# GIT Department of Computer Engineering
# CSE 222/505 - Spring 2020
# Homework 8 Question 2 Report

**Melihcan Çilek**
**1801042092**

## 1. CLASS DIAGRAM

Class diagram is given named 1801042092_Class_Diagram.png because there is no enough space for class diagram

## 2. PROBLEM SOLUTION APPROACH

For this implementation, Extend the graph ADT defined in the book so that it includes the following operations.

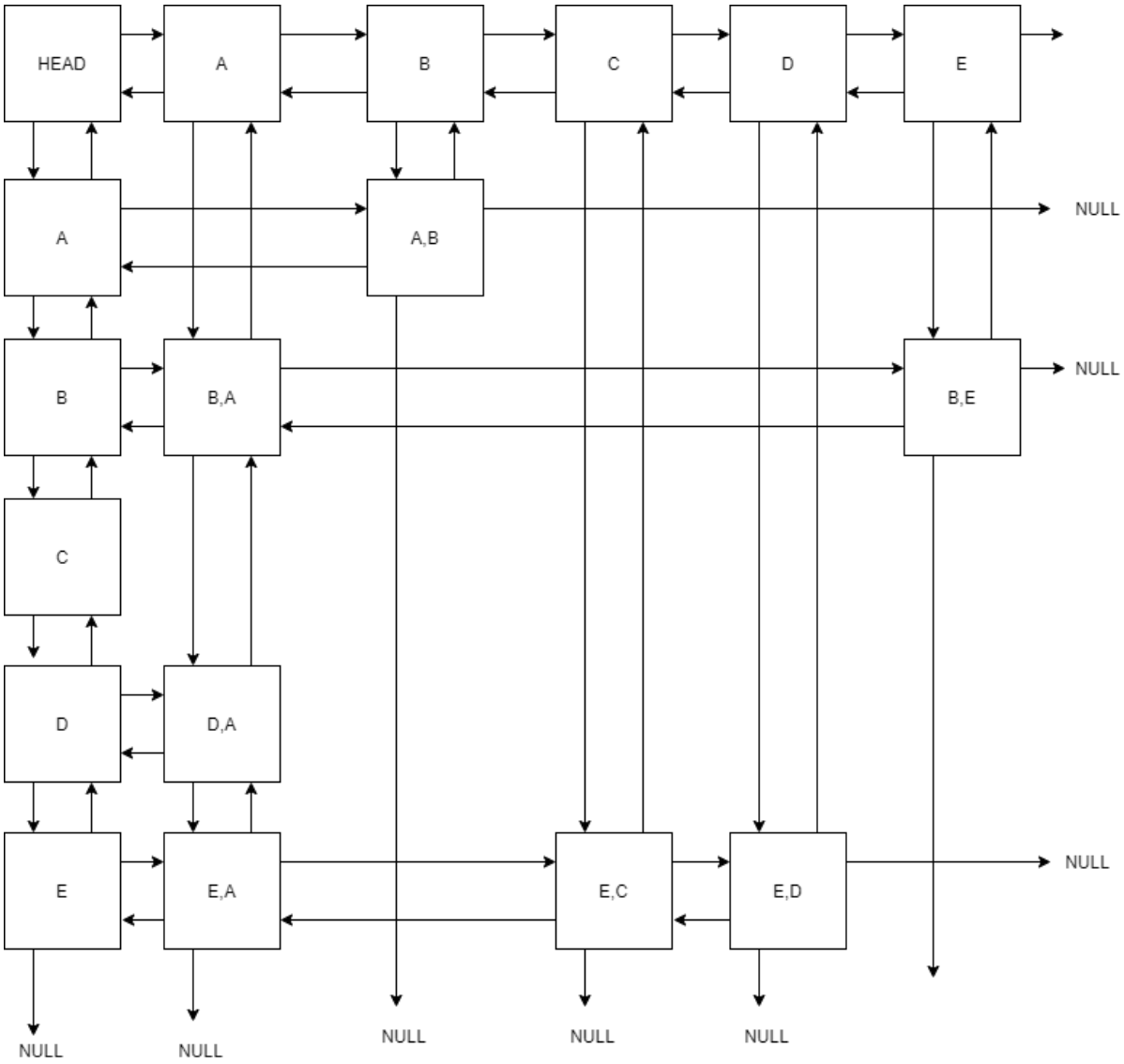
-Deletion of an individual edge.

-Insertion and deletion of an individual vertex. Give proper definition of the operations. Note that if you can delete a node, node IDs cannot be from 0…(n-1) anymore.

-Perform breadth-first search of the graph

-Perform depth-first search of the graph.

Implement the extended graph ADT using 2-D linked-list structure. In 2-D linked-list structure, each node has two row links (rprev to row-predecessor and rnext to row-successor) and two column links (cprev to column-predecessor and cnext to column-successor). In our graph representation, each row linked-list represents adjacent vertices to a vertex and each column linked-list represents vertices adjacent to a vertex. There are some problems and design choices while implementing this data structure. One of the design choice is implementing top and left nodes as Java Linkedlist data structure and implementing a Node class that has rprev, rnext, cprev and cnext Node class inside that Node class and controlling and managing nodes as while insertion and deletion edges or vertexes. Second design choice is implementing all of the nodes top and left as Node class that is defined private static class inside of ExtendedGraph class defined. What I choose as second representation because while implementing first representation, we have to use extra 2 class for top and left nodes and nodes that connects nodes between top and left linkedlist so that we have to use more memory. I did't choose that representation. At the picture below, my representation is more clear if I'm not being able to explain.




Right and bottom sides of head represents a linkedlist and there are connections between nodes

### 3. TEST CASES

| Test Case Number | Test Case | Test Data | Expected Result | Result | Decission |
|---|---|---|---|---|---|
| Test1 | Creating undirected, nonweighted graphs and insertion | 3,4 | 3,4 and 4,3 edges will be add as weight 1.0 | (3 , 4 , weight=1,0), (4 , 3 , weight=1,0) | PASS |

| | | | | | |
|---|---|---|---|---|---|
| Test2 | Creating directed, nonweighted graphs and insertion | 3,4 | 3,4 edge will be add as weight 1.0 | (3 , 4 , weight=1,0) | PASS |
| Test3 | Creating undirected, weighted graphs and insertion | 3,4,123.98 | 3,4 and 4,3 edge will be add as weight 123.98 | (3 , 4 , weight=124,0), (4 , 3 , weight=124,0) | PASS |
| Test4 | Creating directed, weighted graphs and insertion | 3,4,123.98 | 3,4 edge will be add as weight 123.98 | (3 , 4 , weight=123.98 | PASS |
| Test5 | Creating undirected, not weighted graphs and insertion with same nodes | 2,1 | 2,1 or 1,2 will not be add to graph | Not added | PASS |
| Test6 | Creating directed, not weighted graphs and insertion with same nodes | 2,1 | 2,1 will not be added to graph | (2 , 1 , weight=1,000000) Already Exists | PASS |
| Test7 | Creating undirected, weighted graphs and insertion with same nodes | 3 , 5, 23.4 | 3 , 5, 23.4 will not be added to graph | (3 , 5 , weight=23,400000) Already Exists | PASS |

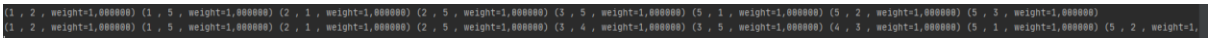
| | | | | | |
|---|---|---|---|---|---|
| Test8 | Creating directed, weighted graphs and insertion with same nodes | 3 , 5, 23.4 | 3 , 5, 23.4 and 5,4,23.4 will not be added to graph | (3 , 5 , weight=23,400000) Already Exists<br>(5 , 3 , weight=23,400000) Already Exists | PASS |
| Test9 | Creating undirected, not weighted graphs and deletion edge | 3 , 5 | 3,5 and 5,3 edges will be deleted | Deletion printed | PASS |
| Test10 | Creating directed, not weighted graphs and deletion edge | 3 , 5 | 3,5 edge will be deleted | Deletion printed | PASS |
| Test11 | Creating undirected, weighted graphs and deletion edge | 3 , 5 | 3,5 and 5,3 edges will be deleted | Deletion printed | PASS |
| Test12 | Creating directed, not weighted graphs and deletion edge | 3,5 | 3,5 edge will be deleted | Deletion printed | PASS |
| Test13 | Creating undirected, not weighted graphs and deletion edge that is | 1,1 | 1,1 will not be deleted | No edge 1 , 1 | PASS |

| | | | | | |
|---|---|---|---|---|---|
| | not in the graph | | | | |
| Test14 | Creating undirected, weighted graphs and deletion edge that is not in the graph | 1,1 | 1,1 will not be deleted | No edge 1 , 1 | PASS |
| Test15 | Individual vertex insertion | 8 | Vertex with 8 will be inserted to ExtendedGraph | 5<br>6 | PASS |
| Test16 | Individual vertex insertion and creating edge | 8<br>8,2 | Vertex with 8 will be inserted to ExtendedGraph and we can be able to create edge with vertex 8 | (8 , 2 , weight=1,000000) | PASS |
| Test17 | Individual vertex deletion and with that, all edges connected to that vertex will be deleted | 3 | (3 , 5 , weight=1,000000) will be deleted and number of vertex's will be decrease | Number of vertices = 4 | PASS |
| Test18 | Breath First Search with starting node 2 | 2 | Parent Array will be displayed | [-1, 2, -1, -1, -1, 2, -1] | PASS |
| Test19 | Depth First Search with starting node 2 | 2 | Parent Array will be displayed | [-1, 2, -1, -1, -1, 2, -1] | PASS |

| | | | | | |
|---|---|---|---|---|---|
| Test20 | Iterator | Printing all edges with same source | All edges will be displayed | (2 , 1 , weight=1,000000)<br>(2 , 3 , weight=1,000000)<br>(2 , 5 , weight=1,000000) | PASS |

## 4. RUNNING AND RESULTS

Test1:




Test2:

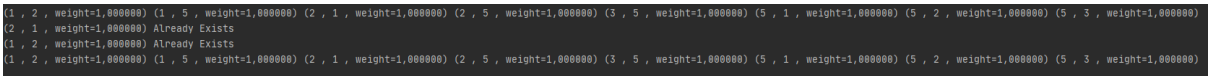
```
(2 , 1 , weight=1,000000) (2 , 5 , weight=1,000000) (3 , 5 , weight=1,000000) (5 , 1 , weight=1,000000)
(2 , 1 , weight=1,000000) (2 , 5 , weight=1,000000) (3 , 4 , weight=1,000000) (3 , 5 , weight=1,000000) (5 , 1 , weight=1,000000)
```

Test3:


Test4:

```
 (2 , 1 , weight=2,500000) (2 , 5 , weight=5,400000) (3 , 5 , weight=23,400000) (5 , 1 , weight=4,560000)
 (2 , 1 , weight=2,500000) (2 , 5 , weight=5,400000) (3 , 4 , weight=123,980000) (3 , 5 , weight=23,400000) (5 , 1 , weight=4,560000)
```

Test5:




Test6:

```
(2 , 1 , weight=1,000000) (2 , 5 , weight=1,000000) (3 , 5 , weight=1,000000) (5 , 1 , weight=1,000000)
(2 , 1 , weight=1,000000) Already Exists
(2 , 1 , weight=1,000000) (2 , 5 , weight=1,000000) (3 , 5 , weight=1,000000) (5 , 1 , weight=1,000000)
```

Test7:

```
(2 , 1 , weight=2,500000) (2 , 5 , weight=5,400000) (3 , 5 , weight=23,400000) (5 , 1 , weight=4,560000)
(3 , 5 , weight=23,400000) Already Exists
(2 , 1 , weight=2,500000) (2 , 5 , weight=5,400000) (3 , 5 , weight=23,400000) (5 , 1 , weight=4,560000)
```

Test8:

```
(1 , 2 , weight=2,500000) (1 , 5 , weight=4,560000) (2 , 1 , weight=2,500000) (2 , 5 , weight=5,400000) (3 , 5 , weight=23,400000) (5 , 1 , weight=4,560000) (5 , 2 , weight=5,400000) (5 , 3 , weight=23,400000)
(3 , 5 , weight=23,400000) Already Exists
(5 , 3 , weight=23,400000) Already Exists
(1 , 2 , weight=2,500000) (1 , 5 , weight=4,560000) (2 , 1 , weight=2,500000) (2 , 5 , weight=5,400000) (3 , 5 , weight=23,400000) (5 , 1 , weight=4,560000) (5 , 2 , weight=5,400000) (5 , 3 , weight=23,400000)
```

Test9:

```
(1 , 2 , weight=1,000000) (1 , 5 , weight=1,000000) (2 , 1 , weight=1,000000) (2 , 5 , weight=1,000000) (3 , 5 , weight=1,000000) (5 , 1 , weight=1,000000) (5 , 2 , weight=1,000000) (5 , 3 , weight=1,000000)
(1 , 2 , weight=1,000000) (1 , 5 , weight=1,000000) (2 , 1 , weight=1,000000) (2 , 5 , weight=1,000000) (5 , 1 , weight=1,000000) (5 , 2 , weight=1,000000)
```

Test10:

```
(2 , 1 , weight=1,000000) (2 , 5 , weight=1,000000) (3 , 5 , weight=1,000000) (5 , 1 , weight=1,000000)
(2 , 1 , weight=1,000000) (2 , 5 , weight=1,000000) (5 , 1 , weight=1,000000)
```

Test11:

```
(1 , 2 , weight=2,500000) (1 , 5 , weight=4,560000) (2 , 1 , weight=2,500000) (2 , 5 , weight=5,400000) (3 , 5 , weight=23,400000) (5 , 1 , weight=4,560000) (5 , 2 , weight=5,400000) (5 , 3 , weight=23,400000)
(1 , 2 , weight=2,500000) (1 , 5 , weight=4,560000) (2 , 1 , weight=2,500000) (2 , 5 , weight=5,400000) (5 , 1 , weight=4,560000) (5 , 2 , weight=5,400000)
```

Test12:

```
(2 , 1 , weight=1,000000) (2 , 5 , weight=1,000000) (3 , 5 , weight=1,000000) (5 , 1 , weight=1,000000)
(2 , 1 , weight=1,000000) (2 , 5 , weight=1,000000) (5 , 1 , weight=1,000000)
```

Test13:

```
(1 , 2 , weight=1,000000) (1 , 5 , weight=1,000000) (2 , 1 , weight=1,000000) (2 , 5 , weight=1,000000) (3 , 5 , weight=1,000000) (5 , 1 , weight=1,000000) (5 , 2 , weight=1,000000) (5 , 3 , weight=1,000000)
No edge 1 , 1
(1 , 2 , weight=1,000000) (1 , 5 , weight=1,000000) (2 , 1 , weight=1,000000) (2 , 5 , weight=1,000000) (3 , 5 , weight=1,000000) (5 , 1 , weight=1,000000) (5 , 2 , weight=1,000000) (5 , 3 , weight=1,000000)
```

Test14:

```
(1 , 2 , weight=2,500000) (1 , 5 , weight=4,560000) (2 , 1 , weight=2,500000) (2 , 5 , weight=5,400000) (3 , 5 , weight=23,400000) (5 , 1 , weight=4,560000) (5 , 2 , weight=5,400000) (5 , 3 , weight=23,400000)
No edge 1 , 1
(1 , 2 , weight=2,500000) (1 , 5 , weight=4,560000) (2 , 1 , weight=2,500000) (2 , 5 , weight=5,400000) (3 , 5 , weight=23,400000) (5 , 1 , weight=4,560000) (5 , 2 , weight=5,400000) (5 , 3 , weight=23,400000)
```

Test15:

```
5
6
```

Test16:

```
Number of vertices = 5
(2 , 1 , weight=1,000000) (2 , 5 , weight=1,000000) (3 , 5 , weight=1,000000) (5 , 1 , weight=1,000000)
No vertex with 8
Number of vertices = 5
(2 , 1 , weight=1,000000) (2 , 5 , weight=1,000000) (3 , 5 , weight=1,000000) (5 , 1 , weight=1,000000)
Number of vertices = 6
(2 , 1 , weight=1,000000) (2 , 5 , weight=1,000000) (3 , 5 , weight=1,000000) (5 , 1 , weight=1,000000) (8 , 2 , weight=1,000000)
```

Test17:

```
Number of vertices = 5
(2 , 1 , weight=1,000000) (2 , 5 , weight=1,000000) (3 , 5 , weight=1,000000) (5 , 1 , weight=1,000000)
Number of vertices = 4
(2 , 1 , weight=1,000000) (2 , 5 , weight=1,000000) (5 , 1 , weight=1,000000)
```

Test18:

```
[-1, 2, -1, -1, -1, 2, -1]
```

Test19:

```
[2, 5, 1, 0, 0]
[0, 0, 1, 5, 2]
```

Test20:

```
(2 , 1 , weight=1,000000)
(2 , 3 , weight=1,000000)
(2 , 5 , weight=1,000000)
```