School of Computer Science

COMP47470

# Lab 6
# Graph Processing with Hadoop MapReduce

| | |
|---|---|
| **Teaching Assistant:** | Patrick Cormac English |
| **Coordinator:** | Dr Anthony Ventresque |
| **Date:** | Wednesday 31st March, 2021 |
| **Total Number of Pages:** | 6 |

In this lab, you will first execute a simple graph processing algorithm on a simple graph and then you will implement your own algorithm(s).

This lab is based on the Docker container we created during Lab 4 for the WordCount exercise on Hadoop MapReduce. Get yourself familiar with Lab 4 if needed.

## 1 Graph Processing on Hadoop MapReduce

Download a simple graph in your container from here using `wget`. Only take the gz file not the full dataset. Make sure this is downloaded on your home directory (or make sure you know where it is for the rest of this practical).

The website `https://snap.stanford.edu/data/` contains lot of interesting graph datasets that I encourage you to explore if you're interested in graph processing.

The dataset you have downloaded contains information obtained from an email network. A line contains information in the format

$$< from\_vertex > \quad < to\_vertex >$$

which means that $< from\_vertex >$ has emailed $< to\_vertex >$.

Once you've downloaded the dataset, unzip the file:

```
$ gunzip email-Enron.txt.gz
```

and remove the first lines at the top of the file (the comments).

Download the dataset in HDFS (e.g. at `/graphs-input/email-Enron.txt`). and create 3 files for (i) a driver (main) class, (ii) a mapper class and (iii) a reducer class:
**MyDriver.java**

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
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.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

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

public class MyDriver {

    public static void main(String[] arg0) throws Exception{

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Give a nice name to the job
        ↪  here");
```

```
            job.setJarByClass(MyDriver.class);
            job.setMapperClass(MyMapper1.class);
            job.setReducerClass(MyReducer1.class);
            job.setMapOutputKeyClass(LongWritable.class);
            job.setMapOutputValueClass(Text.class);
            FileInputFormat.addInputPath(job, new Path(arg0[0]));
            FileOutputFormat.setOutputPath(job, new Path(arg0[1]));

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

**MyMapper1.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.mapreduce.Mapper;

    public class MyMapper1 extends Mapper<LongWritable, Text, LongWritable,
    ↪   Text> {

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

        @Override
        protected void map(LongWritable key, Text value, Context context)
        ↪   throws IOException, InterruptedException {


            long fromNode, toNode;
            fromNode = toNode = -1;

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

            try{
            if(tokens.length == 2){
                fromNode = Long.parseLong(tokens[0]);
                toNode = Long.parseLong(tokens[1]);
            }
            }catch(NumberFormatException npe){
                System.err.println("Non numeric Node ; ignoring it");
                npe.getCause();
                return;
            }

            if(fromNode != -1 && toNode != -1){
                context.write(new LongWritable(toNode), new
                ↪   Text(String.valueOf(fromNode)));
            }

        }
    }
```

**MyReducer1.java**

```
    import java.io.IOException;
    import org.apache.hadoop.io.LongWritable;
    import org.apache.hadoop.io.Text;
    import org.apache.hadoop.mapreduce.Reducer;

    public class MyReducer1 extends Reducer<LongWritable,Text,
     ↪   LongWritable, Text>{


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

            StringBuilder nNodes = new StringBuilder();
            if(key.get() == 1)
                 nNodes.append("(O),");
            else
                nNodes.append("(I),");
            for(Text n: values){
                nNodes.append(n.toString()).append(",");
            }
            context.write(key, new Text(nNodes.toString()));
        }

    }
```

Compile, package and execute your code:

```
$ export HADOOP_CLASSPATH=$JAVA_HOME/lib/tools.jar
$ hadoop com.sun.tools.javac.Main My*.java
$ jar cf graph.jar My*.class
$ hadoop jar graph.jar MyDriver /graphs-input/email-Enron.txt output1
```

(Remember that you have to use a non-existent output repository name when you run your Hadoop job)
Check the output:

```
$ hdfs dfs -cat output1/part-r-00000
```

What does this algorithm do?

## 2 Try your own Algorithm(s)

Now write your own algorithm(s) on the simple graph. We give you some examples here:

- reverse the adjacency list representing the graph. It is currently "from-to", make it "to-from".
- eliminate some of the nodes in the graph, for instance the ones with odd IDs
- eliminate some of the edges in the graph, for instance those between two odd numbers.

**Solutions:**

- reverse the adjacency list representing the graph. It is currently "from-to", make it "to-from".

  For this question we can change either the mapper or reducer to swap the key and value. If modifying the mapper then we can just use the identity reducer. Let's illustrate how we can modify the mapper:

  **MyReducer1.java**

```
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MyReducer1 extends Reducer<LongWritable,Text,
↪  LongWritable, Text>{


    protected void reduce(LongWritable key, Iterable<Text> values,
            Context context) throws IOException,
            ↪  InterruptedException {
        for(Text n: values){
            context.write(new
            ↪  LongWritable(Long.parseLong(n.toString())), new
            ↪  Text(key.toString()));
        }

    }
}
```

- eliminate some of the nodes in the graph, for instance the ones with odd IDs

  To eliminate nodes we can filter them in the reducer. For example, filtering out odd nodes:

  **MyReducer1.java**

```
import java.io.IOException;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MyReducer1 extends Reducer<LongWritable,Text,
↪  LongWritable, Text>{
    protected void reduce(LongWritable key, Iterable<Text> values,
            Context context) throws IOException,
             ↪  InterruptedException {

        StringBuilder nNodes = new StringBuilder();
        if(key.get() % 2 == 0) {
            for(Text n: values){
                if(Long.parseLong(n.toString()) % 2 == 0)
                    nNodes.append(n.toString()).append(",");
            }
        context.write(key, new Text(nNodes.toString()));
        }
    }

}
```

- eliminate some of the edges in the graph, for instance those between two odd numbers.

  To eliminate edges we can filter them in the mapper. For example, filtering out edges between two odd nodes:

  **MyMapper1.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.mapreduce.Mapper;

    public class MyMapper1 extends Mapper<LongWritable, Text,
    ↪  LongWritable, Text> {

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

        @Override
        protected void map(LongWritable key, Text value, Context
        ↪  context) throws IOException, InterruptedException {

            long fromNode, toNode;
            fromNode = toNode = -1;

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

            try{
            if(tokens.length == 2){
                fromNode = Long.parseLong(tokens[0]);
                toNode = Long.parseLong(tokens[1]);
            }
            }catch(NumberFormatException npe){
                System.err.println("Non numeric Node ; ignoring it");
                npe.getCause();
                return;
            }

            if(fromNode != -1 && toNode != -1 && !(fromNode % 2 == 1
            ↪  && toNode % 2 == 1)){
                context.write(new LongWritable(toNode), new
                 ↪  Text(String.valueOf(fromNode)));
            }
        }
    }
```