Project 7: (Java) Given a connected graph, G = <N, E>, the task is find a minimum spanning tree of G, using the Kruskal's algorithm.

****

Language: Java
Project points: 10pts
Due Date: Soft copy (*.zip) and hard copies (*.pdf):

10/10 on time: 11/19/2020 Thursday before midnight
+1 early submission: 11/16/2020 Monday before midnight
-1 for 1 day late: 11/20/2020 0 Friday before midnight
-2 for 2 days late: 11/21/2020 Saturday before midnight
-10/10: after 10/30/2020 Tuesday after midnight
-5/10: does not pass compilation
0/10: program produces no output
0/10: did not submit hard copy.

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

Include in your hard copy:
- cover page
- draw the diagrams for the construction of MST
- source code
- print debugFile
- print MSTfile

****

I. Input:
inFile (args [0]): A text file contains a connected undirected graph, represented by a list of edges with costs, $<n_i, n_j, c>$, where the first text line is the number of nodes, N, in G, follows by a list of triplets, $<n_i, n_j, cost>$; $n_i > 0$ and $n_j > 0$.
For example:

```
5          // there are 5 nodes in the graph
1  5 10 // an undirected edge: i.e.,  an edge <1, 5, 10> and an edge <5, 1, 10>
2  3  5 // an undirected edge: i.e.,  an edge <2, 3, 5> and an edge <3, 2, 5>
1  2  20 // an undirected edge: i.e.,   an edge <1, 2, 20> and an edge <2, 1, 20>
3  5  2 // an undirected edge: i.e.,  an edge <3, 5, 2> and an edge < 5, 3, 2>
          :
          :
```

II. Outputs:
a) MSTfile (args [1]):
A text file contain a minimum spanning tree (MST) of G, represented by a list of undirected edges with costs where the first text line is the number nodes, N, the same as N in G,
follows by the list of edges with costs in MST of G; at the end of the list, print the total cost of the MST.

For example:

```
*** A Kruskal's MST of the input graph is given below: ***
          5
          2  3  5
          3  5  2
          :
          :
```

*** The total cost of a Kruskal's MST is:

b) debugFile (args[2]) : For all debugging outputs.

****
III. Data structure:
****

1) An edgeNode class

- Ni (int) // an integer 1 to N
- Nj (int) // an integer 1 to N
- cost (int) // a positive integer > 0
- next (edgeNode)

Methods:
- printNode (node) // print node.Ni, node.Nj, node.cost, node.next
- constructor (n1, n2, cost) // to create a new edgeNode of given data: n1, n2, cost and next set to null.

2) A KruskalMST class

- numNodes (int) //number of nodes in G
- inWhichSet (int[]) // a 1-D array, size of N+1; inWhichSet[i] means node i belongs to set of inWhichSet[i]
  // to indicate which set each node belongs; dynamically allocated, and set to itself initially.
  // as a singleton set.

- numSets (int) // to indicate the remaining number of sets in the computation, initially set to numNodes.
- totalMSTCost (int) // initially set to zero

- listHeadEdge (edgeNode) //This linked list (with a dummy node) is to store all the edges in the graph G.
  // for constructing the MST of G.
  // the list is sorted in ascending order with respect to the cost of edges in the list.
  // the dummy node is set to <0,0,0, null> when created.

- listHeadMST (edgeNode) // This linked list (with a dummy node) is to store the list of edgeNode in MST
  // the list is sorted in ascending order with respect to the cost of edges in the list.
  // the dummy node is set to <0,0,0, null> when created.

Method:
- insert (node, listHead)
  // insert the given node into the given list, using insert sort, in ascending order w.r.t. the edge. cost.

- (edgeNode) removedEdge (listHead) // removes *and* returns the front node of the given linked list.

- merge2Sets (node1, node2) // if node1 < node2
  inWhichSet (node2) ← node1
  Else
  inWhichSet (node1) ← node2

- printAry (…) // print inWhichSet array to debugFile, with proper caption.

-printList (listHead, file) // print the given linked list, listHead, to a given file
// Format: listHead → <0, 0, 0> → <N1, N1, edgeCost> → <N2, N2, edgeCost> …

****
IV. main(…)
****

Step 0: numNodes ← inFile
numSets ← numNodes
inWhichSet ← allocate space, size of numNodes_1, set inWhichSet[i] to i, i from 1 to numNodes+1
listHeadEdge ← create a linked list of edgeNode with a dummy node
listHeadMST ← create a linked list of edgeNode with a dummy node
totalMSTCost ← 0

printAry(…)

Step 1: <Ni, Nj, cost> ← read from inFile
newEdge ← get a new edgeNode (Ni, Nj, cost)

Step 2: insert (newEdge, listHeadEdge)

Step 3: printList (listHeadEdge, debugFile)

Step 4: repeat step 1 to step 3 while inFile is not empty

Step 5: nextEdge ← removedEdge (listHeadEdge)

Step 6: repeat Step 5 if inWhichSet [nextEdge.Ni] == inWhichSet [nextEdge.Nj] // Ni and Nj are in the same set

Step 7: insert(nextEdge, listHeadMST)
totalMSTCost += nextEdge.cost
merge2Sets (Ni, Nj) // now, Ni, Nj are in the same set
numSets --

Step 8: printAry(inWhichSet)

Step 9: printList (listHeadMST, debugFile)
printList (listHeadEdge, debugFile)

Step 10: repeat step 5 – step 8 while numSets is > 1

Step 11: printList (listHeadMST, MSTfile)

Step 12: close all files.