**Task0**

生成JAR:

cp -R /home/public/WordCount.java /home/student126/Project5/Part\_1/Task0/WordCount.java

mkdir WordCount\_classes

javac -classpath /usr/local/hadoop/hadoop-core-1.2.1.jar:./WordCount\_classes -d WordCount\_classes WordCount.java

jar -cvf WordCount.jar -C WordCount\_classes/ .

复制文件：

hadoop dfs -copyFromLocal /home/public/words.txt /user/student126/input/Task0

运行JAR：

hadoop jar WordCount.jar org.myorg.WordCount /user/student126/input/Task0 /user/student126/output/Task0

检查生成文件：

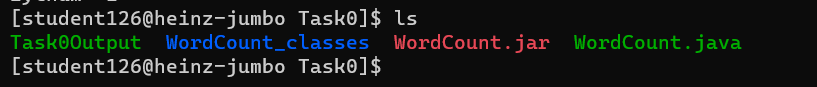
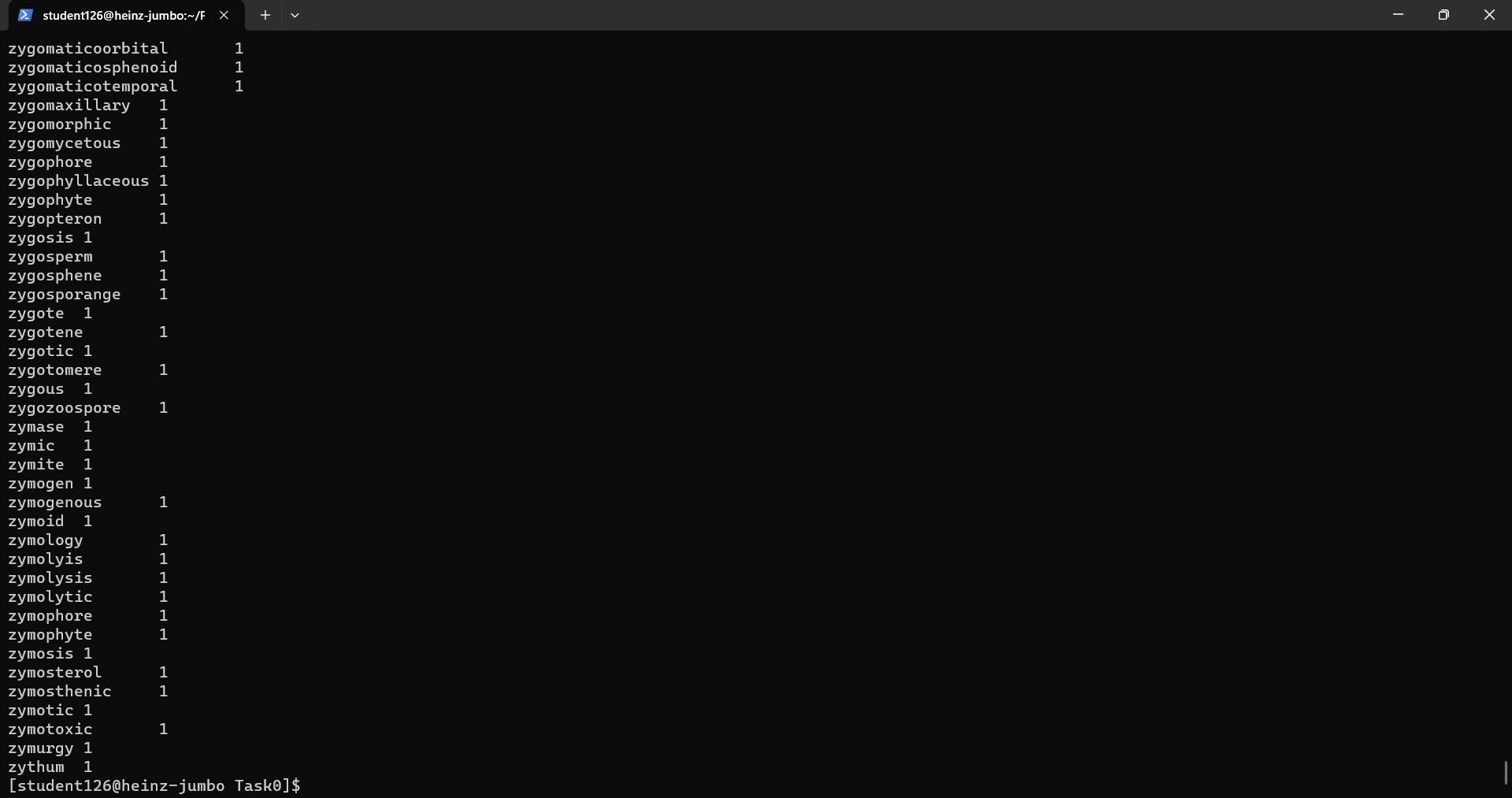
hadoop dfs -ls /user/student126/output/Task0

hadoop dfs -cat /user/student126/output/Task0/part-r-00000

合并本地目录：

hadoop dfs -getmerge /user/student126/output/Task0 ~/Project5/Part\_1/Task0/Task0Output

cat ~/Project5/Part\_1/Task0/Task0Output



**Task1**

cd /home/student126/Project5/Part\_1/Task1

touch LetterCounter.java

chmod 777 LetterCounter.java

mkdir LetterCounter\_classes

更改文件内容：nano /home/student126/Project5/Part\_1/Task1/LetterCounter.java

Java内容更新如下：

package org.myorg;

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 LetterCounter extends Configured implements Tool {

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

{

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

{

String line = value.toString();

StringTokenizer tokenizer = new StringTokenizer(line);

while(tokenizer.hasMoreTokens())

{

String nextWord = tokenizer.nextToken();

for (char c : nextWord.toCharArray()) {

String helper = String.*valueOf*(c);

word.set(helper);

context.write(word, *one*);

}

}

}

}

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

{

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

{

int sum = 0;

for(IntWritable value: values)

{

sum += value.get();

}

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

}

}

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

Job job = new Job(getConf());

job.setJarByClass(org.myorg.LetterCounter.class);

job.setJobName("wordcount");

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

job.setMapperClass(LetterCounterMap.class);

job.setCombinerClass(LetterCounterReducer.class);

job.setReducerClass(LetterCounterReducer.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 {

// TODO Auto-generated method stub

int result = ToolRunner.run(new org.myorg.LetterCounter(), args);

System.*exit*(result);

}

}

javac -classpath /usr/local/hadoop/hadoop-core-1.2.1.jar:./LetterCounter\_classes -d LetterCounter\_classes LetterCounter.java

jar -cvf LetterCounter.jar -C LetterCounter\_classes/ .

复制文件：

hadoop dfs -copyFromLocal /home/public/words.txt /user/student126/input/Task1

运行JAR：

hadoop jar LetterCounter.jar org.myorg.LetterCounter /user/student126/input/Task1 /user/student126/output/Task1

检查生成文件：

hadoop dfs -ls /user/student126/output/Task1

hadoop dfs -cat /user/student126/output/Task1/part-r-00000

合并本地目录：

hadoop dfs -getmerge /user/student126/output/Task1 ~/Project5/Part\_1/Task1/Task1Output

排序：

sort -k 2nr Task1Output -o Task1Output

cat ~/Project5/Part\_1/Task1/Task1Output

**Task2**

touch FindPattern.java

chmod 777 FindPattern.java

mkdir FindPattern\_classes

更改文件内容：nano /home/student126/Project5/Part\_1/Task2/FindPattern.java

// ======================= FindPattern.java ==========================================

package org.myorg;

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.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 {

public static class FindPatternMap extends Mapper<LongWritable, Text, Text, Text> {

private Text word = new Text();

@Override

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

String line = value.toString();

StringTokenizer tokenizer = new StringTokenizer(line);

while (tokenizer.hasMoreTokens()) {

String currentWord = tokenizer.nextToken();

if (currentWord.toLowerCase().contains("fun")) {

word.set(currentWord);

context.write(word, new Text(""));

}

}

}

}

public static class FindPatternReducer extends Reducer<Text, Text, Text, Text> {

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

// Output the word that contains the string "fun"

context.write(key, new Text(""));

}

}

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

Job job = new Job(getConf());

job.setJarByClass(FindPattern.class);

job.setJobName("findpattern");

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

job.setMapperClass(FindPatternMap.class);

job.setReducerClass(FindPatternReducer.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 FindPattern(), args);

System.*exit*(result);

}

}

javac -classpath /usr/local/hadoop/hadoop-core-1.2.1.jar:./FindPattern\_classes -d FindPattern\_classes FindPattern.java

jar -cvf FindPattern.jar -C FindPattern\_classes/ .

复制文件：

hadoop dfs -copyFromLocal /home/public/words.txt /user/student126/input/Task2

运行JAR：

hadoop jar FindPattern.jar org.myorg.FindPattern /user/student126/input/Task2 /user/student126/output/Task2

检查生成文件：

hadoop dfs -ls /user/student126/output/Task2

hadoop dfs -cat /user/student126/output/Task2/part-r-00000

合并本地目录：

hadoop dfs -getmerge /user/student126/output/Task2 ~/Project5/Part\_1/Task2/Task2Output

排序：

sort -k 2nr Task1Output -o Task1Output

cat ~/Project5/Part\_1/Task2/Task2Output

**Task3**

cp -R /home/public/MaxTemperature.java /home/student126/Project5/Part\_1/Task3

cp -R /home/public/MaxTemperatureMapper.java /home/student126/Project5/Part\_1/Task3

cp -R /home/public/MaxTemperatureReducer.java /home/student126/Project5/Part\_1/Task3

mkdir temperature\_classes

javac -classpath /usr/local/hadoop/hadoop-core-1.2.1.jar:./temperature\_classes -d temperature\_classes MaxTemperatureMapper.java

javac -classpath /usr/local/hadoop/hadoop-core-1.2.1.jar:./temperature\_classes -d temperature\_classes MaxTemperatureReducer.java

javac -classpath /usr/local/hadoop/hadoop-core-1.2.1.jar:./temperature\_classes -d temperature\_classes MaxTemperature.java

jar -cvf temperature.jar -C temperature\_classes/ .

复制文件：

hadoop dfs -copyFromLocal /home/public/combinedYears.txt /user/student126/input/Task3

运行JAR：

hadoop jar temperature.jar edu.cmu.andrew.mm6.MaxTemperature /user/student126/input/Task3 /user/student126/output/Task3

检查生成文件：

hadoop dfs -ls /user/student126/output/Task3

hadoop dfs -cat /user/student126/output/Task3/part-00000

hadoop dfs -cat /user/student126/output/Task3/part-00001

hadoop dfs -cat /user/student126/output/Task3/part-00002

合并本地目录：

hadoop dfs -getmerge /user/student126/output/Task3 ~/Project5/Part\_1/Task3/Task3Output

cat ~/Project5/Part\_1/Task3/Task3Output

**Task4**

touch MinTemperature.java

chmod 777 MinTemperature.java

nano MinTemperature.java

touch MinTemperatureMapper.java

chmod 777 MinTemperatureMapper.java

nano MinTemperatureMapper.java

touch MinTemperatureReducer.java

chmod 777 MinTemperatureReducer.java

nano MinTemperatureReducer.java

mkdir temperature\_classes

javac -classpath /usr/local/hadoop/hadoop-core-1.2.1.jar:./temperature\_classes -d temperature\_classes MinTemperatureMapper.java

javac -classpath /usr/local/hadoop/hadoop-core-1.2.1.jar:./temperature\_classes -d temperature\_classes MinTemperatureReducer.java

javac -classpath /usr/local/hadoop/hadoop-core-1.2.1.jar:./temperature\_classes -d temperature\_classes MinTemperature.java

jar -cvf temperature.jar -C temperature\_classes/ .

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

/\*\*

\* MinTemperature.java

\*

\* This is a Hadoop MapReduce program that calculates the minimum temperature from a dataset.

\* The program uses MapReduce to process the data and find the minimum temperature for each year.

\*

\* @goal Calculate the minimum temperature for each year in the given dataset.

\* @param args Command line arguments: <input path> <output path>

\* - <input path>: Path to the input dataset.

\* - <output path>: Path to store the output results.

\* @author Xinyuan Yang(xy3)

\* @email xy3@andrew.cmu.edu

\* Work Cited:https://github.com/CMU-Heinz-95702/Project5

\*/

package edu.cmu.andrew.mm6;

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 java.io.\*;

public class MinTemperature {

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

if (args.length != 2) {

System.err.println("Usage: MinTemperature <input path> <output path>");

System.exit(-1);

}

JobConf conf = new JobConf(edu.cmu.andrew.mm6.MinTemperature.class);

conf.setJobName("Min temperature");

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

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

conf.setMapperClass(MinTemperatureMapper.class);

conf.setReducerClass(MinTemperatureReducer.class);

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

JobClient.runJob(conf);

}

}

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

/\*\*

\* MinTemperatureMapper.java

\*

\* This is the Mapper class for a Hadoop MapReduce program that calculates the minimum temperature from a dataset.

\*

\* The Mapper processes each line of the input data, extracts the year and temperature,

\* and outputs key-value pairs with the year as the key and the temperature as the value.

\*

\* @goal Process input data and emit intermediate key-value pairs with the year as the key

\* and the temperature as the value for further processing by the reducer.

\* @param key The file offset (ignored in this case).

\* @param value The line of input data containing temperature readings.

\* @param output The output collector for emitting intermediate key-value pairs.

\* @param reporter The reporter for reporting progress or status information to the Hadoop framework.

\* @throws IOException Signals that an I/O exception has occurred.

\*

\* @author Xinyuan Yang(xy3)

\* @email xy3@andrew.cmu.edu

\* Work Cited:https://github.com/CMU-Heinz-95702/Project5

\*/

package edu.cmu.andrew.mm6;

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) == '+') { // 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)); }

}

}

// =========== MinTemperatureReducer.java ====================================================

/\*\*

\* MinTemperatureReducer.java

\*

\* This is the Reducer class for a Hadoop MapReduce program that calculates the minimum temperature from a dataset.

\*

\* The Reducer processes intermediate key-value pairs, where the key is the year and the values

\* are a list of temperatures. It finds the minimum temperature for each year and emits

\* key-value pairs with the year as the key and the minimum temperature as the value.

\*

\* @goal Process intermediate key-value pairs and emit final key-value pairs with the year as the key

\* and the minimum temperature as the value.

\* @param key The year as the key.

\* @param values An iterator over the list of temperatures for the given year.

\* @param output The output collector for emitting final key-value pairs.

\* @param reporter The reporter for reporting progress or status information to the Hadoop framework.

\* @throws IOException Signals that an I/O exception has occurred.

\*

\* @author Xinyuan Yang(xy3)

\* @email xy3@andrew.cmu.edu

\* Work Cited:https://github.com/CMU-Heinz-95702/Project5

\*/

package edu.cmu.andrew.mm6;

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));

}

}

复制文件：

hadoop dfs -copyFromLocal /home/public/combinedYears.txt /user/student126/input/Task4

运行JAR：

hadoop jar temperature.jar edu.cmu.andrew.mm6.MinTemperature /user/student126/input/Task4 /user/student126/output/Task4

检查生成文件：

hadoop dfs -ls /user/student126/output/Task4

hadoop dfs -cat /user/student126/output/Task4/part-00000

hadoop dfs -cat /user/student126/output/Task4/part-00001

hadoop dfs -cat /user/student126/output/Task4/part-00002

合并本地目录：

hadoop dfs -getmerge /user/student126/output/Task4 ~/Project5/Part\_1/Task4/Task4Output

cat ~/Project5/Part\_1/Task4/Task4Output

**Task5**

mkdir Crime\_classes

cp /home/public/P1V.txt /home/studentXXX/Project5/Part\_1/Task5

hadoop dfs -copyFromLocal /home/student126/Project5/Part\_1/Task5/P1V.txt /user/student126/input/Task5/P1V.txt

javac -classpath /usr/local/hadoop/hadoop-core-1.2.1.jar:./Crime\_classes -d Crime\_classes Crime.java

jar -cvf Crime.jar -C Crime\_classes/ .

hadoop jar /home/student126/Project5/Part\_1/Task5/Crime.jar org.myorg.CountCrime /user/student126/input/Task5/P1V.txt /user/student126/output/Task5

hadoop dfs -getmerge /user/studentXXX/Task5 /home/studentXXX/Project5/Part\_1/Task5/Task5Output

package org.myorg;

/\*\*

\* CountCrime is a MapReduce program that counts the occurrences of specific crimes

\* such as "Aggravated Assault" and "Robbery" in the input data.

\*

\* @goal Count occurrences of specific crimes in a dataset.

\* @param args Command line arguments, where args[0] is the input path and args[1] is the output path.

\* @author Xinyuan Yang(xy3)

\* @email xy3@andrew.cmu.edu

\* Work Cited: https://github.com/CMU-Heinz-95702/Project5

\*/

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.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 CountCrime extends Configured implements Tool {

/\*\*

\* CountCrimeMap is the Mapper class that processes input records and emits key-value pairs.

\*/

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

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

private Text word = new Text("#TotalofCrime");

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

String[] helper = value.toString().split("\\t");

if (helper.length > 4) {

String crime = helper[4].toUpperCase();

if (crime.equalsIgnoreCase("AGGRAVATED ASSAULT") || crime.equalsIgnoreCase("ROBBERY")) {

context.write(word, one);

}

}

}

}

/\*\*

\* CountCrimeReducer is the Reducer class that aggregates intermediate key-value pairs.

\*/

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

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

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

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

}

}

/\*\*

\* Run the MapReduce job to count the occurrences of specific crimes.

\*

\* @param args Command line arguments, where args[0] is the input path and args[1] is the output path.

\* @return 0 if the job completes successfully, 1 otherwise.

\* @throws Exception If an error occurs during job execution.

\*/

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

Job job = new Job(getConf());

job.setJarByClass(CountCrime.class);

job.setJobName("CountCrime");

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

job.setMapperClass(CountCrimeMap.class);

job.setReducerClass(CountCrimeReducer.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;

}

/\*\*

\* Main method to run the CountCrime MapReduce job.

\*/

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

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

System.exit(result);

}

}

hadoop dfs -getmerge /user/student126/output/Task5 ~/Project5/Part\_1/Task4/Task5Output

cat ~/Project5/Part\_1/Task4/Task5Output

**Task6**

cd Task6

touch CountCrime.java

chmod 777 CountCrime.java

nano CountCrime.java

package org.myorg;

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.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;

/\*\*

\* CountCrime is a MapReduce program that counts the occurrences of specific crimes

\* such as "Aggravated Assault" within a specified distance from a target location.

\*

\* @goal Count occurrences of specific crimes within proximity to a target location.

\* @param args Command line arguments, where args[0] is the input path and args[1] is the output path.

\* @author Xinyuan Yang(xy3)

\* @email xy3@andrew.cmu.edu

\* Work Cited: Feet to meters conversion: https://www.google.com/search?q=feet+to+meters&oq=feet+to&gs\_lcrp=EgZjaHJvbWUqCggAEAAYsQMYgAQyCggAEAAYsQMYgAQyCggBEAAYsQMYgAQyBggCEEUYOTIKCAMQABixAxiABDIKCAQQABixAxiABDIKCAUQABixAxiABDIKCAYQABixAxiABDIHCAcQABiABDIKCAgQABixAxiABDIHCAkQABiABKgCALACAA&sourceid=chrome&ie=UTF-8

\*/

public class CountCrime extends Configured implements Tool {

private static final double TARGET\_X = 1354326.897;

private static final double TARGET\_Y = 411447.7828;

private static final double MAX\_DISTANCE\_METERS = 350;

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

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

private Text word = new Text("#TotalofCrime occurred within 350 meters of 3803 Forbes Avenue in Oakland");

/\*\*

\* Maps input key-value pairs to intermediate key-value pairs.

\*

\* @param key Input key (line number).

\* @param value Input value (line of text).

\* @param context Context object to write output key-value pairs.

\* @throws IOException If an I/O error occurs.

\* @throws InterruptedException If the task is interrupted.

\*/

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

String[] helper = value.toString().split("\\t");

if (helper.length > 4) {

String crime = helper[4].toUpperCase();

if (crime.equalsIgnoreCase("AGGRAVATED ASSAULT") && isWithinDistance(helper)) {

context.write(word, one);

}

}

}

/\*\*

\* Checks if a crime occurred within the specified distance from the target location.

\*

\* @param data Array of crime data.

\* @return True if the crime occurred within distance, false otherwise.

\*/

private boolean isWithinDistance(String[] data) {

if (data.length >= 2) {

double x = Double.parseDouble(data[0]);

double y = Double.parseDouble(data[1]);

double distance = calculateDistance(x, y, TARGET\_X, TARGET\_Y);

//Work Cited: Feet to meters:https://www.google.com/search?q=feet+to+meters&oq=feet+to&gs\_lcrp=EgZjaHJvbWUqCggAEAAYsQMYgAQyCggAEAAYsQMYgAQyCggBEAAYsQMYgAQyBggCEEUYOTIKCAMQABixAxiABDIKCAQQABixAxiABDIKCAUQABixAxiABDIKCAYQABixAxiABDIHCAcQABiABDIKCAgQABixAxiABDIHCAkQABiABKgCALACAA&sourceid=chrome&ie=UTF-8

return distance \* 0.3048 <= MAX\_DISTANCE\_METERS;

}

return false;

}

/\*\*

\* Calculates the distance between two points.

\*

\* @param x1 X-coordinate of the first point.

\* @param y1 Y-coordinate of the first point.

\* @param x2 X-coordinate of the second point.

\* @param y2 Y-coordinate of the second point.

\* @return Euclidean distance between the two points.

\*/

private double calculateDistance(double x1, double y1, double x2, double y2) {

// Pythagorean theorem to calculate distance

return Math.pow((x1-x2)\*(x1-x2) + (y1-y2)\*(y1-y2),0.5);

}

}

/\*\*

\* Reducer class that aggregates intermediate key-value pairs.

\*/

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

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

throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

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

}

}

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

Job job = new Job(getConf());

job.setJarByClass(CountCrime.class);

job.setJobName("CountCrime");

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

job.setMapperClass(CountCrimeMap.class);

job.setReducerClass(CountCrimeReducer.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 CountCrime(), args);

System.exit(result);

}

}

mkdir Crime\_classes

cp /home/public/P1V.txt /home/student126/Project5/Part\_1/Task6

hadoop dfs -copyFromLocal /home/student126/Project5/Part\_1/Task6/P1V.txt /user/student126/input/Task6/P1V.txt

javac -classpath /usr/local/hadoop/hadoop-core-1.2.1.jar:./Crime\_classes -d Crime\_classes CountCrime.java

jar -cvf Crime.jar -C Crime\_classes/ .

hadoop jar /home/student126/Project5/Part\_1/Task6/Crime.jar org.myorg.CountCrime /user/student126/input/Task6/P1V.txt /user/student126/output/Task6

hadoop dfs -ls /user/student126/output/Task6

hadoop dfs -cat /user/student126/output/Task6/part-r-00000

hadoop dfs -cat /user/student126/output/Task6/part-r-00001

hadoop dfs -cat /user/student126/output/Task6/part-r-00002

hadoop dfs -getmerge /user/student126/output/Task6 ~/Project5/Part\_1/Task6/Task6Output

cat ~/Project5/Part\_1/Task6/Task6Output

Task7

package org.myorg;

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.\*;

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;

/\*\*

\* CountCrime is a MapReduce program that counts the occurrences of specific crimes

\* such as "Aggravated Assault" within a specified distance from a target location.

\*

\* @goal Count occurrences of specific crimes within proximity to a target location.

\* @param args Command line arguments, where args[0] is the input path and args[1] is the output path.

\* @author Xinyuan Yang(xy3)

\* @email xy3@andrew.cmu.edu

\* Work Cited: https://earth.google.com/web/@36.0069993,123.19495793,-11130.55352784a,22262883.32198381d,35y,174.03555629h,0t,0r/data=OgMKATA

\* Work Cited: https://github.com/CMU-Heinz-95702/Project5

\* Work Cited: Feet to meters conversion: https://www.google.com/search?q=feet+to+meters&oq=feet+to&gs\_lcrp=EgZjaHJvbWUqCggAEAAYsQMYgAQyCggAEAAYsQMYgAQyCggBEAAYsQMYgAQyBggCEEUYOTIKCAMQABixAxiABDIKCAQQABixAxiABDIKCAUQABixAxiABDIKCAYQABixAxiABDIHCAcQABiABDIKCAgQABixAxiABDIHCAkQABiABKgCALACAA&sourceid=chrome&ie=UTF-8

\*/

public class CountCrime extends Configured implements Tool {

private static final double TARGET\_X = 1354326.897;

private static final double TARGET\_Y = 411447.7828;

private static final double MAX\_DISTANCE\_METERS = 350;

public static class CountCrimeMap extends Mapper<LongWritable, Text, NullWritable, Text> {

/\*\*

\* Maps input key-value pairs to intermediate key-value pairs.

\*

\* @param key Input key (line number).

\* @param value Input value (line of text).

\* @param context Context object to write output key-value pairs.

\* @throws IOException If an I/O error occurs.

\* @throws InterruptedException If the task is interrupted.

\*/

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

// Parsing the text

String[] helper = value.toString().split("\\t");

if (helper.length > 8) {

try {

double x = Double.parseDouble(helper[0]);

double y = Double.parseDouble(helper[1]);

String crime = helper[4].toUpperCase();

double latitude = Double.parseDouble(helper[7]);

double longitude = Double.parseDouble(helper[8]);

if (crime.equals("AGGRAVATED ASSAULT") && isWithinDistance(helper)) {

String mark = KMLmark(latitude, longitude, "Attached to the ground. Intelligently places itself at the height of the underlying terrain.");

context.write(NullWritable.get(), new Text(mark));

}

} catch (NumberFormatException e) {

}

}

}

/\*\*

\* Checks if a crime occurred within the specified distance from the target location.

\*

\* @param data Array of crime data.

\* @return True if the crime occurred within distance, false otherwise.

\*/

private boolean isWithinDistance(String[] data) {

if (data.length >= 2) {

double x = Double.parseDouble(data[0]);

double y = Double.parseDouble(data[1]);

double distance = calculateDistance(x, y, TARGET\_X, TARGET\_Y);

//Work Cited: Feet to meters:https://www.google.com/search?q=feet+to+meters&oq=feet+to&gs\_lcrp=EgZjaHJvbWUqCggAEAAYsQMYgAQyCggAEAAYsQMYgAQyCggBEAAYsQMYgAQyBggCEEUYOTIKCAMQABixAxiABDIKCAQQABixAxiABDIKCAUQABixAxiABDIKCAYQABixAxiABDIHCAcQABiABDIKCAgQABixAxiABDIHCAkQABiABKgCALACAA&sourceid=chrome&ie=UTF-8

return distance \* 0.3048 <= MAX\_DISTANCE\_METERS;

}

return false;

}

/\*\*

\* Calculates the distance between two points.

\*

\* @param x1 X-coordinate of the first point.

\* @param y1 Y-coordinate of the first point.

\* @param x2 X-coordinate of the second point.

\* @param y2 Y-coordinate of the second point.

\* @return Euclidean distance between the two points.

\*/

private double calculateDistance(double x1, double y1, double x2, double y2) {

// Pythagorean theorem to calculate distance

return Math.pow((x1-x2)\*(x1-x2) + (y1-y2)\*(y1-y2),0.5);

}

private String KMLmark(double latitude, double longitude, String sentence) {

return String.format("<Placemark>\n<name>Simple placemark</name>\n<description>%s</description><Point><coordinates>%f,%f,0</coordinates></Point></Placemark>",

sentence, longitude, latitude);

}

}

/\*\*

\* The reduce method generates a complete KML (Keyhole Markup Language) document by combining an XML header, KML start tags, the content provided in the values, and KML end tags.

\*/

public static class CrimeReducer extends Reducer<NullWritable, Text, NullWritable, Text>

{

//Work Cited: I first created a blank KML from the website Google Earth, and then

//I downloaded the file, and I formatted the following code with it.

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

String xmlHeader = "<?xml version=\"1.0\" encoding=\"UTF-8\"?>";

String kmlStart = "<kml xmlns=\"http://www.opengis.net/kml/2.2\">";

String documentStart = "<Document>";

// Writing XML header and KML start tags

context.write(NullWritable.get(), new Text(xmlHeader));

context.write(NullWritable.get(), new Text(kmlStart));

context.write(NullWritable.get(), new Text(documentStart));

// Writing content from values

for (Text val : values) {

context.write(NullWritable.get(), val);

}

// Writing KML end tags

String documentEnd = "</Document>";

String kmlEnd = "</kml>";

context.write(NullWritable.get(), new Text(documentEnd));

context.write(NullWritable.get(), new Text(kmlEnd));

}

}

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

Job job = new Job(getConf());

job.setJarByClass(CountCrime.class);

job.setJobName("CountCrimeKML");

job.setMapperClass(CountCrimeMap.class);

job.setReducerClass(CrimeReducer.class);

job.setOutputKeyClass(NullWritable.class);

job.setOutputValueClass(Text.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 CountCrime(), args);

System.exit(result);

}

}