**Exercise - 4**

**wordcount.java**

import org.apache.hadoop.fs.Path;

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

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

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

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

public class WordCount extends Configured implements Tool {

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

JobConf conf = new JobConf(getConf(), WordCount.class);

conf.setJobName("WordCount");

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

conf.setMapperClass(WordCountMapper.class);

conf.setReducerClass(WordCountReducer.class);

Path inp = new Path(args[0]);

Path out = new Path(args[1]);

FileInputFormat.addInputPath(conf, inp);

FileOutputFormat.setOutputPath(conf, out);

JobClient.runJob(conf);

return 0;

}

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

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

System.exit(res);

}

}

**wordCountMapper.java**

import java.io.IOException;

import java.util.StringTokenizer;

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

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

public class WordCountMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {

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

private Text word = new Text();

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

String line = value.toString();

StringTokenizer tokenizer = new StringTokenizer(line);

while (tokenizer.hasMoreTokens()) {

word.set(tokenizer.nextToken());

output.collect(word, one);

}

}

}

**wordCountReducer.java**

import java.io.IOException;

import java.util.Iterator;

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

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

public class WordCountReducer extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {

public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {

int sum = 0;

while (values.hasNext()) {

sum += values.next().get();

}

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

}

}

**Weather(5)**

package bda;

import java.io.IOException;

import java.util.Iterator;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

import org.apache.hadoop.conf.Configuration;

public class MyMaxMin {

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

public static final int MISSING = 9999;

@Override

public void map(LongWritable arg0, Text Value, Context context) throws IOException, InterruptedException {

String line = Value.toString();

if (!(line.length() == 0)) {

String date = line.substring(6, 14);

float temp\_Max = Float.parseFloat(line.substring(39, 45).trim());

float temp\_Min = Float.parseFloat(line.substring(47, 53).trim());

if (temp\_Max > 30.0) {

context.write(new Text("The Day is Hot Day :" + date),

new Text(String.valueOf(temp\_Max)));

}

if (temp\_Min < 15) {

context.write(new Text("The Day is Cold Day :" + date),

new Text(String.valueOf(temp\_Min)));

}

}

}

}

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

public void reduce(Text Key, Iterator<Text> Values, Context context)

throws IOException, InterruptedException {

String temperature = Values.next().toString();

context.write(Key, new Text(temperature));

}

}

@SuppressWarnings("deprecation")

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

Configuration conf = new Configuration();

Job job = new Job(conf, "weather example");

job.setJarByClass(MyMaxMin.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(Text.class);

job.setMapperClass(MaxTemperatureMapper.class);

job.setReducerClass(MaxTemperatureReducer.class);

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

Path OutputPath = new Path(args[1]);

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

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

OutputPath.getFileSystem(conf).delete(OutputPath);

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

**Shortest path (6)**

package shortpath;

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.util.HashMap;

import java.util.Iterator;

import org.apache.hadoop.fs.FileSystem;

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.\*;

import org.apache.hadoop.mapreduce.lib.output.\*;

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

public class Dijkstra extends Configured implements Tool {

public static String OUT = "outfile";

public static String IN = "inputlarger";

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

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

Text word = new Text();

String line = value.toString();

String[] sp = line.split(" ");

int distanceadd = Integer.parseInt(sp[1]) + 1;

String[] PointsTo = sp[2].split(":");

for (int i = 0; i < PointsTo.length; i++) {

word.set("VALUE " + distanceadd);

context.write(new LongWritable(Integer.parseInt(PointsTo[i])), word);

word.clear();

}

word.set("VALUE " + sp[1]);

context.write(new LongWritable(Integer.parseInt(sp[0])), word);

word.clear();

word.set("NODES " + sp[2]);

context.write(new LongWritable(Integer.parseInt(sp[0])), word);

word.clear();

}

}

public static class TheReducer extends Reducer<LongWritable, Text, LongWritable, Text> {

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

String nodes = "UNMODED";

Text word = new Text();

int lowest = 10009;

for (Text val : values) {

String[] sp = val.toString().split(" ");

if (sp[0].equalsIgnoreCase("NODES")) {

nodes = null;

nodes = sp[1];

} else if (sp[0].equalsIgnoreCase("VALUE")) {

int distance = Integer.parseInt(sp[1]);

lowest = Math.min(distance, lowest);

}

}

word.set(lowest + " " + nodes);

context.write(key, word);

word.clear();

}

}

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

getConf().set("mapred.textoutputformat.separator", " ");

IN = args[0];

OUT = args[1];

String infile = IN;

String outputfile = OUT + System.nanoTime();

boolean isdone = false;

boolean success = false;

HashMap<Integer, Integer> \_map = new HashMap<Integer, Integer>();

while (isdone == false) {

Job job = new Job(getConf());

job.setJarByClass(Dijkstra.class);

job.setJobName("Dijkstra");

job.setOutputKeyClass(LongWritable.class);

job.setOutputValueClass(Text.class);

job.setMapperClass(TheMapper.class);

job.setReducerClass(TheReducer.class);

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(TextOutputFormat.class);

FileInputFormat.addInputPath(job, new Path(infile));

FileOutputFormat.setOutputPath(job, new Path(outputfile));

success = job.waitForCompletion(true);

if (infile != IN) {

String indir = infile.replace("part-r-00000", "");

Path ddir = new Path(indir);

FileSystem dfs = FileSystem.get(ddir.toUri(), getConf());

dfs.delete(ddir, true);

}

infile = outputfile + "/part-r-00000";

outputfile = OUT + System.nanoTime();

isdone = true;

Path ofile = new Path(infile);

FileSystem fs = FileSystem.get(ofile.toUri(), new Configuration());

BufferedReader br = new BufferedReader(new InputStreamReader(fs.open(ofile)));

HashMap<Integer, Integer> imap = new HashMap<Integer, Integer>();

String line = br.readLine();

while (line != null) {

String[] sp = line.split(" ");

int node = Integer.parseInt(sp[0]);

int distance = Integer.parseInt(sp[1]);

imap.put(node, distance);

line = br.readLine();

}

if (\_map.isEmpty()) {

isdone = false;

} else {

Iterator<Integer> itr = imap.keySet().iterator();

while (itr.hasNext()) {

int key = itr.next();

int val = imap.get(key);

if (\_map.get(key) != val) {

isdone = false;

}

}

}

if (isdone == false) {

\_map.putAll(imap);

}

}

return success ? 0 : 1;

}

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

System.exit(ToolRunner.run(new Dijkstra(), args));

}

}

**Friends of friends(7)**

package bda;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

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

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.output.FileOutputFormat;

import java.util.\*;

public class MutualFriends {

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

Text user = new Text();

Text friends = new Text();

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

throws IOException, InterruptedException {

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

String userId = split[0];

if (split.length == 1) {

return;

}

String[] friendIds = split[1].split(",");

for (String friend : friendIds) {

if (userId.equals(friend)) {

continue;

}

String userKey = (Integer.parseInt(userId) < Integer.parseInt(friend)) ?

userId + "," + friend : friend + "," + userId;

String regex = "((\\b" + friend + "[^\\w]+)|\\b,?" + friend + "$)";

friends.set(split[1].replaceAll(regex, ""));

user.set(userKey);

context.write(user, friends);

}

}

}

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

private String matchingFriends(String firstList, String secondList) {

if (firstList == null || secondList == null) {

return null;

}

String[] list1 = firstList.split(",");

String[] list2 = secondList.split(",");

LinkedHashSet<String> firstSet = new LinkedHashSet<String>();

for (String user : list1) {

firstSet.add(user);

}

LinkedHashSet<String> secondSet = new LinkedHashSet<String>();

for (String user : list2) {

secondSet.add(user);

}

firstSet.retainAll(secondSet);

return firstSet.toString().replaceAll("\\[|\\]", "");

}

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

throws IOException, InterruptedException {

String[] friendsList = new String[2];

int index = 0;

for (Text value : values) {

friendsList[index++] = value.toString();

}

String mutualFriends = matchingFriends(friendsList[0], friendsList[1]);

if (mutualFriends != null && mutualFriends.length() != 0) {

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

}

}

}

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

Configuration conf = new Configuration();

String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();

if (otherArgs.length != 2) {

System.err.println("Usage: MutualFriends <input file name> <output file name>");

System.exit(2);

}

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

job.setJarByClass(MutualFriends.class);

job.setMapperClass(Map.class);

job.setReducerClass(Reduce.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

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

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

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

**Page rank(8)**

package bigdata;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.DoubleWritable;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.NullWritable;

import org.apache.hadoop.io.ObjectWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class PageRankDriver extends Configured implements Tool {

public static double BETA = 0.8;

public static enum Counter {

totalNodes,

sumOfPageRanks;

}

@Override

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

if (args.length != 2) {

System.out.printf("Usage: " + this.getClass().getName() + " <input dir> <output dir>\n");

return -1;

}

int iterationCount = 0;

while (true) {

Job job\_s = Job.getInstance(getConf());

job\_s.setJarByClass(PageRankDriver.class);

job\_s.setJobName("S\_Calculation");

FileSystem fs = FileSystem.get(new Configuration());

fs.delete(new Path("/Temp"), true);

fs.delete(new Path(args[1]), true);

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

FileOutputFormat.setOutputPath(job\_s, new Path("/Temp"));

job\_s.setReducerClass(S\_Calculation\_Reducer.class);

job\_s.setMapperClass(S\_Calculation\_Mapper.class);

job\_s.setMapOutputKeyClass(IntWritable.class);

job\_s.setMapOutputValueClass(DoubleWritable.class);

job\_s.setOutputKeyClass(DoubleWritable.class);

job\_s.setOutputValueClass(NullWritable.class);

job\_s.setNumReduceTasks(1);

job\_s.waitForCompletion(true);

double sumOfPageRanks = (double) job\_s.getCounters().findCounter(Counter.sumOfPageRanks).getValue() / (double) 10000.0;

fs.delete(new Path("/Temp"), true);

Job job = Job.getInstance(getConf());

job.setJarByClass(PageRankDriver.class);

job.getConfiguration().setDouble("sumOfPageRanks", sumOfPageRanks);

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

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

job.setReducerClass(PageRankReducer.class);

job.setMapperClass(PageRankMapper.class);

job.setMapOutputKeyClass(Text.class);

job.setMapOutputValueClass(ObjectWritable.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(Text.class);

boolean success = job.waitForCompletion(true);

iterationCount++;

if (iterationCount > 2) {

fs.delete(new Path(args[0]), true);

System.out.println("Total Number of Times the Job was run= " + iterationCount);

System.out.println("total Nodes Processed: " + job.getCounters().findCounter(Counter.totalNodes).getValue());

return (success == true ? 1 : 0);

} else {

fs.delete(new Path(args[0]), true);

fs.rename(new Path(args[1]), new Path(args[0]));

}

}

}

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

int exitCode = ToolRunner.run(new Configuration(), new PageRankDriver(), args);

System.exit(exitCode);

}

}