Project 6: All paths shortest paths, by using Dijkstra's algorithm for the Single-Source-Shortest Paths problem N times.

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 all other 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++
****
Project points:12 pts
Due Date: Soft copy (*.zip) and hard copies (*.pdf):

12/12 on time: 10/30/2020 Friday before midnight
+1 early submission: 10/27/2020 Tuesday before midnight
-1 for 1 day late: 10/31/2020 Saturday before midnight
-2 for 2 days late: 11/1/2020 Sunday before midnight
-12/12 : after 11/1/2020 Sunday after midnight
-6/12: does not pass compilation
0/12: program produces no output
0/12: did not submit hard copy.

*** Follow "Project Submission Requirement" to submit your project.

Include in your hard copy:
a) cover page
b) draw an illustrations of iterations (step 5 to step 6) of Dijkstra's' tree search **(ONLY from source node is 5),** including:
- cost matrix
- the source node id
- the bestCostAry
- the markedAry
- fatherAry.

c) source code
d) SSSfile
e) deBugFile

****
I. inFile (argv [1]): a directed graph, represented by a list of edges with costs, {<n_i, n_j, c>}
// You may assume that nodes' Id is from 1 to N (0 is not used)

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, <n_i, n_j, cost>
For example:

```
5           // there are 5 nodes in the graph
1  5 10     // an  edge  from node 1 to node 5, the cost is 10
2  3  5
1  2  20
3  5  2
```

:

:

****

II. Outputs:

a) SSSfile (argv [2]) : for the result of all pairs shortest paths. The format is given below:

// If there are 7 nodes in the graph G. Then your output will be as follows:

```
================================
There are 7 nodes in the input graph. Below are the all pairs of shortest paths:
 =================================
Source node = 1

The path from 1 to 1 :  1 ← 1 : cost = 0
The path from 1 to 2 :  2 ← … ← 1: cost = whatever
The path from 1 to 3 :  3 ← … ← 1: cost = whatever
:
:
The path from 1 to 7 :  7 ← … ← 1: cost = whatever

=================================
The source node = 2

The path from 2 to 1 :  1 ← … ← 2 : cost = whatever
The path from 2 to 2 :  2 ←  2: cost = 0
The path from 2 to 3 :  3 ← … ← 2: cost = whatever
:
:
The path from 2 to 7 :  7 ← … ← 2: cost = whatever
=================================
:
:
=================================
The source node = 7

The path from 7 to 1 :  1 ← … ← 7 : cost = whatever
The path from 7 to 2 :  2 ← … ← 7 : cost = whatever
The path from 7 to 3 :  3 ← … ← 7 : cost = whatever
:
:
The path from 7 to 7 :  7 ← 7: cost = 0
```

b) deBugFile (argv [3]): For all debugging outputs. <u>You do NOT need to print outFile2 in your hard copies.</u>

****

III. Data structure:

1) A DijktraSSS class

- numNodes (int) //number of nodes in G
- sourceNode (int)
- minNode (int)

- currentNode (int)
- newCost (int)
- costMatrix (int **)
  // a 2-D cost matrix (integer array), size of N+1 X N+1, should be dynamically allocated.
  // Initially, costMatrix[i][i] set to zero and all others set to infinity, 99999
  // Note: 0 is not used for node Id.

- fatherAry (int*) // a 1-D integer array, size of N+1, should be dynamically allocated.
  // initially set to itself, i.e., father[i] = i

- markedAry(int*) // 1-D integer array, size of N+1, should be dynamically allocated.
  // initially set to 0 (not marked)
- bestCostAry (int*) // a 1-D integer array, size of N+1, should be dynamically allocated.
  // initially set to 9999 (infinity)

Methods:

- loadCostMatrix (. . .)// read from input file and fill the costMatrix,
- setBestCostAry (sourceNode) // copy the row of source node from costMatrix,
- setFatherAry (…) // set all to itself
- setMarkedAry (sourceNode) // set sourceNode to 1 and all other to 0
- int findMinNode (. . .) // find an *unmarked* node with minimum cost from bestCostAry
  // Algorithm is given below
- int computeCost (minNode, currentNode)
  // computes the best cost for currentNode, which is
  // bestCostAry [minNode] plus the edge cost from minNode to currentNode, i.e.,
  // costMatrix [minNode, currentNode], it returns the computed best cost for currentNode
- debugPrint (…) // This method for you to debug your program.
  // Prints sourceNode to deBugFile (with proper heading, i.e., the sourceNode is:)
  // Prints fatherAry to deBugFile (with proper heading)
  // Prints bestCostAry to deBugFile (with proper heading)
  // Prints markedAry to deBugFile (with proper heading)
- printShortestPath (currentNode, sourceNode, SSSfile) // on your own.
  // The method traces from currentNode back to sourceNode (via fatherAry),
  // print to SSSfile, the shortest path from
  // currentNode to sourceNode with the total cost, using the format given in the above
  // You should know how to do this method.

****

V. main (…)

****

```
step 0:  open inFile, SSSfile, deBugFile
         numNodes ← get from inFile
         Allocate and initialize all members in the DijktraSSS class accordingly

step 1:  loadCostMatrix (inFile)
         sourceNode ← 1

step 2:  setBestCostAry (sourceNode)
```

```
          setFatherAry (…)
          setMarkedAry (sourceNode)
step 3: minNode ← findMinNode(…)
         markedAry[minNode] ← 1
        debugPrint (…)

step 4: // expanding the minNode
        currentNode ← 1

step 5: if markedAry[currentNode] == 0
                    newCost ← computeCost(minNode, currentNode)
                     if newCost < bestCostAry [currentNode]
                              bestCostAry[currentNode] ←  newCost
                               fatherAry[currentNode] ← minNode
                               debugPrint (…)

Step 6: currentNode ++

Step 7: repeat step 5 to step 6 while currentNode <= numNodes

step 8: repeat step 3 to step 7 until all nodes are marked

          // begin printing the paths
step 9: currentNode ← 1

step 10:  printShortestPath (currentNode, sourceNode, SSSfile)

step 11:  currentNode ++

step 12: repeat 10 and step 11 while currentNode <= numNodes

step 13: sourceNode ++

step 14:  repeat step 2 to step 13 while sourceNode <= numNodes

step 15: close all files
```

****

V. int findMinNode ()

****

```
Step 0: minCost ← 99999
          minNode ← 0

Step 1: index ← 1

Step 2: if markedAry[index] == 0 // unmarked
              if bestCostAry[index] < minCost
                   minCost ← bestCostAry[index]
                   minNode ←index
step3: index++

step 4: repeat step 2 – step 3 while index <= numNodes
step 5: return minNode
```