Project 8: Dijkstra’s algorithm for the Single-Source-Shortest Paths problem

Problem Statement: Given a directed graph, G = <N, E>, and the source node, S, in G, the task is find the shortest paths from S to every nodes in G, using the Dijkstra’s algorithm.

Please note that in your program, the source node will be 1, 2, 3, …, N. // i.e., Your program will produce all pairs shortest paths.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Language: C++ and Java

Due date: C++ soft copy: 4/28/2016 Thursday before Midnight

Due date: C++ hard copy: 5/3/2016 Tuesday in class

Due date: Java soft copy: 5/1/2016 Sunday before Midnight

Due date: Java hard copy: 5/3/2016 Tuesday in class

I. Input:

a) argv[1]: a directed graph, represented by a list of edges with costs, {<ni, nj, c>}

// You may assume that nodes’ ID is from 1 to N.

The format of the input file is as follows:

The first text line is the number of nodes, N, follows by a list of triplets, <ni, nj, cost>

For example:

5 // there are 5 nodes in the graph

1 5 10 // an edge <1, 5, 10>

2 3 5 // an edge <2, 3, 5>

1 2 20 // an edge <1, 2, 20>

3 5 2 // an edge <3, 5, 2>

:

:

II. Outputs

a) argv[2]. : This is for the output of all paris shortest paths.

For example, if there are 5 nodes in the graph G. Then your output will be as follows:

===============================

There are 5 nodes in the input graph.

===============================

Source node = 1

The shortest paths from node 1 are:

The path from 1 to 1 : 1 🡪 1: cost = 0

The path from 1 to 1 : 1 🡪 2 : cost = whatever

The path from 1 to 1 : 3 🡪 3 : cost = whatever

:

:

===============================

The source node = 2

The shortest paths from node 2 are:

The path from 2 to 1 : 2 🡪 1: cost = whatever

The path from 2 to 2 : 2 🡪 2 : cost = 0

The path from 2 to 3 : 2 🡪 3 : cost = whatever

:

:

b) argv[3].: For all debugging outputs.

III. Data structure:

1) A DijktraSSS class

- numNodes (integer) //number of nodes in G

- sourceNode (integer)

- minNode (integer)

- currentNode (integer)

- newCost (integer)

- costMatrix, a 2-D cost matrix (integer array), size of N X N, should be dynamically allocated.

// Initially, costMatrix[i][i] set to zero and all others set to infinity, 99999

- fatherAry, a 1-D integer array, size of N, should be dynamically allocated.

// initially set to itself, i.e., father[i] = i

- markedAry, a 1-D integer array, size of N, should be dynamically allocated.

// initially set to 0 (not marked)

- bestCostAry, a 1-D integer array, size of N, should be dynamically allocated.

// initially set to 9999 (infinity)

Methods:

- loadCostMatrix // read from input file and fill the costMatrix,

- loadBestCostAry (sourceNode) // copy the row of source node from costMatrix,

- loadFatherAry (sourceNode) // set all to source

- loadMarkedAry ( ) // set all to 0,

- computeCost (minNode, currentNode) method,

//it compute the currentNode’s cost ( bestCostArray[minNode] + cost from minNode to currentNode, in costMatrix)

- markMinNode (minNode) // mark the minNode in markedAry

- changeFather (node, minNode) // set node’s father be minNode in fatherAry

- changeCost(node, newCost) // set node’s best cost to newCost in bestCostAry

- debugPrint method, this method

Print sourceNode to output-2 (with proper heading, ie., the sourceNode is: )

Print fatherAry to output-2 (with proper heading)

Print bestCostAry to output-2 (with proper heading)

Print markedAry to output-2 (with proper heading)

- printShorestPaths (sourceNode)

- Dijkstras’ method // the Dijkstra’s algorithm

V. Algorithm steps:

// A lot of debugging statements are embedded in the algorithm as to teach you how to debug your program!

step 0: open input and output files

numNodes 🡨 get from input

Allocate and initialize all members in the DijkstraSSS class accordingly

step 1: loadMatrix

sourceNode 🡨 1

step 2: loadBestCostAry (sourceNode)

loadFatherAry (sourceNode)

loadMarkedAry ( )

step 3: minNode 🡨 find an \*unmarked\* node with minimum cost from bestCostAry

markMinNode(minNode)

call debugPrint

step 4: // expanding the minNode

currentNode 🡨 find next unmarked node in markedAry

newCost 🡨 computeCost(minNode, currentNode)

if newCost < bestCostAry [currentNode]

changeFather (currentNode, minNode)

changeCost (currentNode, newCost)

debugPrint

step 5: repeat step 4 until all unmarked node in markedAry are evaluated

step 6: repeat step 3 to step 5 until all nodes are marked

step 7: currentNode 🡨 1

step 8: printShortestPath (currentNode)

// trace from currentNode back to sourceNode (via fatherAry), and print the shortest path from sourceNode to currentNode with the cost to output-1 (with proper heading)

step 9: currentNode ++

step 10: repeat step 8 and step 9 until currentNode >= numNodes

step11: repeat step 2 to step 10 until sourceNode > numNodes