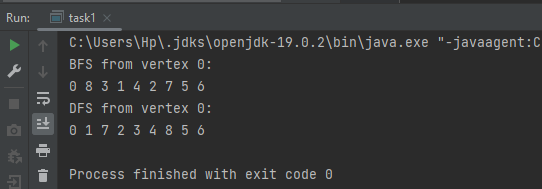
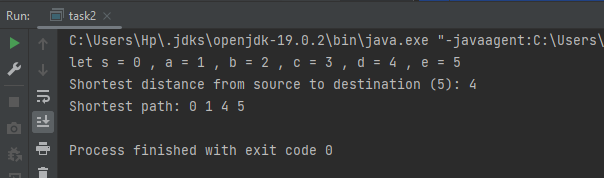
22k-5195 LAB 12

Task 1:

import java.util.LinkedList;  
import java.util.Queue;  
import java.util.Stack;  
  
class Graph {  
 LinkedList<Integer> adj[];  
 public Graph(int v) {  
 adj = new LinkedList[v];  
 for (int i = 0; i < v; ++i)  
 adj[i] = new LinkedList<>();  
 }  
 public void addEdge(int u, int v) {  
 adj[u].add(v);  
 adj[v].add(u);  
 }  
 public void DFS(int begin) {  
 boolean[] visited = new boolean[adj.length];  
 Stack<Integer> stack = new Stack<>();  
  
 stack.push(begin);  
  
 while (!stack.isEmpty()) {  
 int current = stack.pop();  
  
 if (!visited[current]) {  
 visited[current] = true;  
 System.*out*.print(current + " ");  
  
 for (int neighbor : adj[current]) {  
 if (!visited[neighbor]) {  
 stack.push(neighbor);  
 }  
 }  
 }  
 }  
 System.*out*.println();  
 }  
 public void BFS(int begin) {  
 boolean[] visited = new boolean[adj.length];  
 Queue<Integer> queue = new LinkedList<>();  
  
 visited[begin] = true;  
 queue.add(begin);  
  
 while (!queue.isEmpty()) {  
 int current = queue.poll();  
 System.*out*.print(current + " ");  
  
 for (int neighbor : adj[current]) {  
 if (!visited[neighbor]) {  
 visited[neighbor] = true;  
 queue.add(neighbor);  
 }  
 }  
 }  
 System.*out*.println();  
 }  
  
}  
  
public class task1 {  
 public static void main(String[] args) {  
 Graph graph = new Graph(9);  
  
 graph.addEdge(0, 8);  
 graph.addEdge(0, 3);  
 graph.addEdge(0, 1);  
 graph.addEdge(1, 7);  
 graph.addEdge(2, 5);  
 graph.addEdge(3, 4);  
 graph.addEdge(3, 2);  
 graph.addEdge(5, 6);  
 graph.addEdge(7, 2);  
 graph.addEdge(8, 4);  
  
 System.*out*.println("BFS from vertex 0:");  
 graph.BFS(0);  
  
 System.*out*.println("DFS from vertex 0:");  
 graph.DFS(0);  
 }  
}

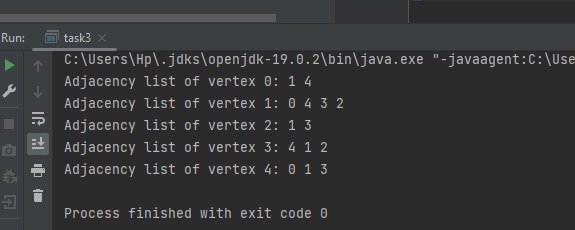


Task2:

import java.lang.\*;  
import java.util.\*;  
  
class SmallestPath {  
 static final int *V* = 6;  
 int minDis(int dist[], Boolean sptSet[])  
 {  
 int min = Integer.*MAX\_VALUE*, min\_index = -1;  
  
 for (int v = 0; v < *V*; v++)  
 if (sptSet[v] == false && dist[v] <= min) {  
 min = dist[v];  
 min\_index = v;  
 }  
  
 return min\_index;  
 }  
 void printPath(int[] pred, int dest) {  
 if (dest == -1) {  
 return;  
 }  
 printPath(pred, pred[dest]);  
 System.*out*.print(dest + " ");  
 }  
 void display(int dist[], int dest, int[] pred) {  
 System.*out*.println("Shortest distance from source to destination (" + dest + "): " + dist[dest]);  
  
 System.*out*.print("Shortest path: ");  
 printPath(pred, dest);  
 System.*out*.println();  
 }  
 void dijkstra(int graph[][], int start, int fin) {  
 int dist[] = new int[*V*];  
 int pred[] = new int[*V*];  
 Boolean sptSet[] = new Boolean[*V*];  
  
 Arrays.*fill*(dist, Integer.*MAX\_VALUE*);  
 Arrays.*fill*(sptSet, false);  
  
 dist[start] = 0;  
 pred[start] = -1;  
  
 for (int count = 0; count < *V* - 1; count++) {  
 int u = minDis(dist, sptSet);  
 sptSet[u] = true;  
  
 for (int v = 0; v < *V*; v++) {  
 if (!sptSet[v] && graph[u][v] != 0 && dist[u] != Integer.*MAX\_VALUE*  
&& dist[u] + graph[u][v] < dist[v]) {  
 dist[v] = dist[u] + graph[u][v];  
 pred[v] = u;  
 }  
 }  
 }  
 display(dist, fin, pred);  
 }  
   
  
   
}  
public class task2{  
 public static void main(String[] args) {  
 int graph[][] = new int[][] {  
 { 0, 1, 5, 0, 0, 0 },  
 { 0, 0, 2, 2, 1, 0 },  
 { 0, 0, 0, 0, 2, 0 },  
 { 0, 0, 0, 0, 3, 1 },  
 { 0, 0, 0, 0, 0, 2 },  
 { 0, 0, 0, 0, 0, 0 }  
 };  
 System.*out*.println("let s = 0 , a = 1 , b = 2 , c = 3 , d = 4 , e = 5");  
 SmallestPath t = new SmallestPath();  
 t.dijkstra(graph, 0, 5);  
 }  
}

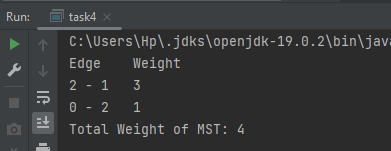
Task3:

import java.util.ArrayList;  
import java.util.LinkedList;  
import java.util.List;  
  
  
class Graph {  
 final int V;  
 final List<List<Integer>> adjList;  
  
 public Graph(int v) {  
 V = v;  
 adjList = new ArrayList<>(V);  
 for (int i = 0; i < V; ++i) {  
 adjList.add(new LinkedList<>());  
 }  
 }  
 public void addEdge(int u, int v) {  
 adjList.get(u).add(v);  
 adjList.get(v).add(u);  
 }  
 public void print() {  
 for (int i = 0; i < V; i++) {  
 System.*out*.print("Adjacency list of vertex " + i + ": ");  
 for (Integer neighbor : adjList.get(i)) {  
 System.*out*.print(neighbor + " ");  
 }  
 System.*out*.println();  
 }  
 }  
}  
public class task3 {  
 public static void main(String[] args) {  
 int V = 5;  
 int E = 7;  
 int[][] edges = {{0, 1}, {0, 4}, {4, 1}, {4, 3}, {1, 3}, {1, 2}, {3, 2}};  
  
 Graph graph = new Graph(V);  
  
 for (int i = 0; i < E; i++) {  
 int u = edges[i][0];  
 int v = edges[i][1];  
 graph.addEdge(u, v);  
 }  
 graph.print();  
 }  
}



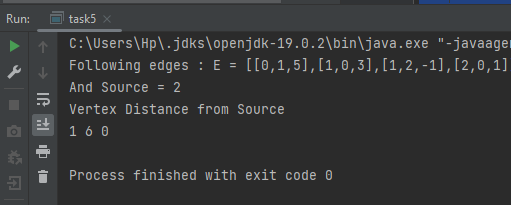
Task 4:

import java.util.Arrays;  
  
class MinSpanningTree {  
 static final int *V* = 3;  
 int minKey(int key[], Boolean mstSet[]) {  
 int min = Integer.*MAX\_VALUE*, min\_index = -1;  
  
 for (int v = 0; v < *V*; v++)  
 if (mstSet[v] == false && key[v] < min) {  
 min = key[v];  
 min\_index = v;  
 }  
  
 return min\_index;  
 }  
 void display(int parent[], int graph[][]) {  
 System.*out*.println("Edge \tWeight");  
 int totalWeight = 0;  
 for (int i = 1; i < *V*; i++) {  
 System.*out*.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);  
 totalWeight += graph[i][parent[i]];  
 }  
 System.*out*.println("Total Weight of MST: " + totalWeight);  
 }  
 void primMST(int graph[][]) {  
 int parent[] = new int[*V*];  
 int key[] = new int[*V*];  
 Boolean mstSet[] = new Boolean[*V*];  
  
 Arrays.*fill*(key, Integer.*MAX\_VALUE*);  
 Arrays.*fill*(mstSet, false);  
  
 key[0] = 0;  
 parent[0] = -1;  
  
 for (int count = 0; count < *V* - 1; count++) {  
 int u = minKey(key, mstSet);  
 mstSet[u] = true;  
  
 for (int v = 0; v < *V*; v++)  
 if (graph[u][v] != 0 && mstSet[v] == false && graph[u][v] < key[v]) {  
 parent[v] = u;  
 key[v] = graph[u][v];  
 }  
 }  
  
 display(parent, graph);  
 }  
   
}  
public class task4{  
 public static void main(String[] args) {  
 MinSpanningTree t = new MinSpanningTree();  
 int graph[][] = new int[][] {  
 { 0, 5, 2 },  
 { 5, 0, 3 },  
 { 1, 3, 0 }  
 };  
  
 t.primMST(graph);  
 }  
}



Task 5:

import java.util.\*;  
  
class Graph {  
 class Edge {  
 int source, finish, weight;  
  
 Edge() {  
 source = finish = weight = 0;  
 }  
 };  
  
 int V, E;  
 Edge edge[];  
  
 Graph(int v, int e) {  
 V = v;  
 E = e;  
 edge = new Edge[e];  
 for (int i = 0; i < e; ++i)  
 edge[i] = new Edge();  
 }  
 void BellFord(Graph 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].source;  
 int v = graph.edge[j].finish;  
 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].source;  
 int v = graph.edge[j].finish;  
 int weight = graph.edge[j].weight;  
 if (dist[u] != Integer.*MAX\_VALUE* && dist[u] + weight < dist[v]) {  
 System.*out*.println("Graph contains negative weight cycle");  
 return;  
 }  
 }  
 display(dist, V);  
 }  
 void display(int dist[], int V) {  
 System.*out*.println("Vertex Distance from Source");  
 for (int i = 0; i < V; ++i)  
 System.*out*.print(dist[i] + " ");  
 System.*out*.println();  
 }  
  
  
}  
public class task5{  
 public static void main(String[] args) {  
 int V = 3;  
 int E = 4;  
  
 Graph graph = new Graph(V, E);  
  
 graph.edge[0].source = 0;  
 graph.edge[0].finish = 1;  
 graph.edge[0].weight = 5;  
  
 graph.edge[1].source = 1;  
 graph.edge[1].finish = 0;  
 graph.edge[1].weight = 3;  
  
 graph.edge[2].source = 1;  
 graph.edge[2].finish = 2;  
 graph.edge[2].weight = -1;  
  
 graph.edge[3].source = 2;  
 graph.edge[3].finish = 0;  
 graph.edge[3].weight = 1;  
  
 System.*out*.println("Following edges : E = [[0,1,5],[1,0,3],[1,2,-1],[2,0,1]]");  
 System.*out*.println("And Source = 2");  
 graph.BellFord(graph, 2);  
 }  
}



Task6:

public class task6 {  
   
 public 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;  
 }  
  
 grid[row][col] = 0;  
  
 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 int numberOfIslands(int[][] grid) {  
 if (grid == null || grid.length == 0 || grid[0].length == 0) {  
 return 0;  
 }  
  
 int rows = grid.length;  
 int cols = grid[0].length;  
 int count = 0;  
  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < cols; j++) {  
 if (grid[i][j] == 1) {  
 count++;  
 dfs(grid, i, j);  
 }  
 }  
 }  
 return count;  
 }  
 public static void main(String[] args) {  
 int[][] grid = {  
 {0, 1},  
 {1, 0},  
 {1, 1},  
 {1, 0}  
 };  
  
 task6 islandCounter = new task6();  
 int numIslands = islandCounter.numberOfIslands(grid);  
 System.*out*.println("Grid : \n" +  
 "{0, 1},\n" +  
 "{1, 0},\n" +  
 "{1, 1},\n" +  
 "{1, 0}\n"  
 );  
 System.*out*.println("num of islands: " + numIslands);  
 }  
}

