Single Source Shortest Path Algorithm Implementation Using MapReduce

Instructor: Kunpeng Zhang (kzhang6@uic.edu)
TA: Minghong Xu (mxu29@uic.edu)

The Deadline: 6:00PM, Feb. 25, 2015 (Wednesday)

Given a graph (see Figure 1), we can represent it using the following format (see sample input file). The assignment is to calculate the shortest distances from the source node to all other nodes in an undirected and un-weighted graph. You may need to provide two input parameters for your main driver program: the source node and the input file.

Sample Input File:

1<tab>2,3|0|GRAY|source
2<tab>1,3,4,5|Integer.MAX_VALUE|WHITE|null
3<tab>1,4,2|Integer.MAX_VALUE|WHITE|null
4<tab>2,3|Integer.MAX_VALUE|WHITE|null
5<tab>2|Integer.MAX_VALUE|WHITE|null

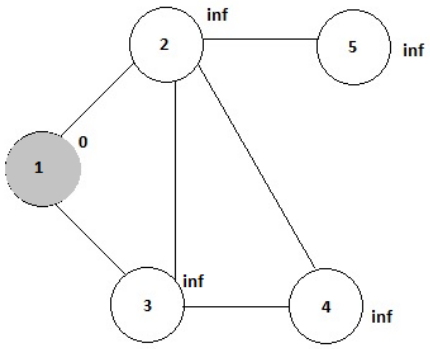

Figure 1: sample graph

To finish this task, you will implement the following two jobs, which can be chained either using method 1 or method 2 that we discussed in class.

(Job 1) Generate a file containing an adjacent list for each node. Within this job, you also need to print out some basic descriptive statistics for the graph to the screen, including the number of nodes and the number of links.

(Job 2) Calculating the shortest path using coloring strategy. The algorithm is described as below.

- At the beginning, all nodes are colored white.
- The source node is colored gray.
- The gray node indicates that it is visited and its neighbors should be processed.
- All the nodes adjacent to a gray node that are white are changed to be gray colored.
- The original gray node is colored.
- The process continues until there are no more gray nodes to process in the graph.

The Map function and the Reduce function are described in the following:

Mapper
The nodes colored WHITE or BLACK are emitted as such. For each node that is colored GRAY, a new node is emitted with the distance incremented by one and the color set to GRAY. The original GRAY colored node is set to BLACK color and it is also emitted.

Reducer
Make a new node that combines all information for this single node id that is for each key. The new node should have the full list of edges, the minimum distance, the darkest Color, and the parent/predecessor node

Within the second job, you need to define global counters to determine if there are more GRAY nodes. In the reduce function, if the color of a node is set to BLACK from GRAY, you need to decrease the counter by 1. If the color of a node is set to GRAY from WHITE, you need to increase the counter by 1.

Requirements:

- You need to create an input file containing at least 100 lines (a graph having at least 100 nodes). It is undirected and un-weighted (the link weight is 1) graph.

- You need to implement a customized Partitioner class. The partition function can be implemented by your own design, but returns at least 3 partitions.

- You need to create a separate Java class representing a graph Node

- You need to save all intermediate files generated from each iteration in your code. The file name should contain iteration number. For example,

the initial file name may be input.txt. After the first iteration, you will generate output_1.txt, after the second iteration, you will generate output_2.txt, … (1, 2, … would represent the iteration number).

- Your codes must be readable and clean.

- When you submit your codes through blackboard, you need to put all source codes (.java files, NOT jar files), the input file, and some other optional files (e.g., a readme file) into one folder and name that folder as <YOUR UID>_ASSIGN1. Assignments not following this rule will not be graded. In addition, no resubmission after TA grades it. Late submission rule: 10% deduction for one day late. Late submission over a week is NOT acceptable.

- DO NOT copy any codes from others. Otherwise, both will be penalized.