Gebze Technical University
Computer Engineering

CSE 222 - 2019 Spring

HOMEWORK 8 REPORT

CANER KARAKAŞ
131044061

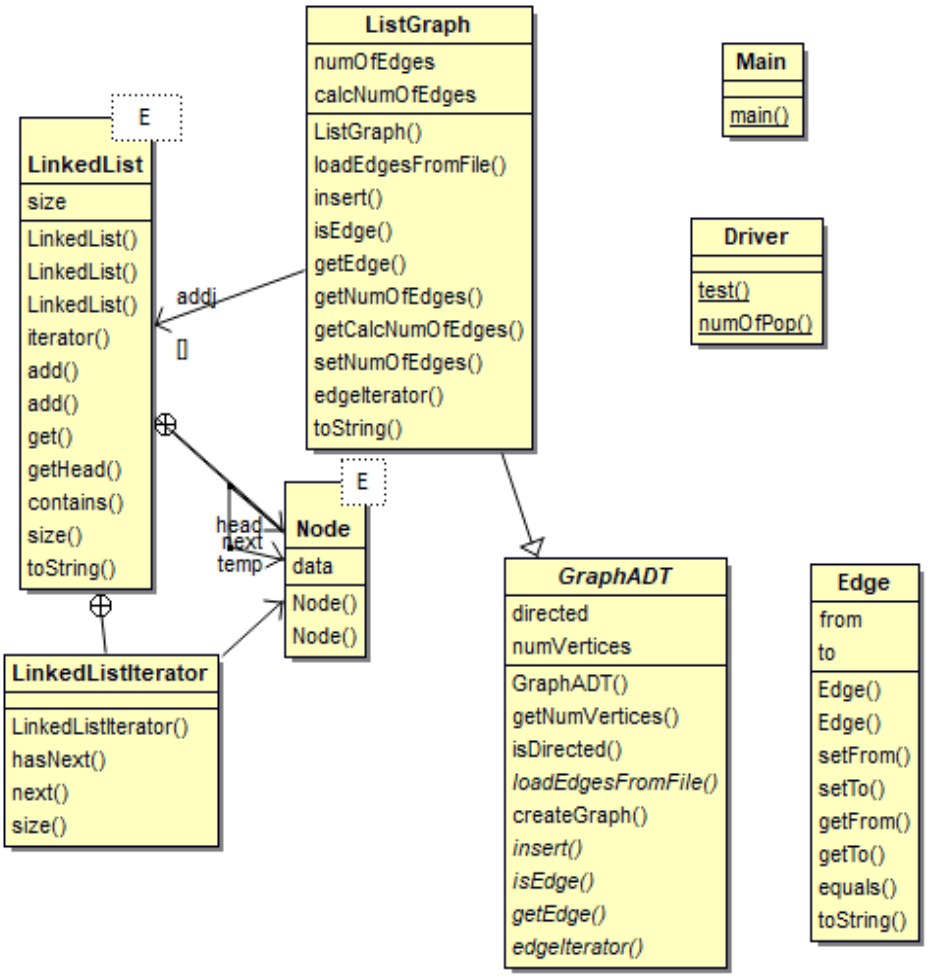
1 INTRODUCTION

1.1 Problem Definition

We have a group of people in which an ordered popularity relation is defined between person pairs. If there exist a relation such that (P1,P2) this means that A thinks that B is popular. The relation 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. You are supposed to write a Java program which finds the people who are considered popular by every other person.

2 METHOD

2.1 Class Diagrams



2.2 Problem Solution Approach

-GraphADT Class

This class has been implemented ADT. We can use adjacency matrix, adjacency list or any other method to implement our graph. Some methods are only implemented, such as loadEdgesFromFile, insert, isEdge,getEdge, edgeIterator.

-ListGraph Class

This class is extended from GraphADT class. It uses adjancecy list which is my LinkedList type. The extended methods are filled.

-numOfPop(ListGraph) Method

This method keeps the edges that are received from the ListGraph object in edges array. Then this method copies the edges in the edge array to my LinkedList object. While doing so, it adds indirectly to the routes that are going as in the tests. Then the search is made in the list starting from the last verticest. The number of elements that the searched element is considered. If the number of elements is one less than the given number of elements, the numOfPop number is increased.

3 RESULT

3.1 Test Cases

```
5 9
1 2
2 1
2 3
4 3
4 5
2 5
1 5
3 2
2 4
```

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

3.2 Running Results

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

1 2
1 1
1 3
1 5
1 4
1 5
2 1
2 2
2 5
2 3
2 2
2 5
2 4
2 3
2 5
3 2
3 1
3 3
3 5
3 4
4 3
4 2
4 5

1
```

```
1 2
2 1
2 3

1 2
1 1
1 3
2 1
2 2
2 3

1
```

TIME COMPLEXITY

Class	Method	Complexty
ListGraph	LoadEdgesFromFile()	O(n)
ListGraph	insert()	O(n)
ListGraph	isEdge()	O(n)
ListGraph	getEdge()	O(n)
ListGraph	getNumOfEdges()	O(1)
ListGraph	getCalcNumOfEdges()	O(1)

ListGraph	setNumOfEdges()	O(1)
ListGraph	edgeIterator()	O(1)
Driver	numOfPop()	O(n*n)