Lab 1: Introduction to Hadoop MapReduce

Today's focus is on:

1. getting acquainted with Hadoop HDFS and MapReduce
2. designing a MapReduce algorithm
3. implementing it in Hadoop

Getting acquainted with Hadoop

To start HDFS, open a shell and run: `start-dfs.sh && start-yarn.sh` (it may take a while...). Download and try the wordcount example. Refer to the HowToRun.txt.

To access HDFS, you can use the following basic commands:

- List directory: `hadoop fs -ls /out`
- Inspect file: `hadoop fs -cat /myfile | less`
- Remove file: `hadoop fs -rm /myfile`
- Remove directory: `hadoop fs -rm -r -f /out`
- Copy file from local file system to HDFS: `hadoop fs -put myfile /myhfdsdir/myfile`
- Copy file from HDFS to local file system: `hadoop fs -get /myhfdsdir/myfile .`
- Here's a comprehensive list of HDFS commands

Useful resources for programming:

- What is MapReduce?
- Hadoop JavaDoc
- HDFS commands

Exercise 1. Finding pairs of nodes at distance 2 in a graph

The goal is to write a MapReduce program that, given a directed graph G=(V,E) computes all pairs of nodes (x,y) such that y is reachable from x in two hops, i.e., there is a node z such that both (x,z) and (z,y) are in E. Notice that (x,y) may or may not be in E.

The input is given as a text file where each line contains an edge with node IDs separated by a space or tab:

```
1 2
0 1
3 2
2 3
...
```

A test input graph is `facebook.txt` (1.7 MB). It would be a good idea to generate a tiny input file for testing purposes. Use the `hadoop fs -put` command to upload graphs to HDFS.

The output should be a list of node pairs x y connected by a path of length exactly 2, one per line:

```
1 3
4 2
...
```

In a first version, it is ok to allow duplicates in the output. As a refinement, you can think of **how to remove them**.

Exercise 2. Computing the in/out degree of the nodes of a graph

The goal is to write a MapReduce program that, given a directed graph G=(V,E) computes the in-degree and the out-degree of each node.

The input is as in Exercise 1. The output should be a list of triples of the form (node, in-degree, out-degree), one per line as follows:

```
1 3 0
4 2 1
...
```

Exercise 3. Checking if a graph is undirected

The goal is to write a MapReduce program that, given a directed graph G=(V,E) allows us to check if it is non-directed, i.e., for each directed edge (x,y), there is also the directed edge (y,x).

The input is as in Exercise 1. What would you produce as output?