Problem 1: Mafia

```
import java.util.*;

public class Mafia {
    static int[] boys;
    static boolean[] visited;
    static List<Integer>[] edges;
    static long ans;

    public static long dfs(int curr) {
        if (visited[curr]) {
            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);
                }
            }
```

```
            System.out.println("Case " + t + ": " + ans);
        }

        sc.close();
    }
}
```

Problem 2: Connected Cells in a Grid

```
import java.io.*;
import java.math.*;
import java.security.*;
import java.text.*;
import java.util.*;
import java.util.concurrent.*;
import java.util.regex.*;

public class Solution {

    // Complete the connectedCell function below.
    static int connectedCell(int[][] matrix) {

        int b=0;
    for(int x=0;x<matrix.length;x++)
    {
        for(int y=0;y<matrix[0].length;y++)
        {
            if(matrix[x][y]==1)
            {
                int i=y;
                int l=1;
                Queue<Integer> q=new LinkedList<>();
                q.add(x*10+i);
                while(!q.isEmpty())
                {
                    int w=q.remove();
                    int xa=w/10;
                    int ia=w%10;
                    matrix[xa][ia]=-1;
                    if(ia!=matrix[0].length-1&&matrix[xa][ia+1]==1&&!q.contains(xa*10+ia+1))
                    {
                        q.add(xa*10+ia+1);
                        l++;
                    }
                    if(x!=0&&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(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();
    }
}
```