**Lý thuyết đồ thị - Buổi 4**

**Bài 1. Liệt kê miền liên thông**

class Graph

{

private int V; // Số đỉnh trong đồ thị

private List<int>[] adj; // Danh sách kề

public Graph(int v)

{

V = v;

adj = new List<int>[V + 1];

for (int i = 1; i <= V; i++)

{

adj[i] = new List<int>();

}

}

public void AddEdge(int v, int w)

{

adj[v].Add(w);

adj[w].Add(v);

}

public List<List<int>> FindConnectedComponents()

{

List<List<int>> connectedComponents = new List<List<int>>();

bool[] visited = new bool[V + 1];

for (int i = 1; i <= V; i++)

{

if (!visited[i])

{

List<int> component = BFS(i, visited);

connectedComponents.Add(component);

}

}

return connectedComponents;

}

private List<int> BFS(int s, bool[] visited)

{

List<int> component = new List<int>();

Queue<int> queue = new Queue<int>();

visited[s] = true;

queue.Enqueue(s);

while (queue.Count != 0)

{

int u = queue.Dequeue();

component.Add(u);

foreach (int v in adj[u])

{

if (!visited[v])

{

visited[v] = true;

queue.Enqueue(v);

}

}

}

return component;

}

}

class Program

{

static void Main(string[] args)

{

string inputFilePath = Path.Combine(Directory.GetParent(Directory.GetCurrentDirectory()).Parent.FullName, "MienLienThongBFS.INP");

string outputFilePath = Path.Combine(Directory.GetParent(Directory.GetCurrentDirectory()).Parent.FullName, "MienLienThongBFS.OUT");

StreamReader reader = new StreamReader(inputFilePath);

StreamWriter writer = new StreamWriter(outputFilePath);

int n = int.Parse(reader.ReadLine());

Graph graph = new Graph(n);

for (int i = 1; i <= n; i++)

{

string[] vertices = reader.ReadLine().Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

foreach (string vertex in vertices)

{

int v = int.Parse(vertex);

graph.AddEdge(i, v);

}

}

List<List<int>> connectedComponents = graph.FindConnectedComponents();

writer.WriteLine(connectedComponents.Count);

foreach (List<int> component in connectedComponents)

{

writer.WriteLine(string.Join(" ", component));

}

reader.Close();

writer.Close();

}

}

**Bài 2. Cạnh cầu**

class Graph

{

private int V; // Số đỉnh trong đồ thị

private List<int>[] adj; // Danh sách kề

private int[] disc; // Thời điểm khám phá của đỉnh

private int[] low; // Thời điểm sớm nhất có thể đạt được từ các đỉnh kề

private int time; // Thời gian khám phá

public Graph(int v)

{

V = v;

adj = new List<int>[V + 1];

disc = new int[V + 1];

low = new int[V + 1];

for (int i = 1; i <= V; i++)

{

adj[i] = new List<int>();

disc[i] = -1;

low[i] = -1;

}

time = 0;

}

public void AddEdge(int v, int w)

{

adj[v].Add(w);

adj[w].Add(v);

}

public bool IsBridge(int u, int v)

{

// Đặt lại thời điểm khám phá và thời điểm sớm nhất của tất cả các đỉnh

for (int i = 1; i <= V; i++)

{

disc[i] = -1;

low[i] = -1;

}

// Gọi đệ quy để tìm cạnh cầu, bắt đầu từ đỉnh u

return DFS(u, v);

}

private bool DFS(int u, int v)

{

disc[u] = low[u] = ++time;

foreach (int w in adj[u])

{

if (w == v)

continue;

if (disc[w] == -1)

{

// Nếu đỉnh w chưa được thăm, tiếp tục đệ quy

if (DFS(w, v))

return true;

// Cập nhật thời điểm sớm nhất của đỉnh u

low[u] = Math.Min(low[u], low[w]);

// Nếu thời điểm sớm nhất của đỉnh w lớn hơn thời điểm khám phá của đỉnh u,

// thì cạnh (u, w) là cạnh cầu

if (low[w] > disc[u])

return true;

}

else

{

// Nếu đỉnh w đã được thăm, cập nhật thời điểm sớm nhất của đỉnh u

low[u] = Math.Min(low[u], disc[w]);

}

}

return false;

}

}

class Program

{

static void Main(string[] args)

{

string inputFilePath = Path.Combine(Directory.GetParent(Directory.GetCurrentDirectory()).Parent.FullName, "CanhCau.INP");

string outputFilePath = Path.Combine(Directory.GetParent(Directory.GetCurrentDirectory()).Parent.FullName, "CanhCau.OUT");

StreamReader reader = new StreamReader(inputFilePath);

StreamWriter writer = new StreamWriter(outputFilePath);

string[] input = reader.ReadLine().Split();

int n = int.Parse(input[0]);

int x = int.Parse(input[1]);

int y = int.Parse(input[2]);

Graph graph = new Graph(n);

for (int i = 1; i <= n; i++)

{

string[] vertices = reader.ReadLine().Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

foreach (string vertex in vertices)

{

int v = int.Parse(vertex);

graph.AddEdge(i, v);

}

}

if (graph.IsBridge(x, y))

writer.WriteLine("YES");

else

writer.WriteLine("NO");

reader.Close();

writer.Close();

}

}

**Bài 3. Đỉnh khớp**

class Graph

{

private int V; // Số đỉnh trong đồ thị

private List<int>[] adj; // Danh sách kề

private int[] disc; // Thời điểm khám phá của đỉnh

private int[] low; // Thời điểm sớm nhất có thể đạt được từ các đỉnh kề

private bool[] visited; // Mảng đánh dấu đỉnh đã thăm

private int time; // Thời gian khám phá

public Graph(int v)

{

V = v;

adj = new List<int>[V + 1];

disc = new int[V + 1];

low = new int[V + 1];

visited = new bool[V + 1];

for (int i = 1; i <= V; i++)

{

adj[i] = new List<int>();

disc[i] = -1;

low[i] = -1;

visited[i] = false;

}

time = 0;

}

public void AddEdge(int u, int v)

{

adj[u].Add(v);

adj[v].Add(u);

}

public bool IsCutVertex(int x)

{

// Đặt lại thời điểm khám phá và thời điểm sớm nhất của tất cả các đỉnh

for (int i = 1; i <= V; i++)

{

disc[i] = -1;

low[i] = -1;

visited[i] = false;

}

// Gọi đệ quy để tìm đỉnh khớp, bắt đầu từ đỉnh x

DFS(x, -1);

int count = 0;

// Đếm số miền liên thông sau khi xóa các cạnh kề của x

for (int i = 1; i <= V; i++)

{

if (!visited[i])

{

count++;

DFS(i, -1);

}

}

// Nếu số miền liên thông tăng lên ít nhất là 2 sau khi xóa các cạnh kề của x, thì x là đỉnh khớp

return count > 1;

}

private void DFS(int u, int parent)

{

disc[u] = low[u] = ++time;

visited[u] = true;

foreach (int v in adj[u])

{

if (v == parent)

continue;

if (!visited[v])

{

DFS(v, u);

low[u] = Math.Min(low[u], low[v]);

}

else

{

low[u] = Math.Min(low[u], disc[v]);

}

}

}

}

class Program

{

static void Main(string[] args)

{

try

{

string inputFilePath = Path.Combine(Directory.GetParent(Directory.GetCurrentDirectory()).Parent.FullName, "DinhKhop.INP");

string outputFilePath = Path.Combine(Directory.GetParent(Directory.GetCurrentDirectory()).Parent.FullName, "DinhKhop.OUT");

StreamReader reader = new StreamReader(inputFilePath);

StreamWriter writer = new StreamWriter(outputFilePath);

string[] input = reader.ReadLine()?.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

if (input == null || input.Length < 2)

{

Console.WriteLine("Invalid input format.");

return;

}

int n, x;

if (!int.TryParse(input[0], out n) || !int.TryParse(input[1], out x))

{

Console.WriteLine("Invalid input format.");

return;

}

Graph graph = new Graph(n);

for (int i = 1; i <= n; i++)

{

string[] vertices = reader.ReadLine()?.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

if (vertices == null)

{

Console.WriteLine("Invalid input format.");

return;

}

foreach (string vertex in vertices)

{

int v;

if (int.TryParse(vertex, out v))

{

graph.AddEdge(i, v);

}

else

{

Console.WriteLine($"Invalid vertex: {vertex}");

return;

}

}

}

if (graph.IsCutVertex(x))

writer.WriteLine("YES");

else

writer.WriteLine("NO");

Console.WriteLine("Processing completed successfully.");

reader.Close();

writer.Close();

}

catch (Exception ex)

{

Console.WriteLine($"An error occurred: {ex.Message}");

}

}

}

**Bài 4. Đi trên lưới**

class Grid

{

private int[,] grid;

private int n;

private int m;

public Grid(int[,] data)

{

grid = data;

n = grid.GetLength(0);

m = grid.GetLength(1);

}

public List<Tuple<int, int>> FindShortestPath(int i1, int j1, int i2, int j2)

{

int[,] distance = new int[n, m];

int[,] parentI = new int[n, m];

int[,] parentJ = new int[n, m];

for (int i = 0; i < n; i++)

{

for (int j = 0; j < m; j++)

{

distance[i, j] = int.MaxValue;

}

}

int[] dx = { -1, 0, 1, 0 };

int[] dy = { 0, 1, 0, -1 };

Queue<int> queueI = new Queue<int>();

Queue<int> queueJ = new Queue<int>();

queueI.Enqueue(i1);

queueJ.Enqueue(j1);

distance[i1, j1] = 0;

while (queueI.Count > 0)

{

int currentI = queueI.Dequeue();

int currentJ = queueJ.Dequeue();

for (int d = 0; d < 4; d++)

{

int ni = currentI + dx[d];

int nj = currentJ + dy[d];

if (ni >= 0 && ni < n && nj >= 0 && nj < m && grid[ni, nj] == 1 && distance[ni, nj] == int.MaxValue)

{

queueI.Enqueue(ni);

queueJ.Enqueue(nj);

distance[ni, nj] = distance[currentI, currentJ] + 1;

parentI[ni, nj] = currentI;

parentJ[ni, nj] = currentJ;

}

}

}

List<Tuple<int, int>> path = new List<Tuple<int, int>>();

int currentI2 = i2;

int currentJ2 = j2;

while (currentI2 != i1 || currentJ2 != j1)

{

path.Add(new Tuple<int, int>(currentI2, currentJ2));

int nextI = parentI[currentI2, currentJ2];

int nextJ = parentJ[currentI2, currentJ2];

currentI2 = nextI;

currentJ2 = nextJ;

}

path.Add(new Tuple<int, int>(i1, j1));

path.Reverse();

return path;

}

}

class Program

{

static void Main(string[] args)

{

string inputFilePath = Path.Combine(Directory.GetParent(Directory.GetCurrentDirectory()).Parent.FullName, "Grid.INP");

string outputFilePath = Path.Combine(Directory.GetParent(Directory.GetCurrentDirectory()).Parent.FullName, "Grid.OUT");

StreamReader reader = new StreamReader(inputFilePath);

StreamWriter writer = new StreamWriter(outputFilePath);

string[] dimensions = reader.ReadLine().Split();

int n = int.Parse(dimensions[0]);

int m = int.Parse(dimensions[1]);

string[] startEndPoints = reader.ReadLine().Split();

int i1 = int.Parse(startEndPoints[0]) - 1;

int j1 = int.Parse(startEndPoints[1]) - 1;

int i2 = int.Parse(startEndPoints[2]) - 1;

int j2 = int.Parse(startEndPoints[3]) - 1;

int[,] gridData = new int[n, m];

for (int i = 0; i < n; i++)

{

string[] row = reader.ReadLine().Split();

for (int j = 0; j < m; j++)

{

gridData[i, j] = int.Parse(row[j]);

}

}

Grid grid = new Grid(gridData);

List<Tuple<int, int>> shortestPath = grid.FindShortestPath(i1, j1, i2, j2);

if (shortestPath.Count == 0)

{

writer.WriteLine("0");

}

else

{

writer.WriteLine(shortestPath.Count);

foreach (var cell in shortestPath)

{

writer.WriteLine($"{cell.Item1 + 1} {cell.Item2 + 1}");

}

}

reader.Close();

writer.Close();

}

}