Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 08 REPORT

AHMED SEMİH ÖZMEKİK
171044039

Course Assistant:

1 INTRODUCTION

1.1 Problem Definition

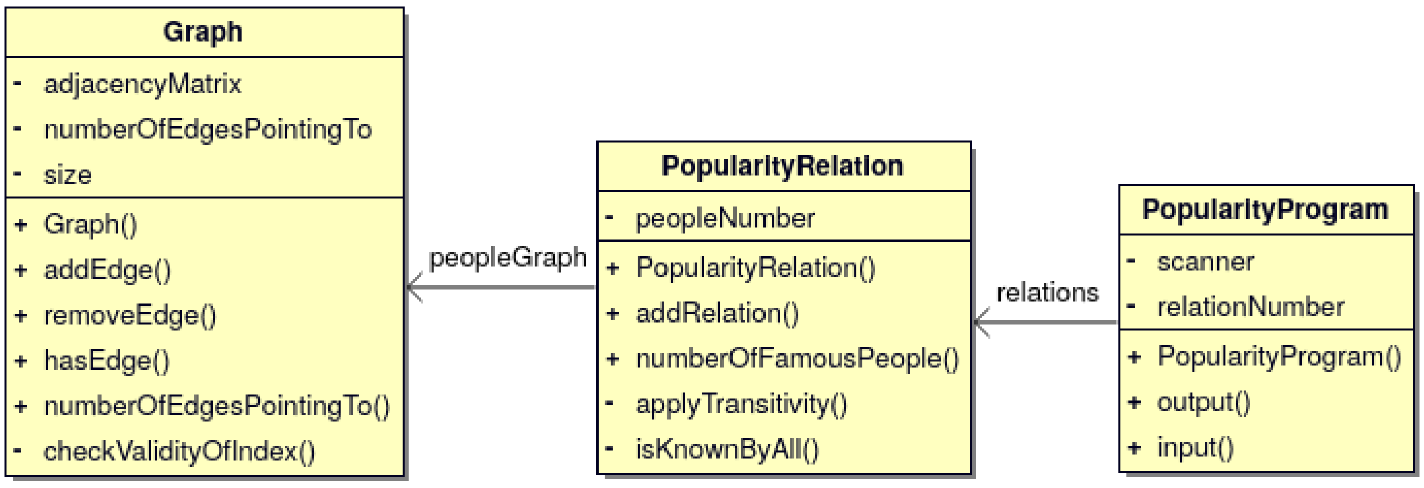
We have transitive relation defined between person pairs. Let’s explain this relation deeply. Say, we have three people in our space: Person A, Person B, Person C. We can define a relation in this space such as (A, B) or (A, C). And the meaning of this relation (A, B) is that, Person A thinks Person B is popular. Since this relation is transivite, if (A, B) and (B, C) is defined, then (A, C) exists. Meaning that, if Person A thinks Person B is popular and Person B thinks Person C is popular, hence Person A thinks Person C is popular. Regarding to the this defined relation and a given person space, our problem is to find the number of people who are considered popular by every other person.

1.2 System Requirements

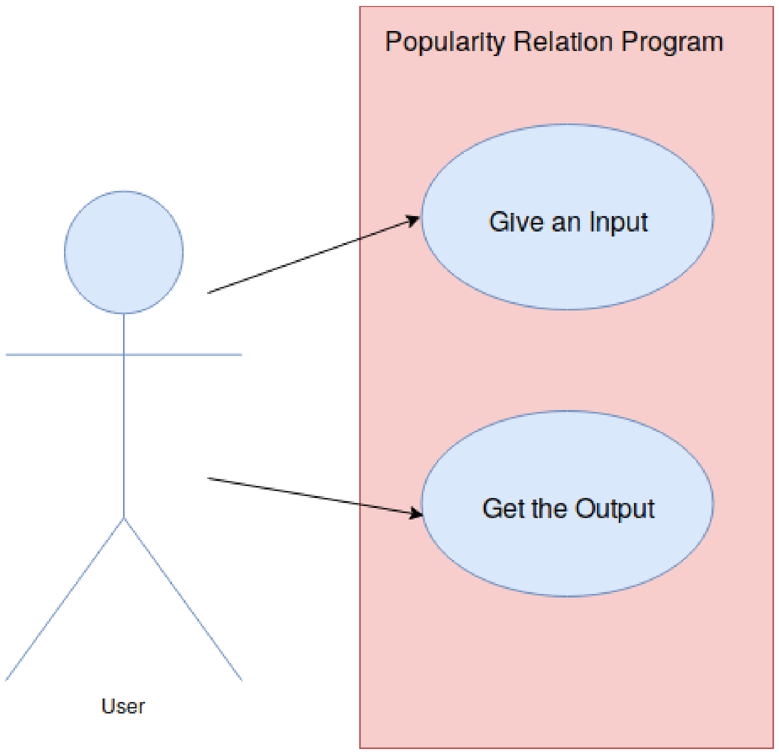
We require a well defined data structure to be able to represent this mentioned relation defined between person pairs. We must have an input file consisting the relations given by user. And further more, we need a robust algorithm to find people who are considered popular by every other person. But if we choose the data structure wisely, all of our other requirements will be satisfied properly. And even in the first look to our problem, we can intuitively say we need a Graph ADT. Of course, details of this choice will be mentioned in the problem solution approach. For to last, our system needs a simple input/output mechanicsm which will satisfy the flow of the program.

2 METHOD

2.1 Class Diagrams



2.2 Use Case Diagrams



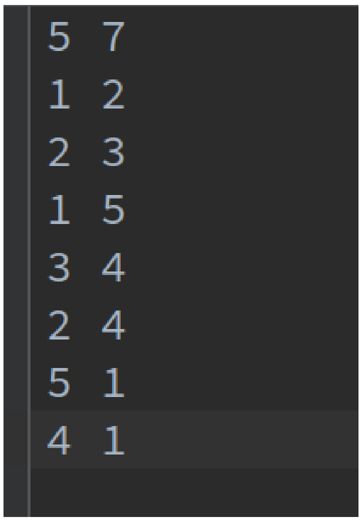
2.3 Problem Solution Approach

Regarding to our system’s requirements and problems, we picked Graph ADT as our underlying data structure for to use and solve our problems. Because the relations between person pairs was perfectly fitting the abstract idea of graphs, it was a well picked choice. We needed to pick an implementation for a Graph ADT. And since our program takes the number of people and number of relations from the user beforehand, adjacency matrix implementation was a better choice rather than adjacency list. Because we know how much space complexity we need in advance and furthermore, we can do lots of operations in constant time with adjacency matrix. So using this advantange, in our approach we concluded that adjacency matrix was a quite better choice. As we can see in our class diagram, we divided our approach into three parts which is kind of similar to the Model-View-Controller (MVC) approach. PopularityProgram only deals with the view of our program which is very basic and as a controller class, PopularityRelation acts a middle guy and handles the details of data structure and adapts the problem to that structure. Our model class which is the Graph implementation is not very specialized for our problem, so PopularityRelations as a role of adapter, specializes the Graph implementation to person pairs relation.

3 RESULT

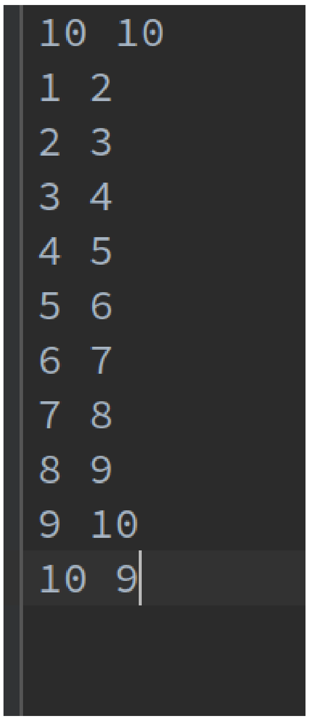
3.1 Test Cases

Case 1:

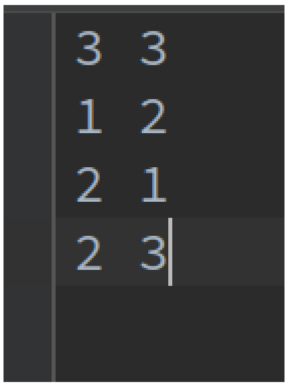


```
5 7
1 2
2 3
1 5
3 4
2 4
5 1
4 1
```

Case 2:

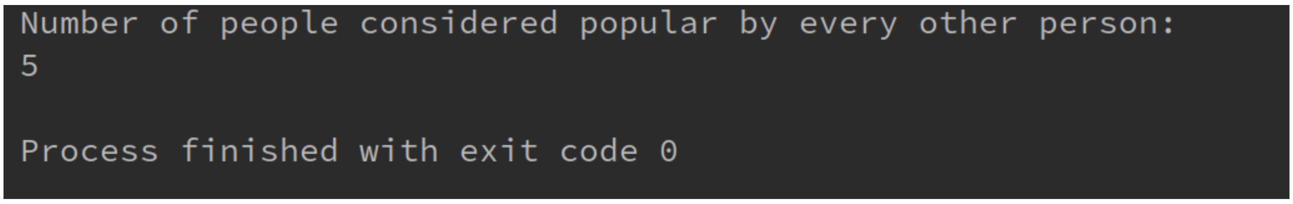

```
10 10
1 2
2 3
3 4
4 5
5 6
6 7
7 8
8 9
9 10
10 9
```

Case 3:


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

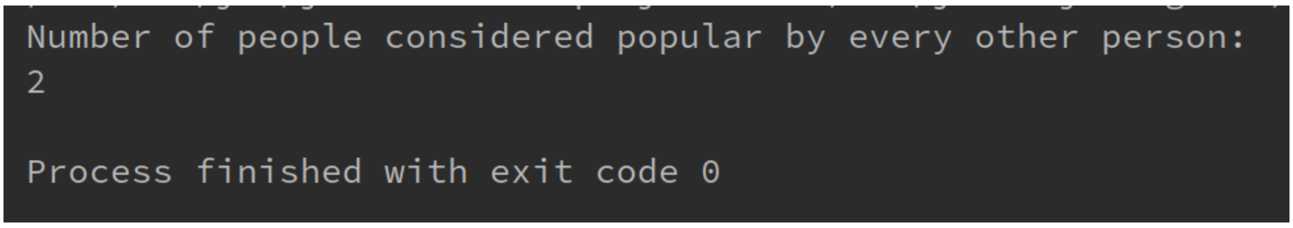
3.2 Running Results

Output Case 1:


```
Number of people considered popular by every other person:
5

Process finished with exit code 0
```

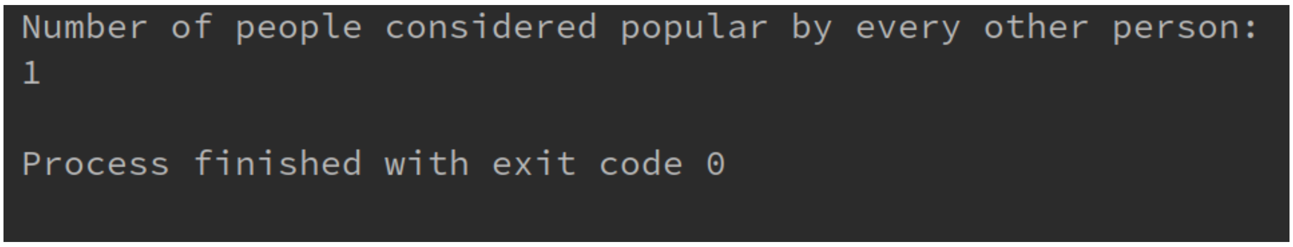
Output Case 2:



```
Number of people considered popular by every other person:
2

Process finished with exit code 0
```

Output Case 3:



```
Number of people considered popular by every other person:
1

Process finished with exit code 0
```