VE572 — Methods and Tools for Big Data

Midterm Rubric — Summer 2019

TA: Yihao and Yanjun (UM-JI)

This document should be kept secret.

Exercise 2 — MapReduce

1. Determine all the FOFs in the following toy example.

```
Ali,Han,Eve,Dan,Fin
Han,Ali,Eve,Gil,Ben
Ben,Han,Gil,Col
Eve,Ali,Han
Gil,Han,Ben,Dan
Fin,Ali,Dan,Col
Dan,Ali,Fin,Gil
Col,Fin,Ben
```

2. Write the Hadoop pseudocode for the first MapReduce Job. Assume a simple input text file with a list of names on each line, the user as first field followed by all his friends. For the output we expect a simple text file where each line is composed of a user and a FOF followed by the number of friends they have in common.

```
package com.ve572.e1;

import org.apache.commons.text.StringTokenizer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;
import java.util.ArrayList;

public class FindFOF {
    private static String formPair(final String a, final String b) {
```

```
20          if (a.compareTo(b) < 0) return a + "," + b;
21          return b + "," + a;
22      }
23
24      public static class Map1 extends Mapper<Object, Text, Text, Text> {
25          private Text resultKey = new Text();
26          private Text resultValue = new Text();
27
28          public void map(Object key, Text value, Context context) throws IOException,
            ↪  InterruptedException {
29              StringTokenizer tokenizer = new StringTokenizer(value.toString(), ",");
30              resultKey.set(tokenizer.nextToken());
31              while (tokenizer.hasNext()) {
32                  resultValue.set(tokenizer.nextToken());
33                  context.write(resultKey, resultValue);
34              }
35          }
36      }
37
38
39      public static class Reduce1 extends Reducer<Text, Text, Text, IntWritable> {
40          private Text resultKey = new Text();
41          private IntWritable resultValue = new IntWritable();
42
43          public void reduce(Text key, Iterable<Text> values, Context context) throws
            ↪  IOException, InterruptedException {
44              ArrayList<String> arrayList = new ArrayList<>();
45              resultValue.set(0);
46              for (Text val : values) {
47                  resultKey.set(formPair(key.toString(), val.toString()));
48                  context.write(resultKey, resultValue);
49                  arrayList.add(val.toString());
50              }
51              resultValue.set(1);
52              for (int i = 0; i < arrayList.size(); i++) {
53                  for (int j = i + 1; j < arrayList.size(); j++) {
54                      resultKey.set(formPair(arrayList.get(i), arrayList.get(j)));
55                      context.write(resultKey, resultValue);
56                  }
57              }
58          }
59      }
60
61      public static class Map2 extends Mapper<Object, Text, Text, IntWritable> {
```

```
62          private Text resultKey = new Text();
63          private IntWritable resultValue = new IntWritable();
64
65          public void map(Object key, Text value, Context context) throws IOException,
            ↪  InterruptedException {
66              StringTokenizer tokenizer = new StringTokenizer(value.toString(), ",");
67              resultKey.set(tokenizer.nextToken() + " " + tokenizer.nextToken());
68              resultValue.set(Integer.parseInt(tokenizer.nextToken()));
69              context.write(resultKey, resultValue);
70          }
71      }
72
73      public static class Reduce2 extends Reducer<Text, IntWritable, Text, IntWritable> {
74          private IntWritable resultValue = new IntWritable();
75
76          public void reduce(Text key, Iterable<IntWritable> values, Context context)
            ↪  throws IOException, InterruptedException {
77              int count = 0;
78              for (IntWritable val : values) {
79                  if (val.get() == 0) return;
80                  count += val.get();
81              }
82              resultValue.set(count);
83              context.write(key, resultValue);
84          }
85      }
86
87      public static void main(String[] args) throws Exception {
88          Configuration conf = new Configuration();
89          conf.set("mapred.textoutputformat.separator", ",");
90  //          conf.set("mapreduce.output.textoutputformat.separator", ",");
91
92          Job job1 = Job.getInstance(conf, "ve572e1ex2.2");
93          job1.setJarByClass(FindFOF.class);
94          job1.setMapperClass(Map1.class);
95          job1.setReducerClass(Reduce1.class);
96
97          job1.setMapOutputValueClass(Text.class);
98          job1.setOutputKeyClass(Text.class);
99          job1.setOutputValueClass(IntWritable.class);
100
101         FileInputFormat.addInputPath(job1, new Path("data.txt"));
102         Path outputPath1 = new Path("output-1");
103         FileSystem fileSystem = outputPath1.getFileSystem(conf);
```

```
104             if (fileSystem.exists(outputPath1)) {
105                 fileSystem.delete(outputPath1, true);
106             }
107             FileOutputFormat.setOutputPath(job1, outputPath1);
108 
109             boolean exitCode = job1.waitForCompletion(true);
110             if (!exitCode) System.exit(1);
111 
112             conf.set("mapred.textoutputformat.separator", " ");
113 
114             Job job2 = Job.getInstance(conf, "ve572e1ex2.2");
115             job2.setJarByClass(FindFOF.class);
116             job2.setMapperClass(Map2.class);
117             job2.setReducerClass(Reduce2.class);
118 
119             job2.setOutputKeyClass(Text.class);
120             job2.setOutputValueClass(IntWritable.class);
121 
122             FileInputFormat.addInputPath(job2, outputPath1);
123             Path outputPath2 = new Path("output-2");
124             if (fileSystem.exists(outputPath2)) {
125                 fileSystem.delete(outputPath2, true);
126             }
127             FileOutputFormat.setOutputPath(job2, outputPath2);
128             exitCode = job2.waitForCompletion(true);
129 
130             System.exit(exitCode ? 0 : 1);
131         }
132 
133 }
```

3. Write the Hadoop pseudocode for the second MapReduce job. Assume the previous output file as input, and as output a simple text file where each line is composed of a user and all his FOF ordered with respect to the number of common friends; for each FOF also display the number of common friends.

```
1 package com.ve572.e1;
2 
3 import org.apache.commons.text.StringTokenizer;
4 import org.apache.hadoop.conf.Configuration;
5 import org.apache.hadoop.fs.FileSystem;
6 import org.apache.hadoop.fs.Path;
7 import org.apache.hadoop.io.Text;
8 import org.apache.hadoop.mapreduce.Job;
```

```java
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.io.IOException;
import java.util.ArrayList;
import java.util.Collections;

public class CountFOF {

    public static class Map extends Mapper<Object, Text, Text, Text> {
        private Text resultKey = new Text();
        private Text resultValue = new Text();

        public void map(Object key, Text value, Context context) throws IOException,
        ↪  InterruptedException {
            StringTokenizer tokenizer = new StringTokenizer(value.toString(), " ");
            String a = tokenizer.nextToken();
            String b = tokenizer.nextToken();
            String count = tokenizer.nextToken();
            resultKey.set(a);
            resultValue.set(b + " " + count);
            context.write(resultKey, resultValue);
            resultKey.set(b);
            resultValue.set(a + " " + count);
            context.write(resultKey, resultValue);
        }
    }

    public static class Reduce extends Reducer<Text, Text, Text, Text> {
        private Text resultValue = new Text();

        public void reduce(Text key, Iterable<Text> values, Context context) throws
        ↪  IOException, InterruptedException {
            ArrayList<String> arrayList = new ArrayList<>();
            for (Text val : values) {
                arrayList.add(val.toString());
            }
            Collections.sort(arrayList);
            resultValue.set(String.join(", ", arrayList));
            context.write(key, resultValue);
        }
    }
```

```
51
52      public static void main(String[] args) throws Exception {
53          Configuration conf = new Configuration();
54          conf.set("mapred.textoutputformat.separator", " ");
55  //          conf.set("mapreduce.output.textoutputformat.separator", ",");
56
57          Job job = Job.getInstance(conf, "ve572e1ex2.3");
58          job.setJarByClass(CountFOF.class);
59          job.setMapperClass(CountFOF.Map.class);
60          job.setReducerClass(CountFOF.Reduce.class);
61
62          job.setOutputKeyClass(Text.class);
63          job.setOutputValueClass(Text.class);
64
65          FileInputFormat.addInputPath(job, new Path("output-2"));
66          Path outputPath = new Path("output-3");
67          FileSystem fileSystem = outputPath.getFileSystem(conf);
68          if (fileSystem.exists(outputPath)) {
69              fileSystem.delete(outputPath, true);
70          }
71          FileOutputFormat.setOutputPath(job, outputPath);
72
73          boolean exitCode = job.waitForCompletion(true);
74          System.exit(exitCode ? 0 : 1);
75      }
76  }
```