Muhammad Ansari

December 12th, 2017

COMP 482 Algorithm Design

Project #3

**Graph.java:**

//COMP 482 Project #3

//By Muhammad Ansari

**import** java.util.\*;

**import** java.io.\*;

**public** **class** Graph {

//------------------------------------------------------

**private** ArrayList<EdgeNode>[] adjList;

**private** **int** nVertices;

**private** **int** nEdges;

**private** String fileName;

Scanner f;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Constructor \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**public** Graph(String filePath) {

fileName = filePath;

//Scan for file

**try** {

File readFile = **new** File(fileName);

f = **new** Scanner(readFile);

}

**catch**(FileNotFoundException e) {

System.***out***.println("ERR 404: File not found");

}

nVertices = f.nextInt(); //Read number of Vertices

adjList = **new** ArrayList[nVertices]; //Initialize adjacency list

**for**(**int** i = 0; i < adjList.length; i++)

adjList[i] = **new** ArrayList<EdgeNode>(); //Initialize edges in adjacency list

nEdges = 0;

//Read remaining file

**while**(f.hasNextInt()) {

**int** parentVertex = f.nextInt(); //Read parent vertex

**int** childVertex = f.nextInt(); //Read child vertex

**int** weight = f.nextInt(); //Read weight

adjList[parentVertex].add(**new** EdgeNode(parentVertex, childVertex, weight)); //Write into adjacency list

nEdges++; //Increment number of Edges

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Print graph method \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**public** **void** printGraph() {

System.***out***.print("Graph: nVertices = " + nVertices + " " + "nEdges = " + nEdges + "\nAdjacency Lists\n");

//Traverse Adjacency List

**for** (**int** i = 0; i < adjList.length; i++) {

System.***out***.print("v= " + i + " [");

//Traverse and Print Each Edge

**for** (**int** j = 0; j < adjList[i].size(); j++) {

**if**(j != adjList[i].size() -1)

System.***out***.print("(" + adjList[i].get(j).vertex1 + "," + adjList[i].get(j).vertex2 + "," + adjList[i].get(j).weight + "), ");

**else**

System.***out***.print("(" + adjList[i].get(j).vertex1 + "," + adjList[i].get(j).vertex2 + "," + adjList[i].get(j).weight + ")]\n");

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* BFS Shortest paths \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**public** SPPacket bfsShortestPaths(**int** start) {

//Array Declaration

**boolean**[] visitedArray = **new** **boolean**[nVertices];

**int**[] distanceArray = **new** **int**[nVertices];

**int**[] parentArray = **new** **int**[nVertices];

ArrayList<ArrayList<Integer>> nodeQueue = **new** ArrayList<>(); //Declare queue stack

nodeQueue.add(**new** ArrayList<Integer>()); //Initialize queue

nodeQueue.get(0).add(**new** Integer(start)); //Add starting vertex to queue

**int** i = 0;

//While queue is not empty...

**while** (!nodeQueue.get(i).isEmpty()) {

nodeQueue.add(**new** ArrayList<Integer>());

//For each vertex in the queue...

**for** (Integer j : nodeQueue.get(i)) {

//For each edge of current vertex...

**for** (EdgeNode edges : adjList[j]) {

**int** destinationVertex = edges.vertex2;

//If destination vertex has not yet been visited...

**if** (!visitedArray[destinationVertex]) {

visitedArray[destinationVertex] = **true**; //..visit it

distanceArray[destinationVertex] = distanceArray[j] + edges.weight; //..update distance

parentArray[destinationVertex] = j; //..update parent

nodeQueue.get(i + 1).add(**new** Integer(destinationVertex)); //..add it into queue

}

}

}

i++;

}

//Setting distance as 0 and parent as -1 (NULL) of starting vertex

distanceArray[start] = 0;

parentArray[start] = -1;

SPPacket result = **new** SPPacket(start, distanceArray, parentArray);

**return** result;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Dijkstra's Shortest Path Algorithm \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**public** SPPacket dijkstraShortestPaths(**int** start) {

//Array Declaration

**boolean**[] visitedArray = **new** **boolean**[nVertices];

**int**[] distanceArray = **new** **int**[nVertices];

**int**[] parentArray = **new** **int**[nVertices];

//Initialize Arrays

**for** (**int** i = 0; i < nVertices; i++)

{

visitedArray[i] = **false**; //No nodes visited yet, initialized as false

distanceArray[i] = Integer.***MAX\_VALUE***; //All vertex distances marked as infinite

parentArray[i] = -1; //All parent vertices marked as NULL

}

distanceArray[start] = 0; //Setting distance of starting vertex as 0

//Traverse Graph

**for** (**int** i = 0; i < nVertices-1; i++) {

**int** minWeight = Integer.***MAX\_VALUE***;

**int** currentVertex = -1;

//Traverse Graph

**for** (**int** j = 0; j < nVertices; j++)

//If vertex has not been visited and if its distance is less than it's minimum weight...

**if** (visitedArray[j] == **false** && distanceArray[j] <= minWeight) {

minWeight = distanceArray[j]; //update vertex's minimum weight value

currentVertex = j; //record the vertex's index number

}

visitedArray[currentVertex] = **true**; //mark as visited

//Traverse Edges of current Vertex

**for** (**int** j = 0; j < adjList[currentVertex].size(); j++) {

**int** destinationVertex = adjList[currentVertex].get(j).vertex2;

**int** edgeWeight = adjList[currentVertex].get(j).weight;

//If destination vertex has not yet been visited and its distance is greater than current weight...

**if** (!visitedArray[destinationVertex]

&& distanceArray[currentVertex] != Integer.***MAX\_VALUE***

&& distanceArray[currentVertex] + edgeWeight < distanceArray[destinationVertex]) {

distanceArray[destinationVertex] = distanceArray[currentVertex] + edgeWeight; //..update distance

parentArray[destinationVertex] = currentVertex; //..update parent

}

}

}

SPPacket result = **new** SPPacket(start, distanceArray, parentArray);

**return** result;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Bellman Ford Shortest Paths \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**public** SPPacket bellmanFordShortestPaths(**int** start) {

//Array Declaration

**int**[] distanceArray = **new** **int**[nVertices];

**int**[] parentArray = **new** **int**[nVertices];

//Initialize Arrays

**for** (**int** i = 0; i < nVertices; i++) {

distanceArray[i] = Integer.***MAX\_VALUE***; //All vertex distances marked as infinite

parentArray[i] = -1; //All parent vertices marked as NULL

}

distanceArray[start] = 0; //Setting distance of starting vertex as 0

//Traverse Graph

**for** (**int** i = 0; i < nVertices; i++) {

//Traverse Edges of current Vertex

**for** (**int** j = 0; j < adjList[i].size(); j++) {

**int** currentVertex = adjList[i].get(j).vertex1;

**int** destinationVertex = adjList[i].get(j).vertex2;

**int** edgeWeight = adjList[i].get(j).weight;

//If destination vertex's distance is greater than current weight...

**if** (distanceArray[currentVertex] != Integer.***MAX\_VALUE***

&& distanceArray[currentVertex] + edgeWeight < distanceArray[destinationVertex]) {

distanceArray[destinationVertex] = distanceArray[currentVertex] + edgeWeight; //..update distance

parentArray[destinationVertex] = currentVertex; //..update parent

}

}

}

//Traverse Graph

**for** (**int** i = 0; i < nVertices; i++){

//Traverse Edges of current Vertex

**for** (**int** j = 0; j < adjList[i].size(); j++) {

**int** currentVertex = adjList[i].get(j).vertex1;

**int** destinationVertex = adjList[i].get(j).vertex2;

**int** edgeWeight = adjList[i].get(j).weight;

//If smaller distances still exist, a negative-weight cycle exists. End Graph Algorithm.

**if** (distanceArray[currentVertex] != Integer.***MAX\_VALUE*** &&

distanceArray[currentVertex] + edgeWeight < distanceArray[destinationVertex]) {

System.***out***.println("Graph contains a negative-weight cycle");

**return** **null**;

}

}

}

SPPacket result = **new** SPPacket(start, distanceArray, parentArray);

**return** result;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Prints shortest paths\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**public** **void** printShortestPaths(SPPacket spp){

System.***out***.println("Shortest Paths from vertex " + spp.source + " to vertex\n");

//Traverse Graph

**for**(**int** i = 0; i < nVertices; i++) {

**int**[] shortestPathArray = **new** **int**[nEdges];

**int** currentVertex = 0;

**int** k = 0;

//While currentVertex is not the source Vertex

**while**(currentVertex != -1) {

shortestPathArray[k] = currentVertex;

currentVertex = spp.parent[currentVertex];

k++;

}

System.***out***.print(i + ": [");

//Traverse Shortest Path backwards and print

**for**(**int** j = k - 1; j >= 0; j--) {

**if** (j != 0)

System.***out***.print(shortestPathArray[j] + ", ");

**else**

System.***out***.print(shortestPathArray[j] + "] Path weight = " + spp.d[i] + "\n");

}

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*isStronglyConnected\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**public** **boolean** isStronglyConnected() {

Boolean visited[] = **new** Boolean[nVertices];

**for** (**int** i = 0; i < nVertices; i++)

visited[i] = **false**;

**for** (**int** i = 0; i < nVertices; i++)

**if** (visited[i] == **false**)

**return** **false**;

**for** (**int** i = 0; i < nVertices; i++)

visited[i] = **false**;

**for** (**int** i = 0; i < nVertices; i++)

**if** (visited[i] == **false**)

**return** **false**;

**return** **false**;

}//end Graph class

//place the EdgeNode class and the SPPacket class inside the Graph.java file

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**class** EdgeNode {

**int** vertex1;

**int** vertex2;

**int** weight;

**public** EdgeNode(**int** v1, **int** v2, **int** w) {

vertex1 = v1;

vertex2 = v2;

weight = w;

}

@Override

**public** String toString() {

String edgeInfo = "Parent Vertex: " + vertex1 + "\nChild Vertex: " + vertex2 + "\nWeight: " + weight;

**return** edgeInfo;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**class** SPPacket {

**int**[] d; //distance array

**int**[] parent; //parent path array

**int** source; //source vertex

**public** SPPacket(**int** start, **int**[] distance, **int**[] pp) {

source = start;

d = distance;

parent = pp;

}

**public** **int**[] getDistance() {

**return** d;

}

**public** **int**[] getParent() {

**return** parent;

}

**public** **int** getSource() {

**return** source;

}

**public** String toString() {

String packetInfo = "Source Vertex: " + source + "\nDistance Array: [";

**for**(**int** i = 0; i < d.length; i++){

**if** (i != d.length - 1)

packetInfo += d[i] + ", ";

**else**

packetInfo += d[i] + "]\nParent Array: [ ";

}

**for**(**int** i = 0; i < parent.length; i++){

**if**(i != parent.length -1)

packetInfo += parent[i] + ", ";

**else**

packetInfo += parent[i] + "]";

}

**return** packetInfo;

}

}

}