Green University of Bangladesh

Department of Computer Science and Engineering (CSE)
Semester: Spring, Year: 2025, B.Sc. in CSE (Day)

Mafia & Connected Cells Problem

Course Title: Algorithm Lab
Course Code: CSE-208
Section: 232_D5

Student Details

Name	**ID**
Md. Shajalal	223002088

Submission Date: 16 May 2025
Course Teacher's Name: Md. Abu Rumman Refat

[For teacher's use only: Don't write anything inside this box]

Lab Project Status	
Marks: **Comments:**	**Signature:** **Date:**

Contents

Chapter 1

Introduction

1.1 Overview

"A set of finite rules or instructions to be followed in calculations or other problem-solving operations" - Algorithm. It's basically a method or rules for solve a problem. By using this algorithm there are given two problems from LightOJ online judge. One is "1219: Mafia", another is "Connected Cells in a Grid" problem. I have to use Java and algorithmic concept for solve this problem. Here both problems are scenario based problem.

1.2 Motivation

For solving a real life problem I do this project. Here one problem is manage mafia for some city. By solve this problem I able to manage traffic police, public servant and business branch managing. By using DFS algorithm I able to solve more specific problem.

1.3 Problem Definition

1.3.1 Problem Statement

The "Mafia" problem presents a scenario where a group of mafia boys must strategically guard a territory consisting of interconnected cities. The goal is to determine the minimum number of moves required to ensure that each city is guarded by at least one mafia boy.
Another Problem is "Connected Cells in a Grid", Here I have to Find the size of the largest connected region of filled cells (1s) in a 2D matrix, where connectivity includes all 8 possible directions (horizontal, vertical, and diagonal).

- You are given a grid where each cell is either 1 (filled) or 0 (empty).
- A region is a group of connected 1s.

- Two 1s are connected if they are adjacent in any of the 8 directions.
- Your task is to identify all such regions and return the number of cells in the largest region.

1.3.2 Complex Engineering Problem

Table 1.1: Summary of the attributes touched by the mentioned projects

Name of the P Attributess	Explain how to address
P1: Depth of knowledge required	Depth Knowledge on Java, Data Structure and Algorithm, Graph and Matrix concept. [1] [2]
P2: Range of conflicting requirements	—-
P3: Depth of analysis required	—-
P4: Familiarity of issues	—-
P5: Extent of applicable codes	For extended code of "Mafia" problem, this application use in Traffic Manage, Police distribution in every city, Business branch create in every city.
P6: Extent of stakeholder involvement and conflicting requirements	City planners, transportation authorities, residents, businesses, emergency services, public servant type people can use this project.
P7: Interdependence	—-

1.4 Design Goals/Objectives

- Develop and implement a correct and efficient DFS algorithm to traverse the graph representing the mafia territory.
- Determine the minimum number of moves required to achieve complete city coverage by the mafia boys.
- Implement the solution using an appropriate programming language Java and data structures (e.g., adjacency list).
- Graph traversal in grid-based data (like flood fill, image processing, etc.).
- Understanding connected components in a 2D space.
- Applying Depth-First Search (DFS) or Breadth-First Search (BFS) in a matrix.

1.5 Application

- Mafia

  - Network Maintenance.
  - Logistics and Delivery.
  - Emergency Response.
  - Traffic Maintenance in city
  - Police distribution in different city.
- Connected Cells in a Grid
  - Image Processing & Computer Vision
  - Satellite Imaging & Land Use Classification
  - Medical Imaging (e.g., MRI, CT scans)
  - Artificial Intelligence & Game Development
  - Disaster Mapping (e.g., Forest Fire Spread Simulation)
  - Network Reliability & Cluster Detection
  - Puzzle Solving & Logical Games

Chapter 2

Design/Development/Implementation of the Project

2.1 Introduction

The "Mafia" project focuses on the implementation and analysis of graph traversal algorithms, specifically Depth-First Search (DFS), through the lens of a captivating problem: the "Mafia" problem. In this scenario, we are tasked with optimizing the movement of a group of mafia boys across a network of interconnected cities. The objective is to determine the minimum number of moves required to ensure that every city in the territory is guarded by at least one mafia boy.
The "Connected Cells in a Grid" project, Given a 2D matrix of 0s and 1s, the goal is to determine the size of the largest region of connected 1s. A cell is considered connected to others if it is adjacent horizontally, vertically, or diagonally — totaling 8 possible directions. Each region is a group of such connected 1s.

2.2 Project Details

2.3 Implementation

Listing 2.1: LOJ-1219 - Mafia Problem Solution

```
1
2  import java.util.*;
3
4  public class Mafia {
5      static int[] boys;
6      static boolean[] visited;
7      static List<Integer>[] edges;
8      static long ans;
9
10     public static long dfs(int curr) {
11         if (visited[curr]) {
12             return 0;
```

```
        }

        long extr = boys[curr] - 1;
        visited[curr] = true;

        for (int child : edges[curr]) {
            long temp = dfs(child);
            extr += temp;
            ans += Math.abs(temp);
        }

        return extr;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int T = sc.nextInt(); // number of test cases
        for (int t = 1; t <= T; t++) {
            int n = sc.nextInt(); // number of cities

            boys = new int[n];
            visited = new boolean[n];
            edges = new ArrayList[n];
            for (int i = 0; i < n; i++) {
                edges[i] = new ArrayList<>();
            }

            ans = 0;

            for (int i = 0; i < n; i++) {
                int k = sc.nextInt() - 1; // city index (0-
                   based)
                boys[k] = sc.nextInt(); // number of mafia boys
                    at city k
                int edge = sc.nextInt(); // number of adjacent
                   cities

                for (int j = 0; j < edge; j++) {
                    int a = sc.nextInt() - 1; // adjacent city
                       index (0-based)
                    edges[k].add(a);
                    edges[a].add(k);
                }
            }

            for (int i = 0; i < n; i++) {
                if (!visited[i]) {
                    dfs(i);
                }
            }
```

```
60
61              System.out.println("Case " + t + ": " + ans);
62          }
63
64          sc.close();
65      }
66  }
```

Listing 2.2: Connected Cells in a Grid

```
69
70  import java.io.*;
71  import java.math.*;
72  import java.security.*;
73  import java.text.*;
74  import java.util.*;
75  import java.util.concurrent.*;
76  import java.util.regex.*;
77
78  public class Solution {
79
80      // Complete the connectedCell function below.
81      static int connectedCell(int[][] matrix) {
82
83          int b=0;
84      for(int x=0;x<matrix.length;x++)
85      {
86          for(int y=0;y<matrix[0].length;y++)
87          {
88              if(matrix[x][y]==1)
89              {
90                  int i=y;
91                  int l=1;
92                  Queue<Integer> q=new LinkedList<>();
93                  q.add(x*10+i);
94                  while(!q.isEmpty())
95                  {
96                      int w=q.remove();
97                      int xa=w/10;
98                      int ia=w%10;
99                      matrix[xa][ia]=-1;
100                     if(ia!=matrix[0].length-1&&matrix[xa][ia
                            +1]==1&&!q.contains(xa*10+ia+1))
101                     {
102                         q.add(xa*10+ia+1);
103                         l++;
104                     }
105                     if(x!=0&&ia!=matrix[0].length-1&&matrix[xa
                            -1][ia+1]==1&&!q.contains((xa-1)*10+ia
                            +1))
106                     {
107                         q.add((xa-1)*10+ia+1);
```

```
                    l++;
                }
                if(x!=0&&ia!=0&&matrix[xa-1][ia-1]==1&&!q.
                   contains((xa-1)*10+ia-1))
                {
                    q.add((xa-1)*10+ia-1);
                    l++;
                }
                if(ia!=0&&matrix[xa][ia-1]==1&&!q.contains
                   ((xa)*10+ia-1))
                {
                    q.add((xa)*10+ia-1);
                    l++;
                }
                if(xa!=0&&matrix[xa-1][ia]==1&&!q.contains
                   ((xa-1)*10+ia))
                {
                    q.add((xa-1)*10+ia);
                    l++;
                }
                if(xa!=matrix.length-1)
                {
                    if(ia!=matrix[0].length-1&&matrix[xa
                       +1][ia+1]==1&&!q.contains((xa+1)*10+
                       ia+1))
                    {
                        q.add((xa+1)*10+ia+1);
                        l++;
                    }
                    if(ia!=0&&matrix[xa+1][ia-1]==1&&!q.
                       contains((xa+1)*10+ia-1))
                    {
                        q.add((xa+1)*10+ia-1);
                        l++;
                    }
                    if(matrix[xa+1][ia]==1&&!q.contains((xa
                       +1)*10+ia))
                    {
                        q.add((xa+1)*10+ia);
                        l++;
                    }
                }
            }
            if(i>=0)
            {

                if(l>b)
                {
                    b=l;
                }
            }
```

```
                }
            }


        }
            return b;
        }

        private static final Scanner scanner = new Scanner(System.
            in);

        public static void main(String[] args) throws IOException {
            BufferedWriter bufferedWriter = new BufferedWriter(new
                FileWriter(System.getenv("OUTPUT_PATH")));

            int n = scanner.nextInt();
            scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

            int m = scanner.nextInt();
            scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

            int[][] matrix = new int[n][m];

            for (int i = 0; i < n; i++) {
                String[] matrixRowItems = scanner.nextLine().split(
                    " ");
                scanner.skip("(\r\n|[\n\r\u2028\u2029\u0085])?");

                for (int j = 0; j < m; j++) {
                    int matrixItem = Integer.parseInt(
                        matrixRowItems[j]);
                    matrix[i][j] = matrixItem;
                }
            }

            int result = connectedCell(matrix);

            bufferedWriter.write(String.valueOf(result));
            bufferedWriter.newLine();

            bufferedWriter.close();

            scanner.close();
        }
}
```

2.3.1 Tools and Technology

Tools and libraries

i Java Programming Language

ii Java Util Package

iii IntelliJ IDEA java compiler

iv LightOJ online judge

v HackerRank online judge

Chapter 3

Performance Evaluation

3.1 Results Analysis/Testing

3.1.1 Result_portion_1

Run this Mafia Code in Java Compiler and give input according to the LightOJ test case. Then get this output.

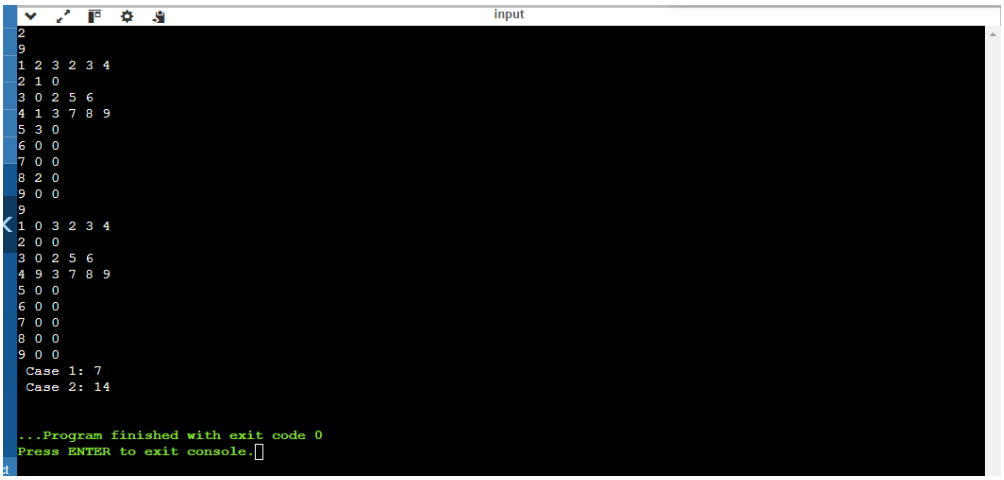

Figure 3.1: Sample input test in Java compiler

3.1.2 Result_portion_2

Submit that code in LightOJ online judge and get accepted.

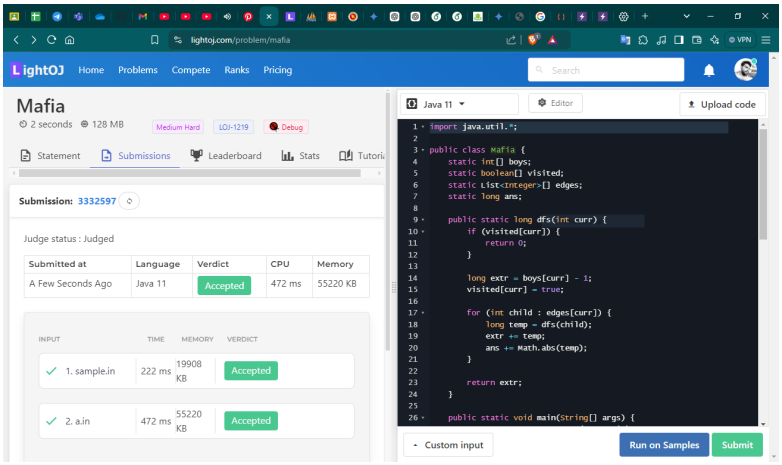

Figure 3.2: Submit code in LightOJ

3.1.3 Result_portion_3

Run this Connected Cells in a Grid Code in Hackerrank. Then get this output.

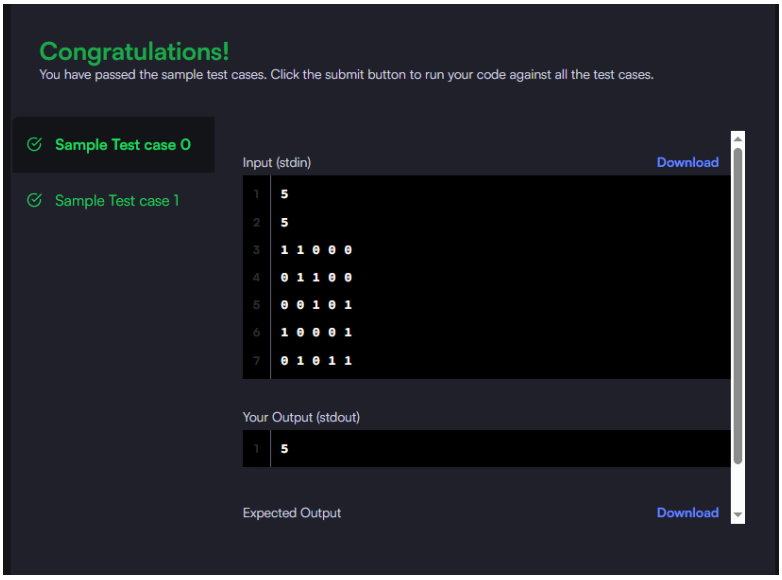

Figure 3.3: Sample input test in Java HackerRank

3.1.4 Result_portion_4

Submit that code in HackerRank online judge and get accepted.

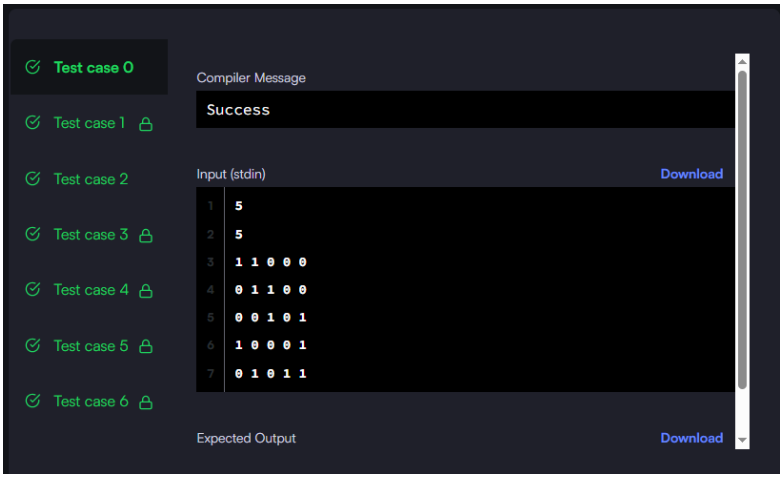

Figure 3.4: Submit code in HackerRank

3.2 Results Overall Discussion

First Output, I run the "Mafia" Problem in my Java compiler. Then give some input according to code and get the proper output. This is my first output. After complete first output I move to the second output, here Submit this code into lightoj online judge and it accepted my code. Then solve the second problem solution efficiently solves the problem using BFS and queue-based traversal, correctly handling all 8 directions. It accurately computes the largest region of connected filled cells in a 2D grid. This problem is a practical example of graph traversal in 2D grids, commonly found in image processing and geographical mapping.

Chapter 4

Conclusion

4.1 Discussion

The implementation of these code little bit easy but exact expected output code was hard. In the first problem, firstly I was tried to understood the problem, then tried to understood the input output formation. Then saw some sample code of this program and then i implement the code. Sometime after run it don't give me expected output. Then I find the problem, troubleshooting code and finally get the proper code. Here most hardest problem was implement DFS logic. This seconds problem solution focuses on identifying the largest region of connected 1s in a 2D grid using a Breadth-First Search (BFS) algorithm. It demonstrates how grid traversal and region detection can solve problems involving spatial connectivity. The solution efficiently explores all 8 directions around each filled cell to find connected regions. This approach has real-world applications in image processing, medical imaging, and satellite mapping, where identifying connected areas is essential. Overall, the project highlights the practical use of graph-based algorithms in solving real-life data structure problems.

References

[1] CPA John Kimani and James Scott. *Introduction to Algorithms Professional Level*. Finstock Evarsity Publishers, 2023.

[2] Robert Sedgewick and Kevin Wayne. *Algorithms*. Addison-wesley professional, 2011.