CSE 222

HOMEWORK 8

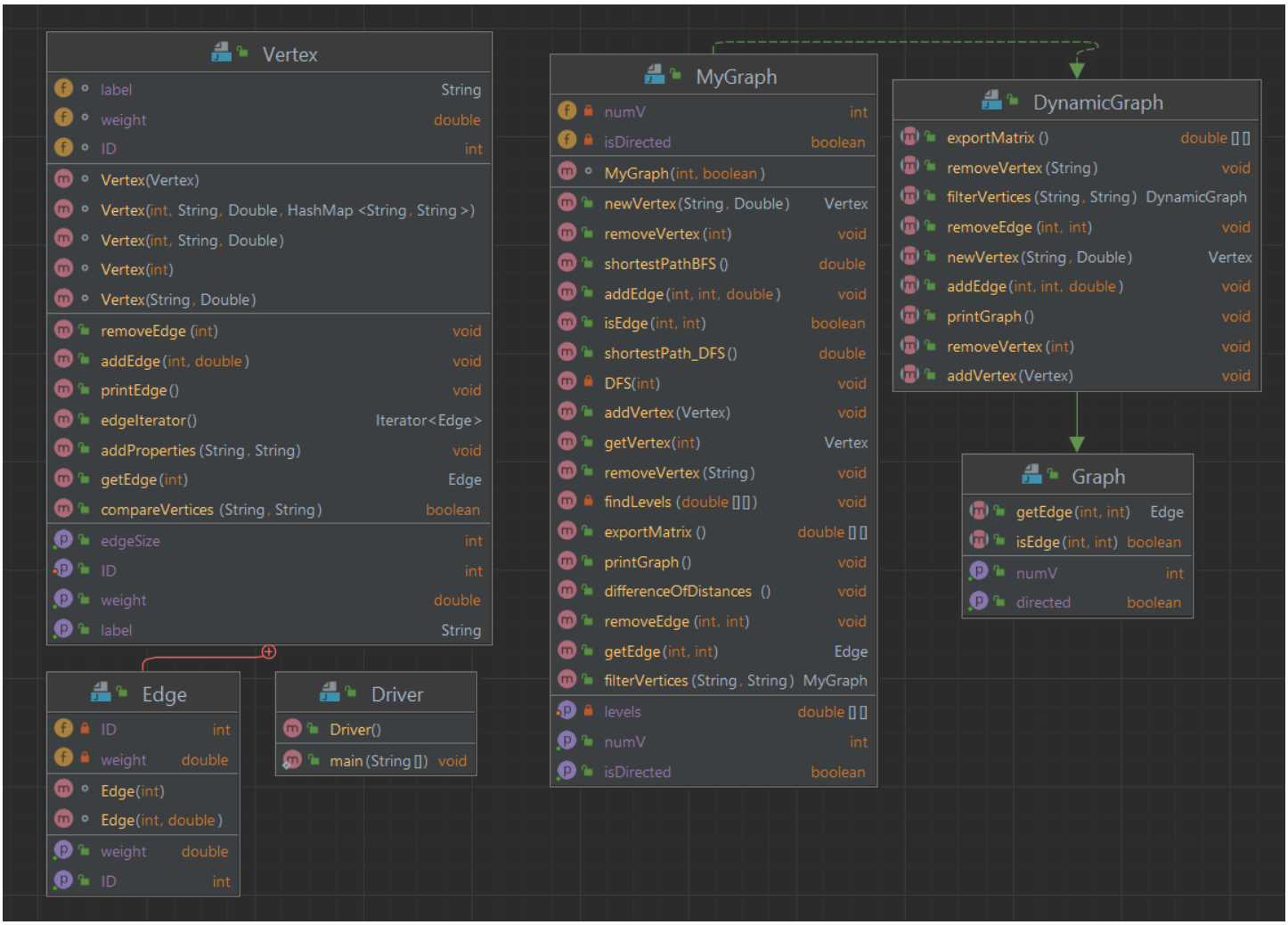
ERSEL CELAL EREN

1901042673

1-SYSTEM REQUIREMENTS

In this task, we are asked to modify the ListGraph class, breadth-first search and depth-first search methods. Normally when ListGraph has only edge class, in Q1 part, we implemented additional methods and Vertex class that has weight value. In the end, new graph class still has Adjacency list structure. For Q2 part we modified the BFS and DFS as returning distance of shortest path.

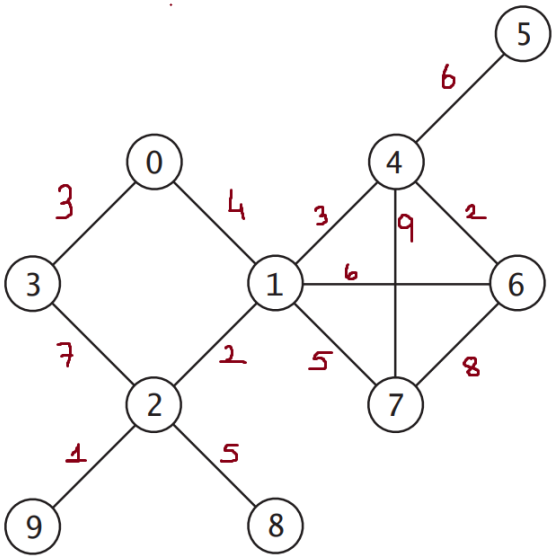
2-CLASS DIAGRAMS



3-PROBLEM SOLUTION APPROACH

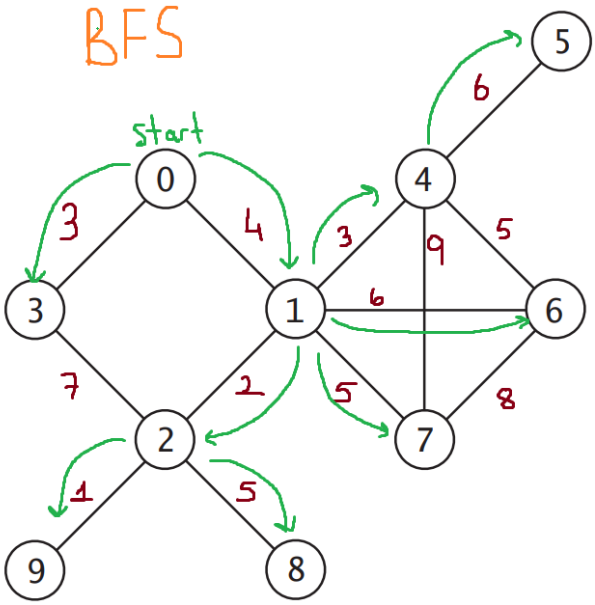
Q1) In this part I used Arraylist of Vertex class to store vertices. Also, Vertex instances have LinkedList of Edge class to represent edges. In this way, I preserved Adjacency list structure. I have used .get(index) method for vertices and list iterator for edges to access them. In remove method, I have used dummy reference and checked dummy reference in addVertex methods. Also, Vertex class has HashMap field for used defined properties.

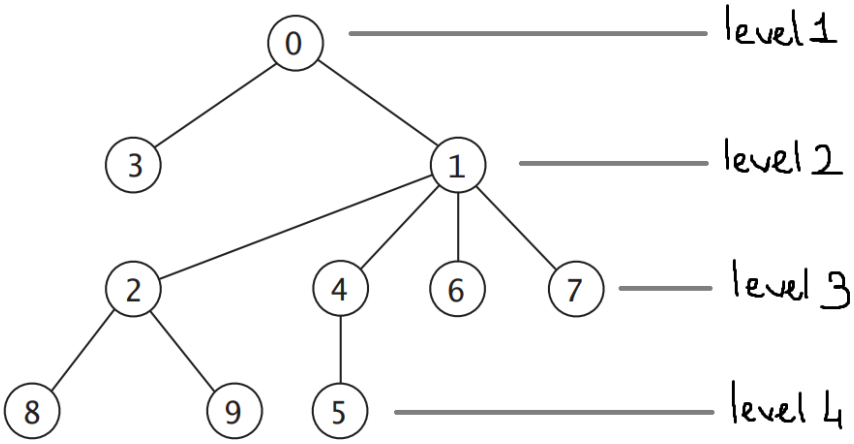
Q2) For second part of task, I have modified BFS and DFS methods that returns shortest path distance without violating their general algorithm.



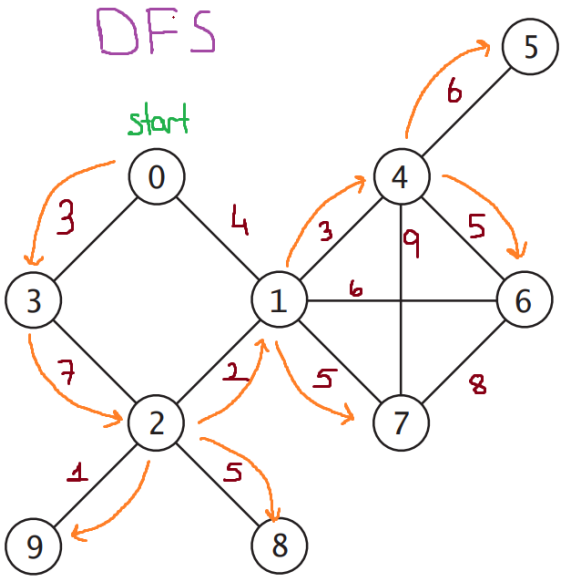
This graph is taken from our textbook. I gave weight to all edges randomly and calculated shortest distance. When BFS's distance is 35, DFS's distance is 37. These green and orange arrows are the path that algorithm follows. So, we can prove result when we sum weight of that edges.

For BFS algorithm, I used parent array of default BFS method and created a level array. Level array shows level of each vertex when we convert it to binary tree. In BFS method, all vertices will be in same level for every possible path.

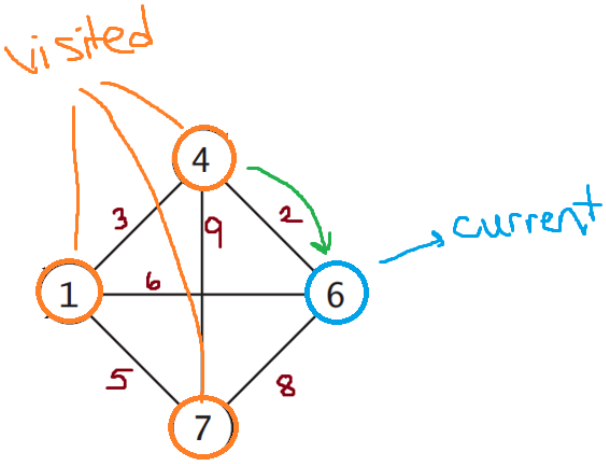




This is the binary tree of graph and levels of vertices.



For DFS algorithm, we don't have any level information. Levels are changing according to different paths. So I made a different algorithm. When we visit a vertex, if there is no unvisited vertex adjacent to our current vertex, we choose edge with the smallest weight. In this example, when it comes to vertex:6, it will choose edge with vertex:4.



4-TEST CASES

Creating undirected graph and adding vertex and edges

```
MyGraph graph1 = new MyGraph(6, false);

graph1.addVertex(new Vertex("label0", 0.0));
graph1.addVertex(new Vertex("label1", 1.0));
graph1.addVertex(new Vertex("label2", 2.0));
graph1.addVertex(new Vertex("label3", 3.0));
graph1.addVertex(new Vertex("label4", 4.0));
graph1.addVertex(new Vertex("label5", 5.0));
graph1.addVertex(new Vertex("label6", 6.0));
graph1.addVertex(new Vertex("label7", 7.0));
graph1.addVertex(new Vertex("label8", 8.0));
graph1.addVertex(new Vertex("label9", 9.0));

graph1.addEdge( 0, 3, 3);
graph1.addEdge( 0, 1, 4 );
graph1.addEdge( 3, 2, 7);
graph1.addEdge( 1, 2, 2);
graph1.addEdge( 2, 9, 1);
graph1.addEdge( 2, 8, 5);
graph1.addEdge( 1, 4, 3);
graph1.addEdge( 1, 6, 6);
graph1.addEdge( 1, 7, 5);
graph1.addEdge( 4, 7, 9);
graph1.addEdge( 4, 6, 5);
graph1.addEdge( 4, 5, 6);
graph1.addEdge( 6, 7, 8);
```

Creating same graph as directed

```
MyGraph graph2 = new MyGraph(6, true);

graph2.addVertex(new Vertex("label0", 0.0));
graph2.addVertex(new Vertex("label1", 1.0));
graph2.addVertex(new Vertex("label2", 2.0));
graph2.addVertex(new Vertex("label3", 3.0));
graph2.addVertex(new Vertex("label4", 4.0));
graph2.addVertex(new Vertex("label5", 5.0));
graph2.addVertex(new Vertex("label6", 6.0));
graph2.addVertex(new Vertex("label7", 7.0));
graph2.addVertex(new Vertex("label8", 8.0));
graph2.addVertex(new Vertex("label9", 9.0));
```

```
        graph2.addEdge(0 , 3 , 3);
        graph2.addEdge( 0, 1, 4 );
        graph2.addEdge( 3, 2, 7 );
        graph2.addEdge( 1, 2, 2);
        graph2.addEdge( 2 , 9, 1);
        graph2.addEdge( 2, 8, 5 );
        graph2.addEdge( 1 , 4, 3);
        graph2.addEdge( 1 , 6 , 6);
        graph2.addEdge( 1 , 7 , 5);
        graph2.addEdge( 4 , 7 , 9);
        graph2.addEdge( 4 , 6 , 2);
        graph2.addEdge( 4 , 5 , 6 );
        graph2.addEdge( 6 , 7 , 8);
```

Removing edges from graph2 with source and destination

```
        graph2.removeEdge(2, 9);
        graph2.removeEdge(0, 3);
```

Removing vertices from graph2 with ID and label

```
        graph2.removeVertex(9);
        graph2.removeVertex("label3");
```

isEdge method test

```
        if(graph2.isEdge(6, 7)) System.out.printf("\n6 to 7 IS EDGE");
        else System.out.printf("\n6 to 7 NOT EDGE");
        if(graph2.isEdge(8, 2)) System.out.printf("\n8 to 2 IS EDGE");
        else System.out.printf("\n8 to 2 NOT EDGE");
```

exportMatrix method test

```
        double[][] array = graph1.exportMatrix();
```

```
System.out.printf("\n ---> BFS and DFS Calculations <---\n");
        graph1.differenceOfDistances();
```

5-RESULTS

```
--->Creating undirected graph and adding vertex and edges<---

 || ID:0 - Weight:0.0 || =======<EDGES>    <|| ID :> 3  - Weight:3.0 ||>  <|| ID :> 1  - Weight:4.0 ||>
 || ID:1 - Weight:1.0 || =======<EDGES>    <|| ID :> 0  - Weight:4.0 ||>  <|| ID :> 2  - Weight:2.0 ||>  <|| ID :> 4  - Weight:3.0 ||>  <|| ID :> 6  - Weight:6.0 ||>  <|| ID :> 7  - Weight:5.0 ||>
 || ID:2 - Weight:2.0 || =======<EDGES>    <|| ID :> 3  - Weight:7.0 ||>  <|| ID :> 1  - Weight:2.0 ||>  <|| ID :> 9  - Weight:1.0 ||>  <|| ID :> 8  - Weight:5.0 ||>
 || ID:3 - Weight:3.0 || =======<EDGES>    <|| ID :> 0  - Weight:3.0 ||>  <|| ID :> 2  - Weight:7.0 ||>
 || ID:4 - Weight:4.0 || =======<EDGES>    <|| ID :> 1  - Weight:3.0 ||>  <|| ID :> 7  - Weight:9.0 ||>  <|| ID :> 6  - Weight:5.0 ||>  <|| ID :> 5  - Weight:6.0 ||>
 || ID:5 - Weight:5.0 || =======<EDGES>    <|| ID :> 4  - Weight:6.0 ||>
 || ID:6 - Weight:6.0 || =======<EDGES>    <|| ID :> 1  - Weight:6.0 ||>  <|| ID :> 4  - Weight:5.0 ||>  <|| ID :> 7  - Weight:8.0 ||>
 || ID:7 - Weight:7.0 || =======<EDGES>    <|| ID :> 1  - Weight:5.0 ||>  <|| ID :> 4  - Weight:9.0 ||>  <|| ID :> 6  - Weight:8.0 ||>
 || ID:8 - Weight:8.0 || =======<EDGES>    <|| ID :> 2  - Weight:5.0 ||>
 || ID:9 - Weight:9.0 || =======<EDGES>    <|| ID :> 2  - Weight:1.0 ||>
-------------------------------------------------------------------------------------------------------------

--->Creating same graph as directed <---

 || ID:0 - Weight:0.0 || =======<EDGES>    <|| ID :> 3  - Weight:3.0 ||>  <|| ID :> 1  - Weight:4.0 ||>
 || ID:1 - Weight:1.0 || =======<EDGES>    <|| ID :> 2  - Weight:2.0 ||>  <|| ID :> 4  - Weight:3.0 ||>  <|| ID :> 6  - Weight:6.0 ||>  <|| ID :> 7  - Weight:5.0 ||>
 || ID:2 - Weight:2.0 || =======<EDGES>    <|| ID :> 9  - Weight:1.0 ||>  <|| ID :> 8  - Weight:5.0 ||>
 || ID:3 - Weight:3.0 || =======<EDGES>    <|| ID :> 2  - Weight:7.0 ||>
 || ID:4 - Weight:4.0 || =======<EDGES>    <|| ID :> 7  - Weight:9.0 ||>  <|| ID :> 6  - Weight:2.0 ||>  <|| ID :> 5  - Weight:6.0 ||>
 || ID:5 - Weight:5.0 || =======<EDGES>
 || ID:6 - Weight:6.0 || =======<EDGES>    <|| ID :> 7  - Weight:8.0 ||>
 || ID:7 - Weight:7.0 || =======<EDGES>
 || ID:8 - Weight:8.0 || =======<EDGES>
 || ID:9 - Weight:9.0 || =======<EDGES>
-------------------------------------------------------------------------------------------------------------

---> Removing edges from graph2 with source and dest <---

 || ID:0 - Weight:0.0 || =======<EDGES>    <|| ID :> 1  - Weight:4.0 ||>
 || ID:1 - Weight:1.0 || =======<EDGES>    <|| ID :> 2  - Weight:2.0 ||>  <|| ID :> 4  - Weight:3.0 ||>  <|| ID :> 6  - Weight:6.0 ||>  <|| ID :> 7  - Weight:5.0 ||>
 || ID:2 - Weight:2.0 || =======<EDGES>    <|| ID :> 8  - Weight:5.0 ||>
 || ID:3 - Weight:3.0 || =======<EDGES>    <|| ID :> 2  - Weight:7.0 ||>
 || ID:4 - Weight:4.0 || =======<EDGES>    <|| ID :> 7  - Weight:9.0 ||>  <|| ID :> 6  - Weight:2.0 ||>  <|| ID :> 5  - Weight:6.0 ||>
 || ID:5 - Weight:5.0 || =======<EDGES>
 || ID:6 - Weight:6.0 || =======<EDGES>    <|| ID :> 7  - Weight:8.0 ||>
 || ID:7 - Weight:7.0 || =======<EDGES>
 || ID:8 - Weight:8.0 || =======<EDGES>
 || ID:9 - Weight:9.0 || =======<EDGES>
```

```
---> Removing vertices from graph2 with ID and label <---

 || ID:0 - Weight:0.0 || =======<EDGES>    <|| ID :> 1  - Weight:4.0 ||>
 || ID:1 - Weight:1.0 || =======<EDGES>    <|| ID :> 2  - Weight:2.0 ||>  <|| ID :> 4  - Weight:3.0 ||>  <|| ID :> 6  - Weight:6.0 ||>  <|| ID :> 7  - Weight:5.0 ||>
 || ID:2 - Weight:2.0 || =======<EDGES>    <|| ID :> 8  - Weight:5.0 ||>
 || ID:4 - Weight:4.0 || =======<EDGES>    <|| ID :> 7  - Weight:9.0 ||>  <|| ID :> 6  - Weight:2.0 ||>  <|| ID :> 5  - Weight:6.0 ||>
 || ID:5 - Weight:5.0 || =======<EDGES>
 || ID:6 - Weight:6.0 || =======<EDGES>    <|| ID :> 7  - Weight:8.0 ||>
 || ID:7 - Weight:7.0 || =======<EDGES>
 || ID:8 - Weight:8.0 || =======<EDGES>
-------------------------------------------------------------------------------------------------------------

---> Adding vertex and edge into graph2 after removing edges and vertices <---

 || ID:0 - Weight:0.0 || =======<EDGES>    <|| ID :> 1  - Weight:4.0 ||>
 || ID:1 - Weight:1.0 || =======<EDGES>    <|| ID :> 2  - Weight:2.0 ||>  <|| ID :> 4  - Weight:3.0 ||>  <|| ID :> 6  - Weight:6.0 ||>  <|| ID :> 7  - Weight:5.0 ||>
 || ID:2 - Weight:2.0 || =======<EDGES>    <|| ID :> 8  - Weight:5.0 ||>
 || ID:3 - Weight:3.2 || =======<EDGES>    <|| ID :> 8  - Weight:3.8 ||>
 || ID:4 - Weight:4.0 || =======<EDGES>    <|| ID :> 7  - Weight:9.0 ||>  <|| ID :> 6  - Weight:2.0 ||>  <|| ID :> 5  - Weight:6.0 ||>
 || ID:5 - Weight:5.0 || =======<EDGES>
 || ID:6 - Weight:6.0 || =======<EDGES>    <|| ID :> 7  - Weight:8.0 ||>
 || ID:7 - Weight:7.0 || =======<EDGES>
 || ID:8 - Weight:8.0 || =======<EDGES>
```

```
6 to 7 IS EDGE
8 to 2 NOT EDGE
5 to 1 NOT EDGE
8 to 11 NOT EDGE
23 to 0 NOT EDGE
1 to 4 IS EDGE
```

```
---> Exporting Weight Matrix of Graph1 <---
    0  1  2  3  4  5  6  7  8  9
0 | -  4  -  3  -  -  -  -  -  -
1 | 4  -  2  -  3  -  6  5  -  -
2 | -  2  -  7  -  -  -  -  5  1
3 | 3  -  7  -  -  -  -  -  -  -
4 | -  3  -  -  -  6  5  9  -  -
5 | -  -  -  -  6  -  -  -  -  -
6 | -  6  -  -  5  -  -  8  -  -
7 | -  5  -  -  9  -  8  -  -  -
8 | -  -  5  -  -  -  -  -  -  -
9 | -  -  1  -  -  -  -  -  -  -
```

```
---> BFS and DFS Calculations <---


Distance of BFS : 35.0
Distance of DFS : 37.0
Difference : -2.0
```

6-COMPLEXITY ANALYSIS

```
public Vertex newVertex(String label, Double weight) {
    return new Vertex(numV,label,weight);
}
```

Constructor is constant time.

>>> Theta(1)

```
public void addVertex(Vertex new_vertex) {
    //checks dummy reference first
    int i;
    if(dummyCount>0){
        for(i=0;i<vertices.size();i++){
            if(vertices.get(i)==dummy) break;
        }
        vertices.set(i, new_vertex);
        dummyCount--;
        size++;
    }
    else{
        vertices.add(new_vertex);
        vertices.get(vertices.size()-1).setID(vertices.size()-1);
    }

}
```

.get(i) .set(…) .add(…) .setID(…) *methods are theta(1)*

addVertex *best case : theta(1)*

addVertex *worst case : theta(n)*

>>> O(n)

```
public void addEdge(int vertexID1, int vertexID2, double weight) {
    vertices.get(vertexID1).addEdge(vertexID2, weight);
    if(!isDirected())
        vertices.get(vertexID2).addEdge(vertexID1, weight);
}
```

isDirected() *and* ***addEdge(…)*** *methods are constant time*

>>> Theta(1)

```java
public void removeEdge(int vertexID1, int vertexID2){
    if(vertexID1 >= 0 && vertexID1 < vertices.size() && vertexID2 >= 0 && vertexID2 < vertices.size())
        vertices.get(vertexID1).removeEdge(vertexID2);

    if(isDirected() == false)
    vertices.get(vertexID2).removeEdge(vertexID1);
}
```

removeEdge(vertexID2) *is O(n) (linkedlist remove)*

>>> O(n)

```java
public void removeVertex(int vertexID) {
    if(vertexID >= 0 && vertexID < numV){
        for(int i=0;i<numV;i++){
            vertices.get(i).removeEdge(vertexID);
        }
    }

    vertices.set(vertexID, dummy);
    dummyCount++;
}
```

removeEdge(vertexID) *is O(n) (arraylist remove)*

>>> O(n)

```java
public void removeVertex(String label) {
    int index;
    for(index=0;index<vertices.size();index++)
        if(vertices.get(index).getLabel().compareTo(label) == 0)
            break;

    removeVertex(index);
}
```

For loop iterates ***n*** *times,* ***removeVertex(index)*** *is O(n)*

>>> O(n^2)

```java
public void printGraph() {
    for(int i=0;i<vertices.size();i++){
        if(vertices.get(i) != dummy){
            System.out.printf("\n || ID:%d - Weight:%.1f || ",i, vertices.get(i).getWeight());
            vertices.get(i).printEdge();
        }

    }
}
```

>>> O(n^2)