W3D4

1,

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F | G | H | I |
| A |  | 1 | 1 |  |  | 1 |  |  |  |
| B | 1 |  |  |  |  | 1 |  |  |  |
| C | 1 |  |  |  |  | 1 | 1 |  |  |
| D |  |  |  |  | 1 |  |  |  | 1 |
| E |  |  |  | 1 |  |  |  |  | 1 |
| F | 1 | 1 | 1 |  |  |  |  | 1 |  |
| G |  |  | 1 |  |  |  |  | 1 |  |
| H |  |  |  |  |  | 1 | 1 |  |  |
| I |  |  |  | 1 | 1 |  |  |  |  |

2, import java.util.InputMismatchException;

import java.util.Scanner;

import java.util.Stack;

public class GraphComponenetsUsingMatrixDFS

{

    private Stack<Integer> stack;

    public GraphComponenetsUsingMatrixDFS()

    {

            stack = new Stack<Integer>();

    }

    public void dfs(int adjacency\_matrix[][])

    {

        int number\_of\_nodes = adjacency\_matrix[0].length;

        int visited[] = new int[number\_of\_nodes];

        int cc = 0;

        for  (int vertex = 0; vertex < number\_of\_nodes; vertex++)

        {

            if (visited[vertex] == 0)

            {

                int element = vertex;

                int i = vertex;

                visited[vertex] = 1;

                cc++;

                stack.push(vertex);

                while (!stack.isEmpty())

                {

                    element = stack.peek();

                    i = element;

                    while (i < number\_of\_nodes)

                    {

                        if (adjacency\_matrix[element][i] == 1 && visited[i] == 0)

                        {

                            stack.push(i);

                            visited[i] = 1;

                            element = i;

                            i = 1;

                            continue;

                            }

                            i++;

                    }

                    stack.pop();

                }

            }

        }

        System.out.println("Number of Connected Components: " + cc);

    }

    public static void main(String...arg)

    {

        int number\_of\_nodes;

        Scanner scanner = null;

        try

        {

            System.out.println("Enter the number of nodes in the graph");

            scanner = new Scanner(System.in);

            number\_of\_nodes = scanner.nextInt();

            int adjacency\_matrix[][] = new int[number\_of\_nodes][number\_of\_nodes];

            System.out.println("Enter the adjacency matrix");

            for (int i = 0; i < number\_of\_nodes; i++)

                for (int j = 0; j < number\_of\_nodes; j++)

                       adjacency\_matrix[i][j] = scanner.nextInt();

            for (int i = 0; i < number\_of\_nodes; i++)

            {

                for (int j = 0; j < number\_of\_nodes; j++)

                {

                    if (adjacency\_matrix[i][j] == 1 && adjacency\_matrix[j][i] == 0)

                    {

                            adjacency\_matrix[j][i] = 1;

                    }

                    }

            }

            GraphComponenetsUsingMatrixDFS undirectedConnectivity= new GraphComponenetsUsingMatrixDFS();

            undirectedConnectivity.dfs(adjacency\_matrix);

        }catch(InputMismatchException inputMismatch)

        {

            System.out.println("Wrong Input format");

        }

        scanner.close();

    }

}

3,

**package** BFS;

**class** Solution {

**int**[] parent;

**int** count;

// union function

**private** **void** union(**int** a, **int** b) {

**int** parentA = parent[a];

**int** parentB = parent[b];

**if** (parentA != parentB) {

parent[parentA] = parentB;

count--;

}

}

// find function

**private** **int** find(**int** x) {

**if** (parent[x] == x) {

**return** x;

}

**return** parent[x] = find(parent[x]);

}

// return count function

**private** **int** query() {

**return** count;

}

**public** **int** countComponents(**int** n, **int**[][] edges) {

**if** (n == 0) {

**return** 0;

} **else** **if** (edges == **null** || edges.length == 0) {

**return** n;

}

count = n;

parent = **new** **int**[n];

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

parent[i] = i;

}

**for** (**int**[] edge : edges) {

**int** x = edge[0];

**int** y = edge[1];

**if** (find(x) != find(y)) {

union(x, y);

}

}

**return** query();

}

**public** **static** **void** main(String args[]) {

**int** n = 5;

**int** edges[][] = {{0, 1}, {1, 2}, {2, 3}, {3, 4}};

// 0 4

// | |

// 1 --- 2 --- 3

// Output: 1

Solution s=**new** Solution();

System.***out***.println(s.countComponents(n,edges));

}

}