# CSCI4180/ESTR4106 (Fall 2018)

### Assignment 2: Iterative MapReduce

Due on November 15 (Thur), 2018, 23:59:59

## 1 Part 1: Single-source Shortest Path Lengths by Parallel Dijkstra's Algorithm (50%)

In this part, you will need to write a MapReduce program to compute the **shortest path lengths** from a given **source node** in a graph dataset extracted from Twitter. We implement the algorithm via the parallel Dijkstra's algorithm. The Twitter dataset is obtained from the following reference:

- Haewoon Kwak, Changhyun Lee, Hosung Park, Sue Moon.
  "What is Twitter, a Social Network or a News Medium".
  19th World-Wide Web (WWW) Conference, April 2010.
  URL: http://an.kaist.ac.kr/traces/WWW2010.html

We model the Twitter network as a directed graph. Each user is represented as a *node* with a unique positive integer as the nodeID. When user 1 "follows" user 2 on Twitter, an *edge* is created from node 1 to node 2 in the graph. Also, we attach a positive integer weight to each edge.

**Problem:** Given a graph $G = (V, E)$ and a source node $v_s \in V$, find the shortest path distance from $v_s$ to every other reachable node in $V$. The source node $v_s$ is provided as a command-line argument.

**Input Format:** Each line contains a tuple of (nodeID, nodeID, weight), separated by spaces. Each tuple indicates a directed edge from the former node to the latter node.

**Output Format:** Each line contains a tuple of (nodeID, distance, prev), separated by spaces, where "distance" means the shortest path distance, and "prev" means the incoming node that lies on the shortest path. Only output tuples for nodes that are reachable from $v_s$ which is given as a command-line argument.

**Sample Command:**

```
hadoop jar [.jar file] ParallelDijkstra [infile] [outdir] [src] [iterations]
```

**Sample Input:**

1 2 7
1 3 20
2 3 3
3 1 5
4 1 9
5 6 10

**Sample Output:**

1 0 nil
2 7 1
3 10 2

**Notes:**

- The sample assumes that the source node is 1.
- Since there is no path going to node 4, 5 or 6 from $v_s$, the tuples corresponding to these nodes should not be shown in the output.
- Your program (call it *ParallelDijkstra.java*) will take a command-line argument `Iterations` to indicate the maximum number of MapReduce iterations (the program may finish earlier if all shortest path distances have been found). If `Iterations` equals 0, it means that the program will keep iterating until all shortest path distances are found. Throughout the iterative map/reduce jobs, feel free to store and retrieve any intermediate files in your own format on HDFS. We would only look at your final output for grading.
- You will need to implement a class of the node structure (call it *PDNodeWritable.java*) to define the node attributes, such as adjacency lists.
- You will need to write a separate MapReduce program (call it *PDPreProcess.java*) to convert the input files into *adjacency list* format first.

**Time Limit:** Note that the program should be completed within a reasonable timeframe for a given dataset. The time limit is set by the TAs. Marks will be deducted if the program runs too long.

## 2 Part 2: PageRank Algorithm (50%)

In this part, you will need to write a MapReduce program to compute the PageRanks of nodes. We still use the same Twitter dataset in Part 1 by treating each node in the dataset as a "page" for the PageRank algorithm.

**Problem:** Given a graph $G = (V, E)$, find the PageRank values of all nodes in $V$. In addition, we include the random jump factor $\alpha$ (which is a command-line parameter) and redistribute all missing PageRank mass $m$ due to dangling nodes. For dangling nodes, we set $p = 0$ during the map phase (i.e., during the distribution of PageRank mass) of the PageRank algorithm, and its PageRank mass is considered to be missed. The missing PageRank mass $m$ (from all dangling nodes) will be added back based on the following equation:

$$p' = \alpha(\frac{1}{|G|}) + (1 - \alpha)(\frac{m}{|G|} + p). \quad (1)$$

**Input Format:** Each line contains a tuple of (nodeID, nodeID, weight), separated by spaces. Each tuple indicates a directed edge from the former node to the latter node. We ignore the edge weights in this problem, as they are not needed by the PageRank algorithm that we taught in class.

**Output Format:** Each line contains a tuple of (nodeID, PageRank value), separated by spaces. Only output tuples for nodes whose PageRank values are above certain threshold, where the threshold value (between 0 and 1) is given as a command-line argument.

**Sample Command:**

```
hadoop jar [.jar file] PageRank [alpha] [threshold] [iteration] [infile] [outdir]
```

**Notes:**

- Your program (call it *PageRank.java*) will execute the PageRank algorithm over a fixed number of iterations. The program will take a command-line argument `Iterations` to indicate the number of MapReduce iterations needed to be executed. We assume that `Iterations` is at least one.

- You need to write a MapReduce program (call it *PRAdjust.java*) to adjust the PageRank values due to random jumps and dangling nodes after each iteration of the PageRank algorithm.
- As in Part 1, we need to implement a class of node structure (call it *PRNodeWritable.java*) and write a MapReduce program to to convert the input files into adjacency list format (call it *PRPre-Process.java*).

## Bonus (5%)

The top 3 groups who have the total shortest running time for their programs in *both Parts 1 and 2* will receive the bonus marks. You may consider optimizing your programs or configuring some parameters in Hadoop to make the programs perform better. If more than 3 groups have the best performance, we will still give out the 5% bonus to each group. Note that the program must return the correct answer in order to be considered for the bonus mark.

## Submission Guidelines

Please at least submit the following files. Additional files are allowed.

**Part 1:**

- ParallelDijkstra.java
- PDNodeWritable.java
- PDPreProcess.java

**Part 2:**

- PageRank.java
- PRAdjust.java
- PRNodeWritable.java
- PRPreProcess.java

Demo will be arranged on the following day of the deadline. Have fun! :)