Gebze Technical University
Computer Engineering

CSE 222 - 2019 Spring

HOMEWORK 8 REPORT

HAMZA YOĞURTCUOĞLU
171044086

Course Assistant: Ayşe Şerbetçi TURAN

1 INTRODUCTION

1.1 Problem Definition

1) We have a group of people in which an ordered popularity relation is defined between person pairs.

2) If there exist a relation such that (Person-1,Person-2) this means that Person-1 thinks that Person-2 is popular. The relation is transitive which means that if the relations (Person-1,Person-2) and (Person-2,Person-3) exist, than (Person-1,Person-3) also exist event if it is not specified by the input pairs.

3) Writing a Java program which finds the people who are considered popular by every other person.

1.2 System Requirements

+ Total of Algoirthm Complexity is work with O(n) (Meaning is depend of your dara (White Point in Image)).

+ All Operating Systems handle this program.

+ Doesnt need a lot of memory acually if have alot of data , memory usage will raise linearly.

+ Pogram can work 128KB of memory (That can be change your data.)

+ This program doest make properly in smartphone it can be work in computer.

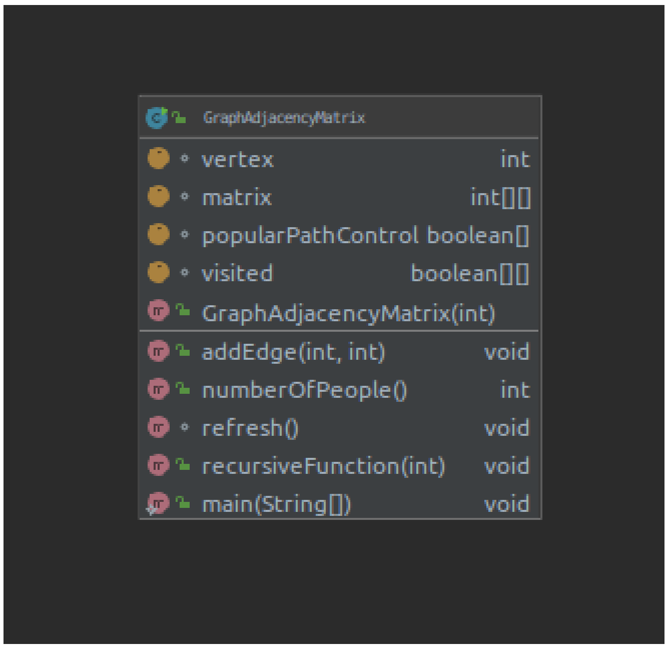
+ Doesnt need a specific of hardware just a computer work.

+ Just take executable file for directly execute program. Then , it works efficently.

2 METHOD

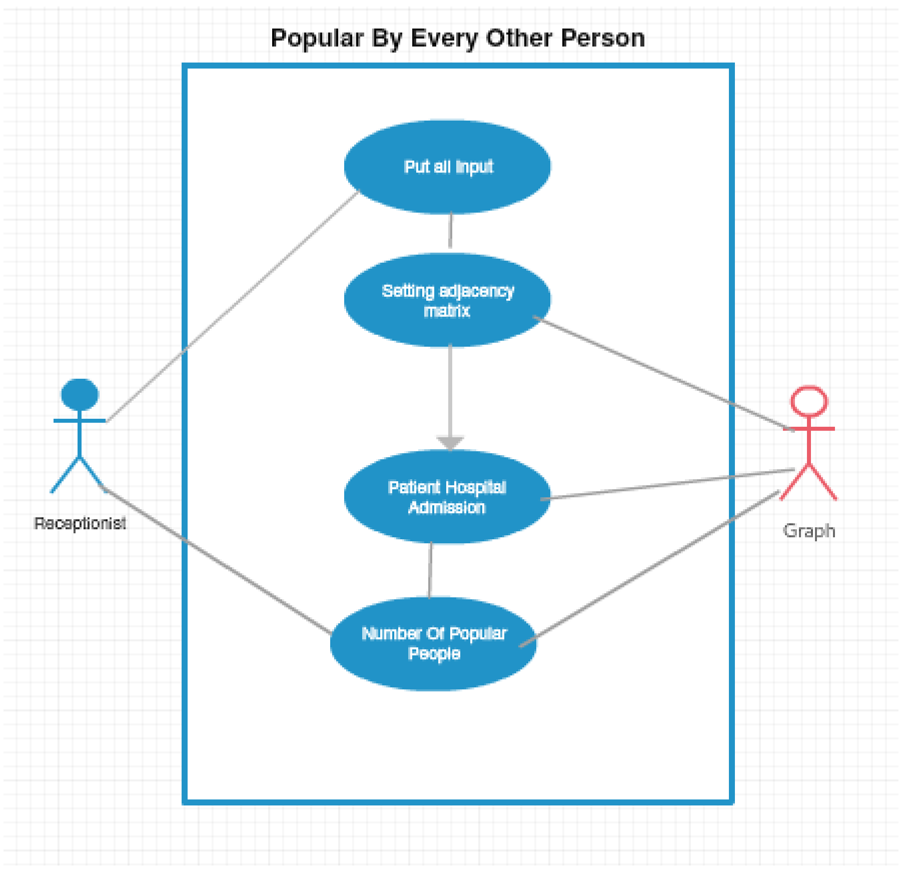
2.1 Class Diagrams

There is just a class. Im using adjacency matrix for Graph.



```
GraphAdjacencyMatrix
vertex                          int
matrix                        int[][]
popularPathControl boolean[]
visited                   boolean[][]
GraphAdjacencyMatrix(int)
addEdge(int, int)               void
numberOfPeople()                 int
refresh()                       void
recursiveFunction(int)          void
main(String[])                  void
```

2.2 Use Case Diagrams

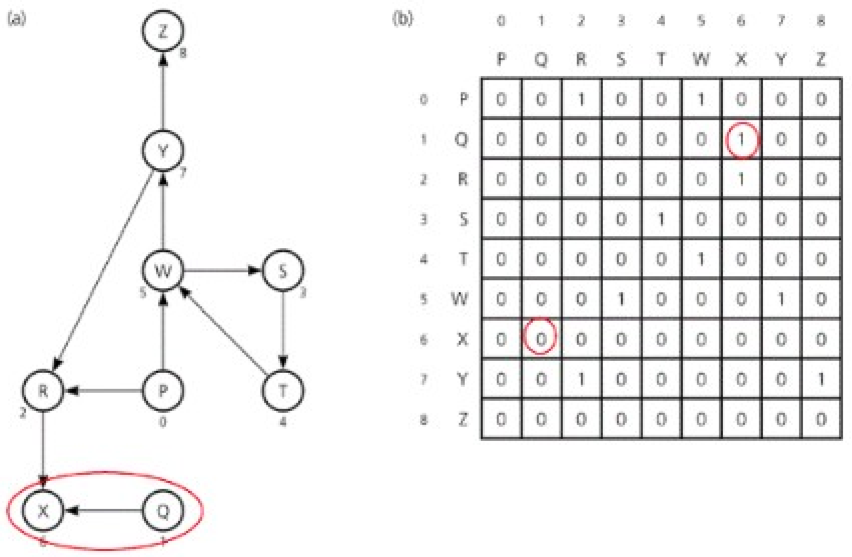


2.3 Problem Solution Approach

I decided Graph Data Structure. Using charts and graphs can help the audience grasp visually the message that has to be conveyed. Charts and graphs are especially useful if there will be a great deal of details that would normally take up too much time explaining and you need to compact the information to a visual summation. Knowing your audience helps creating the right type of charts and graphs to use. For example, if you are addressing a popular of person representative, you want to search for people charts and graphsthat typically a people will understand visually. When utilizing charts and graphs keepin mind that they are to be supplemental to using words. Charts and graphs can be very powerful in visual presentations if done effectively.

Vertex : means that number of person. We can say node number.

Matrix : It keeps all edge between person. According to binary representation.



		0	1	2	3	4	5	6	7	8
		P	Q	R	S	T	W	X	Y	Z
0	P	0	0	1	0	0	1	0	0	0
1	Q	0	0	0	0	0	0	1	0	0
2	R	0	0	0	0	0	0	1	0	0
3	S	0	0	0	0	1	0	0	0	0
4	T	0	0	0	0	0	1	0	0	0
5	W	0	0	0	1	0	0	0	1	0
6	X	0	0	0	0	0	0	0	0	0
7	Y	0	0	1	0	0	0	0	0	1
8	Z	0	0	0	0	0	0	0	0	0

a) A directed graph and b) its adjacency matrix

PopularPathControl:

I get a recursion situation in my algorithm , Recursion must not continue if the node got a recursion call.

Visited:

It is a matrix that keeps all visited node then sign each visited node cordinate. Like true , if not visited 'false'.

GraphAdjacencyMatrix --> Time Complexity - O(1) :

All Attributes are initialized.

- Visited - PopularPathControl - Matrix - Vertex

AddEdge --> Time Complexity - O(1) :

It takes 2 paramater for putting these values in adjacency matrix. Our problem needs a directed path ,so just a directory is put.

NumberOfPeople --> Time Complexity - O(Vertex^2) :

Edges are represented by a two dimensional array, called an adjacency matrix, with |i| rows and |j| columns.

For an adjacency matrix
Step 1 is O(|i|)
Step 2 is O(|j|)

The combination of Steps 1 and 2 represents examining each edge in the graph, giving O(| vertex^2 |) The adjacency list gives better performance in a sparse graph, whereas for a dense graph the performance is the same for both representations
Some graph algorithms are of the form:

```
1. for each vertex u in some subset of the vertices
2.     for each vertex v in some subset of the vertices
3.         if (u, v) is an edge
4.             Do something with edge (u, v)
```

The overall algorithm is O(|Vertex^2 |)

Refresh --> Time Complexity - O(Vertex^2) :

All visited and GraphAdjacencyMatrix are put false value again.

GraphAdjacencyMatrix --> Time Complexity - O(Vertex) :

Recursively choose an adjacent vertex that is not being visited.

Return from the recursion to popular node; all adjacent nodes to popular node are being visited.

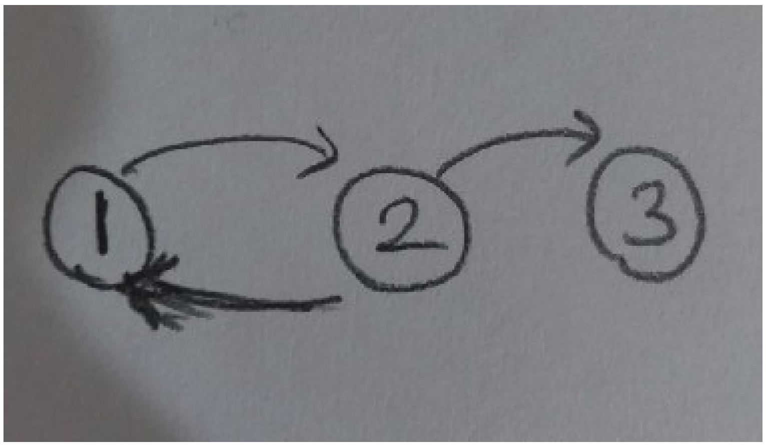
3 RESULT

3.1 Test Cases

3.2 Running Results

1) input.txt

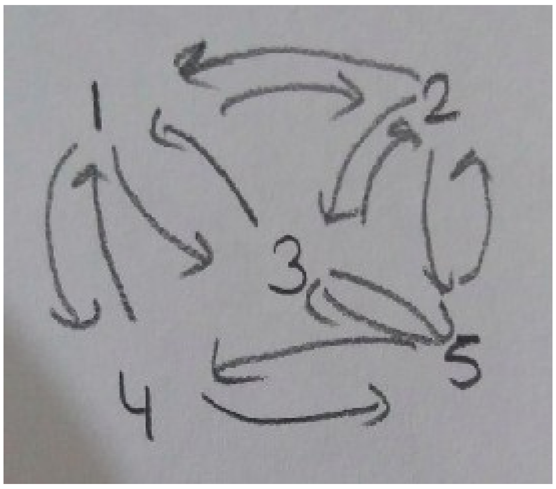
```
3 3
1 2
2 1
2 3
```



Answer : 1

2) input.txt

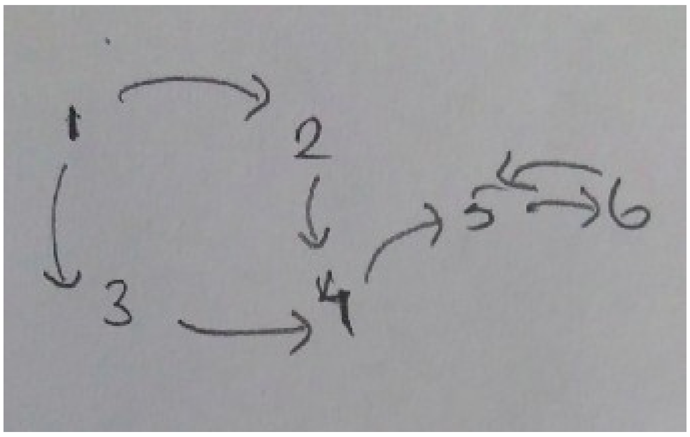
```
5 10
1 3
3 1
1 4
4 1
4 5
5 4
5 2
2 5
4 3
3 4
```



Answer : 5

3) input.txt

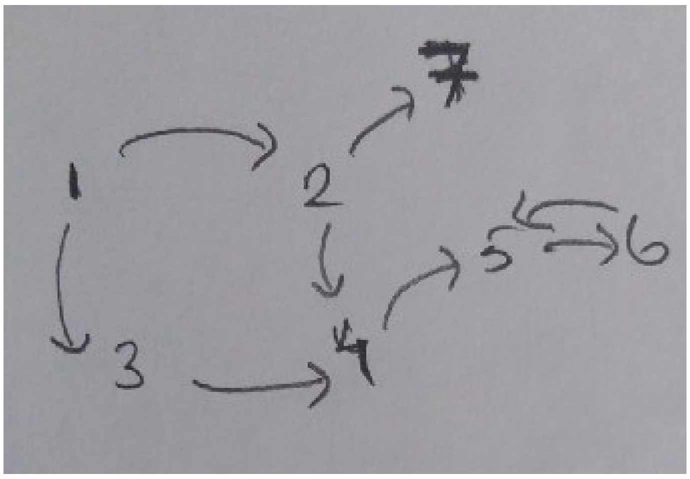
```
6 7
1 2
1 3
3 4
2 4
4 5
5 6
6 5
```



Answer : 2

4) input.txt

```
7 8
1 2
1 3
3 4
2 4
4 5
5 6
6 5
2 7
```



Answer : 0