Competitive Programming Lab - 10

Academic year: 2020-2021 **Semester:** Long Sem

Faculty Name: Dr. D Sumathi mam **Date:** 16/ 7/ 2022

Student name: Taran Mamidala **Reg. no.:** 19BCE7346

MINIMUM STEPS FOR COMPLETING SNAKE AND LADDERS GAME :

Q1.) Markov takes out his Snakes and Ladders game, stares at the board and wonders: "If I can always roll the die to whatever number I want, what would be the least number of rolls to reach the destination?"

Rules The game is played with a cubic die of faces numbered to .
1. Starting from square , land on square with the exact roll of the die. If moving the number rolled would place the player beyond square , no move is made.
2. If a player lands at the base of a ladder, the player must climb the ladder. Ladders go up only.
3. If a player lands at the mouth of a snake, the player must go down the snake and come out through the tail. Snakes go down only

Function Description

Complete the quickestWayUp function in the editor below. It should return an integer that represents the minimum number of moves required.

quickestWayUp has the following parameter(s):
· ladders: a 2D integer array where each contains the start and end cell numbers of a ladder
· snakes: a 2D integer array where each contains the start and end cell numbers of a snake

Input Format :
The first line contains the number of tests .

For each test case: - The first line contains the number of ladders. - Each of the next lines contains two space-separated integers, the start and end of a ladder. - The next line contains the integer , the number of snakes. - Each of the next lines contains two space-separated integers, the start and end of a snake.

Constraints :

The board is always with squares numbered to . Neither square nor square will be the starting point of a ladder or snake. A square will have at most one endpoint from either a snake or a ladder.

Output Format :
For each of the t test cases, print the least number of rolls to move from start to finish on a separate line. If there is no solution, print -1.

Sample Input :

```
2
3
32 62
42 68
12 98
```

```
7
95 13
97 25
93 37
79 27
75 19
49 47
67 17
```

```
4
8 52
6 80
26 42
2 72
```

```
9
51 19
39 11
37 29
81 3
9 5
79 23
53 7
43 33
7 21
```

Sample Output

```
3
5
```

Explanation

For the first test:

The player can roll a and a to land at square . There is a ladder to the square . A roll ends the traverse in rolls.

For the second test:

The player first rolls and climbs the ladder to square . Three rolls of get to square . A final roll of lands on the target square in total rolls.

CODE :

```
//Solved using Dijkstra algorithm
import java.util.*;
import java.io.*;
public class SnknLadMinSteps {
	public static void main(String[] args) {
	Scanner in = new Scanner(System.in);
	int testCase = in .nextInt();
	for (int h = 0; h < testCase; h++) {
		int w[][] = new int[101][101];
		for (int i = 1; i <= 100; i++) {
			for (int j = 1; j <= 100; j++) {
			if (i != j && i < j && Math.abs(j - i) <= 6) {
				w[i][j] = 1;
			} else {
				w[i][j] = 10000;
			}
			}
		}
		int ladder = in.nextInt();
		for (int i = 0; i < ladder; i++) {
			int x = in .nextInt(), y = in .nextInt();
			w[x][y] = 0;
		}
```

```
            int snake = in.nextInt();
            int snakes[] = new int[snake];
            for (int i = 0; i < snake; i++) {
                  int x = in .nextInt(), y = in .nextInt();
                  snakes[i] = x;
                  w[x][y] = 0;
            }
            Arrays.sort(snakes);
            int count = 0;
            for (int temp = 0; temp < snake; temp++) {
                  if (temp > 0 && snakes[temp] - snakes[temp - 1] == 1) {
                  ++count;
                  } else {
                  count = 0;
                  }
                  if (count >= 6) {
                  break;
                  }
            }
            Stack < Integer > t = new Stack < Integer > ();
            int src = 1, des = 100;
            for (int i = 1; i <= 100; i++) {
                  if (i != src) {
                  t.push(i);
                  }
            }
            Stack < Integer > p = new Stack < Integer > ();
            p.push(src);
            while (!t.isEmpty()) {
                  int min = 9999, loc = -1;
                  for (int i = 0; i < t.size(); i++) {
                  w[src][t.elementAt(i)] = Math.min(w[src][t.elementAt(i)],
w[src][p.peek()] + w[p.peek()][t.elementAt(i)]);
                  if (w[src][t.elementAt(i)] <= min) {
                        min = (int) w[src][t.elementAt(i)];
                        loc = i;
                  }
                  }

                  p.push(t.elementAt(loc));
                  t.removeElementAt(loc);
            }
```

```
            if (count >= 6) {
                  System.out.println("-1");
                  continue;
            }
            if (w[src][des] != 10000) {
                  System.out.println(w[src][des]);
            } else {
                  System.out.println("-1");
            }

      }
      }
}
```

OUTPUT:

Result

CPU Time: 0.20 sec(s), Memory: 37252 kilobyte(s)

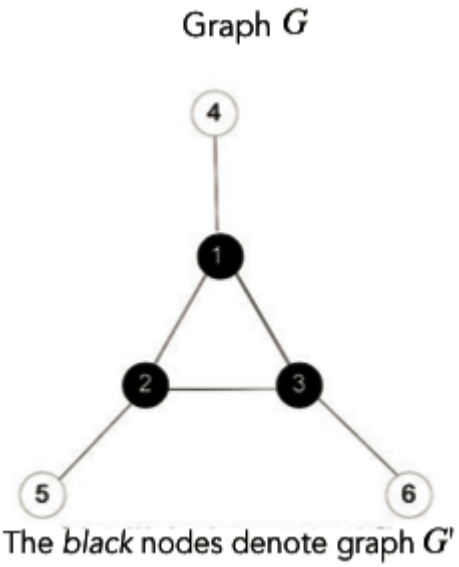
```
3
5
```

NUMBER OF TRIANGLES IN DIRECTED GRAPH

Q2.) We define the following:

· The number of triangles in an undirected graph, , is the number of unordered triples such that , , and are edges in . · Graph is a non-empty vertex-induced subgraph such that the number of triangles divided by the number of nodes in is maximal.

For example, consider the following graph



Given graph , find and print according to the Output Format specified below. If there are multiple such graphs, you may print any one of them.

Input Format

The first line contains an integer denoting (the number of vertices in). Each line of the subsequent lines contains space-separated binary integers where each integer is a if there is an edge between vertices and and a if there is not.

Output Format

On the first line, print an integer, , denoting the number of vertices in . On the second line, print distinct space-separated integers describing the respective ID numbers of the vertices in the graph .

Sample Input 0

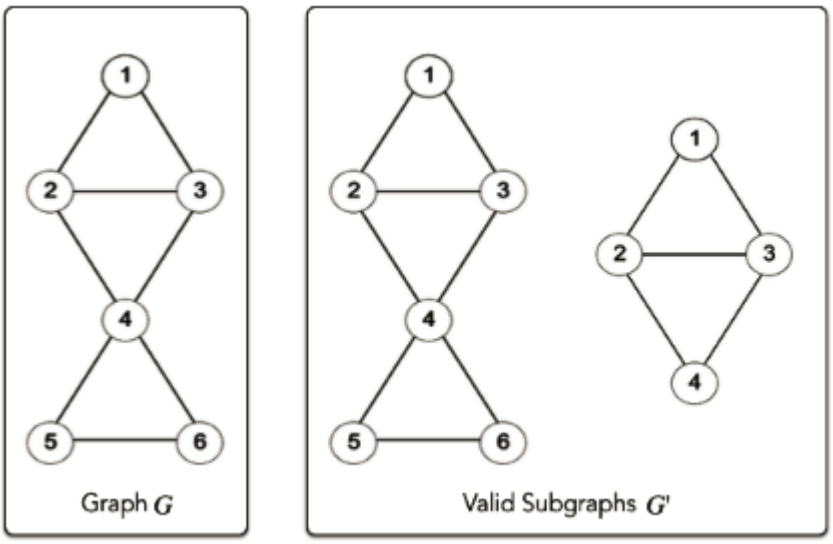
```
6
0 1 1 0 0 0
1 0 1 1 0 0
1 1 0 1 0 0
0 1 1 0 1 1
0 0 0 1 0 1
0 0 0 1 1 0
```

Sample Output 0

```
4
```

```
1 2 3 4
```

Explanation 0



- If we choose vertices , , and , then the induced subgraph contains triangles. We then calculate:
- If we choose vertices , , , and , then the induced subgraph contains triangles (i.e., and). We then calculate:
- If we choose all vertices (i.e., , , , , , and), then the induced subgraph contains triangles. We then calculate:

Because a consisting of either or both result in a maximal fraction, then either of the following are valid answers:

```
4
1 2 3 4
6
1 2 3 4 5 6
```

Sample Input 1

```
6
0 1 1 1 0 0
1 0 1 0 1 0
1 1 0 0 0 1
1 0 0 0 0 0
0 1 0 0 0 0
0 0 1 0 0 0
```

Sample Output 1

```
3
1 2 3
```

Explanation 1

This graph corresponds to the image in Problem Statement above. There is only one possible triangle, and it's formed by the set of nodes . We then calculate:

Because no other exists, we print this graph as our answer.

CODE:

```
import java.io.*;
import java.util.*;

public class TrianglesOfGraph {

     public static void main(String[] args) {
     Scanner in = new Scanner(System.in);
     int n = in.nextInt();
     int[][] g = new int[n][n];
     for(int g_i=0; g_i < n; g_i++){
          for(int g_j=0; g_j < n; g_j++){
               g[g_i][g_j] = in.nextInt();
          }
     }
     double ratio = -1;
     TreeSet<Integer> result = new TreeSet<>();
     for (int i = 0; i < n-2; ++i) {
          for (int j = i+1; j < n-1; ++j) {
               if (g[i][j] == 1) {
               for (int k = j+1; k < n; ++k) {
                    if (g[i][k] == 1) {

                         TreeSet<Integer> temp = new TreeSet(result);
                         temp.add(i);
                         temp.add(j);
                         temp.add(k);
                         double r = GetTriangleOverNodeRatio(temp, g);
                         if (r > ratio) {
```

```
                                result = temp;
                                ratio = r;
                           }
                      }
                 }
                 }
           }
     }
     if (result.size() == 0) {
           System.out.println(n);
           for (int i = 0; i < n; ++i) {
                 System.out.print((i + 1) + " ");
           }
           System.out.println();
     } else {
           System.out.println(result.size());
           while (result.isEmpty() == false) {
                 System.out.print((result.pollFirst() + 1) + " ");
           }
           System.out.println();
     }
     }

     public static double GetTriangleOverNodeRatio(TreeSet<Integer> set, int[][]
g) {
     int[][] g1 = new int[set.size()][set.size()];
     Object[] arr = set.toArray();
     for (int i = 0; i < arr.length - 1; i++) {
           for (int j = i+1; j < arr.length; j++) {
                 if (g[(int)arr[i]][(int)arr[j]] == 1) {
                 g1[i][j] = 1;
                 g1[j][i] = 1;
                 }
           }
     }
     return CountTriangle(g1) * 1.0 / set.size();
     }
     public static int CountTriangle(int[][] g) {
     int[][] path3 = MatrixMult(g, MatrixMult(g, g));
     int s = 0;
     for (int i = 0; i < path3.length; ++i) {
           s+= path3[i][i];
```

```
      }
      return s / 6;
      }
      public static int[][] MatrixMult(int[][] a, int[][] b) {
       int[][] c = new int[a.length][b[0].length];
       for (int i = 0; i < c.length; ++i) {
            for (int j = 0; j < c[i].length; ++j) {
                 c[i][j] = 0;
            }
       }
       for (int i = 0; i < c.length; ++i) {
            for (int j = 0; j < c[i].length; ++j) {
                 for (int k = 0; k < a[0].length; ++k) {
                      c[i][j] += a[i][k] * b[k][j];
                 }
            }
       }
       return c;
    }
}
```

OUTPUT:

Result

CPU Time: 0.28 sec(s), Memory: 38976 kilobyte(s)

```
4
1 2 3 4
```

Input 2:

```
6
0 1 1 0 0 1
1 0 1 1 1 1
1 1 0 1 0 1
0 1 1 0 1 0
0 0 1 1 0 1
1 1 1 0 0 1
```

Result

CPU Time: 0.21 sec(s), Memory: 37264 kilobyte(s)

```
6
1 2 3 4 5 6
```