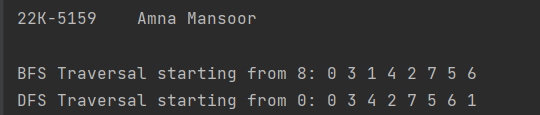
22K-5159 Amna Mansoor BSE-3B LAB-12

Task 1,3, and 6 were done in lab as instructed!

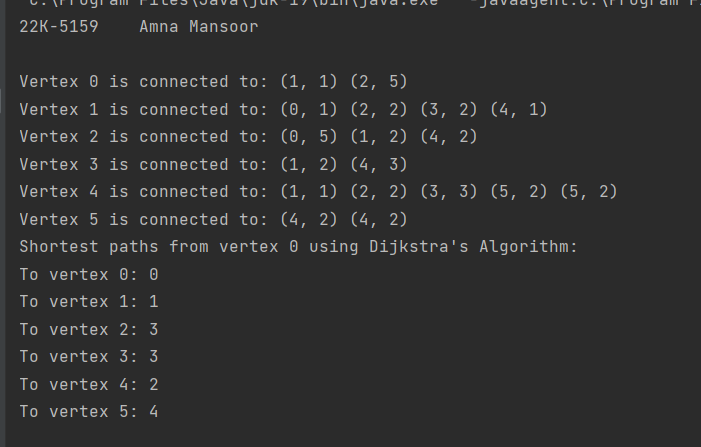
***Task1:***

import java.util.Iterator;  
import java.util.LinkedList;  
public class Task1 {  
 private int vNum;  
 private LinkedList<Integer> adj[];  
  
 Task1(int v) {  
 vNum = v;  
 adj = new LinkedList[v];  
 for (int i = 0; i < v; ++i) {  
 adj[i] = new LinkedList();  
 }  
 }  
  
 void addEdge(int v, int w) {  
 adj[v].add(w);  
 }  
  
 void BreadthFirst(int s) {  
 boolean visited[] = new boolean[vNum];  
 LinkedList<Integer> queue = new LinkedList<Integer>();  
 visited[s] = true;  
 queue.add(s);  
  
 while (queue.size() != 0) {  
 int currentvertex = queue.poll();  
 System.*out*.print(currentvertex + " ");  
  
 Iterator<Integer> i = adj[currentvertex].listIterator();  
 while (i.hasNext()) {  
 int n = i.next();  
 if (!visited[n]) {  
 visited[n] = true;  
 queue.add(n);  
 }  
 }  
 }  
 }  
  
 void DFS(int v, boolean visited[]) {  
 visited[v] = true;  
 System.*out*.print(v + " ");  
 Iterator<Integer> i = adj[v].listIterator();  
 while (i.hasNext()) {  
 int n = i.next();  
 if (!visited[n]) {  
 DFS(n, visited);  
 }  
 }  
 }  
  
 void DepthFirst(int v) {  
 boolean visited[] = new boolean[vNum];  
 DFS(v, visited);  
 }  
  
 public static void main(String[] args) {  
 System.*out*.println("22K-5159 Amna Mansoor\n");  
 Task1 Graph = new Task1(9);  
  
 Graph.addEdge(8, 0);  
 Graph.addEdge(8, 4);  
 Graph.addEdge(0, 3);  
 Graph.addEdge(3, 4);  
 Graph.addEdge(0, 1);  
 Graph.addEdge(3, 2);  
 Graph.addEdge(1, 7);  
 Graph.addEdge(2, 7);  
 Graph.addEdge(2, 5);  
 Graph.addEdge(5, 6);  
  
 System.*out*.printf("BFS Traversal starting from 8: ");  
 Graph.BreadthFirst(0);  
  
 System.*out*.printf("\nDFS Traversal starting from 0: ");  
 Graph.DepthFirst(0);  
 }  
}



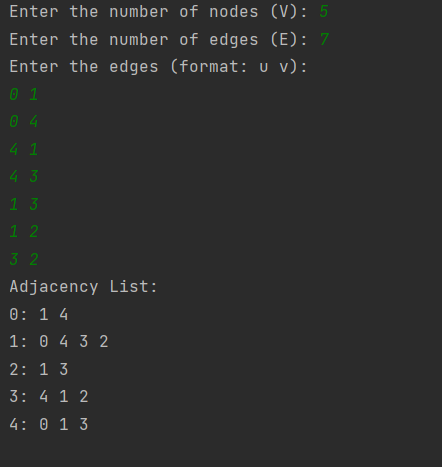
***Task 2:***

import java.util.\*;  
  
class Graph {  
 private int vertices;  
 private Map<Integer, List<Edge>> adjacencyList;  
  
 public Graph(int vertices) {  
 this.vertices = vertices;  
 this.adjacencyList = new HashMap<>();  
 for (int i = 0; i < vertices; i++) {  
 adjacencyList.put(i, new ArrayList<>());  
 }  
 }  
  
 public void insertEdge(int source, int destination, int weight) {  
 adjacencyList.get(source).add(new Edge(destination, weight));  
 adjacencyList.get(destination).add(new Edge(source, weight)); // Assuming it's an undirected graph  
 }  
  
 public void printAdjacencyList() {  
 for (Map.Entry<Integer, List<Edge>> entry : adjacencyList.entrySet()) {  
 System.*out*.print("Vertex " + entry.getKey() + " is connected to: ");  
 for (Edge edge : entry.getValue()) {  
 System.*out*.print("(" + edge.destination + ", " + edge.weight + ") ");  
 }  
 System.*out*.println();  
 }  
 }  
  
 public void dijkstra(int startVertex) {  
 PriorityQueue<Node> priorityQueue = new PriorityQueue<>(vertices, Comparator.*comparingInt*(node -> node.distance));  
 int[] distances = new int[vertices];  
 Arrays.*fill*(distances, Integer.*MAX\_VALUE*);  
 distances[startVertex] = 0;  
 priorityQueue.add(new Node(startVertex, 0));  
  
 while (!priorityQueue.isEmpty()) {  
 int currentVertex = priorityQueue.poll().vertex;  
  
 for (Edge edge : adjacencyList.get(currentVertex)) {  
 int newDistance = distances[currentVertex] + edge.weight;  
  
 if (newDistance < distances[edge.destination]) {  
 distances[edge.destination] = newDistance;  
 priorityQueue.add(new Node(edge.destination, newDistance));  
 }  
 }  
 }  
  
 System.*out*.println("Shortest paths from vertex " + startVertex + " using Dijkstra's Algorithm:");  
 for (int i = 0; i < vertices; i++) {  
 System.*out*.println("To vertex " + i + ": " + distances[i]);  
 }  
 }  
  
 public static void main(String[] args) {  
 System.*out*.println("22K-5159 Amna Mansoor\n");  
 int vertices = 6;  
 Graph g = new Graph(vertices);  
  
 g.insertEdge(0, 1, 1);  
 g.insertEdge(0, 2, 5);  
 g.insertEdge(1, 2, 2);  
 g.insertEdge(1, 3, 2);  
 g.insertEdge(1, 4, 1);  
 g.insertEdge(2, 4, 2);  
 g.insertEdge(3, 4, 3);  
 g.insertEdge(4, 5, 2);  
 g.insertEdge(4, 5, 2);  
  
 g.printAdjacencyList();  
 g.dijkstra(0);  
 }  
}  
  
class Edge {  
 int destination;  
 int weight;  
  
 public Edge(int destination, int weight) {  
 this.destination = destination;  
 this.weight = weight;  
 }  
}  
  
class Node {  
 int vertex;  
 int distance;  
  
 public Node(int vertex, int distance) {  
 this.vertex = vertex;  
 this.distance = distance;  
 }  
}



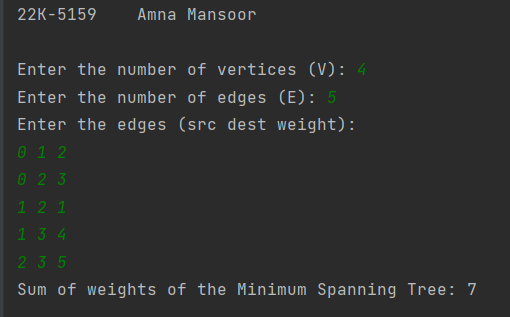
***Task 3:***

import java.util.\*;  
  
public class Task3 {  
 public static List<List<Integer>> createAdjacencyList(int V, int E, int[][] edges) {  
 List<List<Integer>> adjacencyList = new ArrayList<>();  
  
 // Initialize the adjacency list  
 for (int i = 0; i < V; ++i) {  
 adjacencyList.add(new ArrayList<>());  
 }  
  
 // Add edges to the adjacency list  
 for (int i = 0; i < E; ++i) {  
 int u = edges[i][0];  
 int v = edges[i][1];  
  
 // Adding edges for undirected graph  
 adjacencyList.get(u).add(v);  
 adjacencyList.get(v).add(u);  
 }  
  
 return adjacencyList;  
 }  
  
 public static void main(String[] args) {  
 System.*out*.println("22K-5159 Amna Mansoor\n");  
 int V = 5;  
 int E = 7;  
 int[][] edges = {  
 {0, 1},  
 {0, 4},  
 {4, 1},  
 {4, 3},  
 {1, 3},  
 {1, 2},  
 {3, 2}  
 };  
  
 List<List<Integer>> adjacencyList = *createAdjacencyList*(V, E, edges);  
  
 // Print the adjacency list  
 for (int i = 0; i < V; ++i) {  
 System.*out*.print(i + ": ");  
 for (int neighbor : adjacencyList.get(i)) {  
 System.*out*.print(neighbor + " ");  
 }  
 System.*out*.println();  
 }  
 }  
}



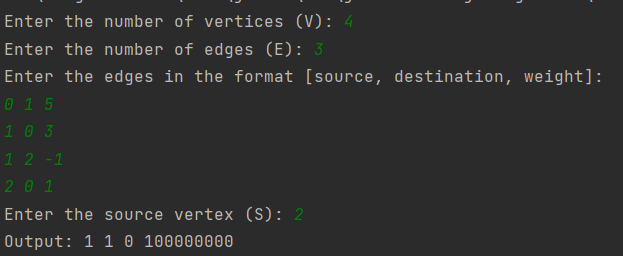
***Task 4:***

import java.util.\*;  
  
class Edge implements Comparable<Edge> {  
 int src, dest, weight;  
  
 public Edge(int src, int dest, int weight) {  
 this.src = src;  
 this.dest = dest;  
 this.weight = weight;  
 }  
  
 @Override  
 public int compareTo(Edge edge) {  
 return this.weight - edge.weight;  
 }  
}  
  
class KruskalMST {  
 private int V, E;  
 private List<Edge> edges;  
  
 public KruskalMST(int V, int E) {  
 this.V = V;  
 this.E = E;  
 edges = new ArrayList<>();  
 }  
  
 public void addEdge(int src, int dest, int weight) {  
 edges.add(new Edge(src, dest, weight));  
 }  
  
 public int kruskalMST() {  
 Collections.*sort*(edges);  
  
 int[] parent = new int[V];  
 for (int i = 0; i < V; i++) {  
 parent[i] = i;  
 }  
  
 int mstWeight = 0;  
 int edgeCount = 0;  
  
 for (Edge edge : edges) {  
 int rootSrc = find(parent, edge.src);  
 int rootDest = find(parent, edge.dest);  
  
 if (rootSrc != rootDest) {  
 mstWeight += edge.weight;  
 union(parent, rootSrc, rootDest);  
 edgeCount++;  
  
 if (edgeCount == V - 1) {  
 break; // MST is complete  
 }  
 }  
 }  
  
 return mstWeight;  
 }  
  
 private int find(int[] parent, int i) {  
 if (parent[i] != i) {  
 parent[i] = find(parent, parent[i]);  
 }  
 return parent[i];  
 }  
  
 private void union(int[] parent, int x, int y) {  
 int rootX = find(parent, x);  
 int rootY = find(parent, y);  
 parent[rootX] = rootY;  
 }  
}  
  
public class Task4{  
 public static void main(String[] args) {  
 System.*out*.println("22K-5159 Amna Mansoor\n");  
 Scanner scanner = new Scanner(System.*in*);  
  
 System.*out*.print("Enter the number of vertices (V): ");  
 int V = scanner.nextInt();  
 System.*out*.print("Enter the number of edges (E): ");  
 int E = scanner.nextInt();  
  
 KruskalMST kruskalMST = new KruskalMST(V, E);  
  
 System.*out*.println("Enter the edges (src dest weight): ");  
 for (int i = 0; i < E; i++) {  
 int src = scanner.nextInt();  
 int dest = scanner.nextInt();  
 int weight = scanner.nextInt();  
 kruskalMST.addEdge(src, dest, weight);  
 }  
  
 int result = kruskalMST.kruskalMST();  
 System.*out*.println("Sum of weights of the Minimum Spanning Tree: " + result);  
 }  
}



***Task 5:***

import java.util.Arrays;  
import java.util.Scanner;  
  
class BellmanFord {  
 static class Edge {  
 int src, dest, weight;  
  
 Edge() {  
 src = dest = weight = 0;  
 }  
 }  
  
 int V, E;  
 Edge edge[];  
  
 BellmanFord(int v, int e) {  
 V = v;  
 E = e;  
 edge = new Edge[e];  
 for (int i = 0; i < e; ++i)  
 edge[i] = new Edge();  
 }  
  
 void bellmanFord(BellmanFord graph, int src) {  
 int V = graph.V, E = graph.E;  
 int dist[] = new int[V];  
  
 Arrays.*fill*(dist, Integer.*MAX\_VALUE*);  
 dist[src] = 0;  
  
 for (int i = 1; i < V; ++i) {  
 for (int j = 0; j < E; ++j) {  
 int u = graph.edge[j].src;  
 int v = graph.edge[j].dest;  
 int weight = graph.edge[j].weight;  
 if (dist[u] != Integer.*MAX\_VALUE* && dist[u] + weight < dist[v]) {  
 dist[v] = dist[u] + weight;  
 }  
 }  
 }  
  
 for (int j = 0; j < E; ++j) {  
 int u = graph.edge[j].src;  
 int v = graph.edge[j].dest;  
 int weight = graph.edge[j].weight;  
 if (dist[u] != Integer.*MAX\_VALUE* && dist[u] + weight < dist[v]) {  
 System.*out*.println("-1");  
 return;  
 }  
 }  
  
 for (int i = 0; i < V; ++i) {  
 if (dist[i] == Integer.*MAX\_VALUE*) {  
 System.*out*.print((int) Math.*pow*(10, 8) + " ");  
 } else {  
 System.*out*.print(dist[i] + " ");  
 }  
 }  
 }  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
  
 System.*out*.print("Enter the number of vertices (V): ");  
 int V = scanner.nextInt();  
 System.*out*.print("Enter the number of edges (E): ");  
 int E = scanner.nextInt();  
  
 BellmanFord graph = new BellmanFord(V, E);  
  
 System.*out*.println("Enter the edges in the format [source, destination, weight]:");  
 for (int i = 0; i < E; i++) {  
 graph.edge[i].src = scanner.nextInt();  
 graph.edge[i].dest = scanner.nextInt();  
 graph.edge[i].weight = scanner.nextInt();  
 }  
 System.*out*.print("Enter the source vertex (S): ");  
 int S = scanner.nextInt();  
  
 System.*out*.print("Output: ");  
 graph.bellmanFord(graph, S);  
 }  
}



***Task 6:***

public class Task6 {  
 public int numIslands(int[][] grid) {  
 if (grid == null || grid.length == 0 || grid[0].length == 0) {  
 return 0;  
 }  
  
 int numIslands = 0;  
 int rows = grid.length;  
 int cols = grid[0].length;  
  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < cols; j++) {  
 if (grid[i][j] == 1) {  
 numIslands++;  
 dfs(grid, i, j);  
 }  
 }  
 }  
  
 return numIslands;  
 }  
  
 private void dfs(int[][] grid, int row, int col) {  
 int rows = grid.length;  
 int cols = grid[0].length;  
  
 if (row < 0 || col < 0 || row >= rows || col >= cols || grid[row][col] == 0) {  
 return;  
 }  
  
 // Mark the current cell as visited  
 grid[row][col] = 0;  
  
 // Explore all 8 directions  
 dfs(grid, row - 1, col);  
 dfs(grid, row + 1, col);  
 dfs(grid, row, col - 1);  
 dfs(grid, row, col + 1);  
 dfs(grid, row - 1, col - 1);  
 dfs(grid, row - 1, col + 1);  
 dfs(grid, row + 1, col - 1);  
 dfs(grid, row + 1, col + 1);  
 }  
  
 public static void main(String[] args) {  
 System.*out*.println("22K-5159 Amna Mansoor\n");  
 Task6 solution = new Task6();  
 int[][] grid = {  
 {0, 1},  
 {1, 0},  
 {1, 1},  
 {1, 0}  
 };  
  
 int result = solution.numIslands(grid);  
 System.*out*.println("Number of Islands: " + result);  
 }  
}

