GTU Department of Computer Engineering
CSE 222/505 - Spring 2022
Homework #08 Report

Ali Kaya

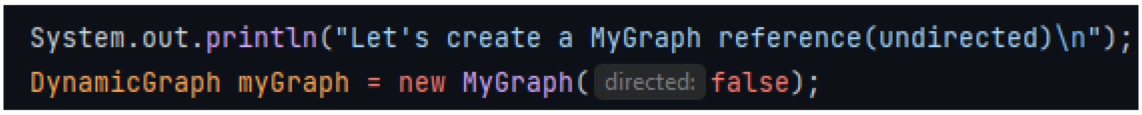
1901042618

a.kaya2019@gtu.edu.tr

1.Detailed System Requirements

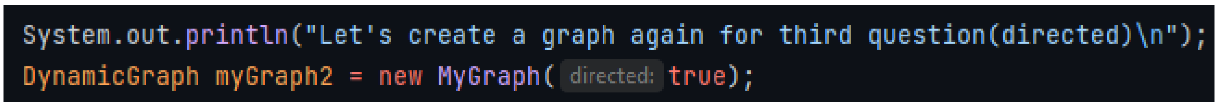
Only requirements for MyGraph class is this:

User must give the graph type information => Directed or Undirected Graph



```
System.out.println("Let's create a MyGraph reference(undirected)\n");
DynamicGraph myGraph = new MyGraph( directed: false);
```

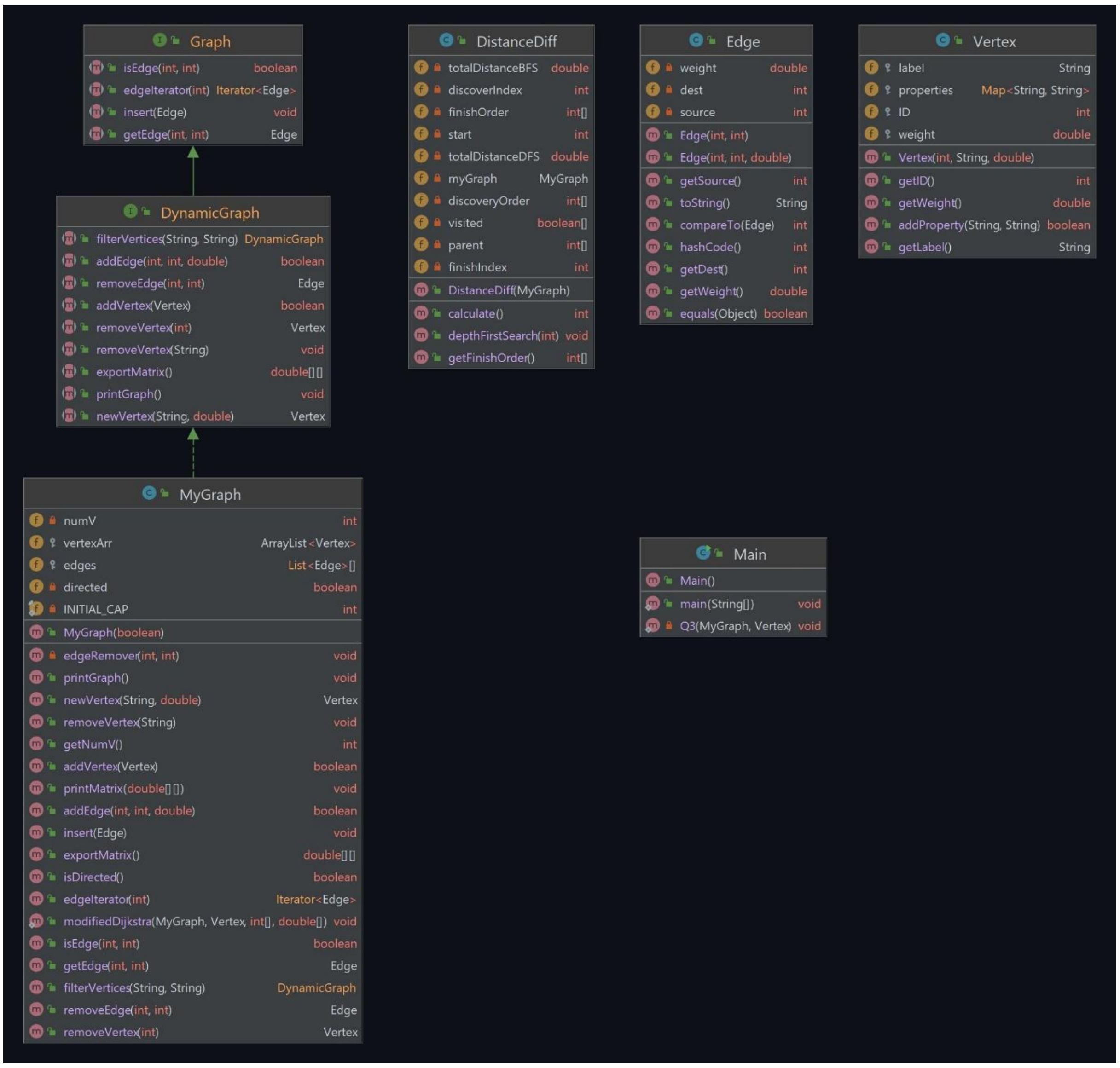
İnput false indicates our graph will be undirected .So ,two edges will be used for each connection between vertices.



```
System.out.println("Let's create a graph again for third question(directed)\n");
DynamicGraph myGraph2 = new MyGraph( directed: true);
```

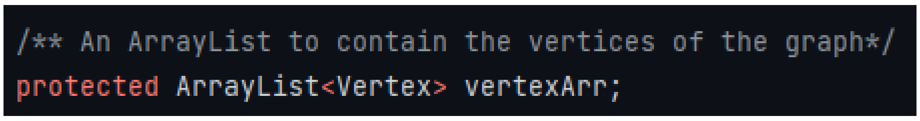
İnput true indicates our graph will be directed .So ,one edge will be used for each connection between vertices.

2.Class Diagrams



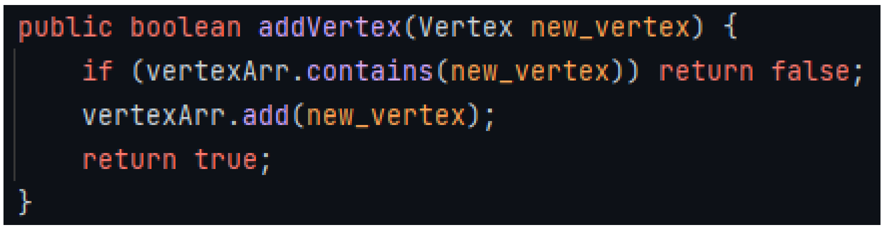
3. Problem solutions approach

=> Since it seems like a better idea to keep the vertices in an ArrayList so that I can access them at constant time, I kept them this way.



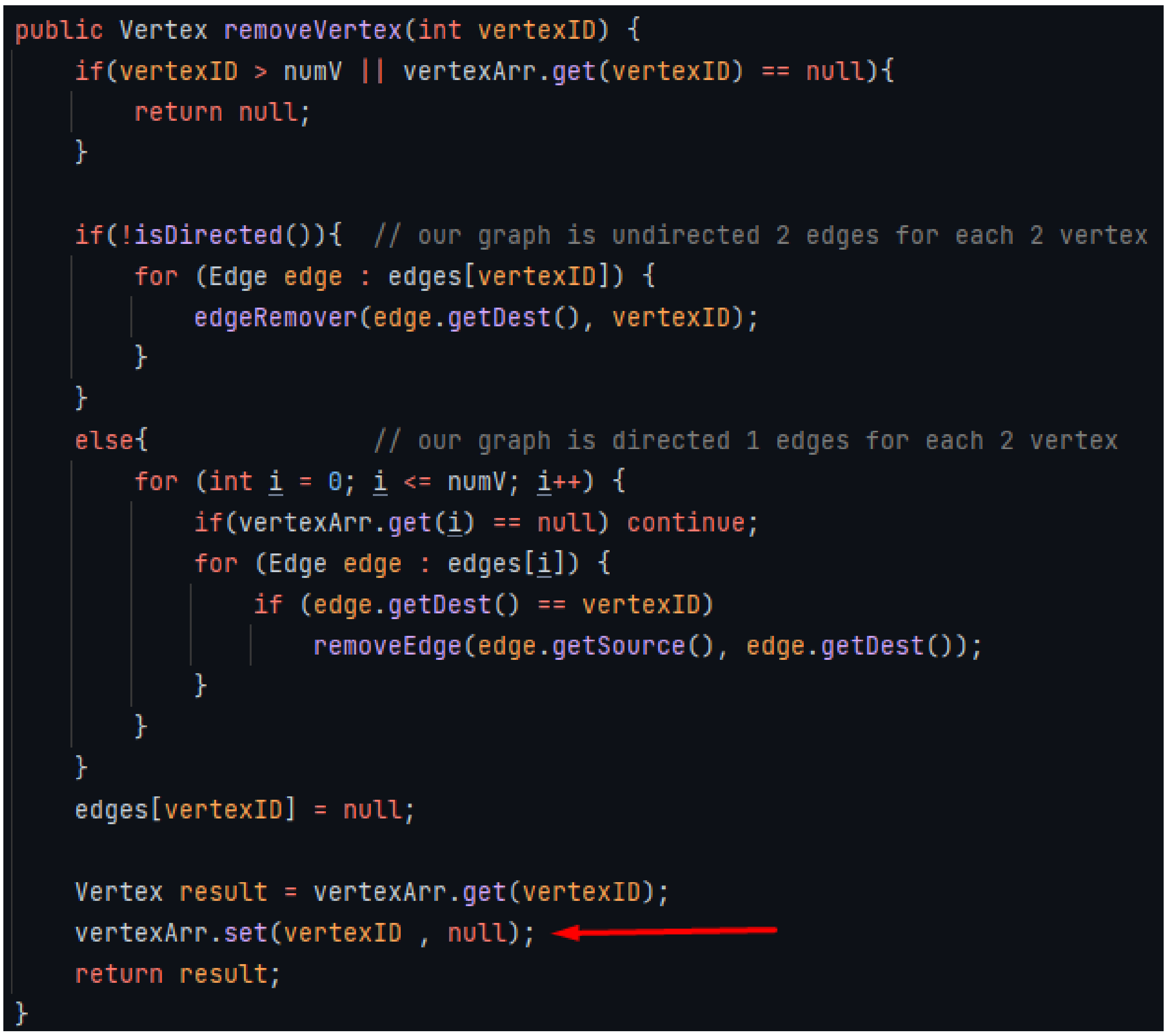
```java
/** An ArrayList to contain the vertices of the graph*/
protected ArrayList<Vertex> vertexArr;
```

Every time a new vertex is added, I add an element to this ArrayList.



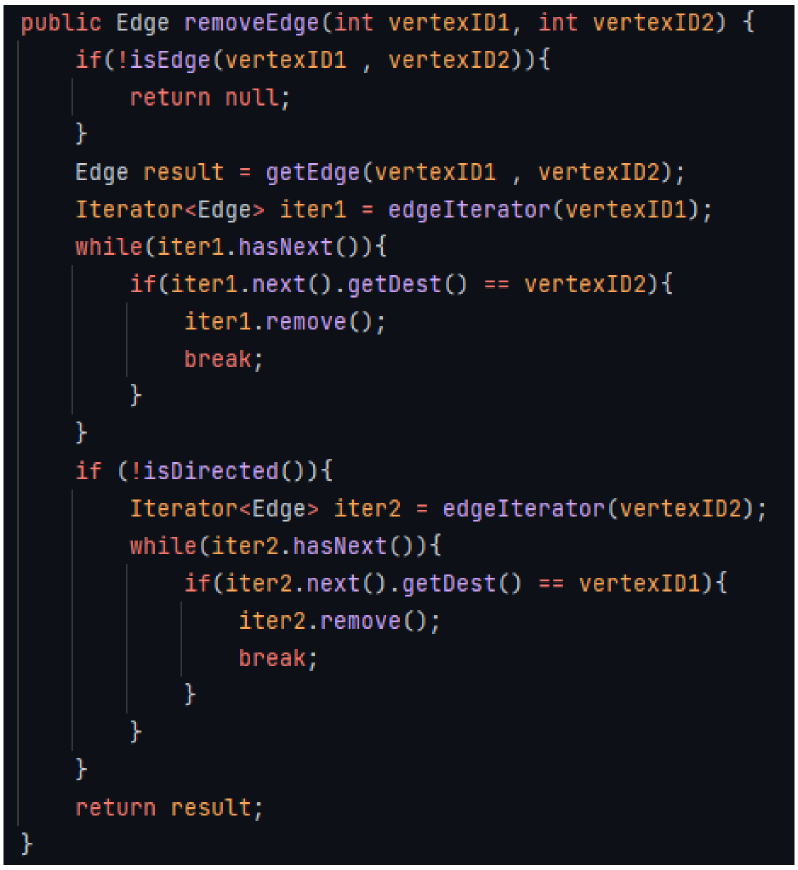
```java
public boolean addVertex(Vertex new_vertex) {
    if (vertexArr.contains(new_vertex)) return false;
    vertexArr.add(new_vertex);
    return true;
}
```

Every time a vertex is deleted, i set the place 'null' in this arraylist which has the specified index(given ID)



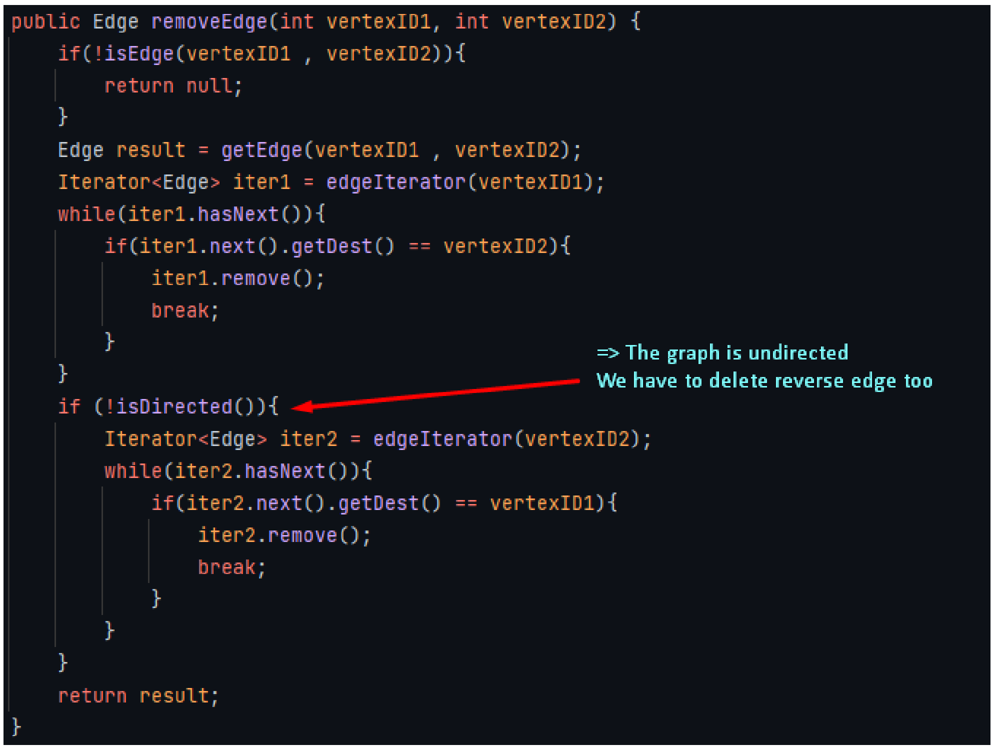
```java
public Vertex removeVertex(int vertexID) {
    if(vertexID > numV || vertexArr.get(vertexID) == null){
        return null;
    }

    if(!isDirected()){  // our graph is undirected 2 edges for each 2 vertex
        for (Edge edge : edges[vertexID]) {
            edgeRemover(edge.getDest(), vertexID);
        }
    }
    else{               // our graph is directed 1 edges for each 2 vertex
        for (int i = 0; i <= numV; i++) {
            if(vertexArr.get(i) == null) continue;
            for (Edge edge : edges[i]) {
                if (edge.getDest() == vertexID)
                    removeEdge(edge.getSource(), edge.getDest());
            }
        }
    }
    edges[vertexID] = null;

    Vertex result = vertexArr.get(vertexID);
    vertexArr.set(vertexID , null);
    return result;
}
```

⇨ While deleting a vertex, I also need to delete the edges connected to the vertex to be deleted.



```
public Edge removeEdge(int vertexID1, int vertexID2) {
    if(!isEdge(vertexID1 , vertexID2)){
        return null;
    }
    Edge result = getEdge(vertexID1 , vertexID2);
    Iterator<Edge> iter1 = edgeIterator(vertexID1);
    while(iter1.hasNext()){
        if(iter1.next().getDest() == vertexID2){
            iter1.remove();
            break;
        }
    }
    if (!isDirected()){
        Iterator<Edge> iter2 = edgeIterator(vertexID2);
        while(iter2.hasNext()){
            if(iter2.next().getDest() == vertexID1){
                iter2.remove();
                break;
            }
        }
    }
    return result;
}
```

When I need to delete an edge, the number of edges I need to delete changes according to the type of graph.



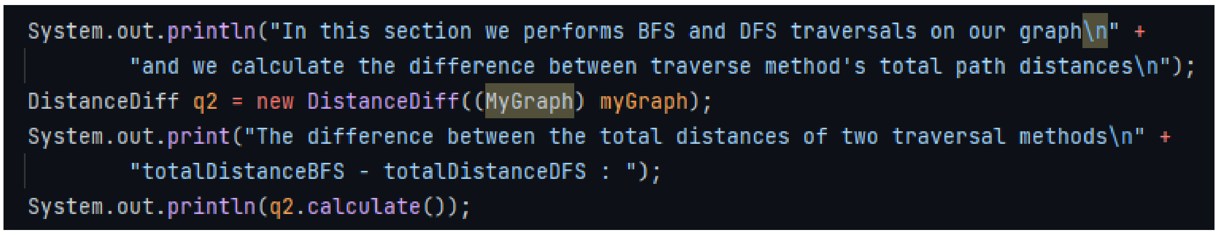
```
public Edge removeEdge(int vertexID1, int vertexID2) {
    if(!isEdge(vertexID1 , vertexID2)){
        return null;
    }
    Edge result = getEdge(vertexID1 , vertexID2);
    Iterator<Edge> iter1 = edgeIterator(vertexID1);
    while(iter1.hasNext()){
        if(iter1.next().getDest() == vertexID2){
            iter1.remove();
            break;
        }
    }
    if (!isDirected()){
        Iterator<Edge> iter2 = edgeIterator(vertexID2);
        while(iter2.hasNext()){
            if(iter2.next().getDest() == vertexID1){
                iter2.remove();
                break;
            }
        }
    }
    return result;
}
```

=> The graph is undirected
We have to delete reverse edge too

#For Q2

I create another class which is DistanceDiff:

In this class we have calculate() method, This method calculates the total distance of the path for accessing each vertex during the traversals(**BFS and DFS**), and it returns the difference between the total distances of two traversal methods.

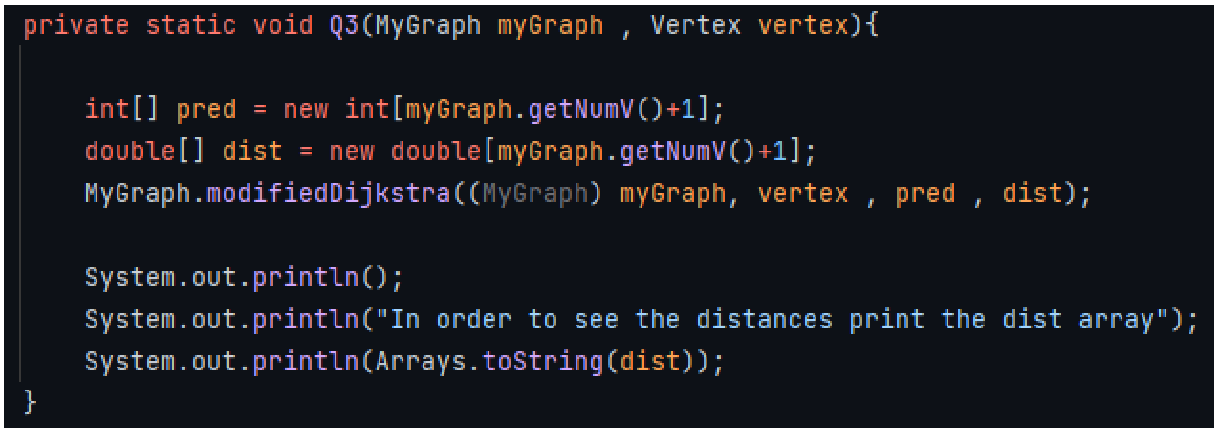


```
System.out.println("In this section we performs BFS and DFS traversals on our graph\n" +
        "and we calculate the difference between traverse method's total path distances\n");
DistanceDiff q2 = new DistanceDiff((MyGraph) myGraph);
System.out.print("The difference between the total distances of two traversal methods\n" +
        "totalDistanceBFS - totalDistanceDFS : ");
System.out.println(q2.calculate());
```

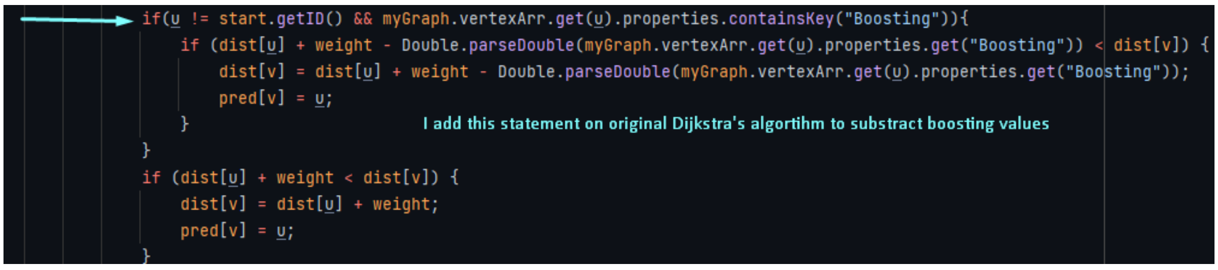
#For Q3

_I create a MyGraph reference which is myGraph2 and call Q3() method with two parameter => mygraph2 , v0 which is a vertex of myGraph2

Q3() method calls modifiedDijkstra() method with additional parameters



```
private static void Q3(MyGraph myGraph , Vertex vertex){

    int[] pred = new int[myGraph.getNumV()+1];
    double[] dist = new double[myGraph.getNumV()+1];
    MyGraph.modifiedDijkstra((MyGraph) myGraph, vertex , pred , dist);

    System.out.println();
    System.out.println("In order to see the distances print the dist array");
    System.out.println(Arrays.toString(dist));
}
```



```
if(u != start.getID() && myGraph.vertexArr.get(u).properties.containsKey("Boosting")){
    if (dist[u] + weight - Double.parseDouble(myGraph.vertexArr.get(u).properties.get("Boosting")) < dist[v]) {
        dist[v] = dist[u] + weight - Double.parseDouble(myGraph.vertexArr.get(u).properties.get("Boosting"));
        pred[v] = u;
    }                        I add this statement on original Dijkstra's algortihm to substract boosting values
}
if (dist[u] + weight < dist[v]) {
    dist[v] = dist[u] + weight;
    pred[v] = u;
}
```

$S = \{ 0, 1, 3, 2, 4 \}$

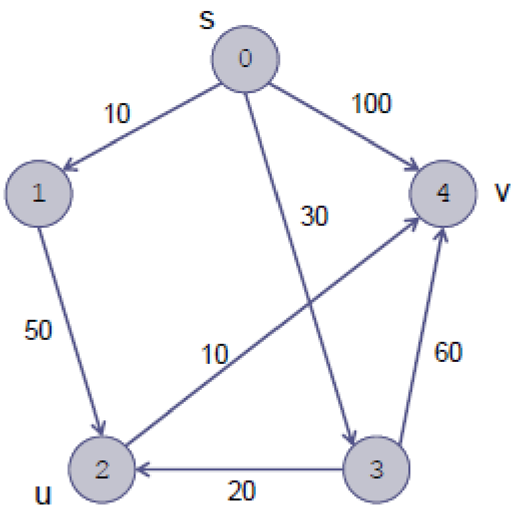
$V\text{-}S = \{ \}$

$u = 2$

dist[]

0	10	50	30	60

v	d[v]	p[v]
1	10	0
2	50	3
3	30	0
4	60	2



This is the original algorithm’s results.

$S = \{ 0, 1, 3, 2, 4 \}$

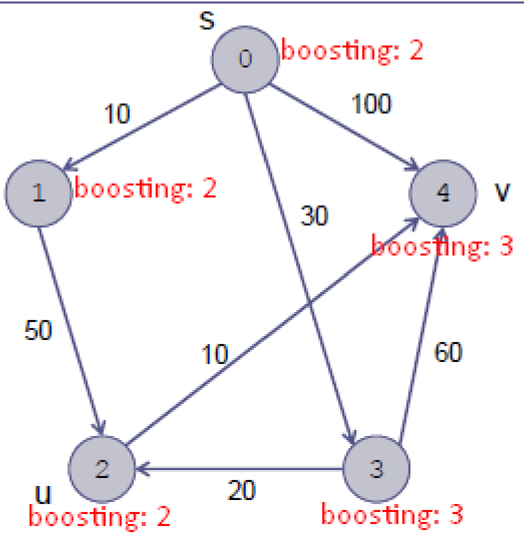
$V\text{-}S = \{ \}$

$u = 2$

dist[]

0	10	47	30	55

v	d[v]	p[v]
1	10	0
→ 2	50	3
3	30	0
→ 4	60	2



This is our modifiedDijkstras method’s results

55 = 30+20+10-3-2

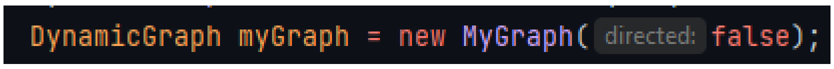
47 = 30+20-3

O(1) 1. newVertex (string label, double weight): Generate a new vertex by given parameters.
O(N) 2. addVertex (Vertex new_vertex): Add the given vertex to the graph.
O(M) 3. addEdge (int vertexID1, int vertexID2, double weight): Add an edge between the given two vertices in the graph.
O(M) 4. removeEdge (int vertexID1, int vertexID2): Remove the edge between the given two vertices.
O(M) 5. removeVertex (int vertexID): Remove the vertex from the graph with respect to the given vertex id.
O(M) 6. removeVertex (string label): Remove the vertices that have the given label from the graph.
Θ(N^2) 7. filterVertices (string key, string filter): Filter the vertices by the given user-defined property and returns a subgraph of the graph.
Θ(N^2) 8. exportMatrix(): Generate the adjacency matrix representation of the graph and returns the matrix.
Θ(M) 9. printGraph(): Print the graph in adjacency list format (You should use the format that can be imported by the method in AbstarctGraph in the book).

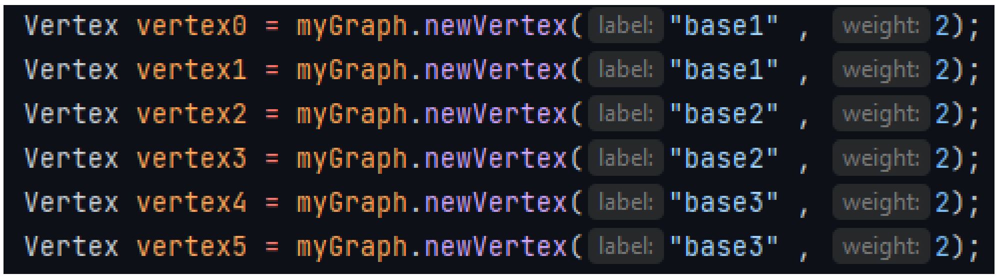
M => number of edges

N => number of vertices

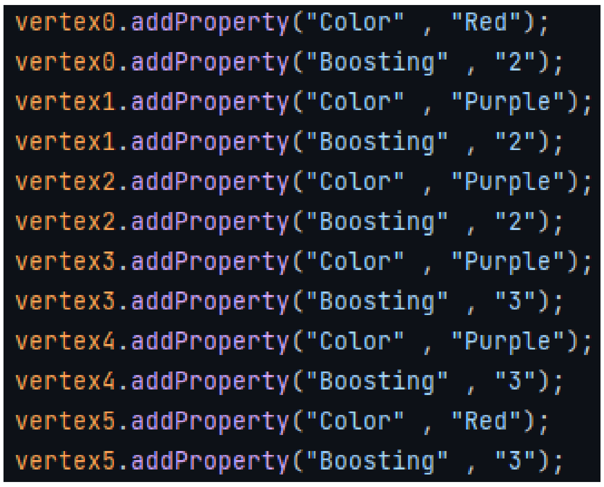
4)Test Cases



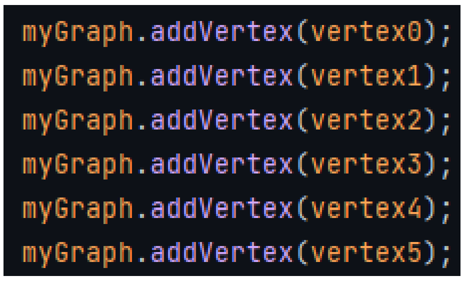
```
DynamicGraph myGraph = new MyGraph( directed: false);
```



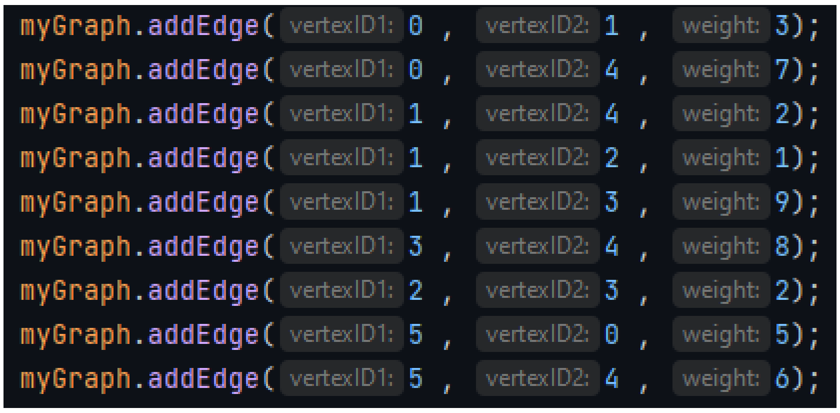
```
Vertex vertex0 = myGraph.newVertex( label: "base1" , weight: 2);
Vertex vertex1 = myGraph.newVertex( label: "base1" , weight: 2);
Vertex vertex2 = myGraph.newVertex( label: "base2" , weight: 2);
Vertex vertex3 = myGraph.newVertex( label: "base2" , weight: 2);
Vertex vertex4 = myGraph.newVertex( label: "base3" , weight: 2);
Vertex vertex5 = myGraph.newVertex( label: "base3" , weight: 2);
```



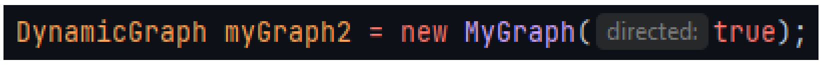
```
vertex0.addProperty("Color" , "Red");
vertex0.addProperty("Boosting" , "2");
vertex1.addProperty("Color" , "Purple");
vertex1.addProperty("Boosting" , "2");
vertex2.addProperty("Color" , "Purple");
vertex2.addProperty("Boosting" , "2");
vertex3.addProperty("Color" , "Purple");
vertex3.addProperty("Boosting" , "3");
vertex4.addProperty("Color" , "Purple");
vertex4.addProperty("Boosting" , "3");
vertex5.addProperty("Color" , "Red");
vertex5.addProperty("Boosting" , "3");
```



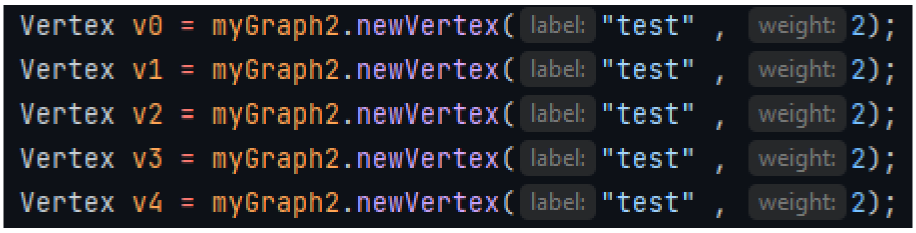
```
myGraph.addVertex(vertex0);
myGraph.addVertex(vertex1);
myGraph.addVertex(vertex2);
myGraph.addVertex(vertex3);
myGraph.addVertex(vertex4);
myGraph.addVertex(vertex5);
```



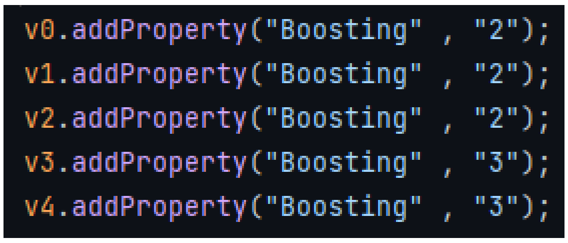
```
myGraph.addEdge( vertexID1: 0 , vertexID2: 1 , weight: 3);
myGraph.addEdge( vertexID1: 0 , vertexID2: 4 , weight: 7);
myGraph.addEdge( vertexID1: 1 , vertexID2: 4 , weight: 2);
myGraph.addEdge( vertexID1: 1 , vertexID2: 2 , weight: 1);
myGraph.addEdge( vertexID1: 1 , vertexID2: 3 , weight: 9);
myGraph.addEdge( vertexID1: 3 , vertexID2: 4 , weight: 8);
myGraph.addEdge( vertexID1: 2 , vertexID2: 3 , weight: 2);
myGraph.addEdge( vertexID1: 5 , vertexID2: 0 , weight: 5);
myGraph.addEdge( vertexID1: 5 , vertexID2: 4 , weight: 6);
```



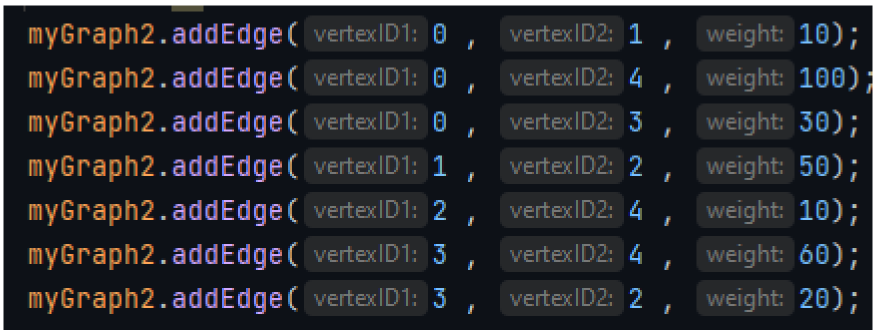
```
DynamicGraph myGraph2 = new MyGraph( directed: true);
```



```
Vertex v0 = myGraph2.newVertex( label: "test" , weight: 2);
Vertex v1 = myGraph2.newVertex( label: "test" , weight: 2);
Vertex v2 = myGraph2.newVertex( label: "test" , weight: 2);
Vertex v3 = myGraph2.newVertex( label: "test" , weight: 2);
Vertex v4 = myGraph2.newVertex( label: "test" , weight: 2);
```

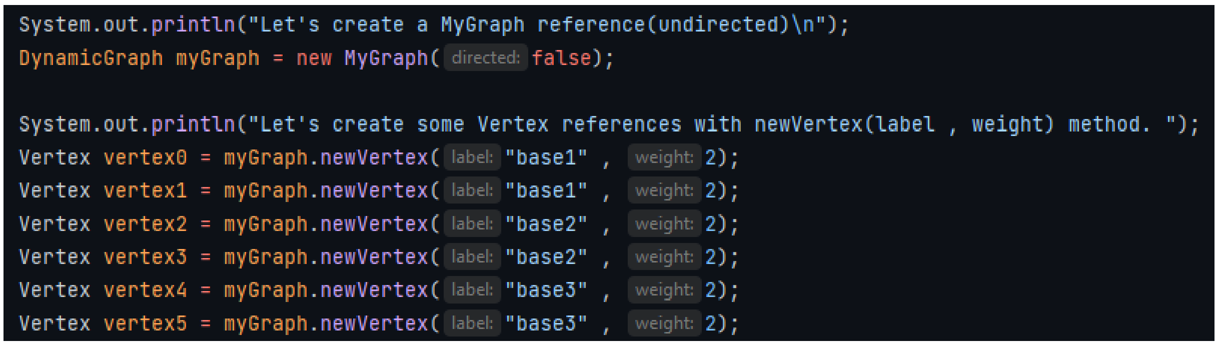


```
v0.addProperty("Boosting" , "2");
v1.addProperty("Boosting" , "2");
v2.addProperty("Boosting" , "2");
v3.addProperty("Boosting" , "3");
v4.addProperty("Boosting" , "3");
```

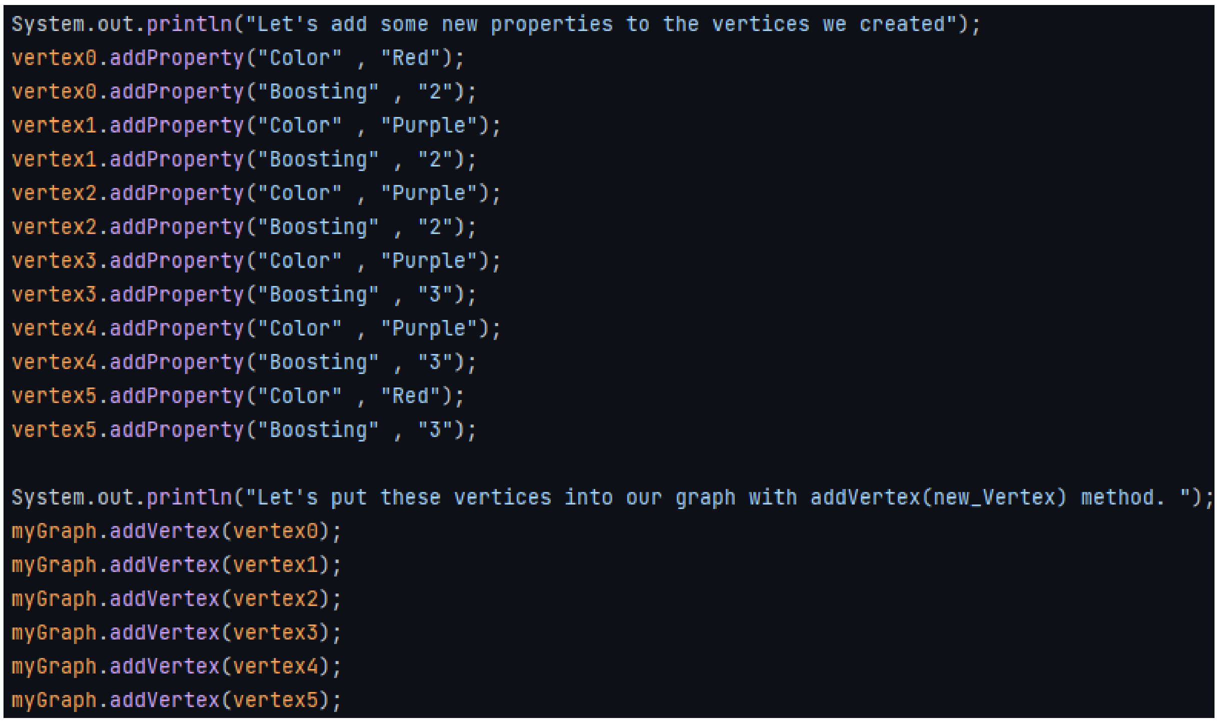


```
myGraph2.addEdge( vertexID1: 0 , vertexID2: 1 , weight: 10);
myGraph2.addEdge( vertexID1: 0 , vertexID2: 4 , weight: 100);
myGraph2.addEdge( vertexID1: 0 , vertexID2: 3 , weight: 30);
myGraph2.addEdge( vertexID1: 1 , vertexID2: 2 , weight: 50);
myGraph2.addEdge( vertexID1: 2 , vertexID2: 4 , weight: 10);
myGraph2.addEdge( vertexID1: 3 , vertexID2: 4 , weight: 60);
myGraph2.addEdge( vertexID1: 3 , vertexID2: 2 , weight: 20);
```

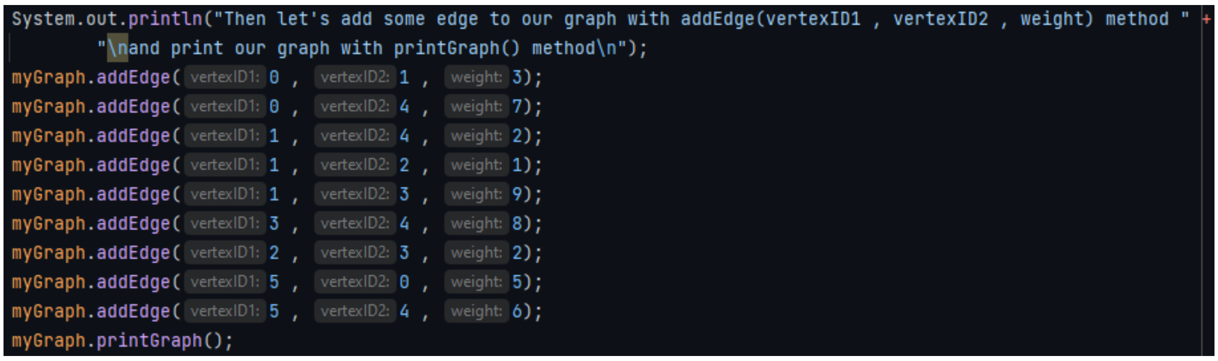
5) Running command and results



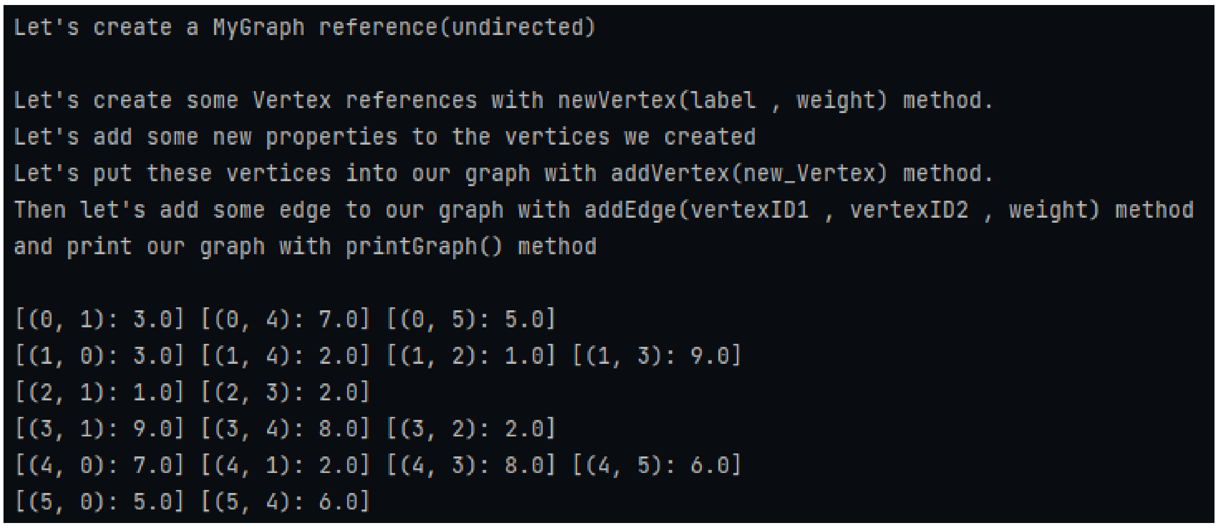
```
System.out.println("Let's create a MyGraph reference(undirected)\n");
DynamicGraph myGraph = new MyGraph( directed: false);

System.out.println("Let's create some Vertex references with newVertex(label , weight) method. ");
Vertex vertex0 = myGraph.newVertex( label: "base1" ,  weight: 2);
Vertex vertex1 = myGraph.newVertex( label: "base1" ,  weight: 2);
Vertex vertex2 = myGraph.newVertex( label: "base2" ,  weight: 2);
Vertex vertex3 = myGraph.newVertex( label: "base2" ,  weight: 2);
Vertex vertex4 = myGraph.newVertex( label: "base3" ,  weight: 2);
Vertex vertex5 = myGraph.newVertex( label: "base3" ,  weight: 2);
```



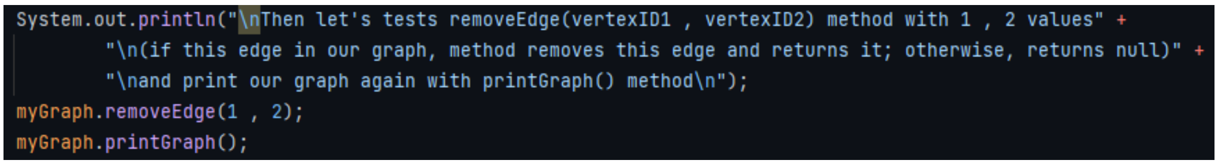
```
System.out.println("Let's add some new properties to the vertices we created");
vertex0.addProperty("Color" , "Red");
vertex0.addProperty("Boosting" , "2");
vertex1.addProperty("Color" , "Purple");
vertex1.addProperty("Boosting" , "2");
vertex2.addProperty("Color" , "Purple");
vertex2.addProperty("Boosting" , "2");
vertex3.addProperty("Color" , "Purple");
vertex3.addProperty("Boosting" , "3");
vertex4.addProperty("Color" , "Purple");
vertex4.addProperty("Boosting" , "3");
vertex5.addProperty("Color" , "Red");
vertex5.addProperty("Boosting" , "3");

System.out.println("Let's put these vertices into our graph with addVertex(new_Vertex) method. ");
myGraph.addVertex(vertex0);
myGraph.addVertex(vertex1);
myGraph.addVertex(vertex2);
myGraph.addVertex(vertex3);
myGraph.addVertex(vertex4);
myGraph.addVertex(vertex5);
```



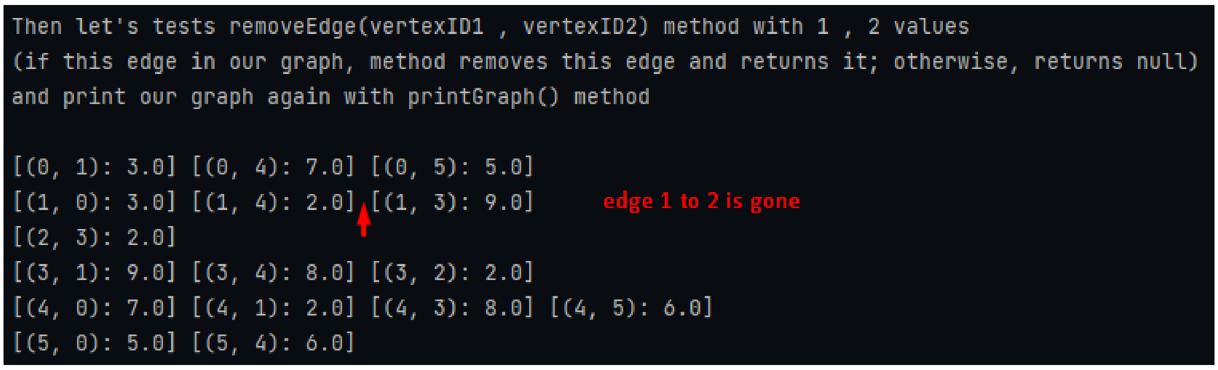
```
System.out.println("Then let's add some edge to our graph with addEdge(vertexID1 , vertexID2 , weight) method " +
        "\nand print our graph with printGraph() method\n");
myGraph.addEdge( vertexID1: 0 ,  vertexID2: 1 ,  weight: 3);
myGraph.addEdge( vertexID1: 0 ,  vertexID2: 4 ,  weight: 7);
myGraph.addEdge( vertexID1: 1 ,  vertexID2: 4 ,  weight: 2);
myGraph.addEdge( vertexID1: 1 ,  vertexID2: 2 ,  weight: 1);
myGraph.addEdge( vertexID1: 1 ,  vertexID2: 3 ,  weight: 9);
myGraph.addEdge( vertexID1: 3 ,  vertexID2: 4 ,  weight: 8);
myGraph.addEdge( vertexID1: 2 ,  vertexID2: 3 ,  weight: 2);
myGraph.addEdge( vertexID1: 5 ,  vertexID2: 0 ,  weight: 5);
myGraph.addEdge( vertexID1: 5 ,  vertexID2: 4 ,  weight: 6);
myGraph.printGraph();
```



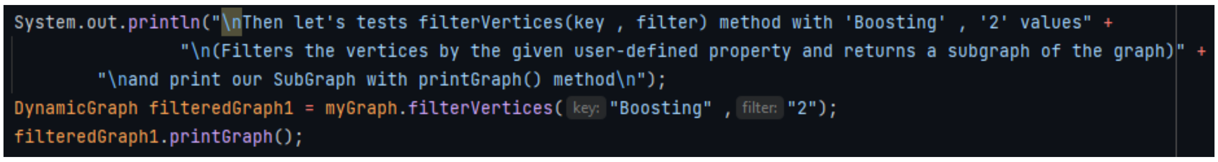
```
Let's create a MyGraph reference(undirected)

Let's create some Vertex references with newVertex(label , weight) method.
Let's add some new properties to the vertices we created
Let's put these vertices into our graph with addVertex(new_Vertex) method.
Then let's add some edge to our graph with addEdge(vertexID1 , vertexID2 , weight) method
and print our graph with printGraph() method

[(0, 1): 3.0] [(0, 4): 7.0] [(0, 5): 5.0]
[(1, 0): 3.0] [(1, 4): 2.0] [(1, 2): 1.0] [(1, 3): 9.0]
[(2, 1): 1.0] [(2, 3): 2.0]
[(3, 1): 9.0] [(3, 4): 8.0] [(3, 2): 2.0]
[(4, 0): 7.0] [(4, 1): 2.0] [(4, 3): 8.0] [(4, 5): 6.0]
[(5, 0): 5.0] [(5, 4): 6.0]
```



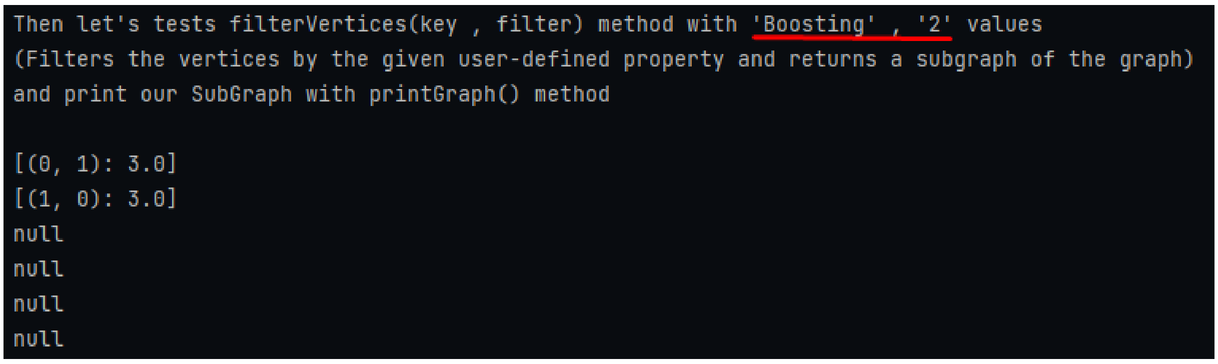
```
System.out.println("\nThen let's tests removeEdge(vertexID1 , vertexID2) method with 1 , 2 values" +
        "\n(if this edge in our graph, method removes this edge and returns it; otherwise, returns null)" +
        "\nand print our graph again with printGraph() method\n");
myGraph.removeEdge(1 , 2);
myGraph.printGraph();
```



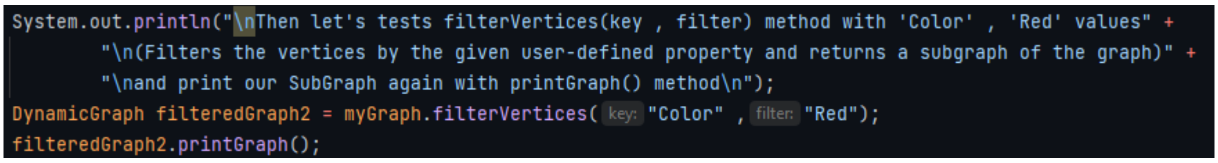
```
Then let's tests removeEdge(vertexID1 , vertexID2) method with 1 , 2 values
(if this edge in our graph, method removes this edge and returns it; otherwise, returns null)
and print our graph again with printGraph() method

[(0, 1): 3.0] [(0, 4): 7.0] [(0, 5): 5.0]
[(1, 0): 3.0] [(1, 4): 2.0] [(1, 3): 9.0]
[(2, 3): 2.0]
[(3, 1): 9.0] [(3, 4): 8.0] [(3, 2): 2.0]
[(4, 0): 7.0] [(4, 1): 2.0] [(4, 3): 8.0] [(4, 5): 6.0]
[(5, 0): 5.0] [(5, 4): 6.0]
```



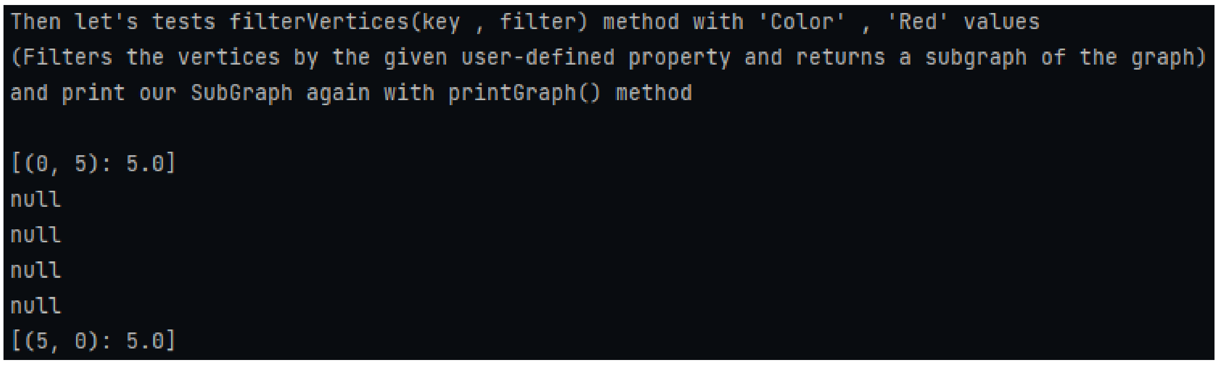
```
System.out.println("\nThen let's tests filterVertices(key , filter) method with 'Boosting' , '2' values" +
                "\n(Filters the vertices by the given user-defined property and returns a subgraph of the graph)" +
        "\nand print our SubGraph with printGraph() method\n");
DynamicGraph filteredGraph1 = myGraph.filterVertices( key: "Boosting" , filter: "2");
filteredGraph1.printGraph();
```



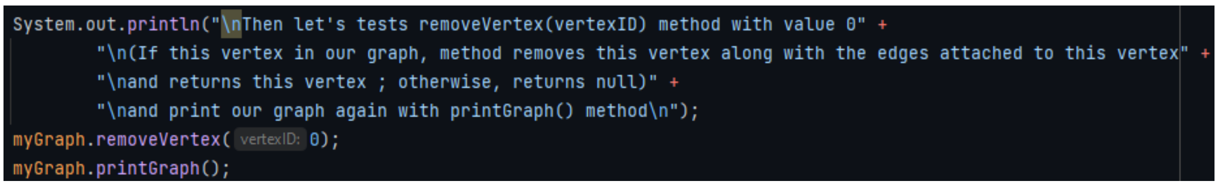
```
Then let's tests filterVertices(key , filter) method with 'Boosting' , '2' values
(Filters the vertices by the given user-defined property and returns a subgraph of the graph)
and print our SubGraph with printGraph() method

[(0, 1): 3.0]
[(1, 0): 3.0]
null
null
null
null
```



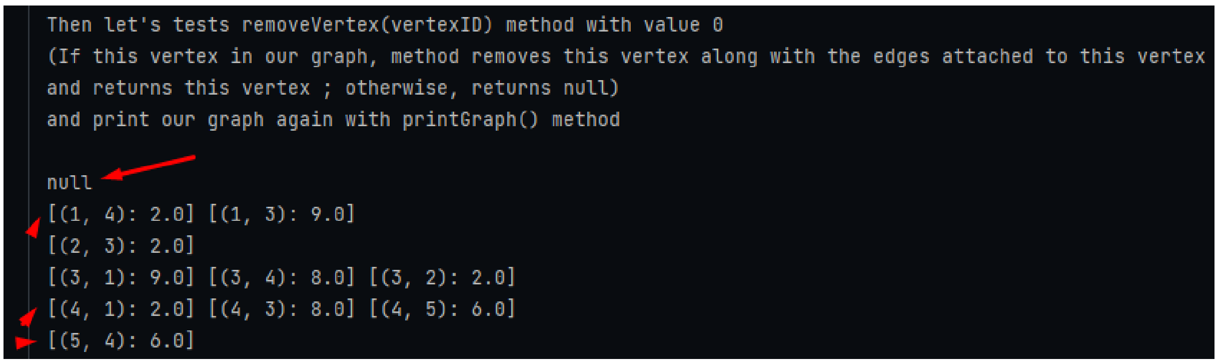
```
System.out.println("\nThen let's tests filterVertices(key , filter) method with 'Color' , 'Red' values" +
        "\n(Filters the vertices by the given user-defined property and returns a subgraph of the graph)" +
        "\nand print our SubGraph again with printGraph() method\n");
DynamicGraph filteredGraph2 = myGraph.filterVertices( key: "Color" , filter: "Red");
filteredGraph2.printGraph();
```



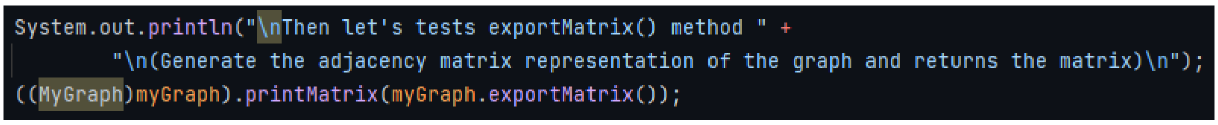
```
Then let's tests filterVertices(key , filter) method with 'Color' , 'Red' values
(Filters the vertices by the given user-defined property and returns a subgraph of the graph)
and print our SubGraph again with printGraph() method

[(0, 5): 5.0]
null
null
null
null
[(5, 0): 5.0]
```



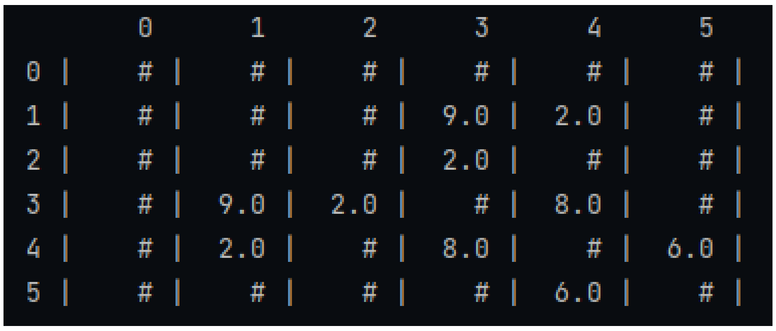
```
System.out.println("\nThen let's tests removeVertex(vertexID) method with value 0" +
        "\n(If this vertex in our graph, method removes this vertex along with the edges attached to this vertex" +
        "\nand returns this vertex ; otherwise, returns null)" +
        "\nand print our graph again with printGraph() method\n");
myGraph.removeVertex( vertexID: 0);
myGraph.printGraph();
```



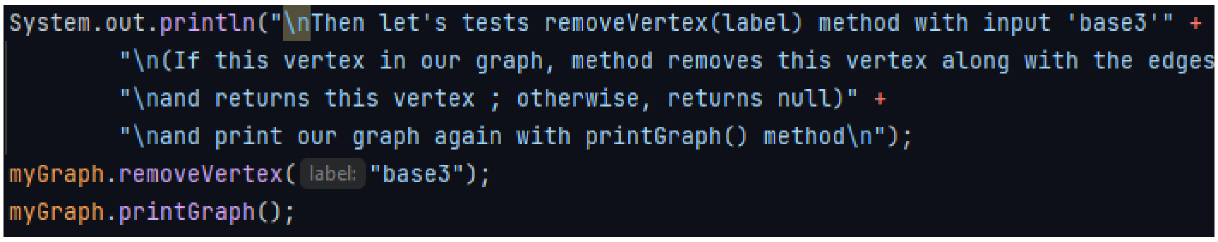
```
Then let's tests removeVertex(vertexID) method with value 0
(If this vertex in our graph, method removes this vertex along with the edges attached to this vertex
and returns this vertex ; otherwise, returns null)
and print our graph again with printGraph() method

null
[(1, 4): 2.0] [(1, 3): 9.0]
[(2, 3): 2.0]
[(3, 1): 9.0] [(3, 4): 8.0] [(3, 2): 2.0]
[(4, 1): 2.0] [(4, 3): 8.0] [(4, 5): 6.0]
[(5, 4): 6.0]
```



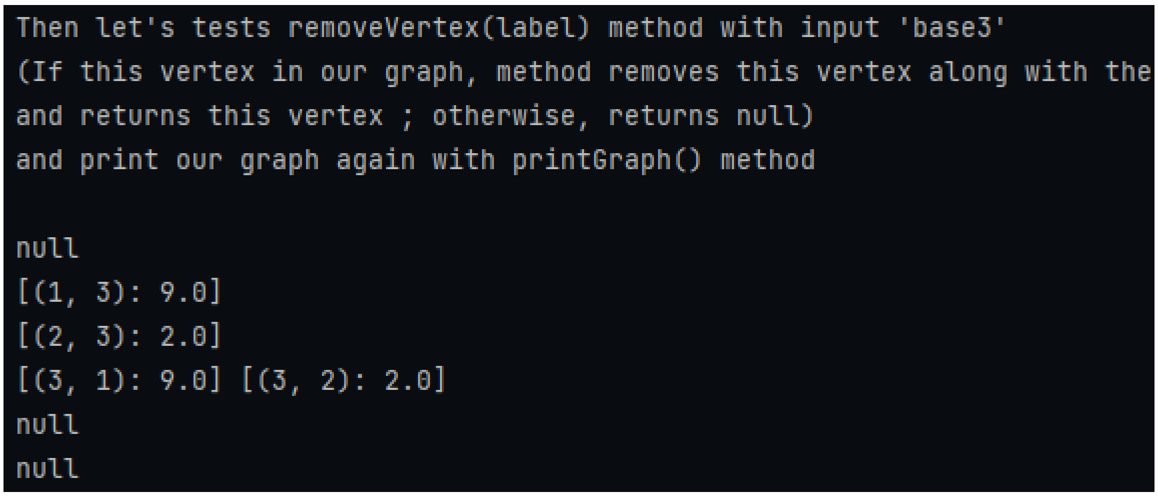
```
System.out.println("\nThen let's tests exportMatrix() method " +
        "\n(Generate the adjacency matrix representation of the graph and returns the matrix)\n");
((MyGraph)myGraph).printMatrix(myGraph.exportMatrix());
```



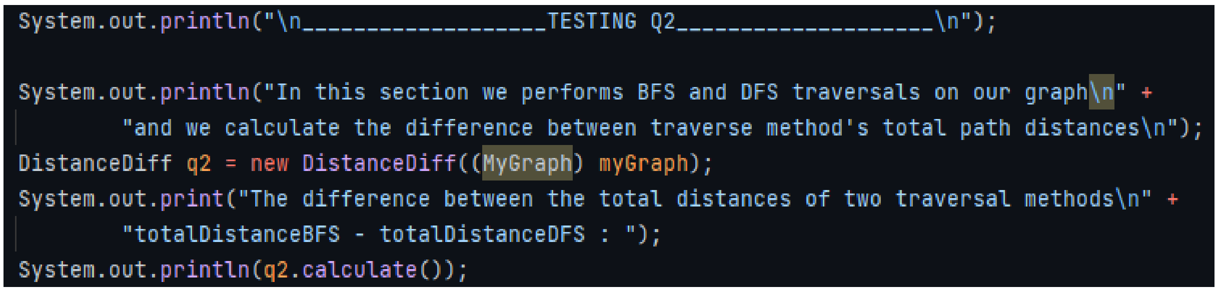
```
            0         1         2         3         4         5
 0 |        # |       # |       # |       # |       # |       # |
 1 |        # |       # |       # |     9.0 |     2.0 |       # |
 2 |        # |       # |       # |     2.0 |       # |       # |
 3 |        # |     9.0 |     2.0 |       # |     8.0 |       # |
 4 |        # |     2.0 |       # |     8.0 |       # |     6.0 |
 5 |        # |       # |       # |       # |     6.0 |       # |
```



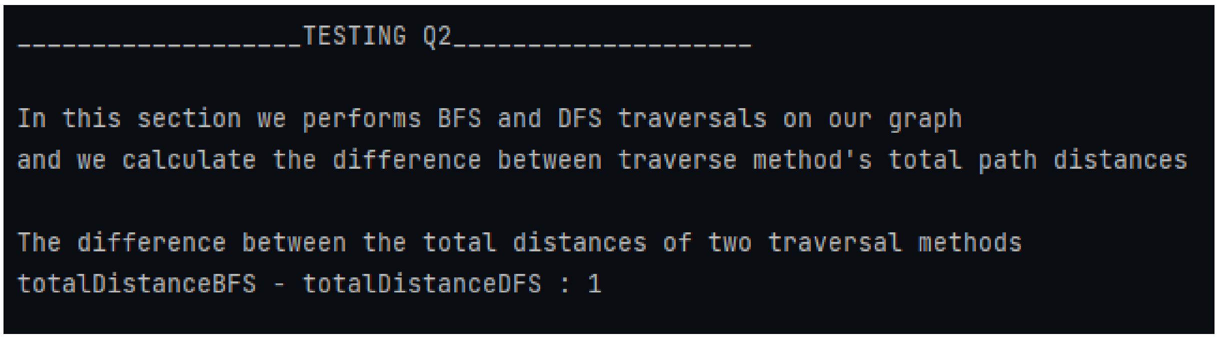
```java
System.out.println("\nThen let's tests removeVertex(label) method with input 'base3'" +
        "\n(If this vertex in our graph, method removes this vertex along with the edges
        "\nand returns this vertex ; otherwise, returns null)" +
        "\nand print our graph again with printGraph() method\n");
myGraph.removeVertex( label: "base3");
myGraph.printGraph();
```



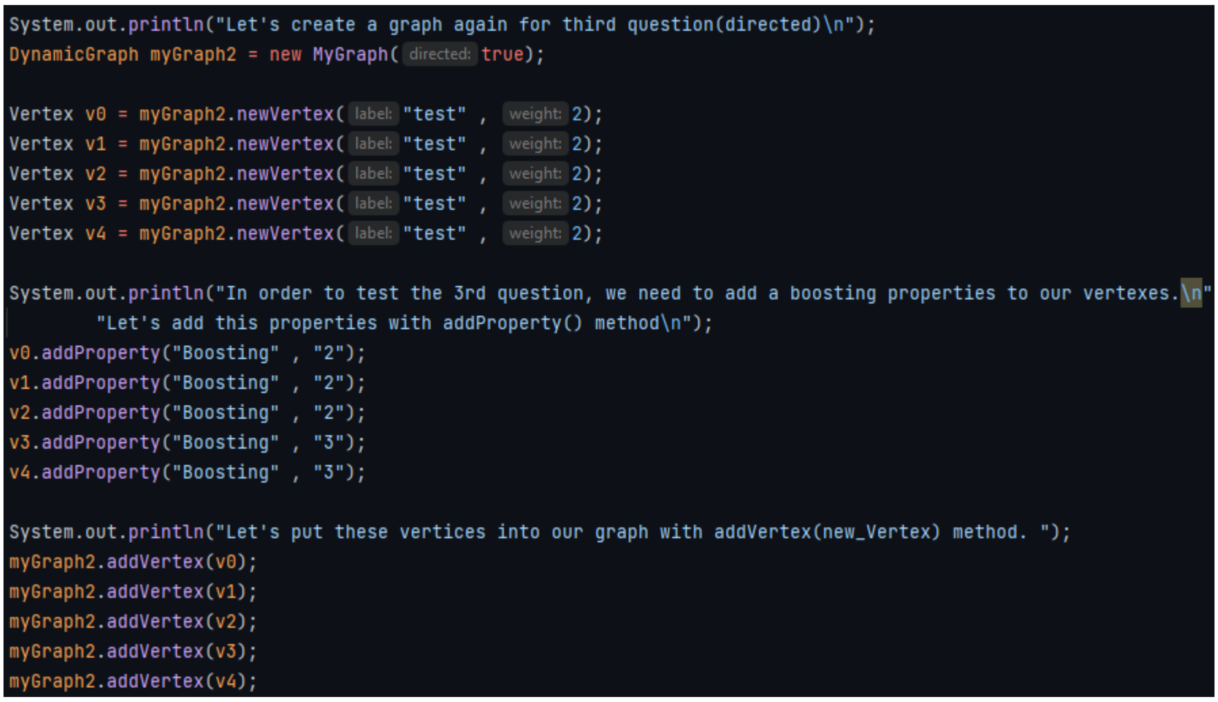
```
Then let's tests removeVertex(label) method with input 'base3'
(If this vertex in our graph, method removes this vertex along with the
and returns this vertex ; otherwise, returns null)
and print our graph again with printGraph() method

null
[(1, 3): 9.0]
[(2, 3): 2.0]
[(3, 1): 9.0] [(3, 2): 2.0]
null
null
```



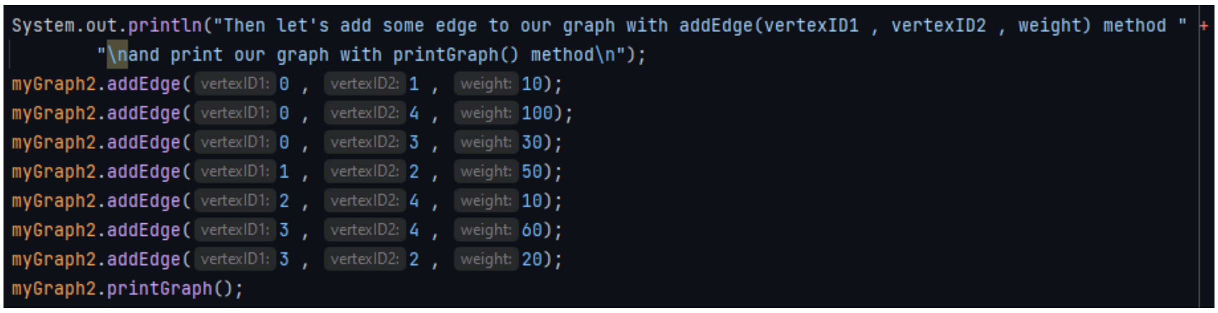
```java
System.out.println("\n___________________TESTING Q2____________________\n");

System.out.println("In this section we performs BFS and DFS traversals on our graph\n" +
        "and we calculate the difference between traverse method's total path distances\n");
DistanceDiff q2 = new DistanceDiff((MyGraph) myGraph);
System.out.print("The difference between the total distances of two traversal methods\n" +
        "totalDistanceBFS - totalDistanceDFS : ");
System.out.println(q2.calculate());
```



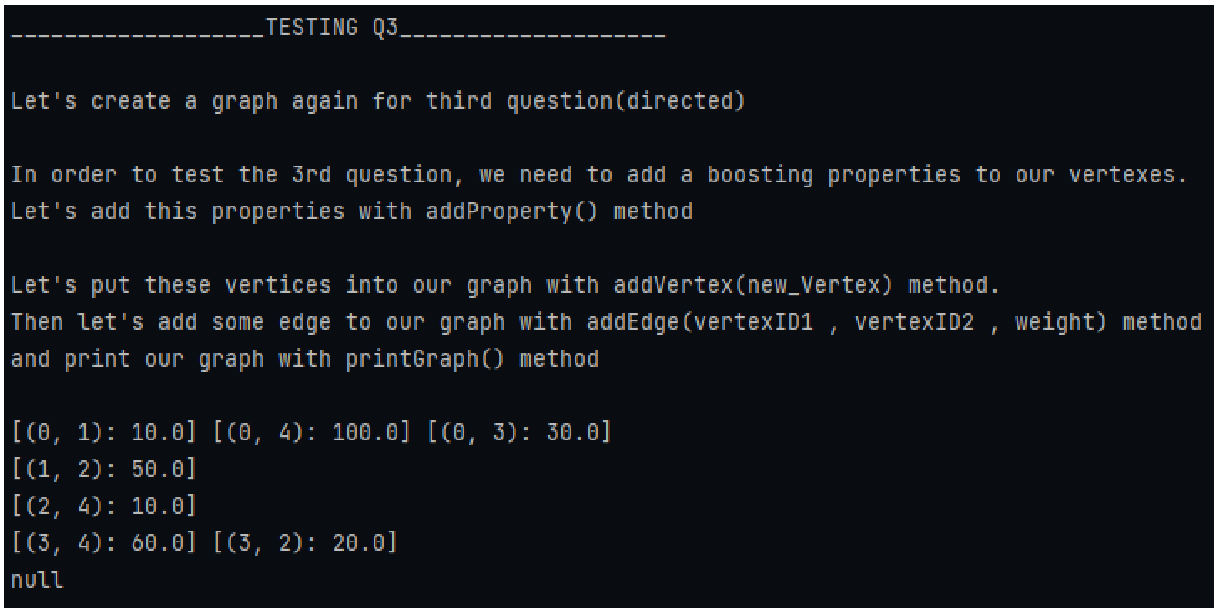
```
___________________TESTING Q2____________________

In this section we performs BFS and DFS traversals on our graph
and we calculate the difference between traverse method's total path distances

The difference between the total distances of two traversal methods
totalDistanceBFS - totalDistanceDFS : 1
```



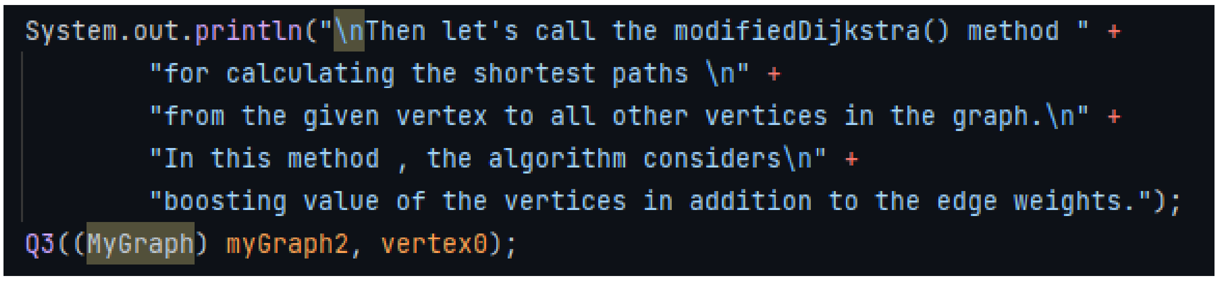
```
System.out.println("Let's create a graph again for third question(directed)\n");
DynamicGraph myGraph2 = new MyGraph( directed: true);

Vertex v0 = myGraph2.newVertex( label: "test" ,  weight: 2);
Vertex v1 = myGraph2.newVertex( label: "test" ,  weight: 2);
Vertex v2 = myGraph2.newVertex( label: "test" ,  weight: 2);
Vertex v3 = myGraph2.newVertex( label: "test" ,  weight: 2);
Vertex v4 = myGraph2.newVertex( label: "test" ,  weight: 2);

System.out.println("In order to test the 3rd question, we need to add a boosting properties to our vertexes.\n"
        "Let's add this properties with addProperty() method\n");
v0.addProperty("Boosting" , "2");
v1.addProperty("Boosting" , "2");
v2.addProperty("Boosting" , "2");
v3.addProperty("Boosting" , "3");
v4.addProperty("Boosting" , "3");

System.out.println("Let's put these vertices into our graph with addVertex(new_Vertex) method. ");
myGraph2.addVertex(v0);
myGraph2.addVertex(v1);
myGraph2.addVertex(v2);
myGraph2.addVertex(v3);
myGraph2.addVertex(v4);
```



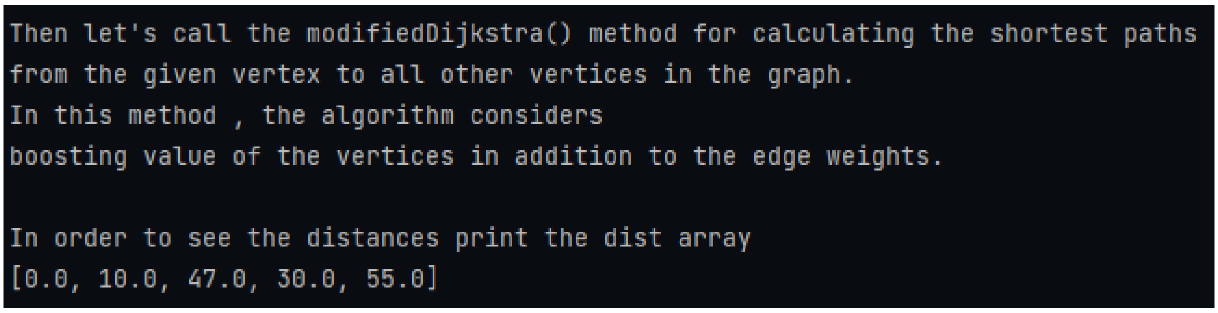
```
System.out.println("Then let's add some edge to our graph with addEdge(vertexID1 , vertexID2 , weight) method " +
        "\nand print our graph with printGraph() method\n");
myGraph2.addEdge( vertexID1: 0 ,  vertexID2: 1 ,  weight: 10);
myGraph2.addEdge( vertexID1: 0 ,  vertexID2: 4 ,  weight: 100);
myGraph2.addEdge( vertexID1: 0 ,  vertexID2: 3 ,  weight: 30);
myGraph2.addEdge( vertexID1: 1 ,  vertexID2: 2 ,  weight: 50);
myGraph2.addEdge( vertexID1: 2 ,  vertexID2: 4 ,  weight: 10);
myGraph2.addEdge( vertexID1: 3 ,  vertexID2: 4 ,  weight: 60);
myGraph2.addEdge( vertexID1: 3 ,  vertexID2: 2 ,  weight: 20);
myGraph2.printGraph();
```



```
___________________TESTING Q3___________________

Let's create a graph again for third question(directed)

In order to test the 3rd question, we need to add a boosting properties to our vertexes.
Let's add this properties with addProperty() method

Let's put these vertices into our graph with addVertex(new_Vertex) method.
Then let's add some edge to our graph with addEdge(vertexID1 , vertexID2 , weight) method
and print our graph with printGraph() method

[(0, 1): 10.0] [(0, 4): 100.0] [(0, 3): 30.0]
[(1, 2): 50.0]
[(2, 4): 10.0]
[(3, 4): 60.0] [(3, 2): 20.0]
null
```



```java
System.out.println("\nThen let's call the modifiedDijkstra() method " +
        "for calculating the shortest paths \n" +
        "from the given vertex to all other vertices in the graph.\n" +
        "In this method , the algorithm considers\n" +
        "boosting value of the vertices in addition to the edge weights.");
Q3((MyGraph) myGraph2, vertex0);
```



```
Then let's call the modifiedDijkstra() method for calculating the shortest paths
from the given vertex to all other vertices in the graph.
In this method , the algorithm considers
boosting value of the vertices in addition to the edge weights.

In order to see the distances print the dist array
[0.0, 10.0, 47.0, 30.0, 55.0]
```