GIT Department of Computer Engineering CSE 222/505 - Spring 2022
Homework #8
Report

Sefa Çiçek
1801042657

1. System Requirements

- jdk and jre are requested from operating system to execute this java program.

- User has to run makefile. (src/makefile)

2. Problem Solution Approach

Q1:

I've held the vertices with arrayList and edges with edge array. I've used helper functions on MyGraph class for edge and vertex operations such as isEdge, insert etc. I would like to briefly touch on the functions.

- newVertex : I've simply create vertex object with given parameters.
- addVertex : I've checked the vertex id, if id is not unique, vertex cannot be added to the graph.
- addEdge : If given vertices are exist, I call helper "insert" function to add edge.
- removeEdge : I've called helper "isEdge" function to determine whether given vertices have edge or not. Afterwards, removing took place according to the returned boolean value.
- removeVertex (id) : I've removed the vertex which has given id from vertices list. (==) is used for comparison.
- removeVertex (label) : I've removed the vertex which has given label from vertices list. (.equals) is used for comparison.
- filterVertices : I've used "Field" to get the data fields of Vertex class. Firstly, I've found which field is to be wanted with given "key" parameter then I've added the vertices on the graph to the subgraph according to their data value. If data value equals "filter", I've added them. Lastly, I've created edges between vertices on the subgraph. I've used helper "isEdge" function for that.
- exportMatrix : I've used helper "getEdge" function to get edges. I've simply added their weights to the double array. Since 0 is a valid weight, I've used Double.POSITIVE_INFINITY to represent the absence of an edge.
- printGraph : I've used edge iterator to traverse the list. I've assigned the last vertex id to an integer value to set the limit value of the loop.

Q2:

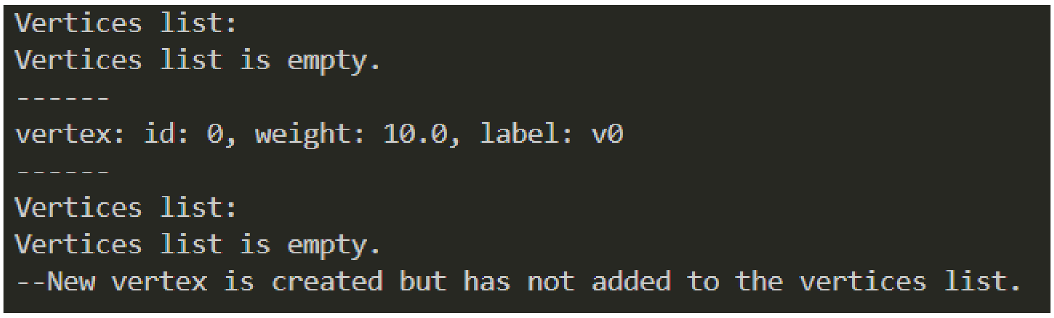
- BFS : Firstly, I've changed "theQueue" type to the arrayList to access its indices. Also, "identified" list holds the edges, not boolean values. If an edge has already added to the "identified" list, I've compared its weight with the current edge. If current edge weight is less than "identified edge", I remove the vertex from "theQueue" and add new vertex to the queue. Total distance is calculated again.
- DFS : I've sorted the edges according to the their weights. So, "visited" array gets the edge which has smaller value each time. The weight of the edge is added to the total distance.

Q3:
I’ve get the boost value of the current vertex and subtract it from the distance + weight value. If result is less than distance value of v(vertex), I’ve assigned result to the distance of v. Lastly, I’ve printed the “dist array – holds distances- ” to display shortest distances from source vertex.

3. Test Cases and Running commands and results

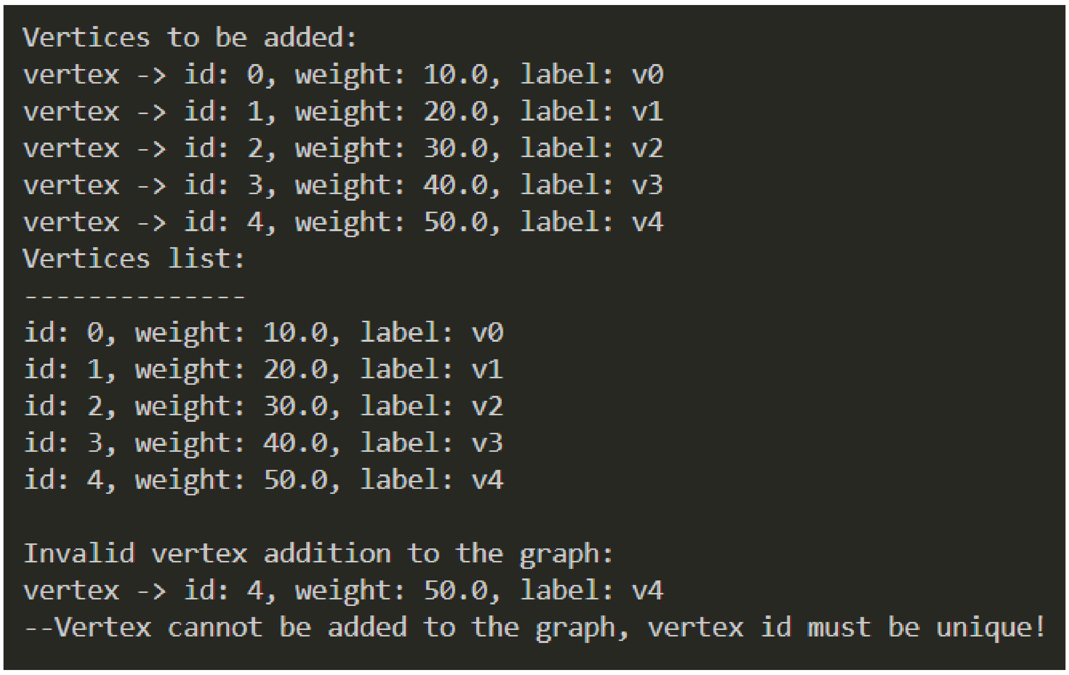
Q1

- Generate a new vertex by given parameters but not add it to the vertices.



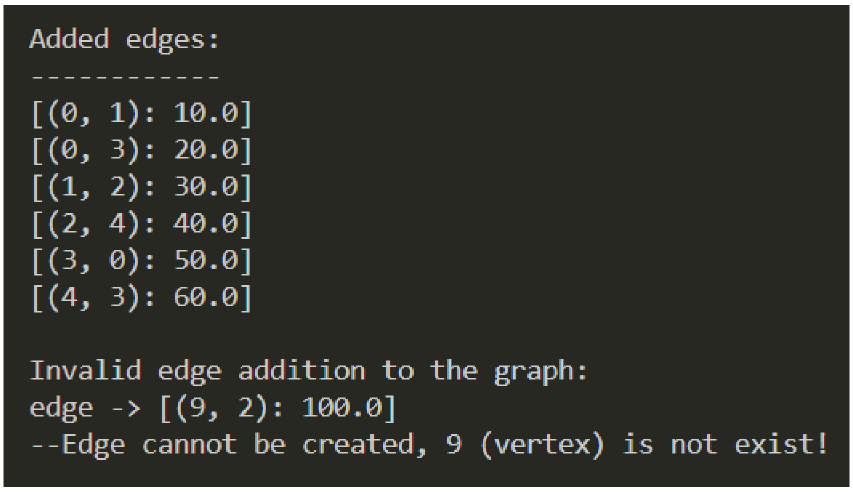
```
Vertices list:
Vertices list is empty.
------
vertex: id: 0, weight: 10.0, label: v0
------
Vertices list:
Vertices list is empty.
--New vertex is created but has not added to the vertices list.
```

- Vertex id check for adding vertex.



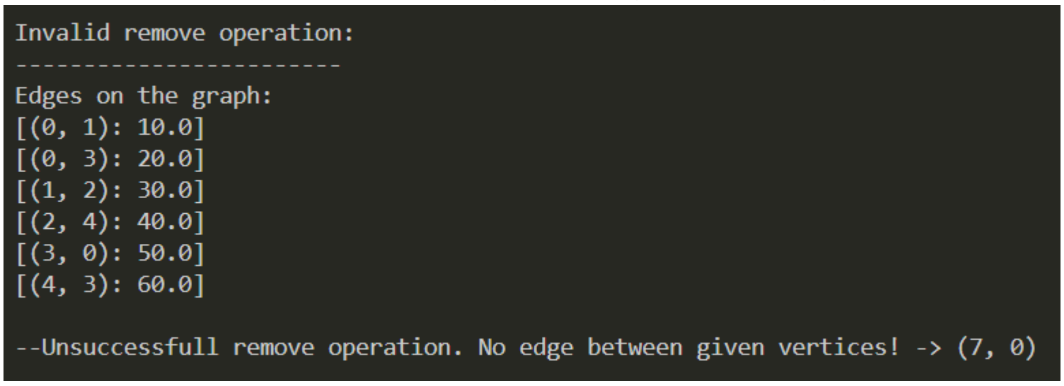
```
Vertices to be added:
vertex -> id: 0, weight: 10.0, label: v0
vertex -> id: 1, weight: 20.0, label: v1
vertex -> id: 2, weight: 30.0, label: v2
vertex -> id: 3, weight: 40.0, label: v3
vertex -> id: 4, weight: 50.0, label: v4
Vertices list:
--------------
id: 0, weight: 10.0, label: v0
id: 1, weight: 20.0, label: v1
id: 2, weight: 30.0, label: v2
id: 3, weight: 40.0, label: v3
id: 4, weight: 50.0, label: v4

Invalid vertex addition to the graph:
vertex -> id: 4, weight: 50.0, label: v4
--Vertex cannot be added to the graph, vertex id must be unique!
```

- Invalid edge addition to the graph (one or both vertex are not exist).



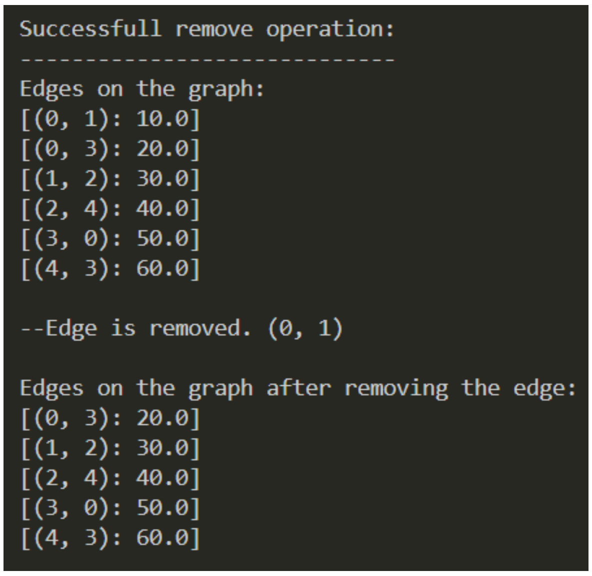
```
Added edges:
------------
[(0, 1): 10.0]
[(0, 3): 20.0]
[(1, 2): 30.0]
[(2, 4): 40.0]
[(3, 0): 50.0]
[(4, 3): 60.0]

Invalid edge addition to the graph:
edge -> [(9, 2): 100.0]
--Edge cannot be created, 9 (vertex) is not exist!
```

- Invalid remove operation for edge (no edge between given vertices).



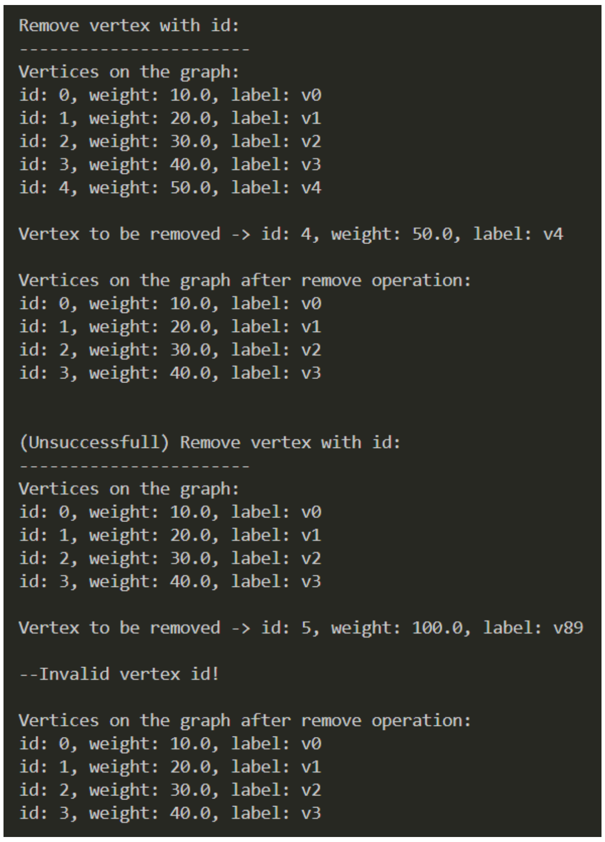
```
Invalid remove operation:
------------------------
Edges on the graph:
[(0, 1): 10.0]
[(0, 3): 20.0]
[(1, 2): 30.0]
[(2, 4): 40.0]
[(3, 0): 50.0]
[(4, 3): 60.0]

--Unsuccessfull remove operation. No edge between given vertices! -> (7, 0)
```

- Removing edge without exception.



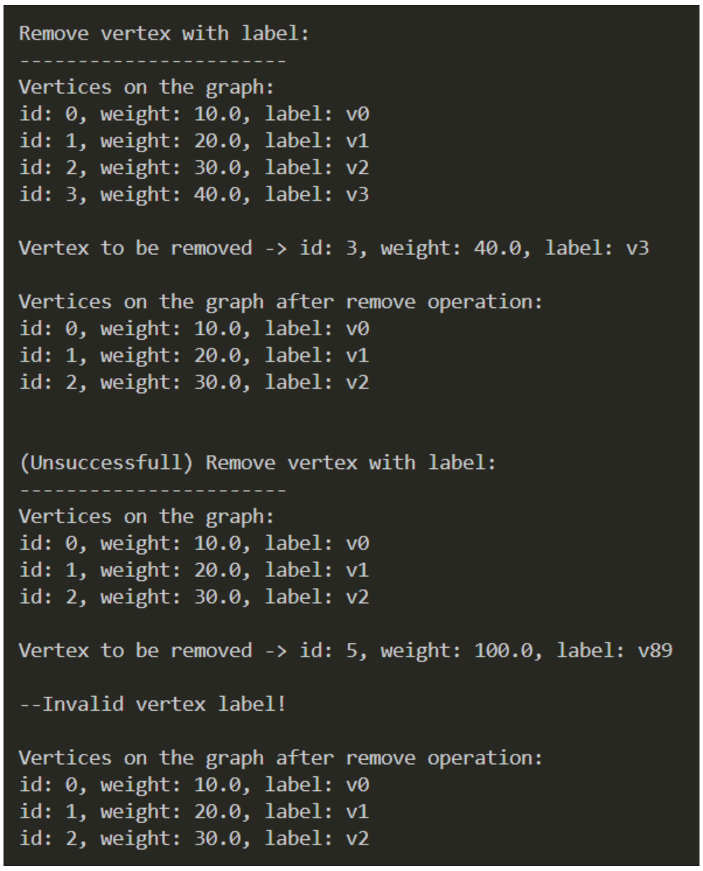
```
Successfull remove operation:
----------------------------
Edges on the graph:
[(0, 1): 10.0]
[(0, 3): 20.0]
[(1, 2): 30.0]
[(2, 4): 40.0]
[(3, 0): 50.0]
[(4, 3): 60.0]

--Edge is removed. (0, 1)

Edges on the graph after removing the edge:
[(0, 3): 20.0]
[(1, 2): 30.0]
[(2, 4): 40.0]
[(3, 0): 50.0]
[(4, 3): 60.0]
```

- Invalid vertex “id” for remove operation (vertex).



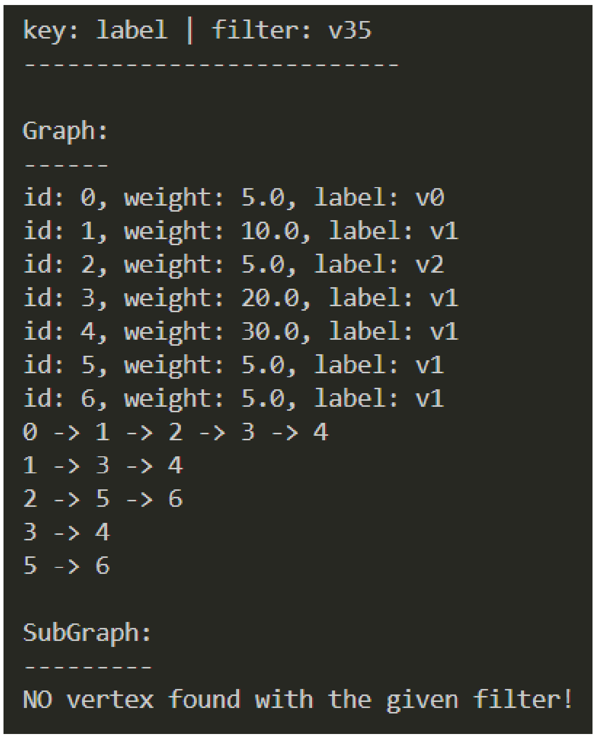
```
Remove vertex with id:
-----------------------
Vertices on the graph:
id: 0, weight: 10.0, label: v0
id: 1, weight: 20.0, label: v1
id: 2, weight: 30.0, label: v2
id: 3, weight: 40.0, label: v3
id: 4, weight: 50.0, label: v4

Vertex to be removed -> id: 4, weight: 50.0, label: v4

Vertices on the graph after remove operation:
id: 0, weight: 10.0, label: v0
id: 1, weight: 20.0, label: v1
id: 2, weight: 30.0, label: v2
id: 3, weight: 40.0, label: v3


(Unsuccessfull) Remove vertex with id:
-----------------------
Vertices on the graph:
id: 0, weight: 10.0, label: v0
id: 1, weight: 20.0, label: v1
id: 2, weight: 30.0, label: v2
id: 3, weight: 40.0, label: v3

Vertex to be removed -> id: 5, weight: 100.0, label: v89

--Invalid vertex id!

Vertices on the graph after remove operation:
id: 0, weight: 10.0, label: v0
id: 1, weight: 20.0, label: v1
id: 2, weight: 30.0, label: v2
id: 3, weight: 40.0, label: v3
```

- Invalid vertex “label” for remove operation (vertex).



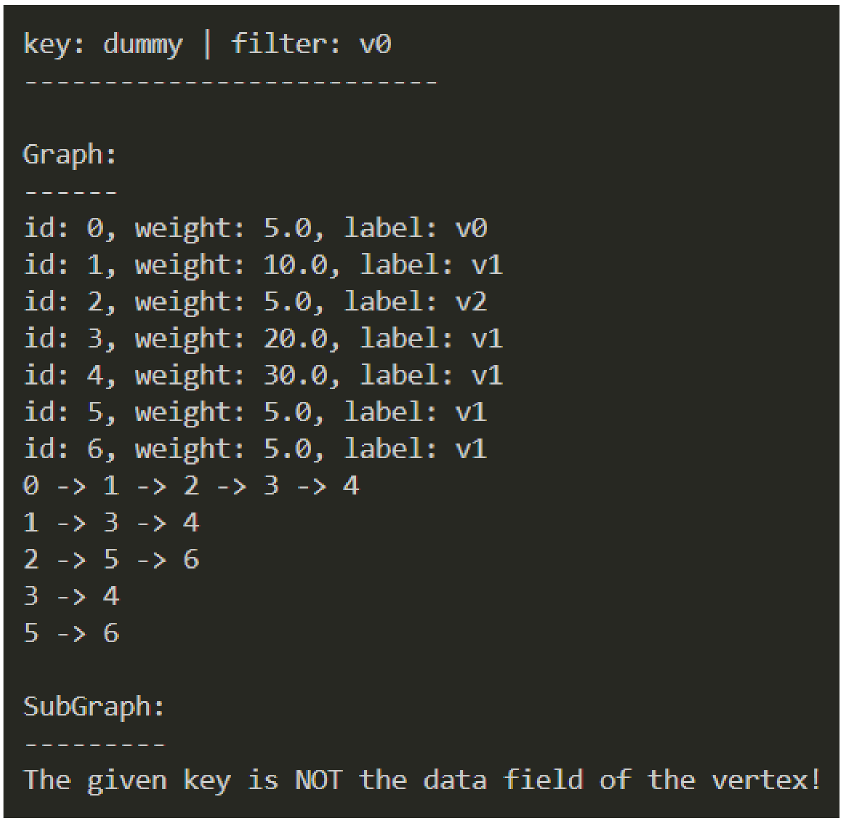
```
Remove vertex with label:
-----------------------
Vertices on the graph:
id: 0, weight: 10.0, label: v0
id: 1, weight: 20.0, label: v1
id: 2, weight: 30.0, label: v2
id: 3, weight: 40.0, label: v3

Vertex to be removed -> id: 3, weight: 40.0, label: v3

Vertices on the graph after remove operation:
id: 0, weight: 10.0, label: v0
id: 1, weight: 20.0, label: v1
id: 2, weight: 30.0, label: v2


(Unsuccessfull) Remove vertex with label:
-----------------------
Vertices on the graph:
id: 0, weight: 10.0, label: v0
id: 1, weight: 20.0, label: v1
id: 2, weight: 30.0, label: v2

Vertex to be removed -> id: 5, weight: 100.0, label: v89

--Invalid vertex label!

Vertices on the graph after remove operation:
id: 0, weight: 10.0, label: v0
id: 1, weight: 20.0, label: v1
id: 2, weight: 30.0, label: v2
```

- No vertex found with the given filter(“v35).



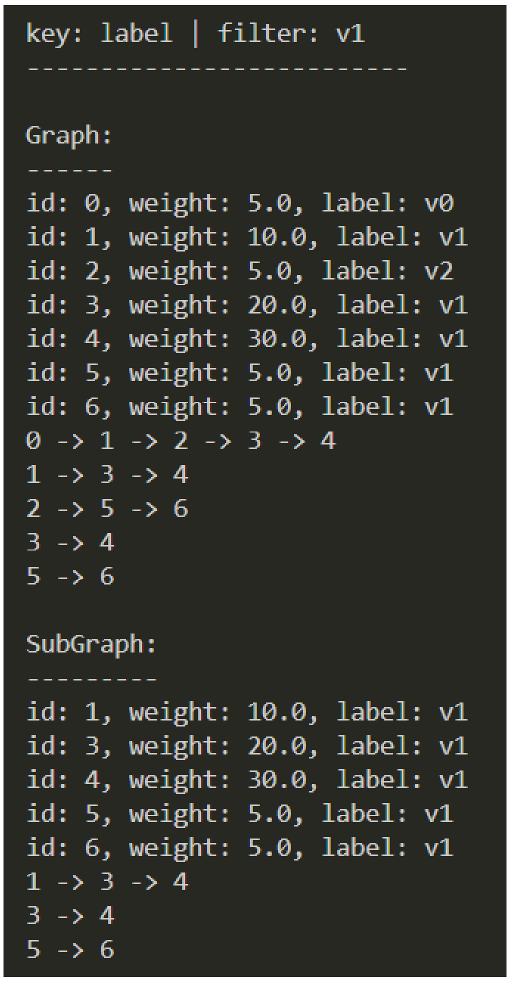
```
key: label | filter: v35
-------------------------

Graph:
------
id: 0, weight: 5.0, label: v0
id: 1, weight: 10.0, label: v1
id: 2, weight: 5.0, label: v2
id: 3, weight: 20.0, label: v1
id: 4, weight: 30.0, label: v1
id: 5, weight: 5.0, label: v1
id: 6, weight: 5.0, label: v1
0 -> 1 -> 2 -> 3 -> 4
1 -> 3 -> 4
2 -> 5 -> 6
3 -> 4
5 -> 6

SubGraph:
---------
NO vertex found with the given filter!
```

- The given key(“dummy”) is not the data field of the vertex.

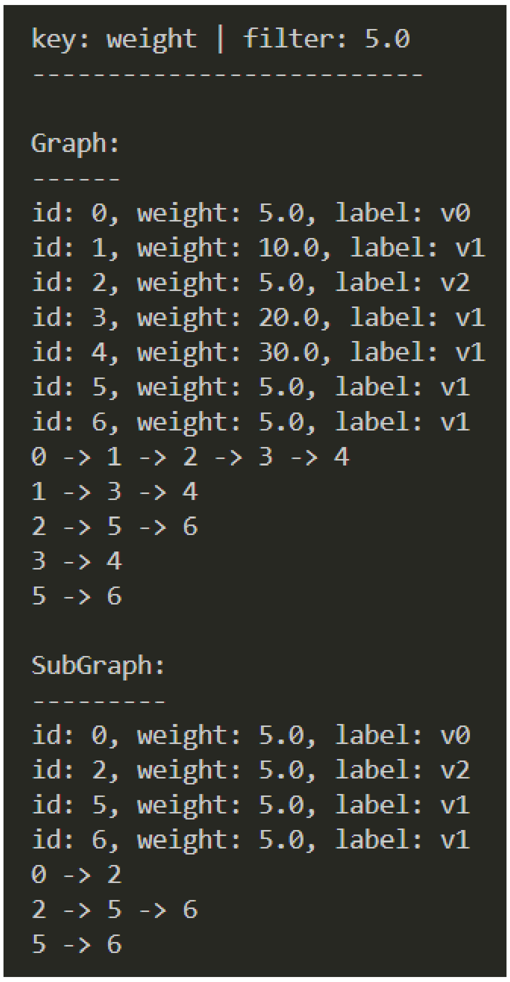


```
key: dummy | filter: v0
--------------------------

Graph:
------
id: 0, weight: 5.0, label: v0
id: 1, weight: 10.0, label: v1
id: 2, weight: 5.0, label: v2
id: 3, weight: 20.0, label: v1
id: 4, weight: 30.0, label: v1
id: 5, weight: 5.0, label: v1
id: 6, weight: 5.0, label: v1
0 -> 1 -> 2 -> 3 -> 4
1 -> 3 -> 4
2 -> 5 -> 6
3 -> 4
5 -> 6

SubGraph:
---------
The given key is NOT the data field of the vertex!
```

- Successfull filterVertices() operations

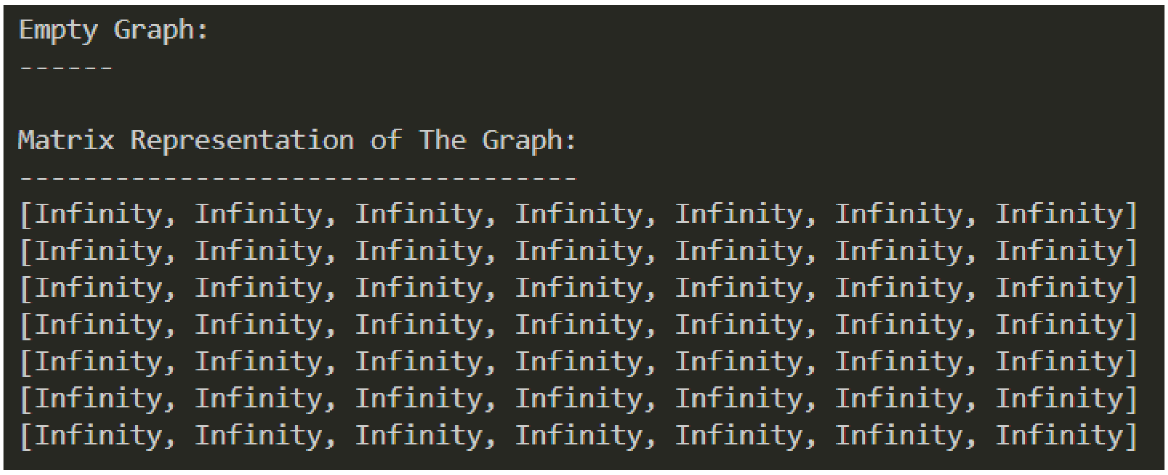


```
key: label | filter: v1
--------------------------

Graph:
------
id: 0, weight: 5.0, label: v0
id: 1, weight: 10.0, label: v1
id: 2, weight: 5.0, label: v2
id: 3, weight: 20.0, label: v1
id: 4, weight: 30.0, label: v1
id: 5, weight: 5.0, label: v1
id: 6, weight: 5.0, label: v1
0 -> 1 -> 2 -> 3 -> 4
1 -> 3 -> 4
2 -> 5 -> 6
3 -> 4
5 -> 6

SubGraph:
---------
id: 1, weight: 10.0, label: v1
id: 3, weight: 20.0, label: v1
id: 4, weight: 30.0, label: v1
id: 5, weight: 5.0, label: v1
id: 6, weight: 5.0, label: v1
1 -> 3 -> 4
3 -> 4
5 -> 6
```



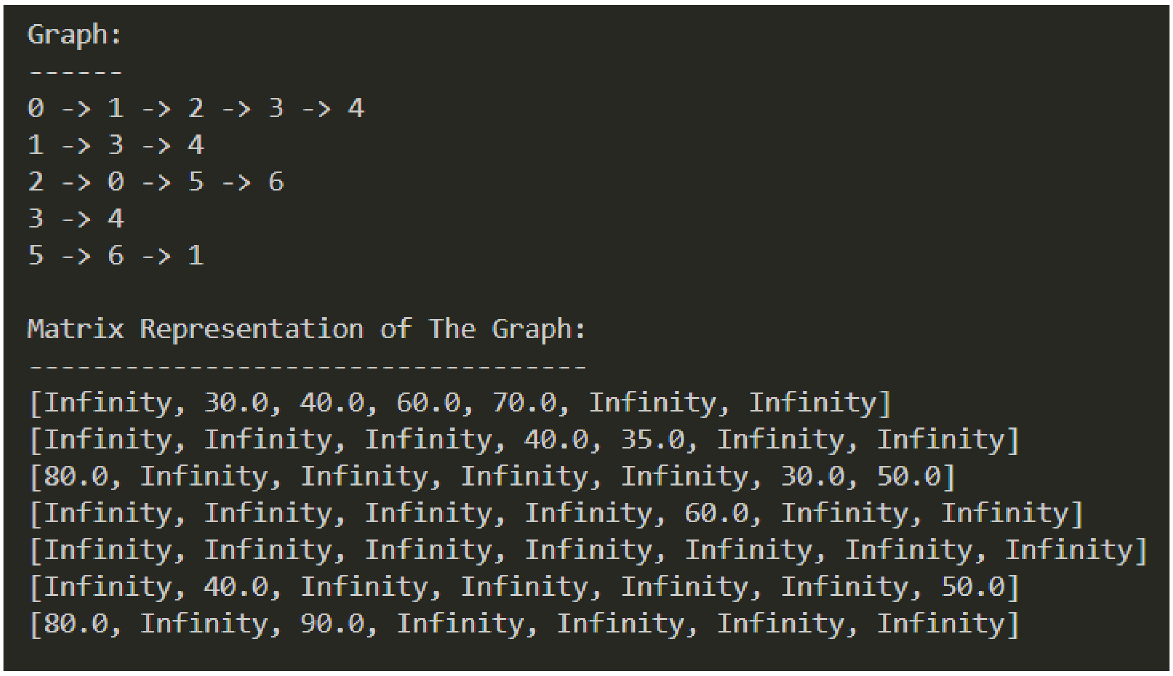
```
key: weight | filter: 5.0
--------------------------

Graph:
------
id: 0, weight: 5.0, label: v0
id: 1, weight: 10.0, label: v1
id: 2, weight: 5.0, label: v2
id: 3, weight: 20.0, label: v1
id: 4, weight: 30.0, label: v1
id: 5, weight: 5.0, label: v1
id: 6, weight: 5.0, label: v1
0 -> 1 -> 2 -> 3 -> 4
1 -> 3 -> 4
2 -> 5 -> 6
3 -> 4
5 -> 6

SubGraph:
---------
id: 0, weight: 5.0, label: v0
id: 2, weight: 5.0, label: v2
id: 5, weight: 5.0, label: v1
id: 6, weight: 5.0, label: v1
0 -> 2
2 -> 5 -> 6
5 -> 6
```

- Empty graph to the matrix graph



```
Empty Graph:
------

Matrix Representation of The Graph:
------------------------------------
[Infinity, Infinity, Infinity, Infinity, Infinity, Infinity, Infinity]
[Infinity, Infinity, Infinity, Infinity, Infinity, Infinity, Infinity]
[Infinity, Infinity, Infinity, Infinity, Infinity, Infinity, Infinity]
[Infinity, Infinity, Infinity, Infinity, Infinity, Infinity, Infinity]
[Infinity, Infinity, Infinity, Infinity, Infinity, Infinity, Infinity]
[Infinity, Infinity, Infinity, Infinity, Infinity, Infinity, Infinity]
[Infinity, Infinity, Infinity, Infinity, Infinity, Infinity, Infinity]
```

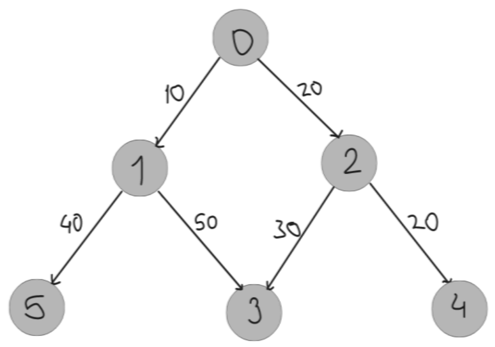
- Filled graph to the matrix graph



```
Graph:
------
0 -> 1 -> 2 -> 3 -> 4
1 -> 3 -> 4
2 -> 0 -> 5 -> 6
3 -> 4
5 -> 6 -> 1

Matrix Representation of The Graph:
-----------------------------------
[Infinity, 30.0, 40.0, 60.0, 70.0, Infinity, Infinity]
[Infinity, Infinity, Infinity, 40.0, 35.0, Infinity, Infinity]
[80.0, Infinity, Infinity, Infinity, Infinity, 30.0, 50.0]
[Infinity, Infinity, Infinity, Infinity, 60.0, Infinity, Infinity]
[Infinity, Infinity, Infinity, Infinity, Infinity, Infinity, Infinity]
[Infinity, 40.0, Infinity, Infinity, Infinity, Infinity, 50.0]
[80.0, Infinity, 90.0, Infinity, Infinity, Infinity, Infinity]
```

Q2

- BFS and DFS Traversals - Example 1



BFS

Queue: ~~1~~, ~~2~~, ~~3~~, ~~5~~, ~~4~~

Identified: 0, 1, 2, 3, 5, 4

10 + 20 + $\boxed{50}$ + 40 + 30 + 20 - 50 = 120

(0,1) (0,2) $\boxed{(1,3)}$ (1,5) 3(2,3) (2,4)

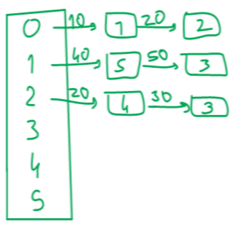
DFS

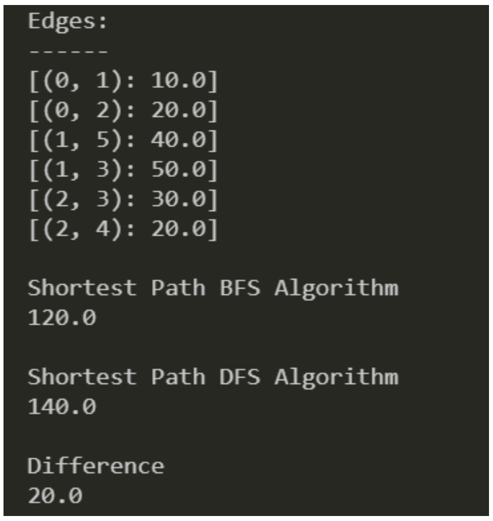
Visit: 0, 1, 5, 3, 2, 4

Finish: 5, 3, 1, 4, 3, 2, 0

10 + 40 + 50 + 20 + 20 = 140

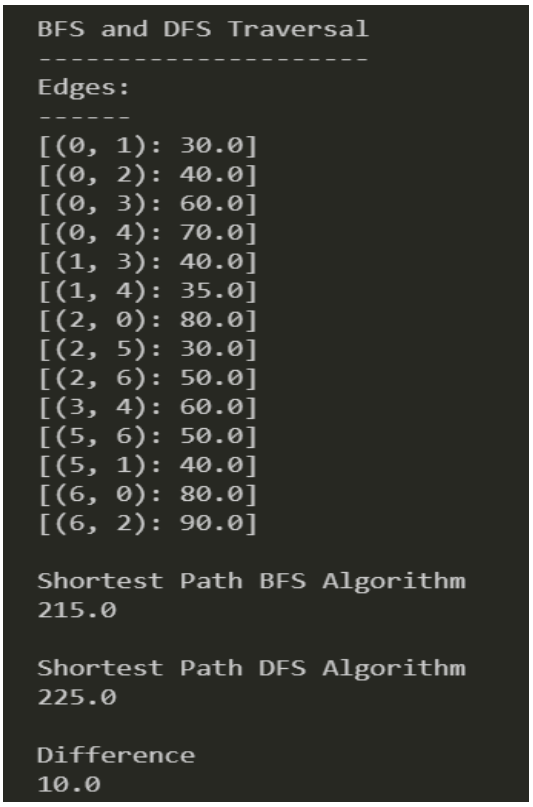
(0,1) (1,5) (1,3) (0,2) (2,4)





```
Edges:
------
[(0, 1): 10.0]
[(0, 2): 20.0]
[(1, 5): 40.0]
[(1, 3): 50.0]
[(2, 3): 30.0]
[(2, 4): 20.0]

Shortest Path BFS Algorithm
120.0

Shortest Path DFS Algorithm
140.0

Difference
20.0
```

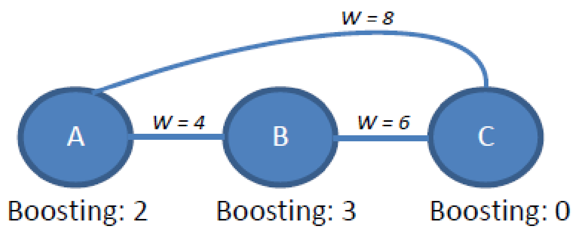
- BFS and DFS Traversals - Example 2

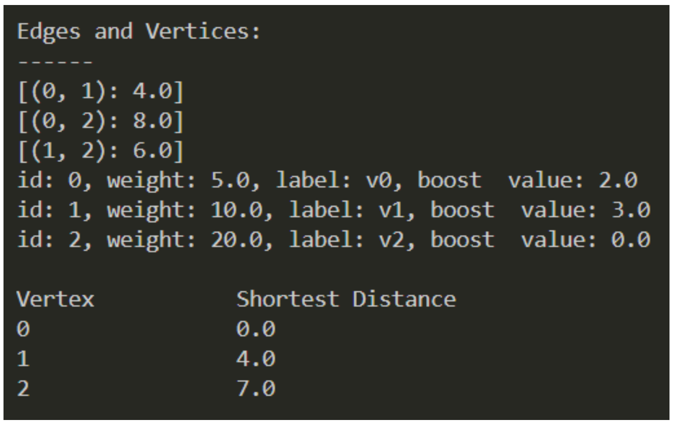


```
BFS and DFS Traversal
---------------------
Edges:
------
[(0, 1): 30.0]
[(0, 2): 40.0]
[(0, 3): 60.0]
[(0, 4): 70.0]
[(1, 3): 40.0]
[(1, 4): 35.0]
[(2, 0): 80.0]
[(2, 5): 30.0]
[(2, 6): 50.0]
[(3, 4): 60.0]
[(5, 6): 50.0]
[(5, 1): 40.0]
[(6, 0): 80.0]
[(6, 2): 90.0]

Shortest Path BFS Algorithm
215.0

Shortest Path DFS Algorithm
225.0

Difference
10.0
```

Q3

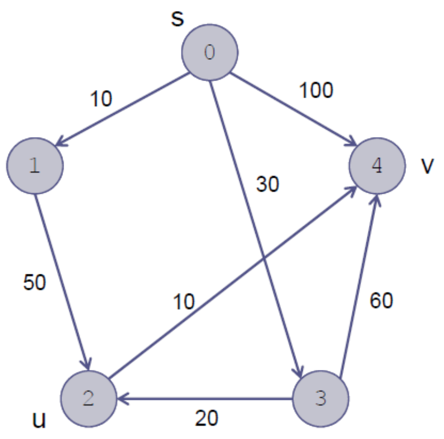
- Basic graph

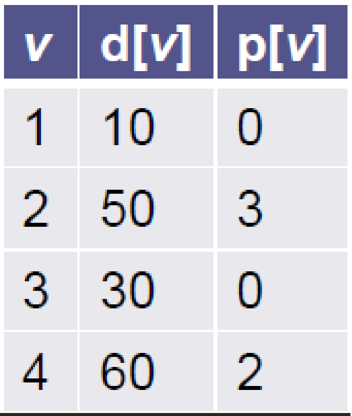




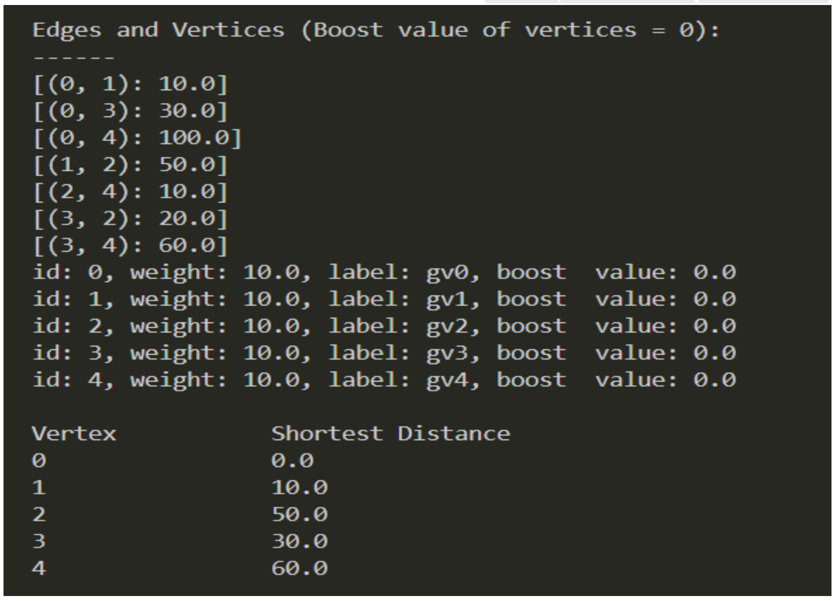
```
Edges and Vertices:
------
[(0, 1): 4.0]
[(0, 2): 8.0]
[(1, 2): 6.0]
id: 0, weight: 5.0, label: v0, boost  value: 2.0
id: 1, weight: 10.0, label: v1, boost  value: 3.0
id: 2, weight: 20.0, label: v2, boost  value: 0.0

Vertex           Shortest Distance
0                0.0
1                4.0
2                7.0
```

- Boost value of vertices = 0



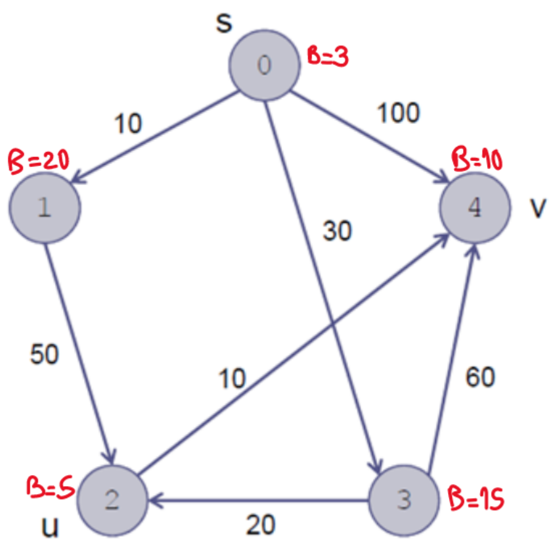


v	d[v]	p[v]
1	10	0
2	50	3
3	30	0
4	60	2



```
Edges and Vertices (Boost value of vertices = 0):
------
[(0, 1): 10.0]
[(0, 3): 30.0]
[(0, 4): 100.0]
[(1, 2): 50.0]
[(2, 4): 10.0]
[(3, 2): 20.0]
[(3, 4): 60.0]
id: 0, weight: 10.0, label: gv0, boost  value: 0.0
id: 1, weight: 10.0, label: gv1, boost  value: 0.0
id: 2, weight: 10.0, label: gv2, boost  value: 0.0
id: 3, weight: 10.0, label: gv3, boost  value: 0.0
id: 4, weight: 10.0, label: gv4, boost  value: 0.0

Vertex           Shortest Distance
0                0.0
1                10.0
2                50.0
3                30.0
4                60.0
```

- Non-zero boost values with the same graph



v2

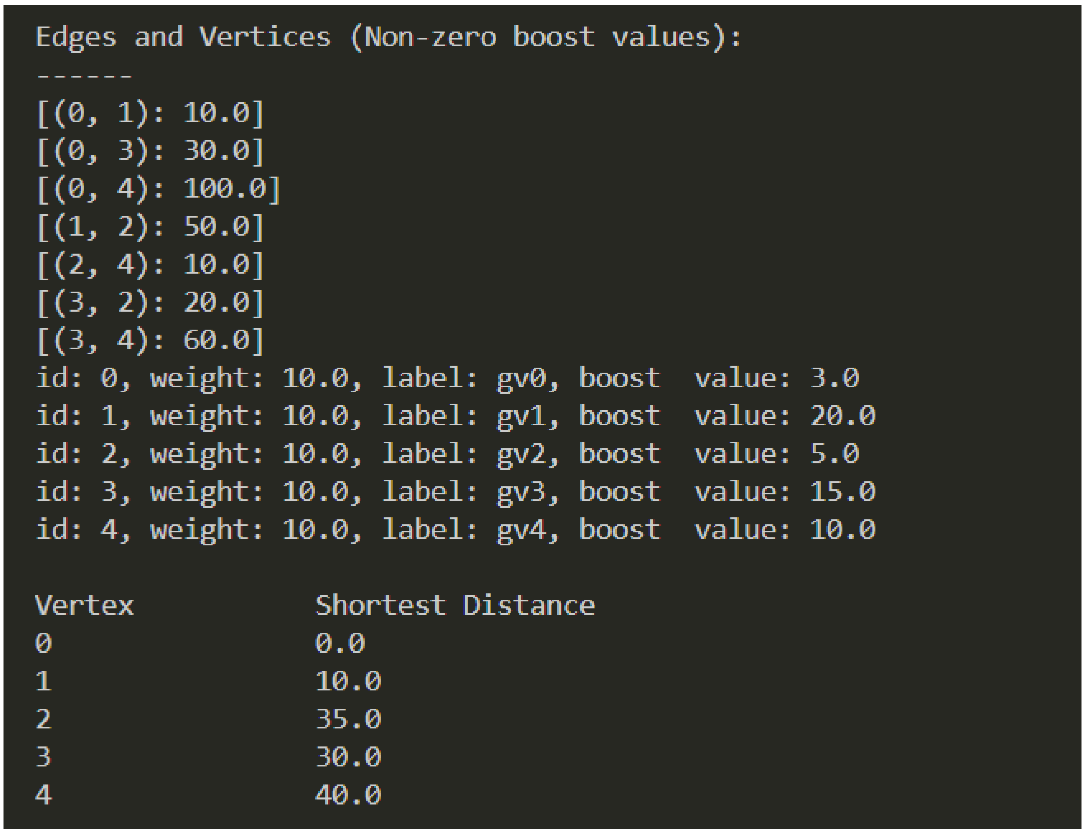
$$0 \xrightarrow{30} \underbrace{3}_{B=15} \xrightarrow{20} 2$$

$$30 + 20 - 15 = \underline{\underline{35}}$$

v4

$$0 \xrightarrow{30} \underbrace{3}_{B=15} \xrightarrow{20} \underbrace{2}_{B=5} \xrightarrow{10} 4$$

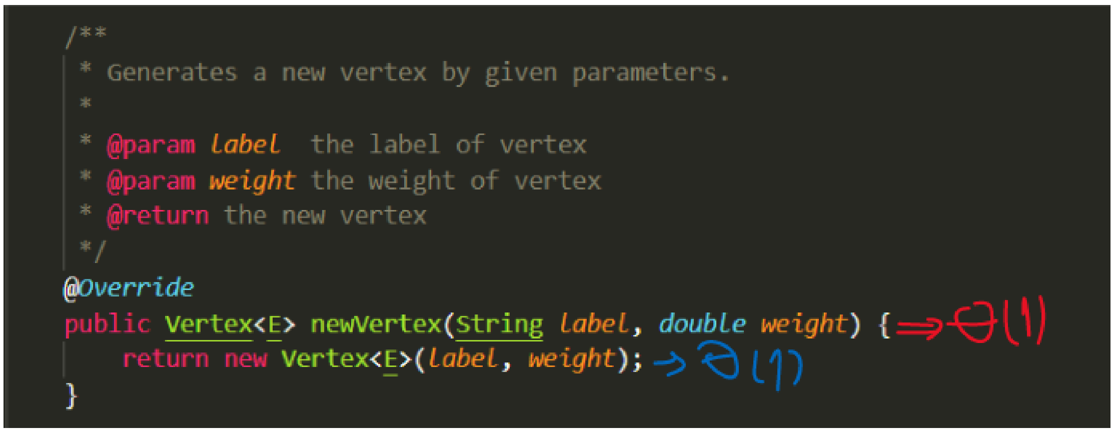
$$30 - 15 + 20 - 5 + 10 = \underline{\underline{40}}$$



```
Edges and Vertices (Non-zero boost values):
------
[(0, 1): 10.0]
[(0, 3): 30.0]
[(0, 4): 100.0]
[(1, 2): 50.0]
[(2, 4): 10.0]
[(3, 2): 20.0]
[(3, 4): 60.0]
id: 0, weight: 10.0, label: gv0, boost  value: 3.0
id: 1, weight: 10.0, label: gv1, boost  value: 20.0
id: 2, weight: 10.0, label: gv2, boost  value: 5.0
id: 3, weight: 10.0, label: gv3, boost  value: 15.0
id: 4, weight: 10.0, label: gv4, boost  value: 10.0

Vertex          Shortest Distance
0               0.0
1               10.0
2               35.0
3               30.0
4               40.0
```

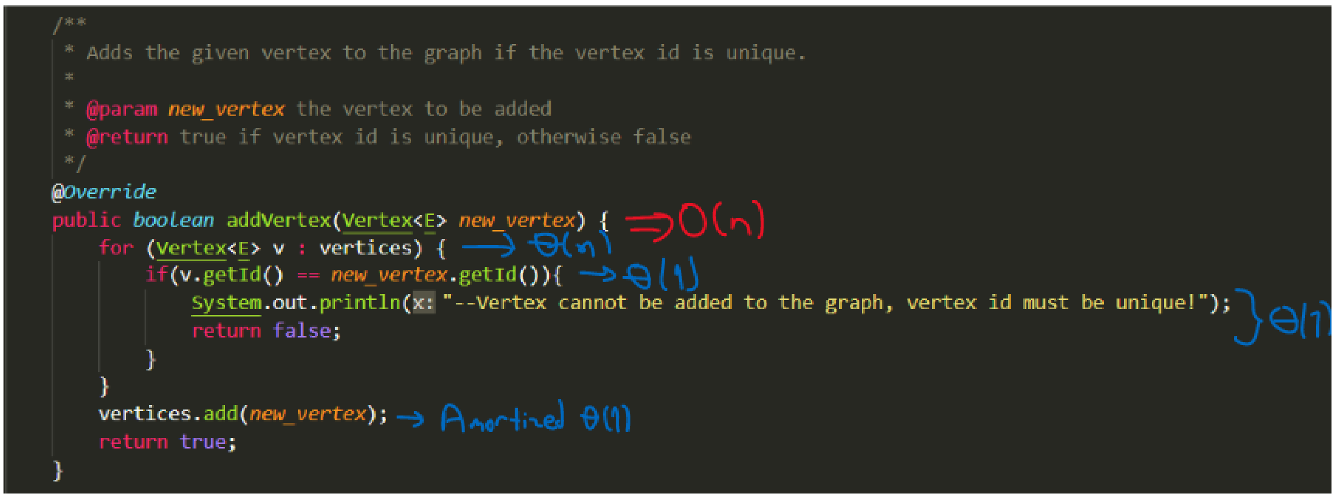
Time Complexities of DynamicGraph Interface Functions

- **newVertex (string label, double weight):**



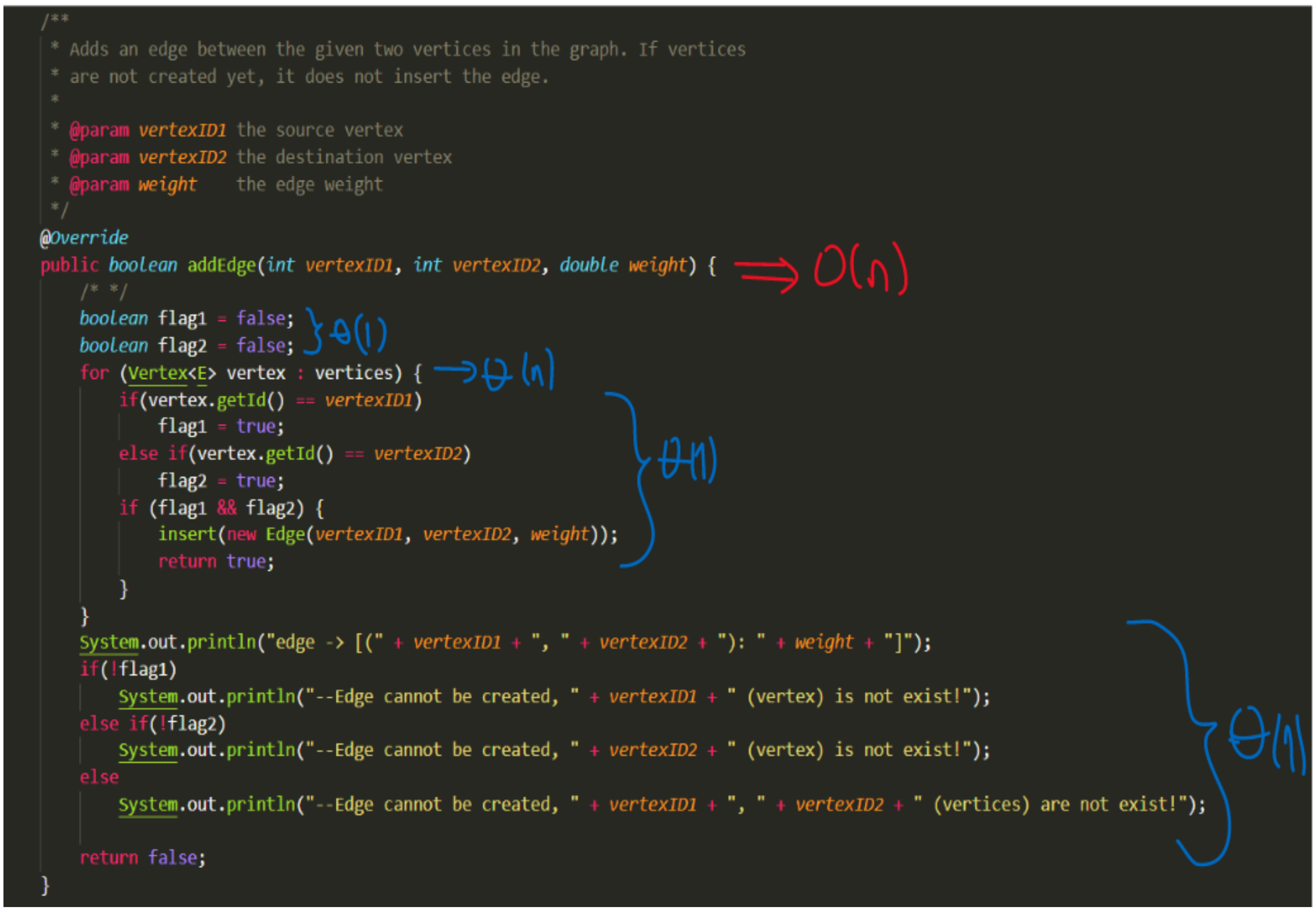
```
/**
 * Generates a new vertex by given parameters.
 *
 * @param label  the label of vertex
 * @param weight the weight of vertex
 * @return the new vertex
 */
@Override
public Vertex<E> newVertex(String label, double weight) {⇒ Θ(1)
    return new Vertex<E>(label, weight); → Θ(1)
}
```

- **addVertex (Vertex new_vertex):**



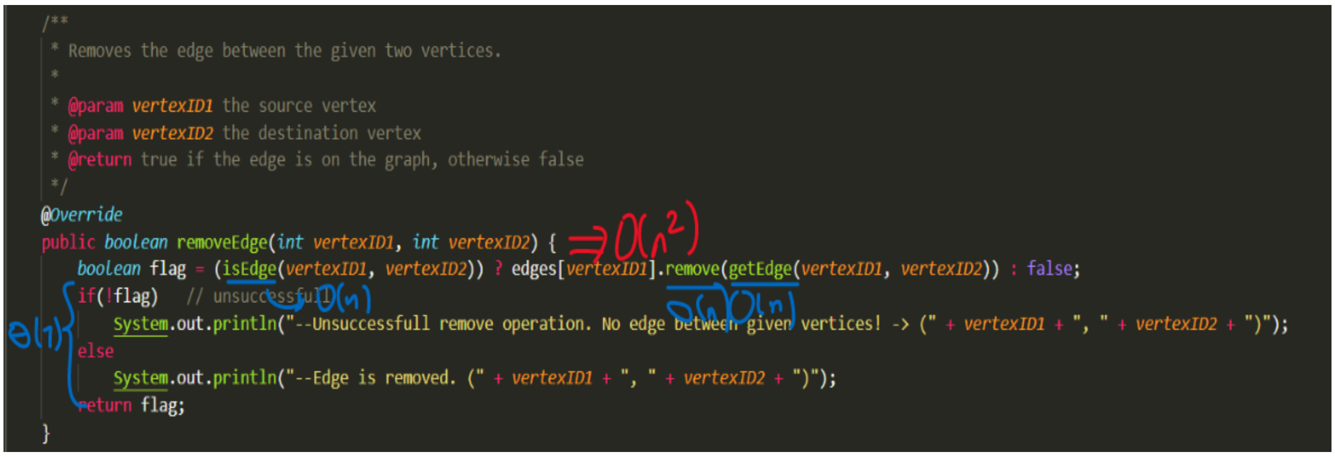
```
/**
 * Adds the given vertex to the graph if the vertex id is unique.
 *
 * @param new_vertex the vertex to be added
 * @return true if vertex id is unique, otherwise false
 */
@Override
public boolean addVertex(Vertex<E> new_vertex) { ⇒ O(n)
    for (Vertex<E> v : vertices) { ⟶ Θ(n)
        if(v.getId() == new_vertex.getId()){ → Θ(1)
            System.out.println(x: "--Vertex cannot be added to the graph, vertex id must be unique!"); } Θ(1
            return false;
        }
    }
    vertices.add(new_vertex); → Amortized Θ(1)
    return true;
}
```

- **addEdge (int vertexID1, int vertexID2, double weight):**



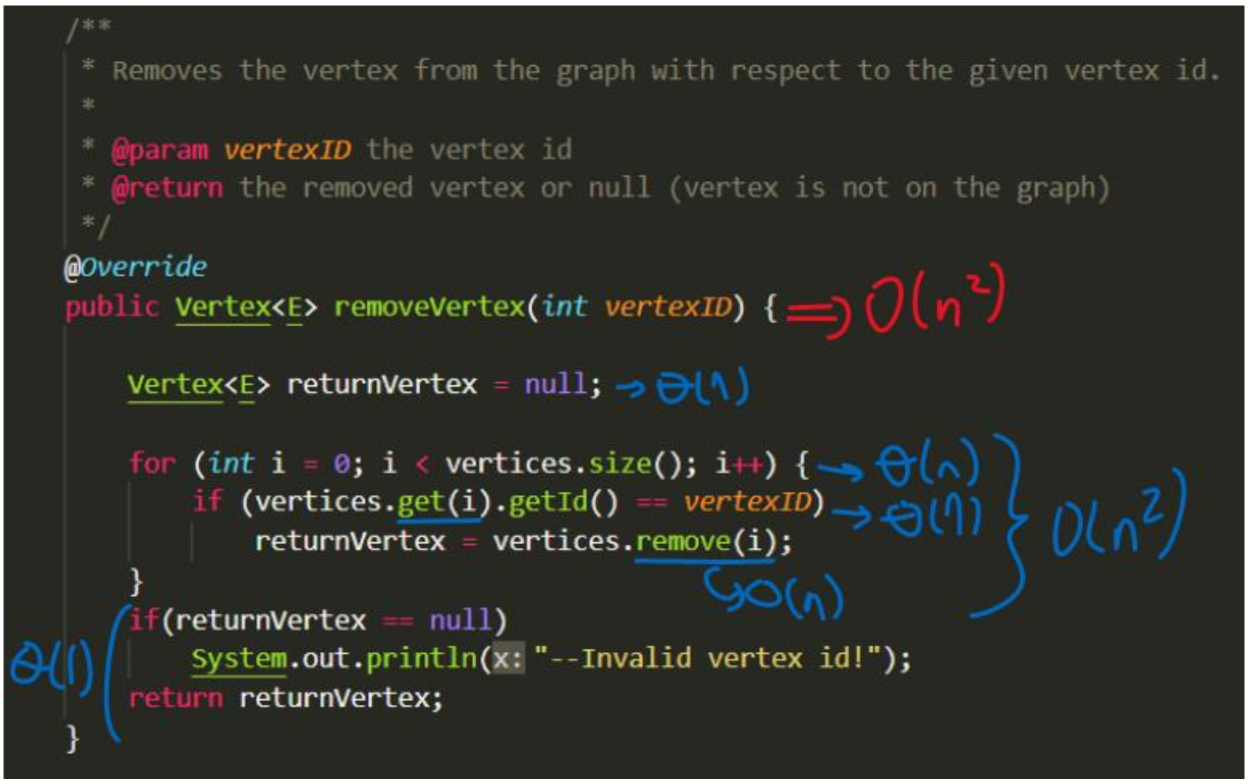
```java
/**
 * Adds an edge between the given two vertices in the graph. If vertices
 * are not created yet, it does not insert the edge.
 *
 * @param vertexID1 the source vertex
 * @param vertexID2 the destination vertex
 * @param weight    the edge weight
 */
@Override
public boolean addEdge(int vertexID1, int vertexID2, double weight) {
    /* */
    boolean flag1 = false;
    boolean flag2 = false;
    for (Vertex<E> vertex : vertices) {
        if(vertex.getId() == vertexID1)
            flag1 = true;
        else if(vertex.getId() == vertexID2)
            flag2 = true;
        if (flag1 && flag2) {
            insert(new Edge(vertexID1, vertexID2, weight));
            return true;
        }
    }
    System.out.println("edge -> [(" + vertexID1 + ", " + vertexID2 + "): " + weight + "]");
    if(!flag1)
        System.out.println("--Edge cannot be created, " + vertexID1 + " (vertex) is not exist!");
    else if(!flag2)
        System.out.println("--Edge cannot be created, " + vertexID2 + " (vertex) is not exist!");
    else
        System.out.println("--Edge cannot be created, " + vertexID1 + ", " + vertexID2 + " (vertices) are not exist!");

    return false;
}
```

- **removeEdge (int vertexID1, int vertexID2):**



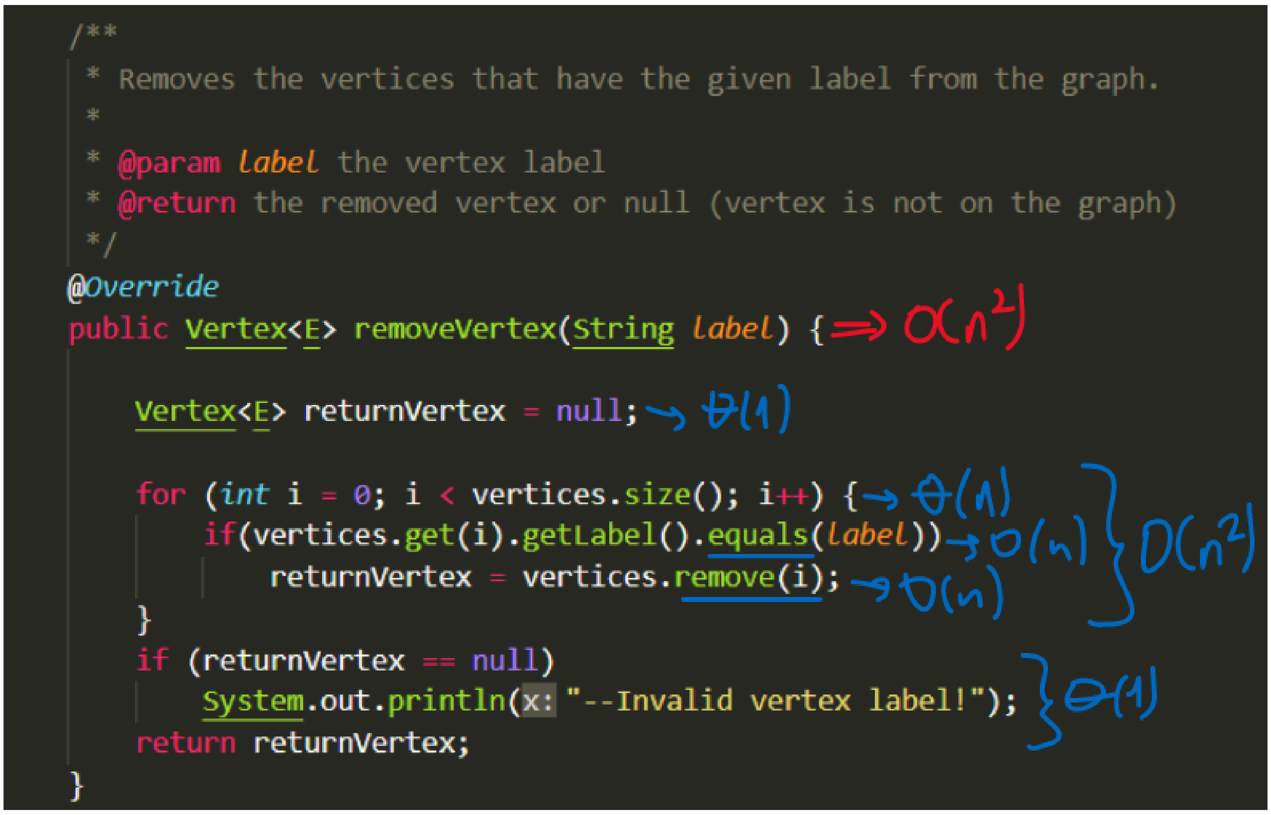
```java
/**
 * Removes the edge between the given two vertices.
 *
 * @param vertexID1 the source vertex
 * @param vertexID2 the destination vertex
 * @return true if the edge is on the graph, otherwise false
 */
@Override
public boolean removeEdge(int vertexID1, int vertexID2) {
    boolean flag = (isEdge(vertexID1, vertexID2)) ? edges[vertexID1].remove(getEdge(vertexID1, vertexID2)) : false;
    if(!flag)   // unsuccessfull
        System.out.println("--Unsuccessfull remove operation. No edge between given vertices! -> (" + vertexID1 + ", " + vertexID2 + ")");
    else
        System.out.println("--Edge is removed. (" + vertexID1 + ", " + vertexID2 + ")");
    return flag;
}
```

- **removeVertex (int vertexID):**



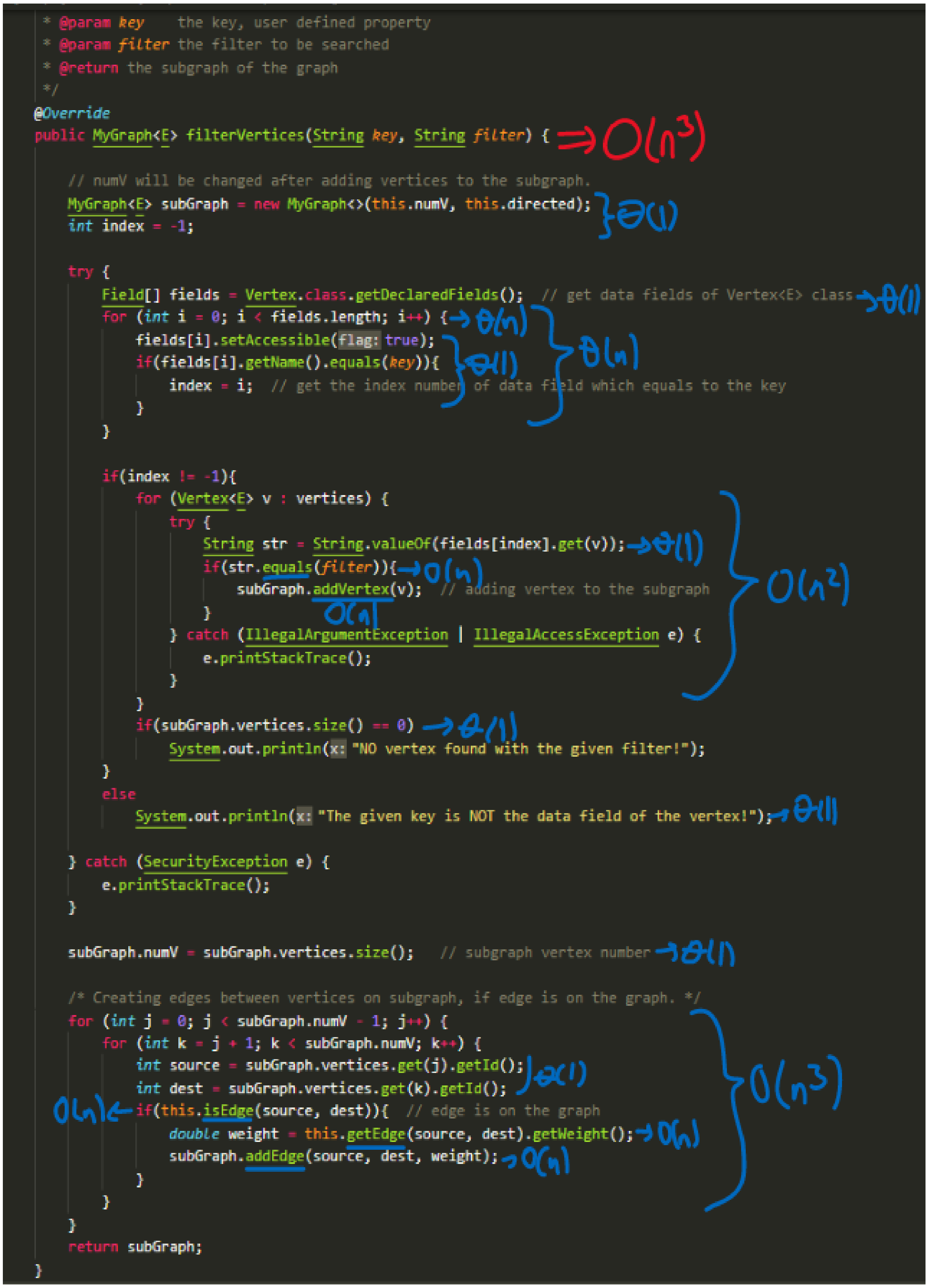
```
/**
 * Removes the vertex from the graph with respect to the given vertex id.
 *
 * @param vertexID the vertex id
 * @return the removed vertex or null (vertex is not on the graph)
 */
@Override
public Vertex<E> removeVertex(int vertexID) { => O(n^2)

    Vertex<E> returnVertex = null; -> Θ(1)

    for (int i = 0; i < vertices.size(); i++) { -> Θ(n)  }
        if (vertices.get(i).getId() == vertexID) -> Θ(1)  } O(n^2)
            returnVertex = vertices.remove(i);           }
                                   ↳ O(n)
    }
      ( if(returnVertex == null)
Θ(1)  (     System.out.println(x: "--Invalid vertex id!");
      ( return returnVertex;
}
```

- **removeVertex (String label):**



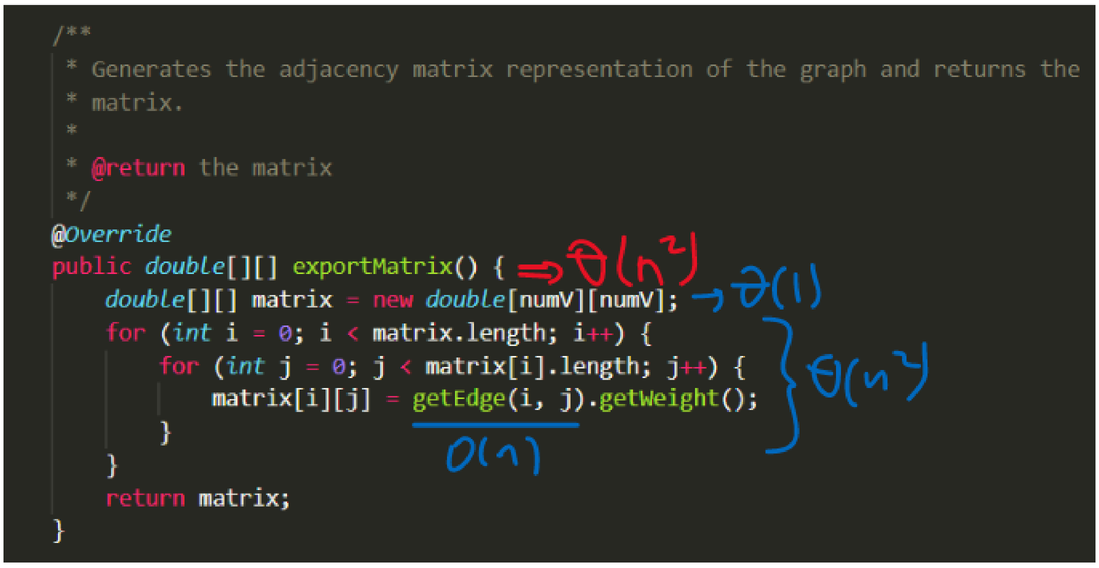
```
/**
 * Removes the vertices that have the given label from the graph.
 *
 * @param label the vertex label
 * @return the removed vertex or null (vertex is not on the graph)
 */
@Override
public Vertex<E> removeVertex(String label) { => O(n^2)

    Vertex<E> returnVertex = null; -> Θ(1)

    for (int i = 0; i < vertices.size(); i++) { -> Θ(n)                    }
        if(vertices.get(i).getLabel().equals(label)) -> O(n)              } O(n^2)
            returnVertex = vertices.remove(i); -> O(n)                     }
    }
    if (returnVertex == null)
        System.out.println(x: "--Invalid vertex label!"); } Θ(1)
    return returnVertex;
}
```

- **filterVertices (string key, string filter):**



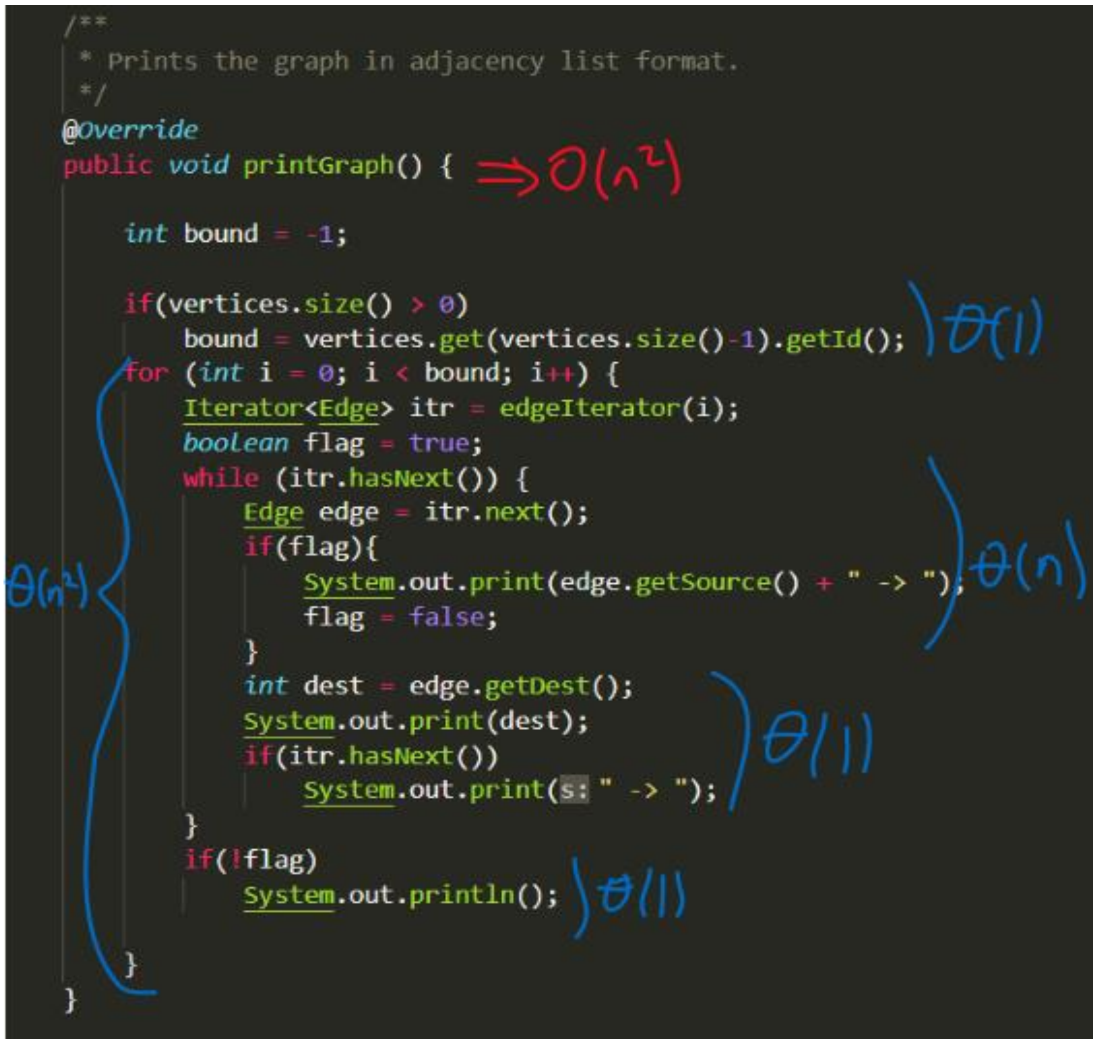
```
     * @param key    the key, user defined property
     * @param filter the filter to be searched
     * @return the subgraph of the graph
     */
    @Override
    public MyGraph<E> filterVertices(String key, String filter) {

        // numV will be changed after adding vertices to the subgraph.
        MyGraph<E> subGraph = new MyGraph<>(this.numV, this.directed);
        int index = -1;

        try {
            Field[] fields = Vertex.class.getDeclaredFields();  // get data fields of Vertex<E> class
            for (int i = 0; i < fields.length; i++) {
                fields[i].setAccessible(flag: true);
                if(fields[i].getName().equals(key)){
                    index = i;  // get the index number of data field which equals to the key
                }
            }

            if(index != -1){
                for (Vertex<E> v : vertices) {
                    try {
                        String str = String.valueOf(fields[index].get(v));
                        if(str.equals(filter)){
                            subGraph.addVertex(v);  // adding vertex to the subgraph
                        }
                    } catch (IllegalArgumentException | IllegalAccessException e) {
                        e.printStackTrace();
                    }
                }
                if(subGraph.vertices.size() == 0)
                    System.out.println(x: "NO vertex found with the given filter!");
            }
            else
                System.out.println(x: "The given key is NOT the data field of the vertex!");

        } catch (SecurityException e) {
            e.printStackTrace();
        }

        subGraph.numV = subGraph.vertices.size();   // subgraph vertex number

        /* Creating edges between vertices on subgraph, if edge is on the graph. */
        for (int j = 0; j < subGraph.numV - 1; j++) {
            for (int k = j + 1; k < subGraph.numV; k++) {
                int source = subGraph.vertices.get(j).getId();
                int dest = subGraph.vertices.get(k).getId();
                if(this.isEdge(source, dest)){  // edge is on the graph
                    double weight = this.getEdge(source, dest).getWeight();
                    subGraph.addEdge(source, dest, weight);
                }
            }
        }
        return subGraph;
    }
```

- **exportMatrix():**



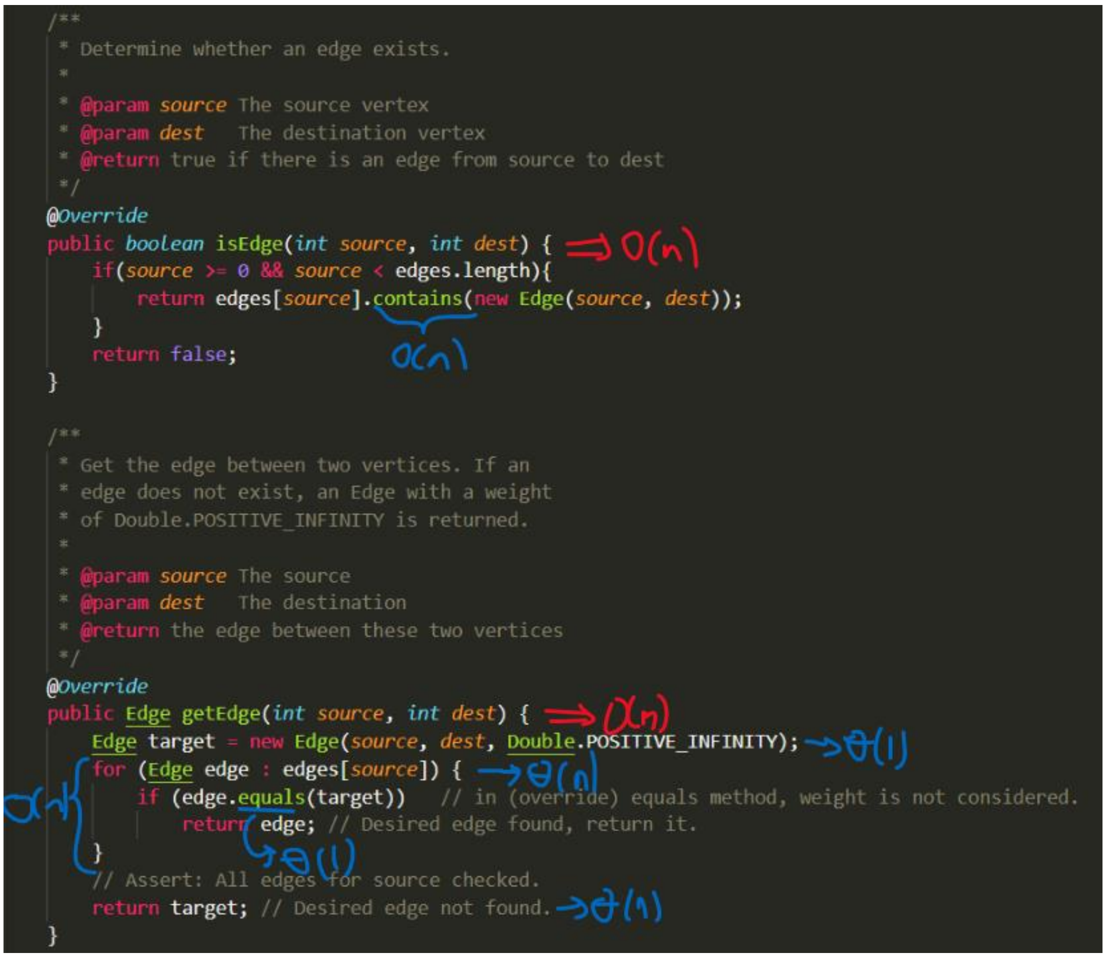
```java
/**
 * Generates the adjacency matrix representation of the graph and returns the
 * matrix.
 *
 * @return the matrix
 */
@Override
public double[][] exportMatrix() {
    double[][] matrix = new double[numV][numV];
    for (int i = 0; i < matrix.length; i++) {
        for (int j = 0; j < matrix[i].length; j++) {
            matrix[i][j] = getEdge(i, j).getWeight();
        }
    }
    return matrix;
}
```

- **printGraph():**



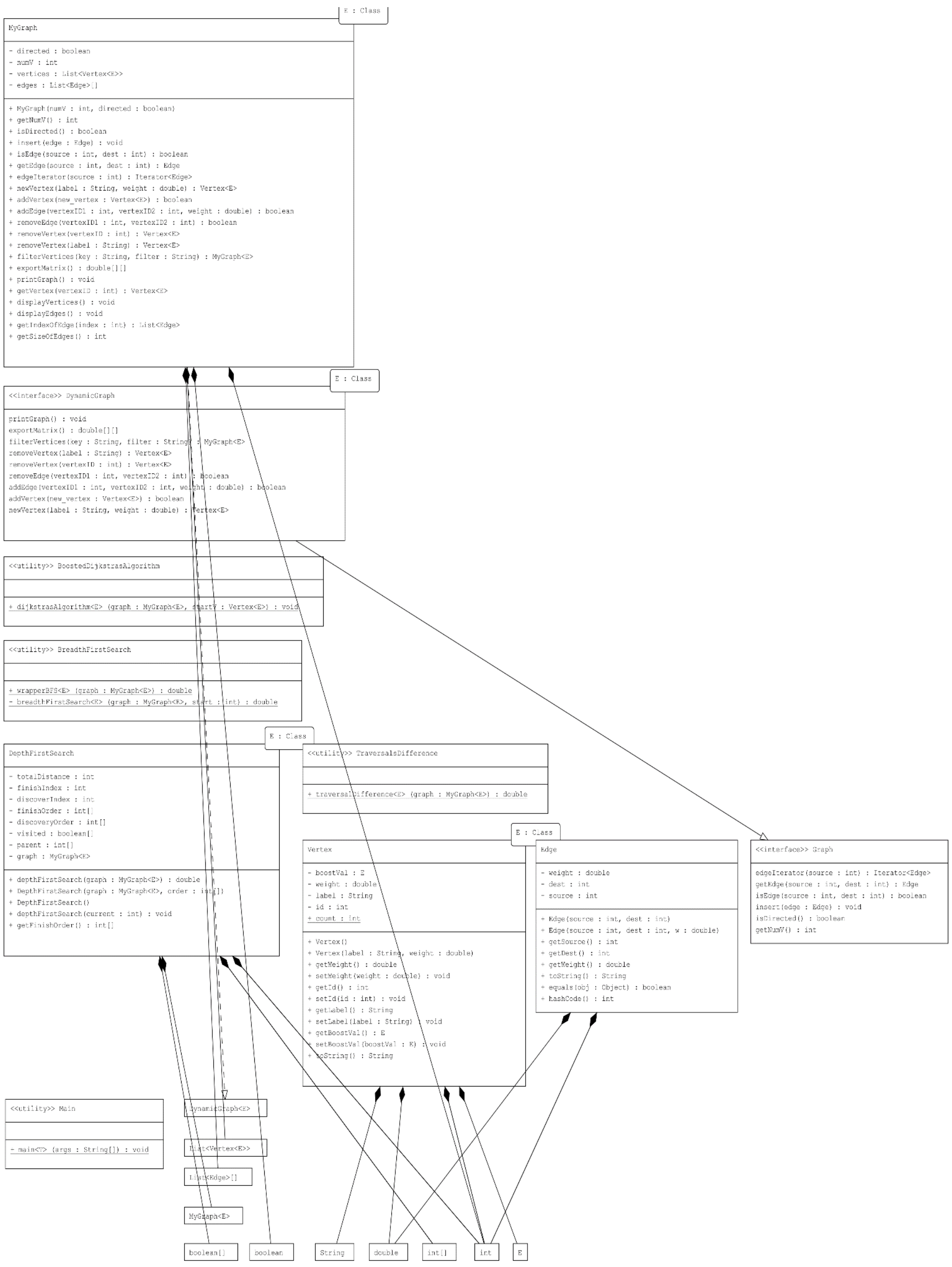
```java
/**
 * Prints the graph in adjacency list format.
 */
@Override
public void printGraph() {

    int bound = -1;

    if(vertices.size() > 0)
        bound = vertices.get(vertices.size()-1).getId();
    for (int i = 0; i < bound; i++) {
        Iterator<Edge> itr = edgeIterator(i);
        boolean flag = true;
        while (itr.hasNext()) {
            Edge edge = itr.next();
            if(flag){
                System.out.print(edge.getSource() + " -> ");
                flag = false;
            }
            int dest = edge.getDest();
            System.out.print(dest);
            if(itr.hasNext())
                System.out.print(s: " -> ");
        }
        if(!flag)
            System.out.println();

    }
}
```

- **Helper functions:**



```java
/**
 * Determine whether an edge exists.
 *
 * @param source The source vertex
 * @param dest   The destination vertex
 * @return true if there is an edge from source to dest
 */
@Override
public boolean isEdge(int source, int dest) {
    if(source >= 0 && source < edges.length){
        return edges[source].contains(new Edge(source, dest));
    }
    return false;
}

/**
 * Get the edge between two vertices. If an
 * edge does not exist, an Edge with a weight
 * of Double.POSITIVE_INFINITY is returned.
 *
 * @param source The source
 * @param dest   The destination
 * @return the edge between these two vertices
 */
@Override
public Edge getEdge(int source, int dest) {
    Edge target = new Edge(source, dest, Double.POSITIVE_INFINITY);
    for (Edge edge : edges[source]) {
        if (edge.equals(target))    // in (override) equals method, weight is not considered.
            return edge; // Desired edge found, return it.
    }
    // Assert: All edges for source checked.
    return target; // Desired edge not found.
}
```

- CLASS DIAGRAM


E : Class
MyGraph
- directed : boolean
- numV : int
- vertices : List<Vertex<E>>
- edges : List<Edge>[]
+ MyGraph(numV : int, directed : boolean)
+ getNumV() : int
+ isDirected() : boolean
+ insert(edge : Edge) : void
+ isEdge(source : int, dest : int) : boolean
+ getEdge(source : int, dest : int) : Edge
+ edgeIterator(source : int) : Iterator<Edge>
+ newVertex(label : String, weight : double) : Vertex<E>
+ addVertex(new_vertex : Vertex<E>) : boolean
+ addEdge(vertexID1 : int, vertexID2 : int, weight : double) : boolean
+ removeEdge(vertexID1 : int, vertexID2 : int) : boolean
+ removeVertex(vertexID : int) : Vertex<E>
+ removeVertex(label : String) : Vertex<E>
+ filterVertices(key : String, filter : String) : MyGraph<E>
+ exportMatrix() : double[][]
+ printGraph() : void
+ getVertex(vertexID : int) : Vertex<E>
+ displayVertices() : void
+ displayEdges() : void
+ getIndexOfEdge(index : int) : List<Edge>
+ getSizeOfEdges() : int
E : Class
<<interface>> DynamicGraph
printGraph() : void
exportMatrix() : double[][]
filterVertices(key : String, filter : String) : MyGraph<E>
removeVertex(label : String) : Vertex<E>
removeVertex(vertexID : int) : Vertex<E>
removeEdge(vertexID1 : int, vertexID2 : int) : boolean
addEdge(vertexID1 : int, vertexID2 : int, weight : double) : boolean
addVertex(new_vertex : Vertex<E>) : boolean
newVertex(label : String, weight : double) : Vertex<E>
<<utility>> BoostedDijkstrasAlgorithm
+ dijkstrasAlgorithm<E> (graph : MyGraph<E>, startV : Vertex<E>) : void
<<utility>> BreadthFirstSearch
+ wrapperBFS<E> (graph : MyGraph<E>) : double
- breadthFirstSearch<E> (graph : MyGraph<E>, start : int) : double
E : Class
DepthFirstSearch
- totalDistance : int
- finishIndex : int
- discoverIndex : int
- finishOrder : int[]
- discoveryOrder : int[]
- visited : boolean[]
- parent : int[]
- graph : MyGraph<E>
+ depthFirstSearch(graph : MyGraph<E>) : double
+ DepthFirstSearch(graph : MyGraph<E>, order : int[])
+ DepthFirstSearch()
+ depthFirstSearch(current : int) : void
+ getFinishOrder() : int[]
<<utility>> TraversalsDifference
+ traversalDifference<E> (graph : MyGraph<E>) : double
E : Class
Vertex
- boostVal : E
- weight : double
- label : String
- id : int
+ count : int
+ Vertex()
+ Vertex(label : String, weight : double)
+ getWeight() : double
+ setWeight(weight : double) : void
+ getId() : int
+ setId(id : int) : void
+ getLabel() : String
+ setLabel(label : String) : void
+ getBoostVal() : E
+ setBoostVal(boostVal : E) : void
+ toString() : String
Edge
- weight : double
- dest : int
- source : int
+ Edge(source : int, dest : int)
+ Edge(source : int, dest : int, w : double)
+ getSource() : int
+ getDest() : int
+ getWeight() : double
+ toString() : String
+ equals(obj : Object) : boolean
+ hashCode() : int
<<interface>> Graph
edgeIterator(source : int) : Iterator<Edge>
getEdge(source : int, dest : int) : Edge
isEdge(source : int, dest : int) : boolean
insert(edge : Edge) : void
isDirected() : boolean
getNumV() : int
<<utility>> Main
+ main<T> (args : String[]) : void
DynamicGraph<E>
List<Vertex<E>>
List<Edge>[]
MyGraph<E>
boolean[]
boolean
String
double
int[]
int
E