323-35 Project 7: MST using Kruskal Java

Student Name: Matthew Smyth

Project due date: 11/19/2020

Algorithm Steps:

****
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.

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;

public class Main {

	public static void main(String[] args) {
		KruskalMST k = new KruskalMST();

		Scanner scan;
		FileWriter MSTFile;
		FileWriter debugFile;
		try {
			scan = new Scanner(new FileReader(args[0]));
			MSTFile = new FileWriter(args[1]);
			debugFile = new FileWriter(args[2]);

			k.numNodes = scan.nextInt();
			k.numSets = k.numNodes;
			k.inWhichSet = new int[k.numNodes + 1];
			for (int i = 1; i < k.numNodes + 1; i++) {
				k.inWhichSet[i] = i;
			}

			k.totalMSTCost = 0;
			k.printAry(debugFile);

			while (scan.hasNext()) {
				int n1 = scan.nextInt();
				int n2 = scan.nextInt();
				int cost = scan.nextInt();

				edgeNode node = new edgeNode(n1, n2, cost);

				k.insert(node, k.listHeadEdge);

				debugFile.write("listHeadEdge");
				k.printList(k.listHeadEdge, debugFile);
			}

			while (k.numSets > 1) {
```

```
                                edgeNode nextEdge;
                                do {
                                        nextEdge = k.removedEdge(k.listHeadEdge);
                                } while (k.inWhichSet[nextEdge.Ni] ==
k.inWhichSet[nextEdge.Nj]);

                                k.insert(nextEdge, k.listHeadMST);
                                k.totalMSTCost += nextEdge.cost;
                                k.merge2Sets(nextEdge.Ni, nextEdge.Nj);
                                k.numSets--;

                                k.printAry(debugFile);

                                debugFile.write("listHeadMST");
                                k.printList(k.listHeadMST, debugFile);

                                debugFile.write("listHeadEdge");
                                k.printList(k.listHeadEdge, debugFile);
                        }

                        k.printMST(k.listHeadMST, MSTFile);

                        scan.close();
                        MSTFile.close();
                        debugFile.close();

                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}

import java.io.FileWriter;

public class edgeNode {
        public int Ni;
        public int Nj;
        public int cost;
        public edgeNode next = null;

        public void printNode(FileWriter outFile) {
                System.out.println(this.Ni + " " + this.Nj + " " + this.cost + "\n");
        }
```

```
	public edgeNode(int n1, int n2, int cost) {
		this.Ni = n1;
		this.Nj = n2;
		this.cost = cost;
	}
}

import java.io.FileWriter;
import java.io.IOException;

public class KruskalMST {
	public int numNodes;
	public int[] inWhichSet;
	public int numSets;
	public int totalMSTCost;
	public LL listHeadEdge = new LL();
	public LL listHeadMST = new LL();

	public void insert(edgeNode node, LL listHead) {
		edgeNode temp = listHead.head;
		while (temp.next != null && temp.next.cost < node.cost) {
			temp = temp.next;
		}
		node.next = temp.next;
		temp.next = node;
	}

	public edgeNode removedEdge(LL listHead) {
		edgeNode temp = listHead.head;
		if (temp.next != null) {
			edgeNode temp2 = temp.next;
			temp.next = temp.next.next;
			return temp2;
		}
		return null;
	}

	public void merge2Sets(int node1, int node2) {
		if (inWhichSet[node1] < inWhichSet[node2]) {
			int temp = inWhichSet[node2];
			for (int i = 1; i < inWhichSet.length; i++) {
				if (inWhichSet[i] == temp) {
					inWhichSet[i] = inWhichSet[node1];
				}
```

```
                }
            } else {
                int temp = inWhichSet[node1];
                for (int i = 1; i < inWhichSet.length; i++) {
                    if (inWhichSet[i] == temp) {
                        inWhichSet[i] = inWhichSet[node2];
                    }
                }
            }
        }

        public void printAry(FileWriter debugFile) {
            for (int i = 1; i < inWhichSet.length; i++) {
                try {
                    debugFile.write("inWhichSet " + i + ": " + inWhichSet[i] + "\n");
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }

        public void printList(LL listHead, FileWriter outFile) {
            edgeNode temp = listHead.head;
            for (int i = 0; i <= listHead.size(); i++) {
                try {
                    outFile.write(" -> <" + temp.Ni + "," + temp.Nj + "," + temp.cost +
">");
                    temp = temp.next;
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
            try {
                outFile.write("\n");
            } catch (IOException e) {
                e.printStackTrace();
            }
        }

        public void printMST(LL listHead, FileWriter outFile) {
            try {
                outFile.write("*** A Kruskal's MST of the input graph is given below:
***\n");
                edgeNode temp = listHead.head.next;
```

```
                        outFile.write(listHeadMST.size() + "\n");
                        for (int i = 1; i <= listHeadMST.size(); i++) {
                                outFile.write(temp.Ni + " " + temp.Nj + " " + temp.cost + "\n");
                                temp = temp.next;
                        }
                        outFile.write("*** The total cost of a Kruskal's MST is: " + totalMSTCost);
                } catch (IOException e) {
                        e.printStackTrace();
                }
        }
}


public class LL {
        edgeNode head;

        LL() {
                head = new edgeNode(0, 0, 0);
        }

        public int size() {
                edgeNode temp = head;
                int count = 0;
                while (temp.next != null) {
                        temp = temp.next;
                        count++;
                }
                return count;
        }
}
```

debugFile:

inWhichSet 1: 1
inWhichSet 2: 2
inWhichSet 3: 3
inWhichSet 4: 4
inWhichSet 5: 5
inWhichSet 6: 6
inWhichSet 7: 7
inWhichSet 8: 8
inWhichSet 9: 9
inWhichSet 10: 10
inWhichSet 11: 11
inWhichSet 12: 12
listHeadEdge -> <0,0,0> -> <6,4,3>
listHeadEdge -> <0,0,0> -> <6,4,3> -> <12,7,4>
listHeadEdge -> <0,0,0> -> <6,4,3> -> <12,7,4> -> <6,12,7>
listHeadEdge -> <0,0,0> -> <6,4,3> -> <12,7,4> -> <10,12,7> -> <6,12,7>
listHeadEdge -> <0,0,0> -> <6,4,3> -> <9,10,4> -> <12,7,4> -> <10,12,7> -> <6,12,7>
listHeadEdge -> <0,0,0> -> <2,4,1> -> <6,4,3> -> <9,10,4> -> <12,7,4> -> <10,12,7> -> <6,12,7>
listHeadEdge -> <0,0,0> -> <2,4,1> -> <6,4,3> -> <9,10,4> -> <12,7,4> -> <9,11,5> -> <10,12,7> -> <6,12,7>
listHeadEdge -> <0,0,0> -> <2,4,1> -> <6,4,3> -> <9,10,4> -> <12,7,4> -> <3,2,5> -> <9,11,5> -> <10,12,7> -> <6,12,7>
listHeadEdge -> <0,0,0> -> <2,4,1> -> <6,4,3> -> <9,10,4> -> <12,7,4> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <10,12,7> -> <6,12,7>
listHeadEdge -> <0,0,0> -> <2,4,1> -> <1,6,3> -> <6,4,3> -> <9,10,4> -> <12,7,4> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <10,12,7> -> <6,12,7>
listHeadEdge -> <0,0,0> -> <2,4,1> -> <8,6,2> -> <1,6,3> -> <6,4,3> -> <9,10,4> -> <12,7,4> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <10,12,7> -> <6,12,7>
listHeadEdge -> <0,0,0> -> <2,4,1> -> <9,8,2> -> <8,6,2> -> <1,6,3> -> <6,4,3> -> <9,10,4> -> <12,7,4> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <10,12,7> -> <6,12,7>
listHeadEdge -> <0,0,0> -> <8,10,1> -> <2,4,1> -> <9,8,2> -> <8,6,2> -> <1,6,3> -> <6,4,3> -> <9,10,4> -> <12,7,4> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <10,12,7> -> <6,12,7>
listHeadEdge -> <0,0,0> -> <8,10,1> -> <2,4,1> -> <5,4,2> -> <9,8,2> -> <8,6,2> -> <1,6,3> -> <6,4,3> -> <9,10,4> -> <12,7,4> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <10,12,7> -> <6,12,7>
listHeadEdge -> <0,0,0> -> <8,10,1> -> <2,4,1> -> <5,4,2> -> <9,8,2> -> <8,6,2> -> <4,3,3> -> <1,6,3> -> <6,4,3> -> <9,10,4> -> <12,7,4> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <10,12,7> -> <6,12,7>
listHeadEdge -> <0,0,0> -> <8,10,1> -> <2,4,1> -> <5,4,2> -> <9,8,2> -> <8,6,2> -> <4,3,3> -> <1,6,3> -> <6,4,3> -> <9,10,4> -> <12,7,4> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <1,2,6> -> <10,12,7> -> <6,12,7>

listHeadEdge -> <0,0,0> -> <8,10,1> -> <2,4,1> -> <5,4,2> -> <9,8,2> -> <8,6,2> -> <4,3,3> -> <1,6,3> -> <6,4,3> -> <9,10,4> -> <12,7,4> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <1,11,6> -> <1,2,6> -> <10,12,7> -> <6,12,7>
listHeadEdge -> <0,0,0> -> <8,10,1> -> <2,4,1> -> <5,4,2> -> <9,8,2> -> <8,6,2> -> <4,3,3> -> <1,6,3> -> <6,4,3> -> <3,5,4> -> <9,10,4> -> <12,7,4> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <1,11,6> -> <1,2,6> -> <10,12,7> -> <6,12,7>
listHeadEdge -> <0,0,0> -> <8,10,1> -> <2,4,1> -> <6,7,2> -> <5,4,2> -> <9,8,2> -> <8,6,2> -> <4,3,3> -> <1,6,3> -> <6,4,3> -> <3,5,4> -> <9,10,4> -> <12,7,4> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <1,11,6> -> <1,2,6> -> <10,12,7> -> <6,12,7>
inWhichSet 1: 1
inWhichSet 2: 2
inWhichSet 3: 3
inWhichSet 4: 4
inWhichSet 5: 5
inWhichSet 6: 6
inWhichSet 7: 7
inWhichSet 8: 8
inWhichSet 9: 9
inWhichSet 10: 8
inWhichSet 11: 11
inWhichSet 12: 12
listHeadMST -> <0,0,0> -> <8,10,1>
listHeadEdge -> <0,0,0> -> <2,4,1> -> <6,7,2> -> <5,4,2> -> <9,8,2> -> <8,6,2> -> <4,3,3> -> <1,6,3> -> <6,4,3> -> <3,5,4> -> <9,10,4> -> <12,7,4> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <1,11,6> -> <1,2,6> -> <10,12,7> -> <6,12,7>
inWhichSet 1: 1
inWhichSet 2: 2
inWhichSet 3: 3
inWhichSet 4: 2
inWhichSet 5: 5
inWhichSet 6: 6
inWhichSet 7: 7
inWhichSet 8: 8
inWhichSet 9: 9
inWhichSet 10: 8
inWhichSet 11: 11
inWhichSet 12: 12
listHeadMST -> <0,0,0> -> <2,4,1> -> <8,10,1>
listHeadEdge -> <0,0,0> -> <6,7,2> -> <5,4,2> -> <9,8,2> -> <8,6,2> -> <4,3,3> -> <1,6,3> -> <6,4,3> -> <3,5,4> -> <9,10,4> -> <12,7,4> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <1,11,6> -> <1,2,6> -> <10,12,7> -> <6,12,7>
inWhichSet 1: 1
inWhichSet 2: 2
inWhichSet 3: 3

inWhichSet 4: 2
inWhichSet 5: 5
inWhichSet 6: 6
inWhichSet 7: 6
inWhichSet 8: 8
inWhichSet 9: 9
inWhichSet 10: 8
inWhichSet 11: 11
inWhichSet 12: 12
listHeadMST -> <0,0,0> -> <2,4,1> -> <8,10,1> -> <6,7,2>
listHeadEdge -> <0,0,0> -> <5,4,2> -> <9,8,2> -> <8,6,2> -> <4,3,3> -> <1,6,3> -> <6,4,3> -> <3,5,4> -> <9,10,4> -> <12,7,4> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <1,11,6> -> <1,2,6> -> <10,12,7> -> <6,12,7>
inWhichSet 1: 1
inWhichSet 2: 2
inWhichSet 3: 3
inWhichSet 4: 2
inWhichSet 5: 2
inWhichSet 6: 6
inWhichSet 7: 6
inWhichSet 8: 8
inWhichSet 9: 9
inWhichSet 10: 8
inWhichSet 11: 11
inWhichSet 12: 12
listHeadMST -> <0,0,0> -> <2,4,1> -> <8,10,1> -> <5,4,2> -> <6,7,2>
listHeadEdge -> <0,0,0> -> <9,8,2> -> <8,6,2> -> <4,3,3> -> <1,6,3> -> <6,4,3> -> <3,5,4> -> <9,10,4> -> <12,7,4> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <1,11,6> -> <1,2,6> -> <10,12,7> -> <6,12,7>
inWhichSet 1: 1
inWhichSet 2: 2
inWhichSet 3: 3
inWhichSet 4: 2
inWhichSet 5: 2
inWhichSet 6: 6
inWhichSet 7: 6
inWhichSet 8: 8
inWhichSet 9: 8
inWhichSet 10: 8
inWhichSet 11: 11
inWhichSet 12: 12
listHeadMST -> <0,0,0> -> <2,4,1> -> <8,10,1> -> <9,8,2> -> <5,4,2> -> <6,7,2>
listHeadEdge -> <0,0,0> -> <8,6,2> -> <4,3,3> -> <1,6,3> -> <6,4,3> -> <3,5,4> -> <9,10,4> -> <12,7,4> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <1,11,6> -> <1,2,6> -> <10,12,7> -> <6,12,7>

inWhichSet 1: 1
inWhichSet 2: 2
inWhichSet 3: 3
inWhichSet 4: 2
inWhichSet 5: 2
inWhichSet 6: 6
inWhichSet 7: 6
inWhichSet 8: 6
inWhichSet 9: 6
inWhichSet 10: 6
inWhichSet 11: 11
inWhichSet 12: 12
listHeadMST -> <0,0,0> -> <2,4,1> -> <8,10,1> -> <8,6,2> -> <9,8,2> -> <5,4,2> -> <6,7,2>
listHeadEdge -> <0,0,0> -> <4,3,3> -> <1,6,3> -> <6,4,3> -> <3,5,4> -> <9,10,4> -> <12,7,4> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <1,11,6> -> <1,2,6> -> <10,12,7> -> <6,12,7>
inWhichSet 1: 1
inWhichSet 2: 2
inWhichSet 3: 2
inWhichSet 4: 2
inWhichSet 5: 2
inWhichSet 6: 6
inWhichSet 7: 6
inWhichSet 8: 6
inWhichSet 9: 6
inWhichSet 10: 6
inWhichSet 11: 11
inWhichSet 12: 12
listHeadMST -> <0,0,0> -> <2,4,1> -> <8,10,1> -> <8,6,2> -> <9,8,2> -> <5,4,2> -> <6,7,2> -> <4,3,3>
listHeadEdge -> <0,0,0> -> <1,6,3> -> <6,4,3> -> <3,5,4> -> <9,10,4> -> <12,7,4> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <1,11,6> -> <1,2,6> -> <10,12,7> -> <6,12,7>
inWhichSet 1: 1
inWhichSet 2: 2
inWhichSet 3: 2
inWhichSet 4: 2
inWhichSet 5: 2
inWhichSet 6: 1
inWhichSet 7: 1
inWhichSet 8: 1
inWhichSet 9: 1
inWhichSet 10: 1
inWhichSet 11: 11
inWhichSet 12: 12

listHeadMST -> <0,0,0> -> <2,4,1> -> <8,10,1> -> <8,6,2> -> <9,8,2> -> <5,4,2> -> <6,7,2> -> <1,6,3> -> <4,3,3>
listHeadEdge -> <0,0,0> -> <6,4,3> -> <3,5,4> -> <9,10,4> -> <12,7,4> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <1,11,6> -> <1,2,6> -> <10,12,7> -> <6,12,7>
inWhichSet 1: 1
inWhichSet 2: 1
inWhichSet 3: 1
inWhichSet 4: 1
inWhichSet 5: 1
inWhichSet 6: 1
inWhichSet 7: 1
inWhichSet 8: 1
inWhichSet 9: 1
inWhichSet 10: 1
inWhichSet 11: 11
inWhichSet 12: 12
listHeadMST -> <0,0,0> -> <2,4,1> -> <8,10,1> -> <8,6,2> -> <9,8,2> -> <5,4,2> -> <6,7,2> -> <6,4,3> -> <1,6,3> -> <4,3,3>
listHeadEdge -> <0,0,0> -> <3,5,4> -> <9,10,4> -> <12,7,4> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <1,11,6> -> <1,2,6> -> <10,12,7> -> <6,12,7>
inWhichSet 1: 1
inWhichSet 2: 1
inWhichSet 3: 1
inWhichSet 4: 1
inWhichSet 5: 1
inWhichSet 6: 1
inWhichSet 7: 1
inWhichSet 8: 1
inWhichSet 9: 1
inWhichSet 10: 1
inWhichSet 11: 11
inWhichSet 12: 1
listHeadMST -> <0,0,0> -> <2,4,1> -> <8,10,1> -> <8,6,2> -> <9,8,2> -> <5,4,2> -> <6,7,2> -> <6,4,3> -> <1,6,3> -> <4,3,3> -> <12,7,4>
listHeadEdge -> <0,0,0> -> <5,7,5> -> <3,2,5> -> <9,11,5> -> <1,11,6> -> <1,2,6> -> <10,12,7> -> <6,12,7>
inWhichSet 1: 1
inWhichSet 2: 1
inWhichSet 3: 1
inWhichSet 4: 1
inWhichSet 5: 1
inWhichSet 6: 1
inWhichSet 7: 1
inWhichSet 8: 1

inWhichSet 9: 1
inWhichSet 10: 1
inWhichSet 11: 1
inWhichSet 12: 1
listHeadMST -> <0,0,0> -> <2,4,1> -> <8,10,1> -> <8,6,2> -> <9,8,2> -> <5,4,2> -> <6,7,2> -> <6,4,3> -> <1,6,3> -> <4,3,3> -> <12,7,4> -> <9,11,5>
listHeadEdge -> <0,0,0> -> <1,11,6> -> <1,2,6> -> <10,12,7> -> <6,12,7>

MSTfile

*** A Kruskal's MST of the input graph is given below: ***
11
2 4 1
8 10 1
8 6 2
9 8 2
5 4 2
6 7 2
6 4 3
1 6 3
4 3 3
12 7 4
9 11 5
*** The total cost of a Kruskal's MST is: 28