****

**Green University of Bangladesh**

**Department of Computer Science and Engineering (CSE)**

**Faculty of Sciences and Engineering**

**Semester: (Spring, Year: 2023), B.Sc. in CSE (Day)**

**CLP-3**

**Course Title: Algorithms Lab**

**Course Code: CSE-206 Section: DC**

**Student Details**

|  |  |  |
| --- | --- | --- |
| **Name** | | **ID** |
|  | Md. Sohanur Rahman | 213902106 |

**Lab Date : 21.03.2023**

**Submission Date : 27.03.2023**

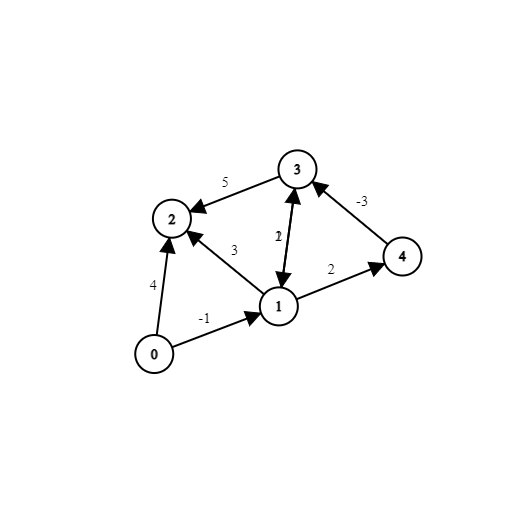
**Course Teacher’s Name : Md. Sultanul Islam Ovi**

**[For Teachers use only: Don’t Write Anything inside this box]**

|  |
| --- |
| **Lab Report Status**  **Marks: ………………………………… Signature: .....................**  **Comments: .............................................. Date: ..............................** |

**Problem Statement:** Implement Bellman Ford Algorithm to find the shortest path and detect negative cycles on a graph of your choosing.

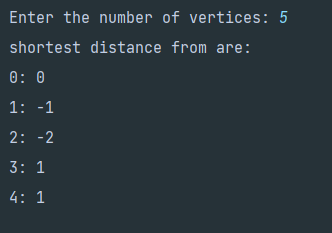
**Graph:**

****

**Code:**

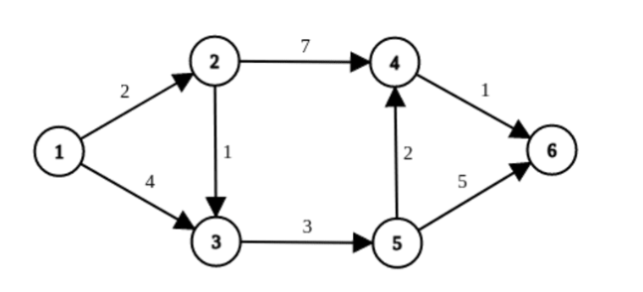
*package* Algorithms\_Lab.CLP\_3;  
  
*import* java.util.ArrayList;  
*import* java.util.*List*;  
*import* java.util.Scanner;  
  
*public class* Task1 {  
 *static int vertices*;  
 *static List*<Edge> *edges*;  
 *static int*[] *distance*;  
 *static int*[] *parent*;  
 *static class* Edge{  
 *int* src,dest,weight;  
 *public* Edge(*int* src,*int* dest,*int* weight){  
 *this*.src=src;  
 *this*.dest=dest;  
 *this*.weight=weight;  
 }  
 }  
 *static boolean* bellmanFord(*int* src) {  
 *distance* = *new int*[*vertices*];  
 *parent* = *new int*[*vertices*];  
 *for* (*int* i = 0; i < *vertices*; i++) {  
 *distance*[i] = Integer.***MAX\_VALUE***;  
 *parent*[i] = -1;  
 }  
 *distance*[src] = 0;  
 *for* (*int* i = 0; i < *vertices* - 1; i++) {  
 *for* (*int* j = 0; j < *edges*.size(); j++) {  
 Edge e = *edges*.get(j);  
 *int* u = e.src;  
 *int* v = e.dest;  
 *int* w = e.weight;  
 *if* (*distance*[u] != Integer.***MAX\_VALUE*** && *distance*[u] + w < *distance*[v]) {  
 *distance*[v] = *distance*[u] + w;  
 *parent*[v] = u;  
 }  
 }  
 }  
 *//check for negative cycle.  
 for* (*int* k = 0; k < *edges*.size(); k++) {  
 Edge e = *edges*.get(k);  
 *int* u = e.src;  
 *int* v = e.dest;  
 *int* w = e.weight;  
 *if* (*distance*[u] != Integer.***MAX\_VALUE*** && *distance*[u] + w < *distance*[v]) {  
 *return false*;  
 }  
 }  
 *return true*;  
 }  
 *//print the path  
 public static void* printPath(*int* src){  
 *if*(*bellmanFord*(src)){  
 System.***out***.println("shortest distance from are: ");  
 *for*(*int* i=0;i<*vertices*;i++){  
 System.***out***.println(i+": "+*distance*[i]);  
 }  
 }  
 *else*{  
 System.***out***.println("Negative cycle detected");  
 }  
 }  
 *public static void* main(String[] args) {  
 *edges*=*new* ArrayList<>();  
 Scanner scan=*new* Scanner(System.***in***);  
 System.***out***.print("Enter the number of vertices: ");  
 *vertices*=scan.nextInt();  
 *edges*.add(*new* Edge(0,1,-1));  
 *edges*.add(*new* Edge(0,2,4));  
 *edges*.add(*new* Edge(1,2,3));  
 *edges*.add(*new* Edge(1,3,-2));  
 *edges*.add(*new* Edge(1,4,2));  
 *edges*.add(*new* Edge(3,2,5));  
 *edges*.add(*new* Edge(3,1,-1));  
 *edges*.add(*new* Edge(4,2,-3));  
 *printPath*(0);  
 }  
}

**Output:**

****

**Problem Statement:** Implement Dijkstra’s algorithm for a single destination shortest path problem. Take node 6 as the destination. Also print each iteration.

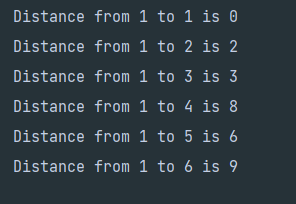
**Graph:**

****

**Code:**

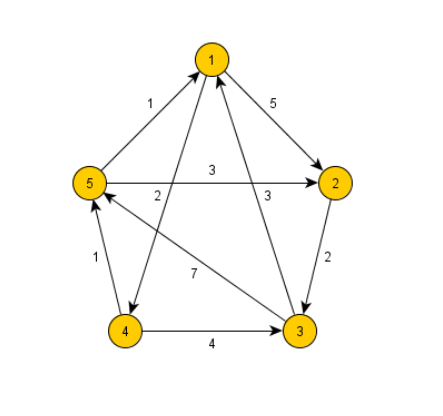
*package* Algorithms\_Lab.CLP\_3;  
  
  
*import* java.util.\*;  
  
*public class* DijkstraAlgorithm {  
 *private static int INF* = Integer.***MAX\_VALUE***;  
 *private int*[] dist;  
 *private boolean*[] visited;  
 *private int* destination;  
 *private* PriorityQueue<Node> pq;  
 *private List*<*List*<Node>> adjList;  
  
 *public* DijkstraAlgorithm(*int* numOfVertices, *int* destination) {  
 *this*.destination = destination;  
 dist = *new int*[numOfVertices];  
 visited = *new boolean*[numOfVertices];  
 pq = *new* PriorityQueue<>(numOfVertices, *new* Node());  
 adjList = *new* ArrayList<>(numOfVertices);  
  
 *for* (*int* i = 0; i < numOfVertices; i++) {  
 dist[i] = *INF*;  
 visited[i] = *false*;  
 adjList.add(*new* ArrayList<>());  
 }  
 }  
  
 *public void* addEdge(*int* source, *int* dest, *int* weight) {  
 adjList.get(source).add(*new* Node(dest, weight));  
 }  
  
 *public void* findShortestPath(*int* start) {  
 dist[start] = 0;  
 pq.offer(*new* Node(start, 0));  
  
 *while* (!pq.isEmpty()) {  
 *int* vertex = pq.poll().node;  
 visited[vertex] = *true*;  
  
 *if* (vertex == destination) {  
 *break*;  
 }  
  
 *for* (Node neighbor : adjList.get(vertex)) {  
 *if* (!visited[neighbor.node]) {  
 *int* newDist = dist[vertex] + neighbor.cost;  
  
 *if* (newDist < dist[neighbor.node]) {  
 dist[neighbor.node] = newDist;  
 pq.offer(*new* Node(neighbor.node, dist[neighbor.node]));  
 }  
 }  
 }  
 }  
 }  
  
 *public int*[] getDistances() {  
 *return* dist;  
 }  
  
 *public static void* main(String[] args) {  
 *int* numOfVertices = 6;  
 *int* destination = 6;  
 DijkstraAlgorithm obj = *new* DijkstraAlgorithm(numOfVertices, destination);  
  
 obj.addEdge(1, 2, 2);  
 obj.addEdge(1, 3, 4);  
 obj.addEdge(2, 3, 1);  
 obj.addEdge(2, 4, 7);  
 obj.addEdge(3, 5, 3);  
 obj.addEdge(4, 6, 1);  
 obj.addEdge(5, 6, 5);  
 obj.addEdge(5, 4, 2);  
  
 obj.findShortestPath(1);  
  
 *int*[] distances = obj.getDistances();  
 *for* (*int* i = 1; i < numOfVertices; i++) {  
 System.***out***.println("Distance from 1 to " + i + " is " + distances[i]);  
 }  
 }  
}  
  
*class* Node *implements Comparator*<Node> {  
 *public int* node;  
 *public int* cost;  
  
 *public* Node() {}  
  
 *public* Node(*int* node, *int* cost) {  
 *this*.node = node;  
 *this*.cost = cost;  
 }  
  
 @Override  
 *public int* compare(Node node1, Node node2) {  
 *if* (node1.cost < node2.cost) {  
 *return* -1;  
 }  
  
 *if* (node1.cost > node2.cost) {  
 *return* 1;  
 }  
  
 *return* 0;  
 }  
}

**Output:**



**Problem Statement:** Implement Floyd Warshall on the following graph. Also print each iteration.

**Graph:**

****

**Code:**

*package* Algorithms\_Lab.CLP\_3;  
  
*import* java.util.Scanner;  
  
*public class* FloydAlgorithm {  
 *static int*[][] *distance*;  
 *private static final int* ***N*** = 919;  
 *public* FloydAlgorithm(){  
 *distance*=*new int*[***N***][***N***];  
 *for*(*int* i=1;i<***N***;i++){  
 *for*(*int* j=1;j<***N***;j++){  
 *if*(i==j){  
 *distance*[i][j]=0;  
 }  
 *else* {  
 *distance*[i][j] = Integer.***MAX\_VALUE***;  
 }  
 }  
 }  
 }  
 *public void* floyd(*int* nodes) {  
 *for* (*int* k = 1; k <= nodes; k++) {  
 *for* (*int* i = 1; i <= nodes; i++) {  
 *for* (*int* j = 1; j <= nodes; j++) {  
 *if* (*distance*[i][k] + *distance*[k][j] < *distance*[i][j]) {  
 *distance*[i][j] = *distance*[i][k] + *distance*[k][j];  
 }  
 }  
 }  
 }  
 *for*(*int* i=1;i<=nodes;i++){  
 *for*(*int* j=1;j<=nodes;j++){  
 *if*(*distance*[i][j]==Integer.***MAX\_VALUE***){  
 System.***out***.print("INF ");  
 }  
 *else*{  
 System.***out***.print(*distance*[i][j]+" ");  
 }  
 }  
 System.***out***.println();  
 }  
 }  
  
 *public static void* main(String[] args) {  
 Scanner scan=*new* Scanner(System.***in***);  
 System.***out***.print("Enter the number of Nodes: ");  
 *int* n=scan.nextInt();  
 System.***out***.print("Enter the number of edges: ");  
 *int* m=scan.nextInt();  
 FloydAlgorithm obj=*new* FloydAlgorithm();  
 *for*(*int* i=1;i<=m;i++){  
 *int* src,dest,weight;  
 src=scan.nextInt();  
 dest=scan.nextInt();  
 weight=scan.nextInt();  
 *distance*[src][dest]=weight;  
 }  
 obj.floyd(n);  
 }  
}

**Output:**

There have some error.