Module 10 Lab: **Hadoop**

Q1. Analyzing a Graph with Hadoop/Java

Follow the **VMSetup instructions** to download, setup a preconfigured virtual machine (VM)image that you will use for this question.

Installing the Virtual Machine

Why using VWs? When developing earlier iterations of this lab, students installed software on their own machines, and we (both students and instructor team) ran into all sorts of issues, some of which could not be resolved satisfactorily. The VM approach allows students to start working on the tasks much more quickly. In future, Docker images may be used to further simplify the setup.

Loading Data into HDFS

Now, let us load our dataset into the HDFS (Hadoop Distributed File System), an abstract file system that stores files in clusters. Your Hadoop code will directly access files on HDFS. Paths on the HDFS look similar to those on the UNIX system, but you cannot explore them directly using standard UNIX commands. Instead, you need to use ***hadoop fs*** commands. For example

```
hadoop fs -ls
```

Download the following two graph files: graph1.tsv[1] (~5MB when unzipped) and graph2.tsv[2] (~1.1GB when unzipped) into the VM and unzip them. Also place the Q1 skeleton code in the VM. Use the following commands to set up a directory on the HDFS to store the two graph datasets. Do not change the directory structure below (data/) since we will grade your homework using the scripts that assume the following directory structure. First, navigate to the folder in your skeleton code that contains the **src** directory, **pom.xml** , **run1.sh**, and **run2.sh** files and enter the following commands.

[1] Graph1 is a modified version of data derived from the LiveJournal social network dataset, with around 30K nodes and 320K edges.
[2] Graph2 is a modified version of data derived from the LiveJournal social network dataset, with around 300K nodes and 69M edges.

```
hadoop fs -mkdir data
hadoop fs -put <insert path to data file>/graph1.tsv data
hadoop fs -put <insert path to data file>/graph2.tsv data
```

Please check the VM setup instructions if you are facing any file permission errors while running the commands.

Now both files (graph1.tsv and graph2.tsv) are on HDFS, at **data/graph1.tsv** and **data/graph2.tsv**. To check this, try:

```
hadoop fs -ls data
```

Setting up Development Environments

We found that compiling and running Hadoop code can be quite complicated. So, we hav e prepared some skeleton code, compilation scripts, and execution scripts for you that you can use, in the Lab10 skeleton folder. You should use this structure to submit your homework.
In the directory of *Q1*, you will find **pom.xml** , **run1.sh, run2.sh** and the **src** directory.

- The **src** directory contains a main Java file that you will primarily work on. We have provided some code to help you get started. Feel free to edit it and add your files in the directory, but the main class should be Q1. Your code will be evaluated using the provided **run1.sh** and **run2.sh** file (details below).

- **pom.xml** contains the necessary dependencies and compile configurations for each question. To compile, you can simply call Maven in the corresponding directory (i.e., Q1 where pom.xml exists) by this command:

  ```
  mvn package
  ```

  It will generate a single JAR file in the target directory (i.e., target/q1 -1.0.jar). Again, we have provided you some necessary configurations to simplify your work for this homework, but you can edit them as long as our run script works and the code can be compiled using mvn package command.

- ***run1.sh***, ***run2.sh*** are the shell script files that run your code over graph1.tsv (run1.sh) , graph2.tsv (run2.sh) and download the output to a local directory. The output files are named based on its question number and graph number (e.g. q1output1.tsv). You can use these r un scripts to test your code. Note that these scripts will be used in grading.

To execute the shell scripts from the command prompt, enter:

```
./run1.sh
./run2.sh
```

Here's what the above scripts do:

1. Run your JAR on Hadoop specifying the input file on HDFS (the first argument) and output directory on HDFS (the second argument)
2. Merge outputs from output directory and download to local file system.
3. Remove the output directory on HDFS.

Analyzing a Graph with Hadoop/Java

Imagine that your boss gives you a large dataset which contains an entire email communication network from a popular social network site. The network is organized as a directed graph where each node represents a person's email address and an edge between two nodes (e.g., address A and address B) has a weight stating how many times A has written to B. You have been tasked with finding the person that each person has written to the most, along with that count (see the example below for more clarification). Your task is to write a MapReduce program in Java to report, for each node X (the "source", or "src" for short) in the graph, the person Y (the "target" or "tgt" for short) that X has written to the most, and the number of times X has written to Y (the outbound "weight", from X to Y). **If a person has written to multiple targets that have exactly the same (largest) number of times, return the target with the smallest node id.**

First, go over the Hadoop word count tutorial to familiarize yourself with Hadoop and some Java basics. You will be able to complete this question with only some knowledge about Java. You should have already loaded two graph files into HDFS and loaded into your HDFS file system in your VM. Each file stores a list of edges as tab-separated-values. Each line represents a single edge consisting of three columns: (source node ID, target node ID, edge weight), each of which is separated by a tab (\t). Node IDs and weights are positive integers. Below is a small toy graph, for illustration purposes (on your screen, the text may appear out of alignment).

```
src   tgt   weight
10    110   3
10    200   1
200   150   30
100   110   10
110   130   15
110   200   67
10    70    3
```

Your program should not assume the edges to be sorted or ordered in any ways (i.e., your program should work even when the edge ordering is random).

Your code should accept two arguments. The first argument (*args[0]*) will be a path for the input graph file on HDFS (e.g., data/graph1.tsv), and the second argument (*args[1]*) will be a path for output directory on HDFS (e.g., data/q1output1). The default output mechanism of Hadoop will create multiple files on the output directory such as part-00000, part-00001, which will be merged and downloaded to a local directory by the supplied run script. Please use the run1.sh and run2.sh scripts for your convenience.

The format of the output: each line contains a node ID, followed by a tab (\t), and the expected "target node ID,weight" tuple (without the quotes; and note there is no space character before and after the comma). Lines do not need to be sorted. The following example result is computed based on the toy graph above. Please exclude nodes that do not have outgoing edges (e.g., those email addresses which have not sent out any communication).

For the toy graph above, the output is as follows.

```
10     70,3
200    150,30
100    110,10
110    200,67
```

Deliverables

1. **Your Maven project directory including Q1.java.** Please see detailed submission guide at the end of this document. You should implement your own MapReduce procedure and should not import external graph processing library.

2. **q1output1.tsv**: the output file of processing graph1.tsv by run1.sh.