Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 8 REPORT

FURKAN OZEV
161044036

Course Assistant: AYSE SERBETCI TURAN

1 INTRODUCTION

1.1 Problem Definition

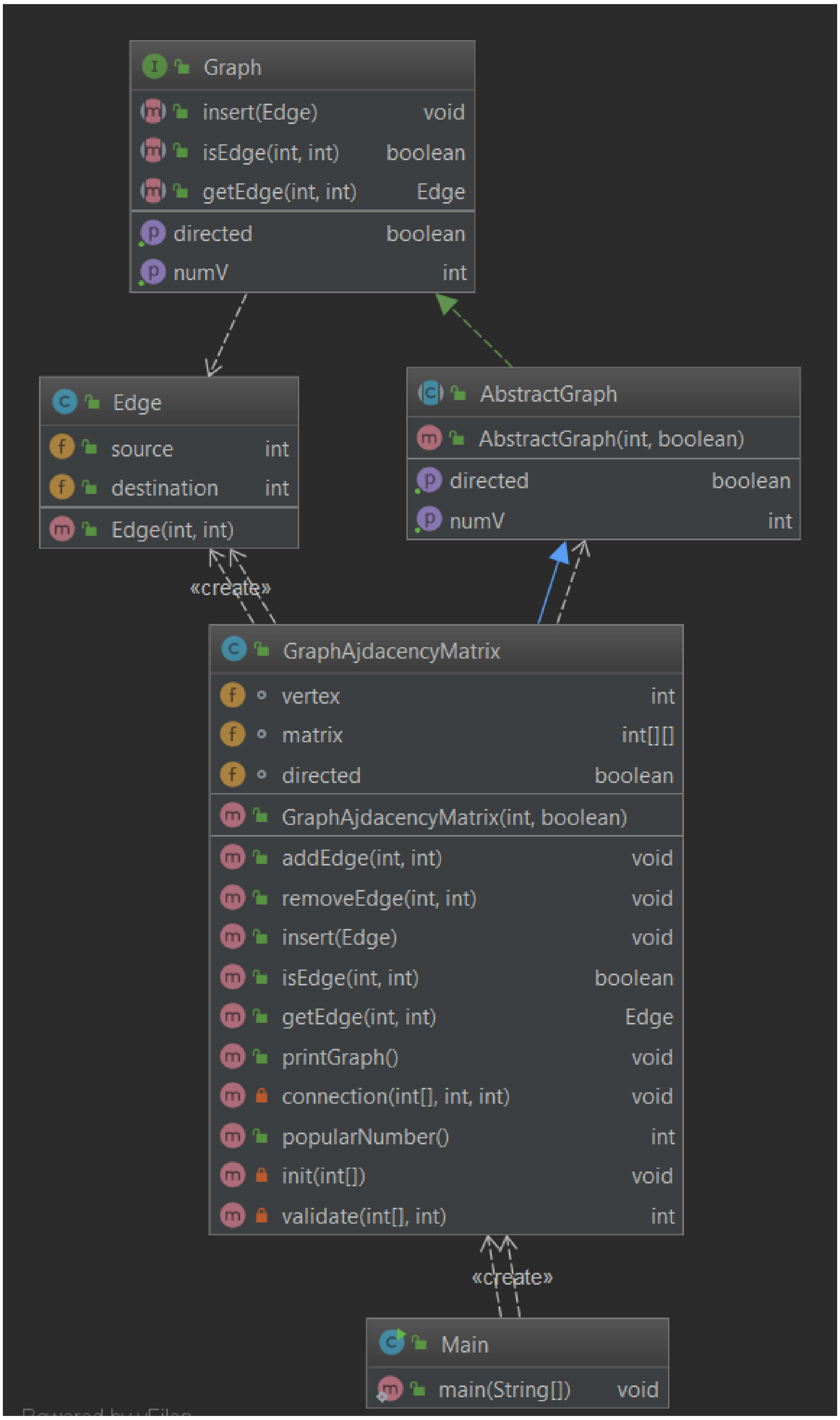
In this assignment, we are asked to create graph and apply some operations requested with the given file. If we look at the problem, We have a group of people in which an ordered popularity relation is defined between person pairs. There may be certain relations between these people. We will call these relations as people seeing each other as popular. For example, If there exist a relation such that (P1,P2) this means that A thinks that B is popular. Also, If there exist a relation such that (P2, P3) this means that B thinks that C is popular. When we consider these relations, We can say that A thinks C is popular. Because relations is transitive which means that if the relations (P1,P2) and (P2,P3) exist, than (P1,P3) also exist event if it is not specified by the input pairs. We are supposed to write a Java program which finds the people who are considered popular by every other person. A file named input.txt will be given as input to the program. Each row of this file has 2 numbers separated by spaces. The first number in the first line indicates the number of people. The second number in the first line indicates the number of ordered relations. So, it indicates total number of edges and also, number of remaining rows. The first number in the other lines indicates source vertex. The second number in the other lines indicates destination vertex. It means that, source vertex thinks destination vertex is popular. After reading the file and establishing the necessary relations, we are asked to print the output on screen. Output is an integer number which represents the number of people who are considered popular by every other person.

1.2 System Requirements

You must have java installed on your operating system to run programs. Running programs with the current version of Java will be useful for efficiency. Programs can work in both windows, linux and other operating systems installed java. The program was written using jdk-11.0.2 in IntelliJ IDEA. It requires 9,48 kb memory to keep source files. Project size is 1.43 mb with javadoc and input file. In order to run the program, additional input file is required. An input file named input.txt created in the specified format must be located in the project folder to perform the requested operation. For the programs to work properly, input file must be appropriate for the program.

2 METHOD

2.1 Class Diagrams



2.2 Use Case Diagrams

An input file named input.txt is required to perform the requested operations. Then the user only runs the program. The results will be seen on the screen. So, Use case diagram is not necessary.

2.3 Problem Solution Approach

First, I implemented the graph class to keep people (as vertices) and their relations (as edges). When implementing, I paid attention to the hierarchy. I have implemented the necessary interface and abstract class. I used the adjacency matrix class because I thought it would be easier for us to solve the problem. I assumed each person as a vertex. I assumed that each relationship as an edge. After that, the queue came to read the file as input and create and fill the graph. After opening the file and reading the first line, I created a graph that has N number of vertex. N is first number in the first line. Then I continued to read M more lines. M is second number in the first line. I've added the relations in graph by reading each one in a row in order of each other. As the problem definition is mentioned, the first digit in each row will be the source, the second digit in each row will be destination. I paid attention to it when adding relations. As we know, We are supposed to finds the people who are considered popular by every other person. The point we have to pay attention to here is that the relations are transitive. For example, If A thinks B is popular and, B thinks C is popular, We can say that A thinks C is popular. So after reading the relations from the file and adding it to the graph, we need to determine the transitive relations. I need to find number of people who are considered popular by every other person. I developed my own algorithm to find this number. I created a one-dimensional array named flag. I repeat the next process for each vertex. I assign 0 to the elements of the flag array. In order to find the relations in the next vertex, I send the vertex with the flag array to a method called connections. The Connections method finds the vertices that see the vertex as popular, and fills the index with the corresponding index 1. It simply does this: First, go to each vertex and see if there is a relation, if there is no relation, the element value that shows the index will be 0. If there is a relation with the index i 1 then fthe element value that shows the index will be 1. After that, call the method again to specifying the transitive relationship and fill the relevant corresponding index 1. Finally, If all the vertices except own thinks vertex that is parameter is popular, it increases the popular number counter 1. After repeating all these steps for each vertex, the popular number counter is returning. Finally the returned counter number is printed on display. We have successfully solved the desired problem.

Problem Solution Complexity:

n is amount of vertices.

```
public int popularNumber()     popularNumber() method complexity will be T(n)
{
    int number = 0;
    int[] flags = new int[vertex];
    for(int i = 0; i < vertex; i++)    complexity of loop is O(n)
    {
        init(flags);                  init() method complexity will be T1(n)
        connection(flags,0, i);      connection() method complexity will be T2(n)
        number += validate(flags, i); validate() method complexity will be T3(n)
    }
    return number;
}
```

T(n) = O(n) * (T1(n) + T2(n) + T3(n))

T(n) = O(n) * max(T1(n) , T2(n) , T3(n))

```
private void init(int[] arr)   init() method complexity is T1(n)
{
    for(int i = 0 ; i < arr.length ; i++)      complexity of loop is O(n)
    {
        arr[i] = 0;                            // arr.length = vertex
    }
}
```

T(n) = O(n) * max(O(n) , T2(n) , T3(n))

```
private int validate(int[] arr, int index)    init() method complexity is T3(n)
{
    for(int i = 0 ; i < arr.length ; i++)      complexity of loop is O(n)
    {
        if(i != index && arr[i] == 0) return 0;
    }                                                  // arr.length = vertex
    return 1;
}
```

T(n) = O(n) * max(O(n) , T2(n) , O(n))

```
private void connection(int[] arr, int index, int target)
//connection() method complexity is T2(n)
{
    if(index < 0 || index > vertex) return;            // O(1)
    else if (validate(arr,target) == 1) return;        // O(1)
    else
    {
        for(int i = 0 ; i < vertex ; i++)              // O(n)
        {
            if(arr[i] != 1)
            {
                if(matrix[i][target] == 1)
                {
                    arr[i] = 1;
                    connection(arr,0,i);               // recursive calling
                }                         // to calculate, use Master Theorem
            }
        }
    }
}
```

T2(n) = 1 * T(n / 1) + O(n)

a = 1 , b = 1 , k = 1

a == b^k (1 == 1^1) so, According to Master theorem, T2(n) = O(n*logn)

T(n) = O(n) * max(O(n) , O(n*logn) , O(n))

T(n) = O(n) * O(n*logn)

T(n) = O(n^2 * logn)

Problem solution’s time complexity is O(n^2 * logn)

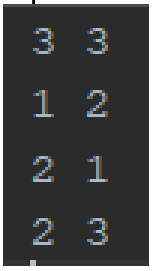
3 RESULT

3.1 Test Cases

I've tested each method one by one after implementing it. I've retested with different vertices and relations. I tested and checked the graphs after a few additions to test if the some methods were working correctly. After making sure that they were working properly, I went to test other methods. I have done the same controls from the beginning to the end for the processes that are printed on the screen. I've tested the correct functioning of the program by printing the 2D matrix and relationships on the screen. I've tried the diffirent vertices and relations when I'm sure there's no problem for this files. When I decided that there was nothing wrong with it, I decided the program was valid. Of course, in all these tests I've encountered certain errors, after correcting these errors I started to test from the beginning. I was convinced that the program was valid.

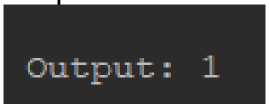
3.2 Running Results

Input 1 :



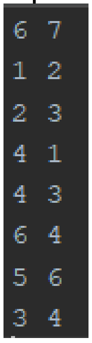
```
3 3
1 2
2 1
2 3
```

Output 1 :



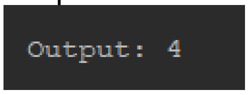
```
Output: 1
```

Input 2 :



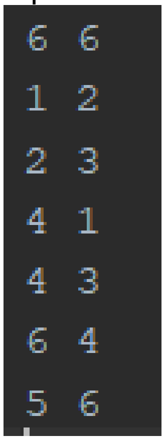
```
6 7
1 2
2 3
4 1
4 3
6 4
5 6
3 4
```

Output 2 :



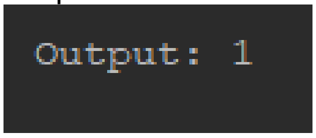
```
Output: 4
```

Input 3 :



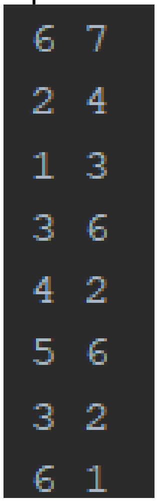
```
6 6
1 2
2 3
4 1
4 3
6 4
5 6
```

Output 3 :



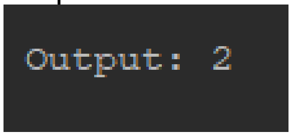
```
Output: 1
```

Input 4 :



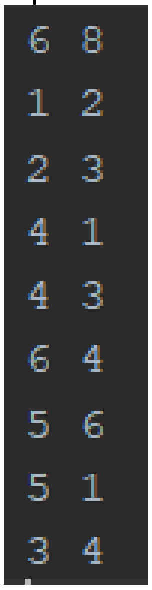
```
6 7
2 4
1 3
3 6
4 2
5 6
3 2
6 1
```

Output 4 :



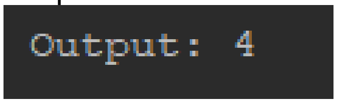
```
Output: 2
```

Input 5 :



```
6 8
1 2
2 3
4 1
4 3
6 4
5 6
5 1
3 4
```

Output 5 :



```
Output: 4
```