# GTU Department of Computer Engineering
# CSE 222/505 - SPRING 2022
# HOMEWORK 8 REPORT

**BURAK ÇİÇEK**
**1901042260**

# System Requirements

- **In MyGraph Class**

`public double adjacencyMatrix[][] = new double[99][99];` **Adjacency Matrix representation of MyGraph Class**

`LinkedList<Vertex> adjacencyList[];` **Adjacency List representation of MyGraph Class**

`int numVertices = 0;` **Vertex number counter.**

`int edgeSize = 0;` **Edge number counter**

`int IDofVertex = 0;` **IDs of vertices starts from 0 and increments when new vertex added.**

`public MyGraph(int node)` **Constructor of MyGraph Class node is size of linkedList Array (Adj List.)**

`public Vertex newVertex(String label, double weight)` **newVertex Method like defined in Homework PDF.**

`public void addVertex(Vertex new_Vertex)` **addVertex defined in Homework PDF and DynamicGraph interface**

`public void addEdge(int vertexID1, int vertexID2, double weight)` **addEdge Method defined in Homework PDF and DynamicGraph interface.**

`public void removeEdge(int vertexID1, int vertexID2)` **removeEdge method defined in Homework PDF and DynamicGraph interface.**

`public void removeVertex(int vertexID)` **removeVertex method defined in Homework PDF and DynamicGraph interface.**

`public void removeVertex(String label)` **removeVertex method defined in Homework PDF and DynamicGraph interface.**

`public MyGraph filterVertices(String key, String filter)` **filterVertices method defined in Homework PDF and DynamicGraph interface.**

`public double[][] exportMatrix()` **exportMatrix method defined in Homework PDF and DynamicGraph interface.**

`public void printGraph()` **printGraph method defined in Homework PDF and DynamicGraph interface.**

---

- ## In Vertex Class

`int index;` **ID for vertices.**

`private String label;` **Label for vertices.**

`private double weight;` **Weight for vertices.**

`HashMap<String, String> properties = new HashMap<~>();` **Hashmap for properties**

`private double boosting = 0;` **Boosting value for vertices. Default value is 0**

---

- ## In DijsktrasAlgortihm Class

```
public static void dijkstrasAlgorithm(MyGraph<Integer> graph,
                                       int start,
                                       int[] pred,
                                       double[] dist) {
```

**Dijsktra's Algorithm method optimized for Q3.**

---

- ## In BreadthFirstSearch Class

`public static void swap(int i1 ,int i2 , List<Edge> list)` **Swap method for index1 and index2 in Edge.**

`public static int[] breadthFirstSearch(MyGraph<Integer> graph, int start)`

**Breadth First Search algorithm.**

---

- ## In DepthFirstSearch Class

`public DepthFirstSearch(MyGraph<Integer> graph)` **Constructor**

`public int[] depthFirstSearch(int current)`

**Depth First Search Algorithm Method.**

# Problem Solution Approach

**For Question 1,**

Since the graph structure is a data structure that I have just learned, I can say that I lost more time than I thought while writing the methods. Since you said that Edge Class is not needed, I had to think about where to keep the weights of the edges, and then I decided to store the weights there since we also keep the edges in the adjacencyMatrix structure. As an extra solution, this value could also be kept in LinkedList nodes in the adjacencyList, but I chose to do it this way because there was no restriction about it :)

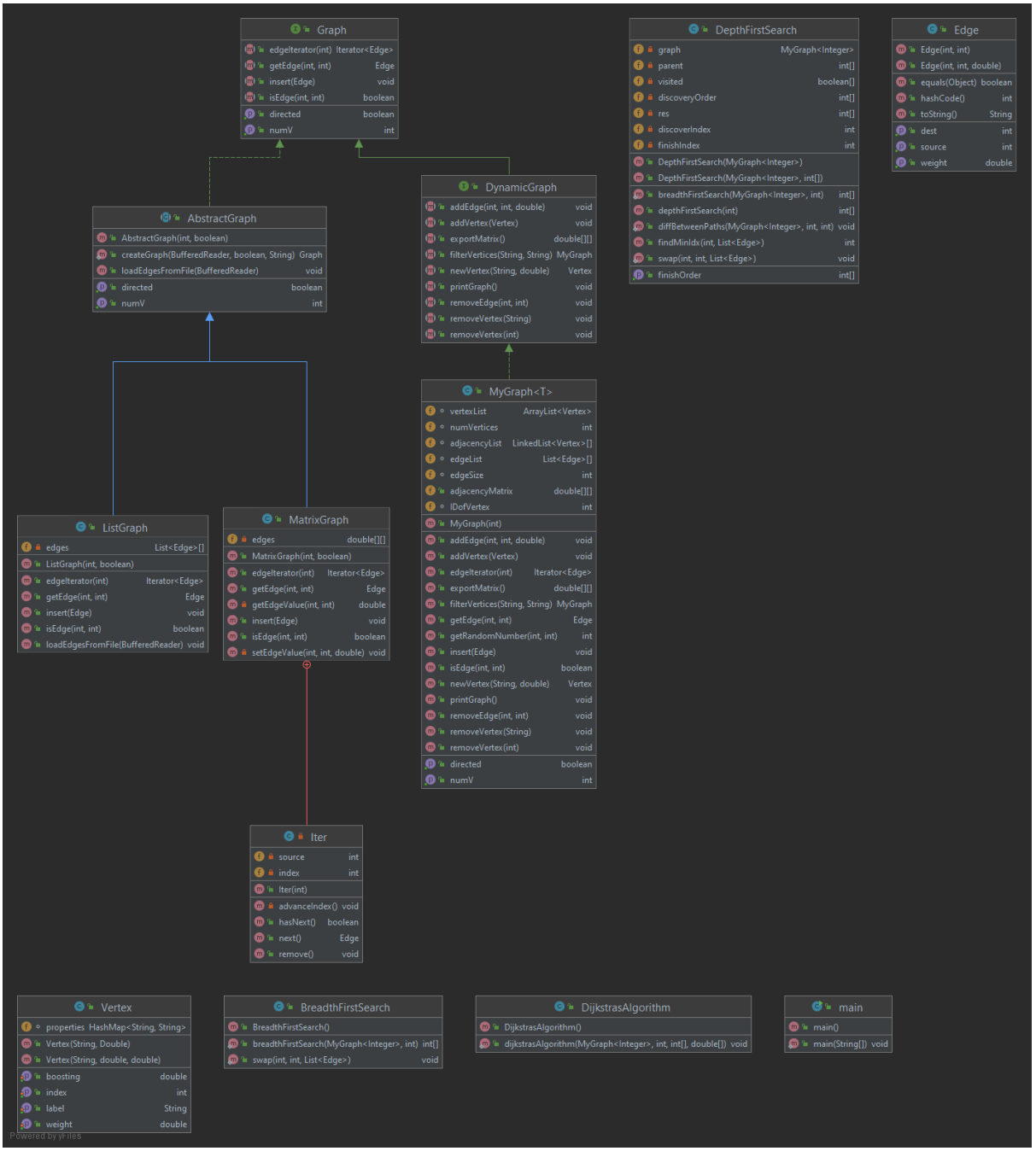
**For Question 2,**

For Question 2, the most difficult part in DFS and BFS algorithms was to implement the shortest path. In this regard, I can say that I had some difficulty in optimizing MyGraph to these algorithms, and even lost a lot of time. But I finally found a way to think about the right approach and answer Question 2 correctly.

**For Question 3,**

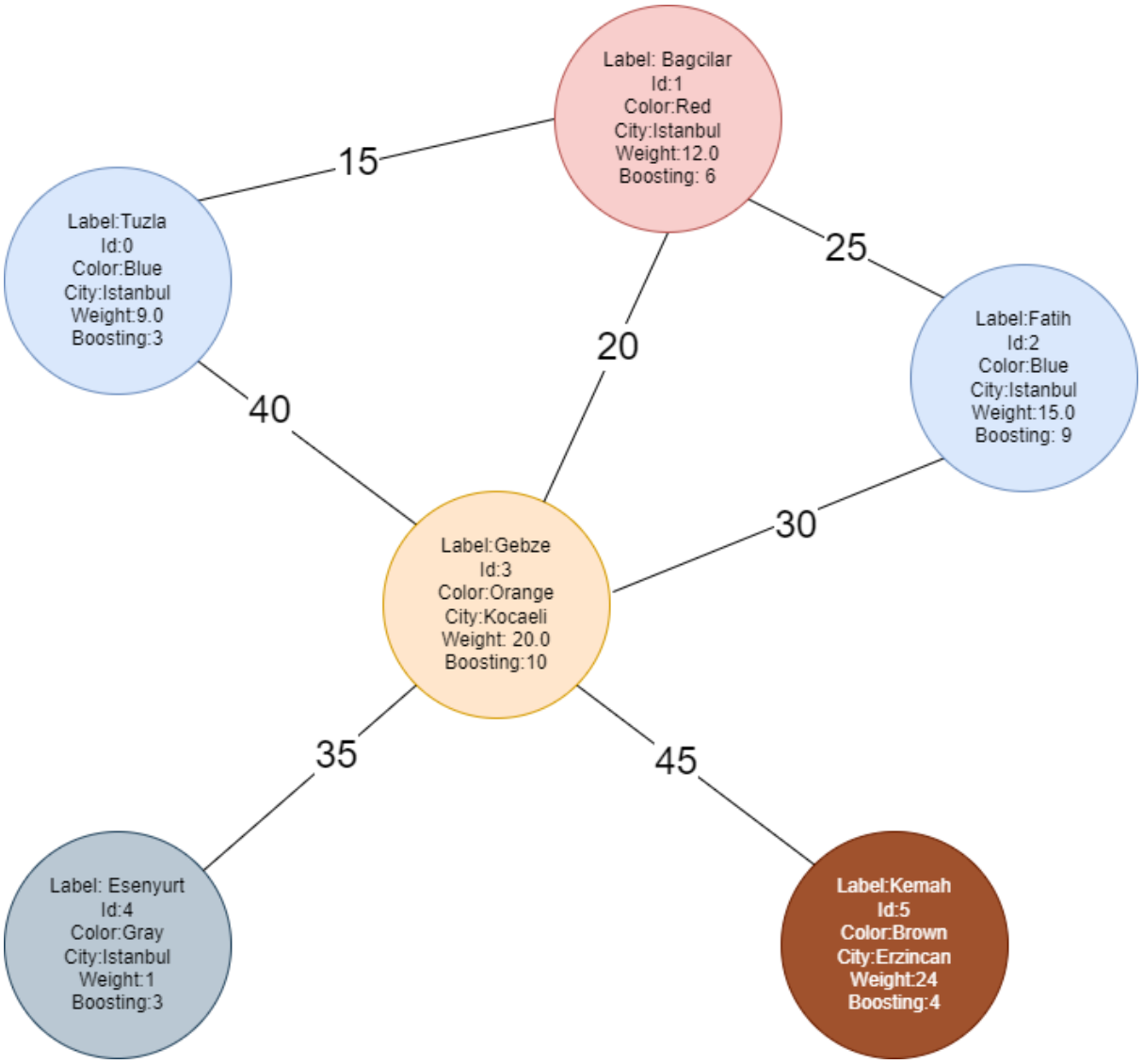
Since it doesn't have an Edge Class, I adapted the parts about the edge class to my own MyGraph class, first of all I did some brainstorming about where to compare the boosting value. I can say that I lost some time in this part, but by debugging the code slowly, I was able to find it right. For this algorithm, I used the DijkstrasAlgorithm class and its implementation in the book.

# Class Diagrams



# Test Cases

**I assume a graph represents below:**




**Let's define our graph in our Java Code**

**First Create Vertices**

```
MyGraph<Integer> myTestGraph = new MyGraph<>( node: 6);
Vertex Tuzla = myTestGraph.newVertex( label: "Tuzla",  weight: 9.0);
Vertex Bagcilar = myTestGraph.newVertex( label: "Bagcilar",  weight: 12.0);
Vertex Fatih = myTestGraph.newVertex( label: "Fatih",  weight: 15.0);
Vertex Gebze = myTestGraph.newVertex( label: "Gebze",  weight: 20);
Vertex Esenyurt = myTestGraph.newVertex( label: "Esenyurt",  weight: 1);
Vertex Kemah= myTestGraph.newVertex( label: "Kemah",  weight: 24);
```

**And properties as well:**

```
Tuzla.properties.put("Color","Blue");
Tuzla.properties.put("City","Istanbul");
Bagcilar.properties.put("Color","Red");
Bagcilar.properties.put("City","Istanbul");
Fatih.properties.put("Color","Blue");
Fatih.properties.put("City","Istanbul");
Gebze.properties.put("Color","Orange");
Gebze.properties.put("City","Kocaeli");
Esenyurt.properties.put("Color","Gray");
Esenyurt.properties.put("City","Istanbul");
Kemah.properties.put("Color","Brown");
Kemah.properties.put("City","Erzincan");
```

**Of course, Boosting values:**

```
Tuzla.setBoosting(3.0);
Bagcilar.setBoosting(6.0);
Fatih.setBoosting(9.0);
Gebze.setBoosting(10.0);
Esenyurt.setBoosting(1.0);
Kemah.setBoosting(4.0);
```

**Than, add it this vertices to our graph:**

```
myTestGraph.addVertex(Tuzla);
myTestGraph.addVertex(Bagcilar);
myTestGraph.addVertex(Fatih);
myTestGraph.addVertex(Gebze);
myTestGraph.addVertex(Esenyurt);
myTestGraph.addVertex(Kemah);
```

**After, create edges:**

```
myTestGraph.addEdge( vertexID1: 0, vertexID2: 1, weight: 15);//Tuzla - Bagcilar - 15
myTestGraph.addEdge( vertexID1: 0, vertexID2: 3, weight: 40);//Tuzla - Gebze - 40
myTestGraph.addEdge( vertexID1: 1, vertexID2: 3, weight: 20);//Bagcilar - Gebze - 20
myTestGraph.addEdge( vertexID1: 1, vertexID2: 2, weight: 25);//Bagcilar - Fatih - 25
myTestGraph.addEdge( vertexID1: 2, vertexID2: 3, weight: 30);//Fatih - Gebze - 30
myTestGraph.addEdge( vertexID1: 3, vertexID2: 4, weight: 35);//Gebze - Esenyurt - 35
myTestGraph.addEdge( vertexID1: 3, vertexID2: 5, weight: 45);//Gebze - Kemah - 45
```

**Lets print our graph!**

```
myTestGraph.printGraph();
```

**Output:**

```
[Vertex0] --> [Vertex1|15.0] --> [Vertex3|40.0]
[Vertex1] --> [Vertex0|15.0] --> [Vertex2|25.0] --> [Vertex3|20.0]
[Vertex2] --> [Vertex1|25.0] --> [Vertex3|30.0]
[Vertex3] --> [Vertex0|40.0] --> [Vertex1|20.0] --> [Vertex2|30.0] --> [Vertex4|35.0] --> [Vertex5|45.0]
[Vertex4] --> [Vertex3|35.0]
[Vertex5] --> [Vertex3|45.0]
```

**TEST SUCCESSFUL**

**I want to test other methods before remove methods because it will be easy to examine.**

---

**Filter City:Istanbul and Print Graph:**

```
MyGraph<Integer> myIstanbulGraph = myTestGraph.filterVertices( key: "City", filter: "Istanbul");
myIstanbulGraph.printGraph();
```

```
[Vertex0] --> [Vertex1|25.0] --> [Vertex2|30.0] --> [Vertex3|19.0]
[Vertex1] --> [Vertex0|25.0] --> [Vertex2|32.0] --> [Vertex3|46.0]
[Vertex2] --> [Vertex0|30.0] --> [Vertex1|32.0] --> [Vertex3|6.0]
[Vertex3] --> [Vertex0|19.0] --> [Vertex1|46.0] --> [Vertex2|6.0]
```

**TEST SUCCESSFUL**

**A little info: After applying the filter, I connect the edges of the returning subgraph with a random value between 0 and 50 and print it on the screen.**

```
filterVerticesGraph.addEdge(filterVerticesGraph.vertexList.get(i).index, filterVerticesGraph.vertexList.get(j).index, this.getRandomNumber(0,50));
```

---

**Let's export our matrix and print it to screen:**

```
double myExportMatrix [][] = myTestGraph.exportMatrix();

for(int i=0; i<myTestGraph.numVertices;i++){
    System.out.print(i + " ");
    for (int j=0; j < myTestGraph.numVertices;j++){
        System.out.print(myExportMatrix[i][j] + " ");
    }
    System.out.println();
}
```

```
0 0.0 15.0 0.0 40.0 0.0 0.0
1 15.0 0.0 25.0 20.0 0.0 0.0
2 0.0 25.0 0.0 30.0 0.0 0.0
3 40.0 20.0 30.0 0.0 35.0 45.0
4 0.0 0.0 0.0 35.0 0.0 0.0
5 0.0 0.0 0.0 45.0 0.0 0.0
```

**TEST SUCCESSFUL**

---

**Try our optimized Dijkstra Algorithm to our Graph Example and select start position to Gebze:**

```
DijkstrasAlgorithm testDijkstrasAlgorithm = new DijkstrasAlgorithm();
int predArray[] = new int[99];
double distArray[] = new double[99];
testDijkstrasAlgorithm.dijkstrasAlgorithm(myTestGraph, start: 3,predArray,distArray);
System.out.println("Gebze to Istanbul Optimized Dijkstra'a Algortihm Value: " + distArray[0]);
System.out.println("Gebze to Bagciler Optimized Dijkstra'a Algortihm Value: " +distArray[1]);
System.out.println("Gebze to Fatih Optimized Dijkstra'a Algortihm Value: " + distArray[2]);
System.out.println("Gebze to Esenyurt Optimized Dijkstra'a Algortihm Value: "+ distArray[4]);
System.out.println("Gebze to Kemah Optimized Dijkstra'a Algortihm Value: " + distArray[5]);
```

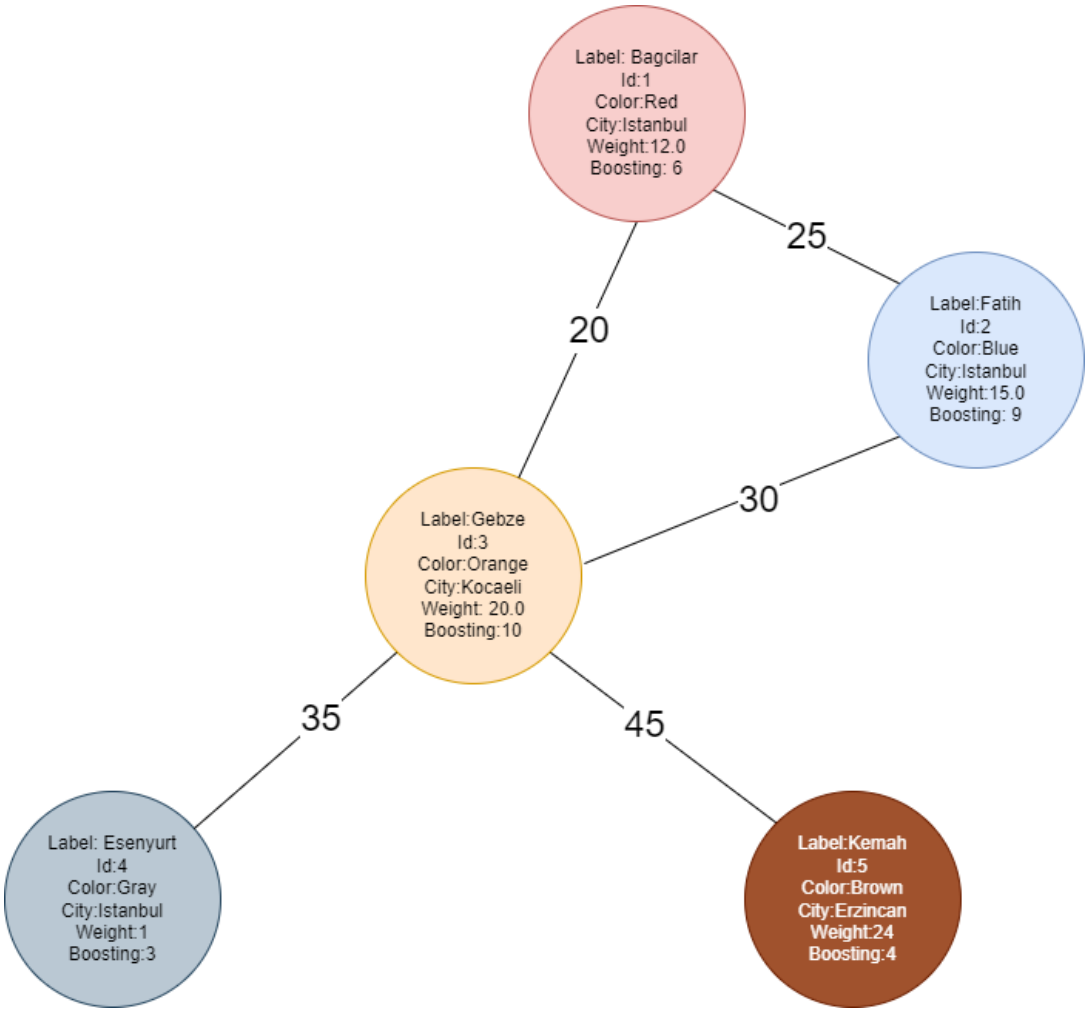
```
Gebze to Istanbul Optimized Dijkstra'a Algortihm Value: 29.0
Gebze to Bagciler Optimized Dijkstra'a Algortihm Value: 20.0
Gebze to Fatih Optimized Dijkstra'a Algortihm Value: 30.0
Gebze to Esenyurt Optimized Dijkstra'a Algortihm Value: 35.0
Gebze to Kemah Optimized Dijkstra'a Algortihm Value: 45.0
```

**Notice that the value of Gebze Istanbul has changed due to the boosting value.**

**TEST SUCCESSFUL**

---

**Lets remove Tuzla Vertex and than our Graph's looks like this:**




**Remove in the code and print graph.**

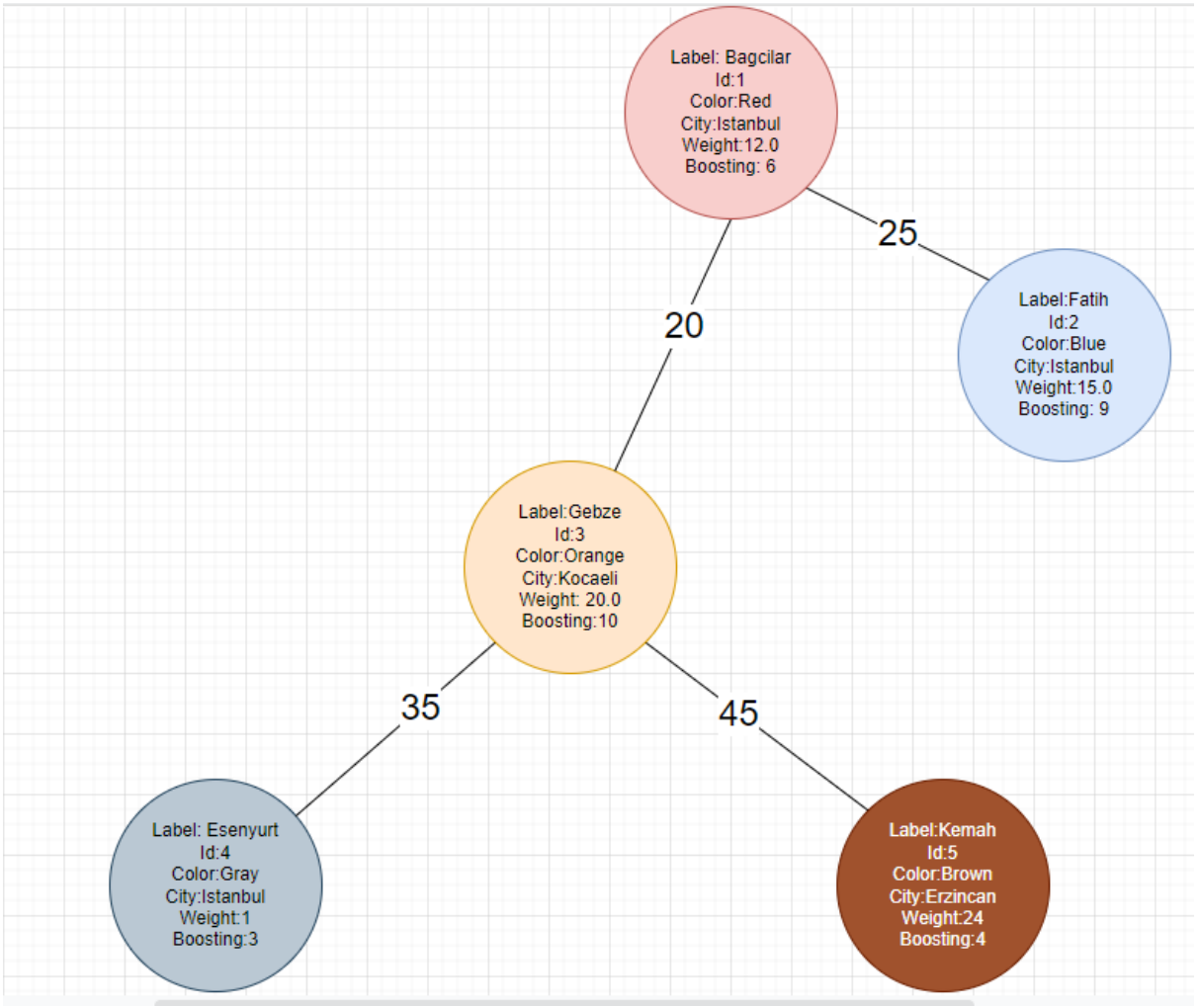
```
myTestGraph.removeVertex( vertexID: 0);
myTestGraph.printGraph();
```

```
[Vertex1] --> [Vertex2|25.0] --> [Vertex3|20.0]
[Vertex2] --> [Vertex1|25.0] --> [Vertex3|30.0]
[Vertex3] --> [Vertex1|20.0] --> [Vertex2|30.0] --> [Vertex4|35.0] --> [Vertex5|45.0]
[Vertex4] --> [Vertex3|35.0]
[Vertex5] --> [Vertex3|45.0]
```

**TEST SUCCESSFUL**

---

### Remove the edge Between Gebze - Fatih and Print Graph



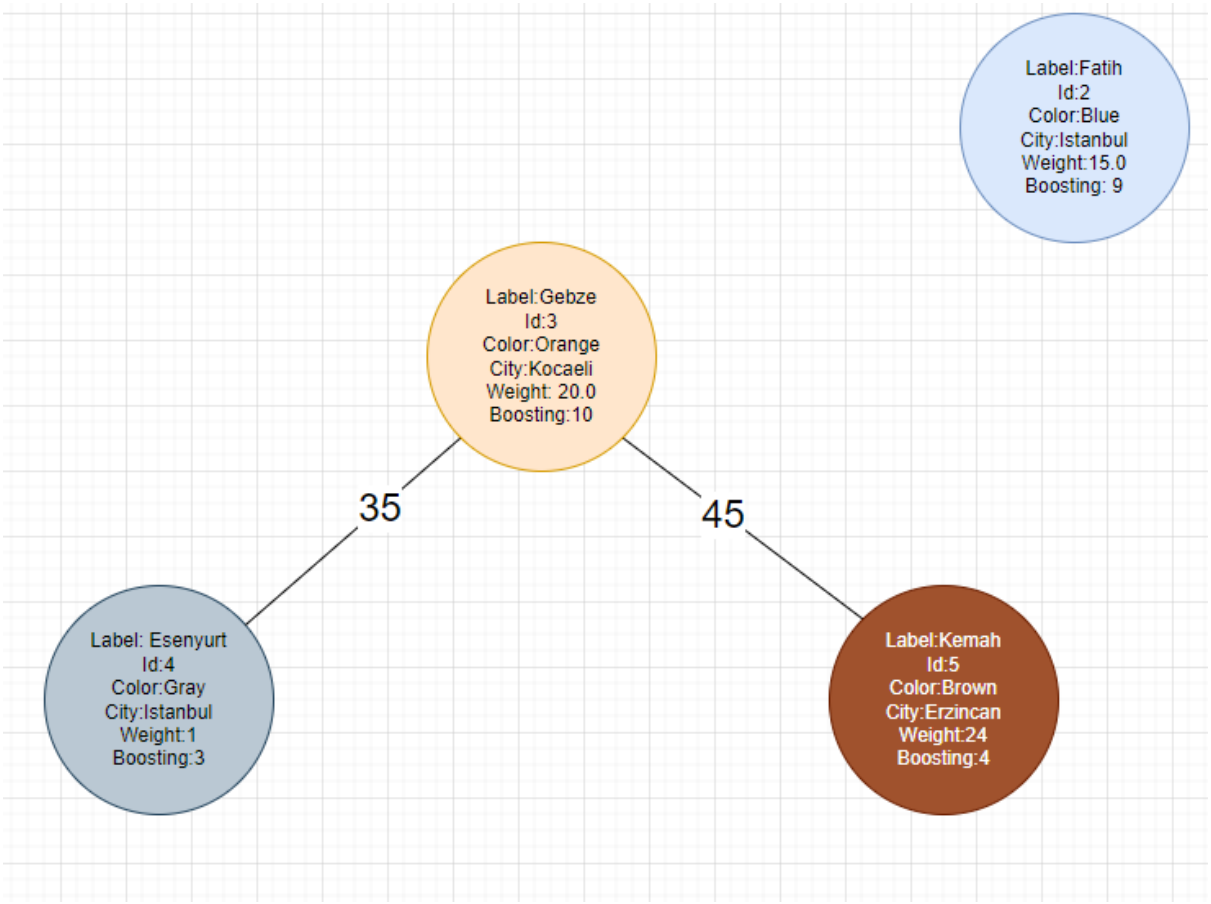

```
myTestGraph.removeEdge(3,2);
myTestGraph.printGraph();
```

```
[Vertex1] --> [Vertex2|25.0] --> [Vertex3|20.0]
[Vertex2] --> [Vertex1|25.0]
[Vertex3] --> [Vertex1|20.0] --> [Vertex4|35.0] --> [Vertex5|45.0]
[Vertex4] --> [Vertex3|35.0]
[Vertex5] --> [Vertex3|45.0]
```

**TEST SUCCESSFUL**

---

**Remove Bagcilar by Label and print Graph**




```
myTestGraph.removeVertex( label: "Bagcilar");
myTestGraph.printGraph();
```

```
[Vertex2]
[Vertex3] --> [Vertex4|35.0] --> [Vertex5|45.0]
[Vertex4] --> [Vertex3|35.0]
[Vertex5] --> [Vertex3|45.0]
```

**TEST SUCCESSFUL**

**For Q2:**
**Lets Start Index: 0 and End Index: 3 than make calculations:**

```
DepthFirstSearch myDFSTest = new DepthFirstSearch(myTestGraph);
int dfsArray [] = myDFSTest.depthFirstSearch( current: 0);
/*
for (int i = 0; i < dfsArray.length; i++){
    System.out.println(dfsArray[i]);
}
*/

myDFSTest.diffBetweenPaths(myTestGraph, start: 0, end: 3);
```

```
DFS Calculation : 70.0
BFS Calculation : 35.0
DFS - BFS : 35.0
```

**TEST SUCCESSFUL**

---

**Test for 0 and 5**

```
myDFSTest.diffBetweenPaths(myTestGraph, start: 0, end: 5);
```

```
DFS Calculation : 160.0
BFS Calculation : 65.0
DFS - BFS : 95.0
```

**TEST SUCCESSFUL**

# Also, I would like to thank the professor and assistants of CSE222 for all their efforts during the term :) ♥