sum

for(IntWritable value: values)

{

sum += value.get();

}

//get word with total count

context.write(key, new IntWritable(sum));

}

}

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

//runs Hadoop job

Job job = new Job(getConf());

job.setJarByClass(WordCount.class);

//sets job name

job.setJobName("wordcount");

//sets output key

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

//sets mapper and reducer class

job.setMapperClass(WordCountMap.class);

job.setReducerClass(WordCountReducer.class);

//sets input and output formatter

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

//sets input and output path

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

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

//returns success

boolean success = job.waitForCompletion(true);

return success ? 0: 1;

}

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

// TODO Auto-generated method stub

//main method to run the program

int result = ToolRunner.run(new WordCount(), args);

System.exit(result);

}

}

Task 1

// ======================= LetterCounter.java ==========================================  
import java.io.IOException;  
  
import org.apache.hadoop.conf.\*;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.util.\*;  
  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.IntWritable;  
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 LetterCounter extends Configured implements Tool {  
//code reference from WordCount.java  
 public static class LetterCounterMap extends Mapper<LongWritable, Text, Text, IntWritable>  
 {  
 //initialize IntWritable and word  
 private final static IntWritable *one* = new IntWritable(1);  
 private Text letter = new Text();  
  
 @Override  
 public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException  
 {  
 //converts text value to string  
 String line = value.toString();  
 //gets character from the line  
 for (char c : line.toCharArray()) {  
 if (Character.*isLetter*(c)) {  
 //outputs key-value pairs of the form (letter, 1)  
 letter.set(String.*valueOf*(Character.*toLowerCase*(c)));  
 context.write(letter, *one*);  
 }  
 }  
 }  
 }  
  
 public static class LetterCounterReducer extends Reducer<Text, IntWritable, Text, IntWritable>  
 {  
 //reducer class  
 @Override  
 public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException  
 {  
 //initialize sum  
 int sum = 0;  
 //iterate through value and get their sum  
 for (IntWritable value : values) {  
 sum += value.get();  
 }  
 //get letters with total count  
 context.write(key, new IntWritable(sum));  
 }  
  
 }  
  
 public int run(String[] args) throws Exception {  
 //runs Hadoop job  
 Job job = new Job(getConf());  
 job.setJarByClass(LetterCounter.class);  
 //sets job name  
 job.setJobName("lettercount");  
 //sets output key  
 job.setOutputKeyClass(Text.class);  
 job.setOutputValueClass(IntWritable.class);  
 //sets mapper and reducer class  
 job.setMapperClass(LetterCounterMap.class);  
 job.setReducerClass(LetterCounterReducer.class);  
 //sets input and output formatter  
 job.setInputFormatClass(TextInputFormat.class);  
 job.setOutputFormatClass(TextOutputFormat.class);  
 //sets input and output path  
 FileInputFormat.setInputPaths(job, new Path(args[0]));  
 FileOutputFormat.setOutputPath(job, new Path(args[1]));  
 //returns success  
 boolean success = job.waitForCompletion(true);  
 return success ? 0: 1;  
 }  
  
 public static void main(String[] args) throws Exception {  
 //main method to run the program  
 int result = ToolRunner.run(new LetterCounter(), args);  
 System.*exit*(result);  
 }  
  
}

nano LetterCounter.java

javac -classpath $(hadoop classpath) LetterCounter.java

jar cf lettercount.jar .

hadoop jar lettercount.jar LetterCounter /user/student101/words.txt /user/student101/letteroutput

hadoop dfs -cat /user/student101/letteroutput/part-r-00000

hadoop fs -cat /user/student101/letteroutput/part-r-\* | sort -k2,2nr > /home/student101/Project5/Part\_1/Task1/Task1Output

Task 2

import java.io.IOException;  
import java.util.HashSet;  
import java.util.StringTokenizer;  
  
import org.apache.hadoop.conf.\*;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.util.\*;  
  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.NullWritable;  
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 FindPattern extends Configured implements Tool {  
//code reference from WordCount.java  
  
 public static class FindPatternMap extends Mapper<LongWritable, Text, Text, NullWritable>  
 {  
 //initialize word and HashSet to avoid duplicates  
 private Text word = new Text();  
 private HashSet<String> words = new HashSet<String>();  
  
 @Override  
 public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException  
 {  
 //converts text value to string  
 String line = value.toString();  
 //tokenize string  
 StringTokenizer tokenizer = new StringTokenizer(line);  
 //iterate tokens and get words containing the pattern  
 while(tokenizer.hasMoreTokens())  
 {  
 String text = tokenizer.nextToken();  
 //if the word contains cool then get that word  
 if (text.toLowerCase().contains("cool")) {  
 if (!words.contains(text)) {  
 words.add(text);  
 word.set(text);  
 //reference  
 //https://stackoverflow.com/questions/15899705/how-to-just-output-value-in-context-writek-v  
 //https://hadoop.apache.org/docs/current/api/org/apache/hadoop/io/NullWritable.html  
 context.write(word, NullWritable.get());  
 }  
 }  
 }  
 }  
 }  
  
 public static class FindPatternReducer extends Reducer<Text, NullWritable, Text, NullWritable>  
 {  
 public void reduce(Text key, Iterable<NullWritable> values, Context context) throws IOException, InterruptedException  
 {  
 //write word to output  
 context.write(key, NullWritable.get());  
 }  
 }  
  
 public int run(String[] args) throws Exception {  
 //runs Hadoop job  
  
 Job job = new Job(getConf());  
 job.setJarByClass(FindPattern.class);  
 //sets job name  
 job.setJobName("findpattern");  
 //sets output key  
 job.setOutputKeyClass(Text.class);  
 job.setOutputValueClass(NullWritable.class);  
 //sets mapper and reducer class  
 job.setMapperClass(FindPatternMap.class);  
 job.setReducerClass(FindPatternReducer.class);  
  
 //sets input and output formatter  
 job.setInputFormatClass(TextInputFormat.class);  
 job.setOutputFormatClass(TextOutputFormat.class);  
  
 //sets input and output path  
 FileInputFormat.setInputPaths(job, new Path(args[0]));  
 FileOutputFormat.setOutputPath(job, new Path(args[1]));  
 //returns success  
 boolean success = job.waitForCompletion(true);  
 return success ? 0: 1;  
 }  
  
 public static void main(String[] args) throws Exception {  
 //main method to run the program  
 int result = ToolRunner.run(new FindPattern(), args);  
 System.*exit*(result);  
 }  
}

nano FindPattern.java

javac -classpath $(hadoop classpath) FindPattern.java

jar cf findpattern.jar .

hadoop jar findpattern.jar FindPattern /user/student101/words.txt /user/student101/findPatternoutput

hadoop fs -cat /user/student101/findPatternoutput/part-r-000\* > /home/student101/Project5/Part\_1/Task2/Task2Output

Task 3

**MaxTemperatureMapper.java**

// ============== MaxTemperatureMapper.java ================================

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 MaxTemperatureMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {

private static final int MISSING = 9999;

public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {

// Get line from input file. This was passed in by Hadoop as value.

// We have no use for the key (file offset) so we are ignoring it.

String line = value.toString();

// Get year when weather data was collected. The year is in positions 15-18.

// This field is at a fixed position within a line.

String year = line.substring(15, 19);

// Get the temperature too.

int airTemperature;

if (line.charAt(87) == '+') { // parseInt doesn't like leading plus signs

airTemperature = Integer.parseInt(line.substring(88, 92));

} else {

airTemperature = Integer.parseInt(line.substring(87, 92));

}

// Get quality of reading. If not missing and of good quality then

// produce intermediate (year,temp).

String quality = line.substring(92, 93);

if (airTemperature != MISSING && quality.matches("[01459]")) {

// for each year in input, reduce will be called with

// (year,[temp,temp,temp, ...])

// They key is year and the list of temps will be placed in an iterator.

output.collect(new Text(year), new IntWritable(airTemperature)); }

}

}

### MaxTemperatureReducer.java

//=========== MaxTemperatureReducer.java ====================================================

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 MaxTemperatureReducer extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {

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

IntWritable> output, Reporter reporter) throws IOException {

// from the list of values, find the maximum

int maxValue = Integer.MIN\_VALUE;

while (values.hasNext()) {

maxValue = Math.max(maxValue, values.next().get());

}

// emit (key = year, value = maxTemp = max for year)

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

}

}

### MaxTemperature.java

// ======= And, to get it all running and tied together: MaxTemperature.java ============

import java.io.IOException;  
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;  
  
public class MaxTemperature {  
 public static void main(String[] args) throws IOException {  
 //checks two are arguments are present  
 if (args.length != 2) {  
 System.*err*.println("Usage: MaxTemperature <input path> <output path>");  
 System.*exit*(-1);  
 }  
 //create job configuration instance  
 JobConf conf = new JobConf(MaxTemperature.class);  
 //set job name  
 conf.setJobName("Max temperature");  
 //set input and output formatter path  
 FileInputFormat.addInputPath(conf, new Path(args[0]));  
 FileOutputFormat.setOutputPath(conf, new Path(args[1]));  
 //set mapper and reducer class  
 conf.setMapperClass(MaxTemperatureMapper.class);  
 conf.setReducerClass(MaxTemperatureReducer.class);  
 //set output key, value class  
 conf.setOutputKeyClass(Text.class);  
 conf.setOutputValueClass(IntWritable.class);  
 //run job  
 JobClient.runJob(conf);  
 }  
}

javac -classpath $(hadoop classpath) MaxTemperature.java MaxTemperatureMapper.java MaxTemperatureReducer.java

jar cf temperature.jar .

hadoop fs -copyFromLocal /home/public/combinedYears.txt /user/student101/combinedYears.txt

hadoop jar temperature.jar MaxTemperature /user/student101/combinedYears.txt /user/student101/temperature\_output

hadoop dfs -getmerge /user/student101/temperature\_output Task3Output

Task 4

MinTemperatureMapper.java

// ============== MinTemperatureMapper.java ================================

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 MinTemperatureMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {  
 private static final int *MISSING* = 9999;  
  
 public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {  
 // Get line from input file. This was passed in by Hadoop as value.  
 // We have no use for the key (file offset) so we are ignoring it.  
 String line = value.toString();  
 // Get year when weather data was collected. The year is in positions 15-18.  
 // This field is at a fixed position within a line.  
 String year = line.substring(15, 19);  
 // Get the temperature too.  
 int airTemperature;  
 if (line.charAt(87) == '+') {  
 airTemperature = Integer.*parseInt*(line.substring(88, 92));  
 } else {  
 airTemperature = Integer.*parseInt*(line.substring(87, 92));  
 }

// convert Celsius to Celsius\*10  
 airTemperature = airTemperature / 10;

// Get quality of reading. If not missing and of good quality then  
 // produce intermediate (year,temp).  
 String quality = line.substring(92, 93);  
 if (airTemperature != *MISSING* && quality.matches("[01459]")) {  
 // for each year in input, reduce will be called with  
 // (year,[temp,temp,temp, ...])  
 // They key is year and the list of temps will be placed in an iterator.  
 output.collect(new Text(year), new IntWritable(airTemperature));  
 }  
 }  
}

MinTemperatureReducer.java

//=========== MaxTemperatureReducer.java ====================================================  
  
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 MinTemperatureReducer extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {  
  
 public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {  
 // from the list of values, find the minimum  
 int minValue = Integer.*MAX\_VALUE*;  
 while (values.hasNext()) {  
 minValue = Math.min(minValue, values.next().get());  
 }  
 // emit (key = year, value = minTemp = min for year)  
 output.collect(key, new IntWritable(minValue));  
 }  
}

MinTemperature.java

// ======= And, to get it all running and tied together: MinTemperature.java ============  
import java.io.IOException;  
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;  
  
public class MinTemperature {  
 public static void main(String[] args) throws IOException {  
 //checks two are arguments are present  
 if (args.length != 2) {  
 System.*err*.println("Usage: MinTemperature <input path> <output path>");  
 System.*exit*(-1);  
 }  
 //create job configuration instance  
 JobConf conf = new JobConf(MinTemperature.class);  
 //set job name  
 conf.setJobName("Min temperature");  
 //set input and output formatter path  
 FileInputFormat.addInputPath(conf, new Path(args[0]));  
 FileOutputFormat.setOutputPath(conf, new Path(args[1]));  
 //set mapper and reducer class  
 conf.setMapperClass(MinTemperatureMapper.class);  
 conf.setReducerClass(MinTemperatureReducer.class);  
 //set output key, value class  
 conf.setOutputKeyClass(Text.class);  
 conf.setOutputValueClass(IntWritable.class);  
 //run job  
 JobClient.runJob(conf);  
 }  
}

javac -classpath $(hadoop classpath) MinTemperature.java MinTemperatureMapper.java MinTemperatureReducer.java

jar cf mintemperature.jar .

hadoop jar mintemperature.jar MinTemperature /user/student101/combinedYears.txt /user/student101/mintemperature\_output

hadoop dfs -getmerge /user/student101/mintemperature\_output Task4Output

Task 5

Crime.java

import java.io.IOException;  
import java.util.StringTokenizer;  
import org.apache.hadoop.conf.\*;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.NullWritable;  
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.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.util.Tool;  
import org.apache.hadoop.util.ToolRunner;  
  
import java.io.IOException;  
  
public class Crime extends Configured implements Tool {  
  
 public static class CrimeMapper extends Mapper<LongWritable, Text, NullWritable, IntWritable> {  
 private final static IntWritable *ONE* = new IntWritable(1);  
  
 @Override  
 public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {  
 // Get line from input file. This was passed in by Hadoop as value.  
 // We have no use for the key (file offset) so we are ignoring it.  
 String line = value.toString();  
 // get the type of crime from the fifth column  
 String type = line.split("\t")[4];  
 // set count to 1 for this record  
 int count = 1;  
 if (type.equalsIgnoreCase("rape") || type.equalsIgnoreCase("robbery")) {  
 // if the type of crime is rape or robbery, emit a key-value pair (type, 1)  
 context.write(NullWritable.get(), *ONE*);  
  
 }  
 }  
 }  
  
  
 public static class CrimeReducer extends Reducer<NullWritable, IntWritable, NullWritable, IntWritable> {  
  
 @Override  
 public void reduce(NullWritable key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {  
 int totalCount = 0;  
 for (IntWritable value : values) {  
 totalCount += value.get();  
 }  
 context.write(NullWritable.get(), new IntWritable(totalCount));  
 }  
 }  
  
 @Override  
 public int run(String[] args) throws Exception {  
 Job job = new Job(getConf());  
 job.setJarByClass(Crime.class);  
 job.setJobName("crime");  
  
 job.setOutputKeyClass(NullWritable.class);  
 job.setOutputValueClass(IntWritable.class);  
  
 job.setMapperClass(Crime.CrimeMapper.class);  
 job.setReducerClass(CrimeReducer.class);  
  
 job.setInputFormatClass(TextInputFormat.class);  
 job.setOutputFormatClass(TextOutputFormat.class);  
  
 FileInputFormat.setInputPaths(job, new Path(args[0]));  
 FileOutputFormat.setOutputPath(job, new Path(args[1]));  
  
 boolean success = job.waitForCompletion(true);  
 return success ? 0 : 1;  
 }  
  
 public static void main(String[] args) throws Exception {  
 int result = ToolRunner.run(new Crime(), args);  
 System.*exit*(result);  
 }  
}

javac -classpath $(hadoop classpath) Crime.java CrimeMapper.java CrimeReducer.java

jar cf rapesplusrobberies.jar .

hadoop fs -copyFromLocal /home/public/P1V.txt /user/student101/P1V.txt

hadoop jar rapesplusrobberies.jar Crime /user/student101/P1V.txt /user/student101/rapeplusrobberyOutput

hadoop dfs -getmerge /user/student101/ eplusrobberyOutput Task5Output

Task 6

import java.io.IOException;  
import java.util.StringTokenizer;  
import org.apache.hadoop.conf.\*;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.NullWritable;  
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.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.util.Tool;  
import org.apache.hadoop.util.ToolRunner;  
  
import java.io.IOException;  
  
public class CrimeAssault extends Configured implements Tool {  
  
 public static class CrimeAssaultMapper extends Mapper<LongWritable, Text, NullWritable, IntWritable> {  
 private final static IntWritable *ONE* = new IntWritable(1);  
  
 @Override  
 public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {  
 // Get line from input file. This was passed in by Hadoop as value.  
 // We have no use for the key (file offset) so we are ignoring it.  
 String line = value.toString();  
 // get the type of crime from the fifth column  
 String type = line.split("\t")[4];  
  
 if (type.equalsIgnoreCase("aggravated assault")) {  
 // if the type of crime is rape or robbery, emit a key-value pair (type, 1)  
 // get the x, y coordinates of the crime location  
 double x = Double.*parseDouble*(line.split("\t")[0]);  
 double y = Double.*parseDouble*(line.split("\t")[1]);  
 // calculate the distance between the crime location and 3803 Forbes Avenue  
 double distance = Math.*sqrt*(Math.*pow*(x - 1354326.897, 2) + Math.*pow*(y - 411447.7828, 2)) \* 0.3048;  
 // emit a key-value pair if the crime occurred within 350 meters of 3803 Forbes Avenue  
 if (distance <= (350.0)) {  
 context.write(NullWritable.get(), *ONE*);  
 }  
 }  
 }  
 }  
  
  
 public static class CrimeAssaultReducer extends Reducer<NullWritable, IntWritable, NullWritable, IntWritable> {  
  
 @Override  
 public void reduce(NullWritable key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {  
 int totalCount = 0;  
 for (IntWritable value : values) {  
 totalCount += value.get();  
 }  
 context.write(NullWritable.get(), new IntWritable(totalCount));  
 }  
 }  
  
 @Override  
 public int run(String[] args) throws Exception {  
 Job job = new Job(getConf());  
 job.setJarByClass(CrimeAssault.class);  
 job.setJobName("crimeassault");  
  
 job.setOutputKeyClass(NullWritable.class);  
 job.setOutputValueClass(IntWritable.class);  
  
 job.setMapperClass(CrimeAssaultMapper.class);  
 job.setReducerClass(CrimeAssaultReducer.class);  
  
 job.setInputFormatClass(TextInputFormat.class);  
 job.setOutputFormatClass(TextOutputFormat.class);  
  
 FileInputFormat.setInputPaths(job, new Path(args[0]));  
 FileOutputFormat.setOutputPath(job, new Path(args[1]));  
  
 boolean success = job.waitForCompletion(true);  
 return success ? 0 : 1;  
 }  
  
 public static void main(String[] args) throws Exception {  
 int result = ToolRunner.run(new CrimeAssault(), args);  
 System.*exit*(result);  
 }  
}

javac -classpath $(hadoop classpath) CrimeAssault.java

jar cf oaklandcrimestats.jar .

hadoop fs -copyFromLocal /home/public/P1V.txt /user/student101/P1V.txt

hadoop jar oaklandcrimestats.jar CrimeAssault /user/student101/P1V.txt /user/student101/crimeassaultstat\_output

hadoop dfs -getmerge /user/student101/crimeassaultstat\_output Task6Output

Task7

import java.io.IOException;  
import java.util.StringTokenizer;  
import org.apache.hadoop.conf.\*;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.NullWritable;  
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.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.util.Tool;  
import org.apache.hadoop.util.ToolRunner;  
  
import java.io.IOException;  
  
public class CrimeKML extends Configured implements Tool {  
  
 public static class CrimeKMLMapper extends Mapper<LongWritable, Text, NullWritable, Text> {  
  
 private Text outputValue = new Text();  
  
 @Override  
 public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {  
  
 String[] fields = value.toString().split("\t");  
 String type = fields[4];  
  
 if (type.equalsIgnoreCase("aggravated assault")) {  
  
 double x = Double.*parseDouble*(fields[0]);  
 double y = Double.*parseDouble*(fields[1]);  
  
 double longitude = Double.*parseDouble*(fields[8]);  
 double latitude = Double.*parseDouble*(fields[7]);  
  
 double distance = Math.*sqrt*(Math.*pow*(x - 1354326.897, 2) + Math.*pow*(y - 411447.7828, 2)) \* 0.3048;  
  
 if (distance <= 350.0) {  
 String kml = "<Documemt>\n" +  
 "<Placemark>\n" +  
 "<name>" + fields[4] + "</name>\n" +  
 "<description>" + fields[3] + "</description>" +  
 "<Point>\n" +  
 "<coordinates>" + longitude + "," + latitude + ",0</coordinates>\n" +  
 "</Point>\n" +  
 "</Placemark>\n" +  
 "</Document>";

outputValue.set(kml);  
 context.write(NullWritable.get(), outputValue);  
 }  
 }  
 }  
 }  
  
  
 public static class CrimeKMLReducer extends Reducer<NullWritable, Text, NullWritable, Text> {  
 private Text result = new Text();  
  
 @Override  
 public void reduce(NullWritable key, Iterable<Text> values, Context context) throws IOException, InterruptedException {  
 StringBuilder sb = new StringBuilder();  
 sb.append("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");  
 sb.append("<kml xmlns=\"http://www.opengis.net/kml/2.2\">\n");  
  
 for (Text value : values) {  
 sb.append(value.toString());  
 }  
  
 sb.append("</kml>\n");  
  
 result.set(sb.toString());  
 context.write(NullWritable.get(), result);  
 }  
 }  
  
 @Override  
 public int run(String[] args) throws Exception {  
 Job job = new Job(getConf());  
 job.setJarByClass(CrimeKML.class);  
 job.setJobName("crimekml");  
  
 job.setOutputKeyClass(NullWritable.class);  
 job.setOutputValueClass(Text.class);  
  
 job.setMapperClass(CrimeKMLMapper.class);  
 job.setReducerClass(CrimeKMLReducer.class);  
  
 job.setInputFormatClass(TextInputFormat.class);  
 job.setOutputFormatClass(TextOutputFormat.class);  
  
 FileInputFormat.setInputPaths(job, new Path(args[0]));  
 FileOutputFormat.setOutputPath(job, new Path(args[1]));  
  
 boolean success = job.waitForCompletion(true);  
 return success ? 0 : 1;  
 }  
  
 public static void main(String[] args) throws Exception {  
 int result = ToolRunner.run(new CrimeKML(), args);  
 System.*exit*(result);  
 }  
}