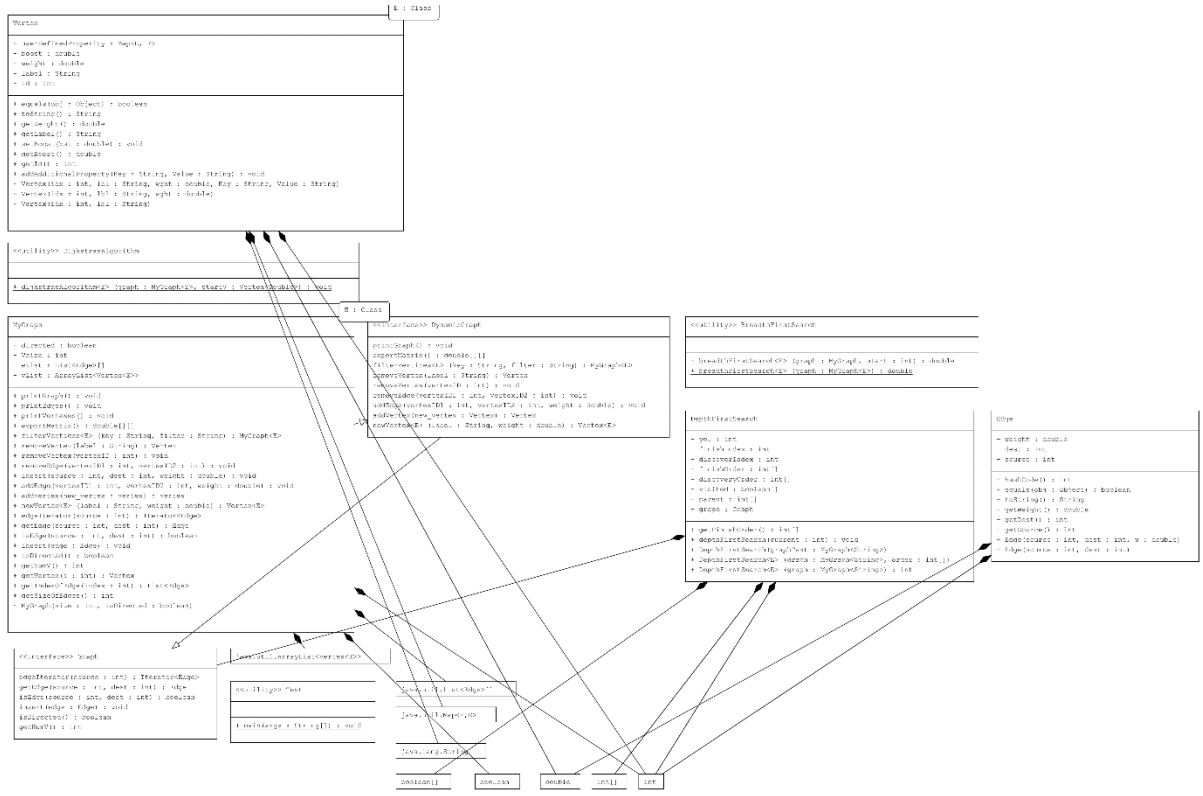
System Requirements:

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

-User must run makefile folder

Class Diagram:



Note: Class diagram is also available in the project folder as png

Q1)

Problem Solution approach:

I used listgraph implementation for edges and wrote my class -Vertex class- for representing vertexes.

Vertex class has id, label, weight, a boost value and userDefinedProperities.

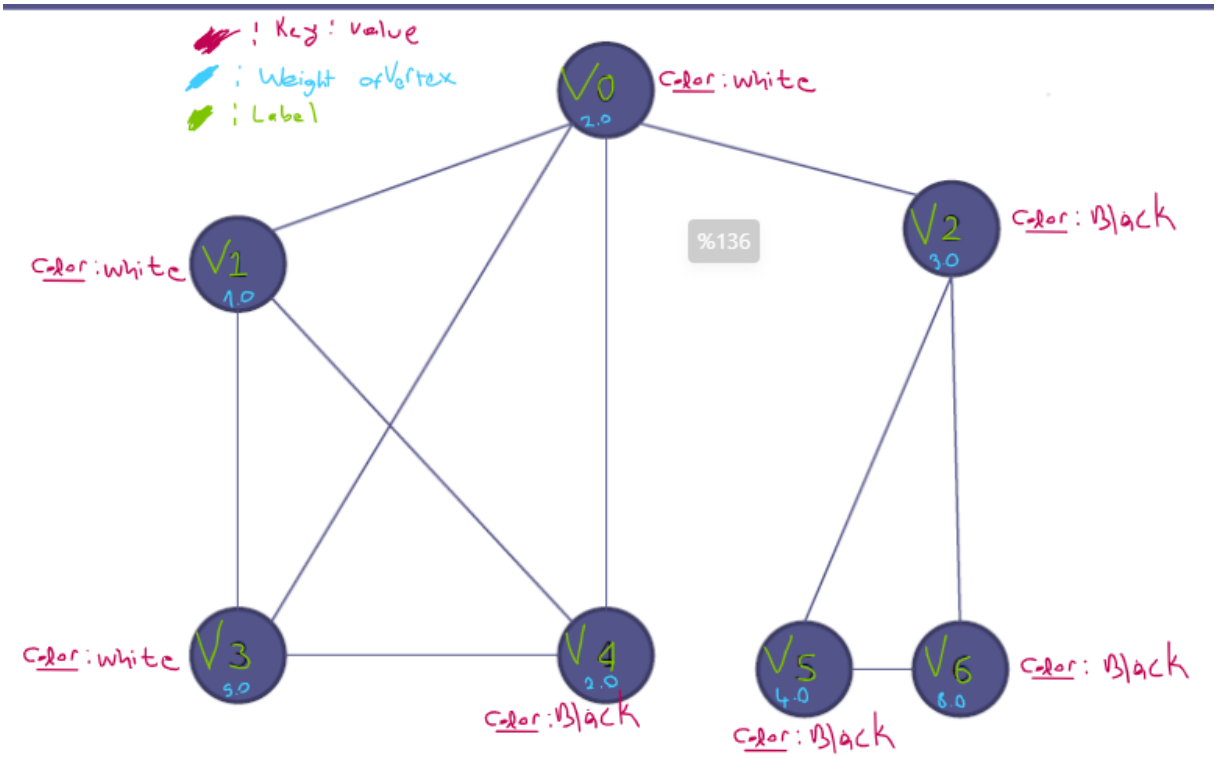
```
private int id;
private String label;
private double weight;
private double boost;

Map<E, E> userDefinedProperity;// = new HashMap<>();
```

The existence of an edge depends on its vertex. If a vertex is not present also the edge will not be able to live. On the other hand, a vertex doesn't depend on an edge to live. I handled it in implementation due to the edge and vertexes are stored in different abstract data types.

Test Cases and Their Results:

- Creating object and Adding Vertex:

```
MyGraph<Integer> graphTest = new MyGraph<Integer>(7, false);
graphTest.addVertex(new Vertex<>(0,"v0",2.0,"Color","white"));
graphTest.addVertex(new Vertex<>(1,"v1",1.0,"Color","white"));
graphTest.addVertex(new Vertex<>(2,"v2",3.0,"Color","black"));
graphTest.addVertex(new Vertex<>(3,"v3",5.0,"Color","white"));
graphTest.addVertex(new Vertex<>(4,"v4",2.0,"Color","black"));
graphTest.addVertex(new Vertex<>(5,"v5",4.0,"Color","black"));
graphTest.addVertex(new Vertex<>(6,"v6",8.0,"Color","black"));
```



Note: Vertexes are not bounded yet.

Number of vertex and edge:

```
System.out.println(graphTest.getNumV());
System.out.println(graphTest.getSizeOfEdges());
```

```
7
0
```

- Adding edge without vertex will give error:

```
MyGraph<Integer> graphTest = new MyGraph<Integer>(7, false);

graphTest.addEdge(0, 1, 30);
graphTest.addEdge(0, 2, 40);
graphTest.addEdge(0, 3, 60);
```

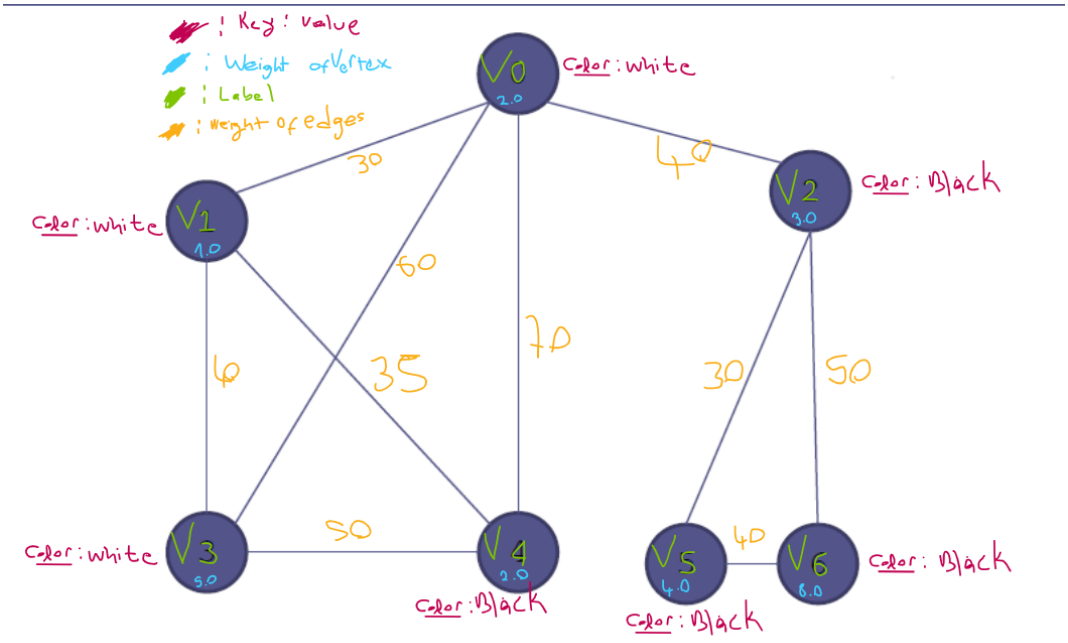
```
edge : [(0, 1): 30.0]
Edge Couldn't be created: Vertex 0 is not exist!
Edge Couldn't be created: Vertex 1 is not exist!

edge : [(0, 2): 40.0]
Edge Couldn't be created: Vertex 0 is not exist!
Edge Couldn't be created: Vertex 2 is not exist!

edge : [(0, 3): 60.0]
Edge Couldn't be created: Vertex 0 is not exist!
Edge Couldn't be created: Vertex 3 is not exist!
```

- Adding edge normally:

```
        graphTest.addEdge(0, 1, 30);
        graphTest.addEdge(0, 2, 40);
        graphTest.addEdge(0, 3, 60);
        graphTest.addEdge(0, 4, 70);
        graphTest.addEdge(1, 3, 40);
        graphTest.addEdge(1, 4, 35);
        graphTest.addEdge(3, 4, 50);
        graphTest.addEdge(2, 5, 30);
        graphTest.addEdge(2, 6, 50);
        graphTest.addEdge(5, 6, 40);
        System.out.println("After Adding edges: ");
        System.out.println(graphTest.getSizeOfEdges());
```

The graph is not directed so edge number will be doubled:

```
After Adding edges:
20
```



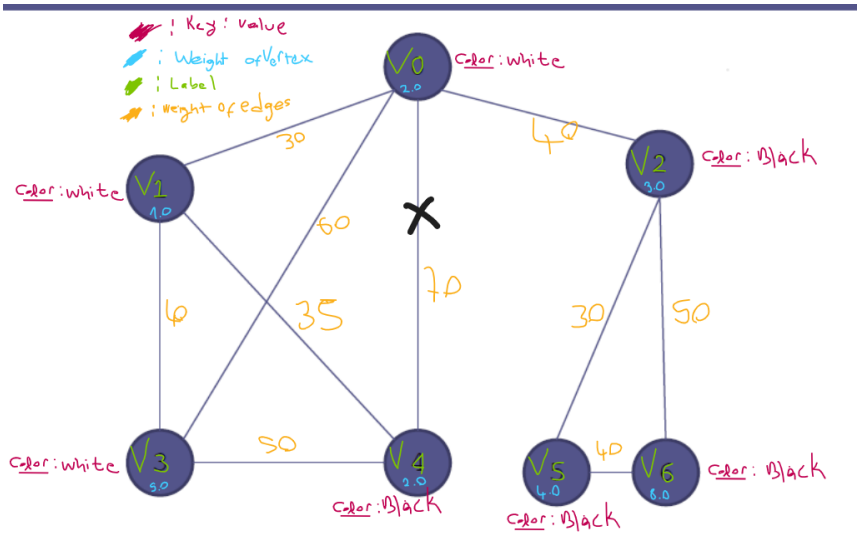
- Print The Graph:

```
System.out.println("\nPrint Result: ");
graphTest.printGraph();
```

```
Print Result:
0 -> 1 -> 2 -> 3 -> 4
1 -> 0 -> 3 -> 4
2 -> 0 -> 5 -> 6
3 -> 0 -> 1 -> 4
4 -> 0 -> 1 -> 3
5 -> 2 -> 6
6 -> 2 -> 5
```

- Remove an Edge:

```
graphTest.removeEdge(0, 4);

System.out.println("\nPrint Result After Removing: ");
graphTest.printGraph();
```



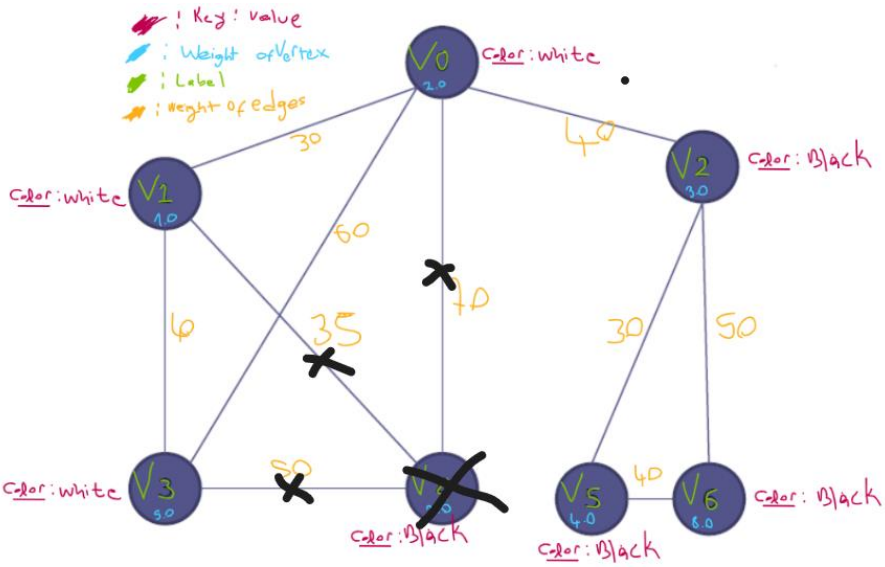

```
Print Result:
0 -> 1 -> 2 -> 3 -> 4
1 -> 0 -> 3 -> 4
2 -> 0 -> 5 -> 6
3 -> 0 -> 1 -> 4
4 -> 0 -> 1 -> 3
5 -> 2 -> 6
6 -> 2 -> 5

Print Result After Removing:
0 -> 1 -> 2 -> 3
1 -> 0 -> 3 -> 4
2 -> 0 -> 5 -> 6
3 -> 0 -> 1 -> 4
4 -> 1 -> 3
5 -> 2 -> 6
6 -> 2 -> 5
```

Before

After

- Remove a Vertex with id:

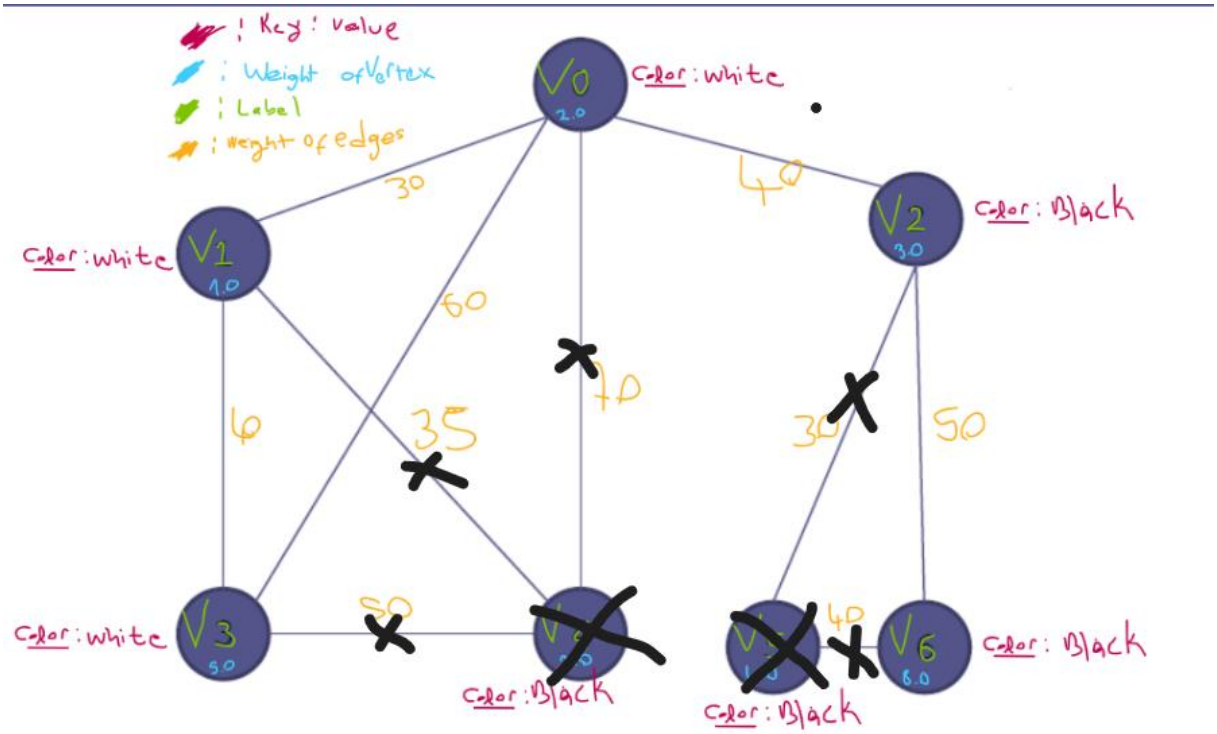
```
        System.out.println("\nPrint Result Before Removing Vertex: ");
        graphTest.printGraph();
        graphTest.removeVertex(4);  //removing 4th vertex
        System.out.println("\nPrint Result After Removing Vertex: ");
        graphTest.printGraph();
```



```
Print Result Before Removing Vertex:
0 -> 1 -> 2 -> 3
1 -> 0 -> 3 -> 4
2 -> 0 -> 5 -> 6
3 -> 0 -> 1 -> 4
4 -> 1 -> 3
5 -> 2 -> 6
6 -> 2 -> 5

Print Result After Removing Vertex:
0 -> 1 -> 2 -> 3
1 -> 0 -> 3
2 -> 0 -> 5 -> 6
3 -> 0 -> 1
5 -> 2 -> 6
6 -> 2 -> 5
```

- Remove a Vertex with Label:

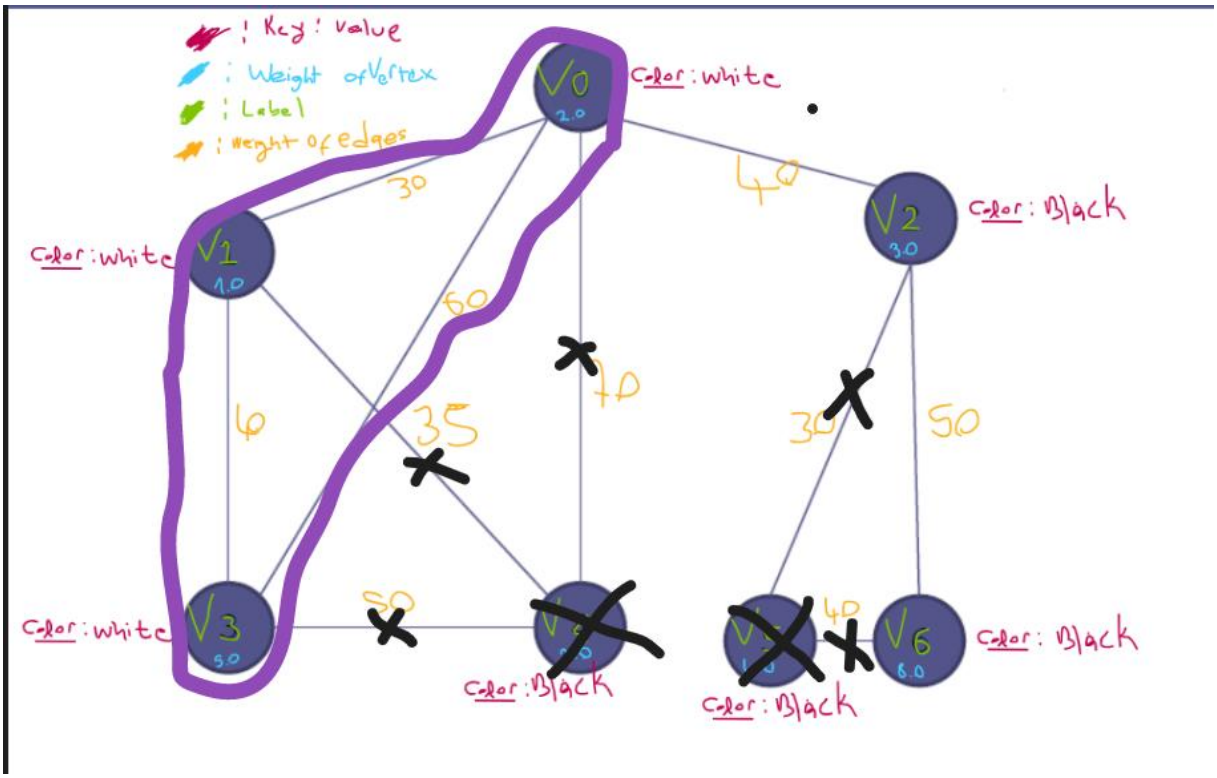
```
System.out.println("\nPrint Result Before Removing Vertex: ");
graphTest.printGraph();
graphTest.removeVertex("v5");  //removing 5th vertex with label
System.out.println("\nPrint Result After Removing Vertex: ");
graphTest.printGraph();
```




```
Print Result Before Removing Vertex:
0 -> 1 -> 2 -> 3
1 -> 0 -> 3
2 -> 0 -> 5 -> 6
3 -> 0 -> 1
5 -> 2 -> 6
6 -> 2 -> 5

Print Result After Removing Vertex:
0 -> 1 -> 2 -> 3
1 -> 0 -> 3
2 -> 0 -> 6
3 -> 0 -> 1
6 -> 2
```

- Filter Vertices:

We will filter vertices according to their color. As you can see in the graph representation there are 3 white and 4 black vertices present. We will take white ones as a sub graph:

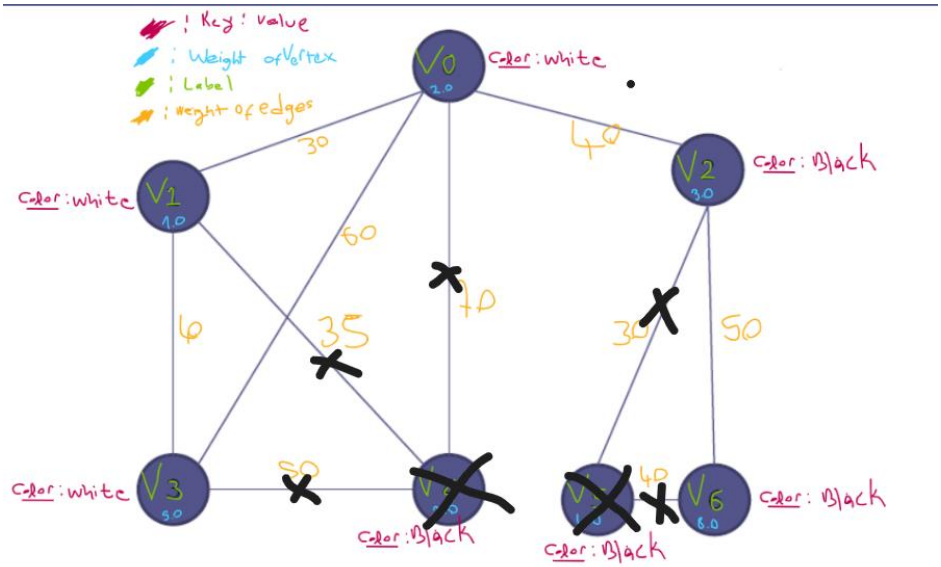



```
System.out.println();
System.out.println("SubGraph: (White Vertexes only)");
MyGraph<Integer> subGraph = graphTest.filterVertices("Color", "white");
subGraph.printGraph();
```

```
SubGraph: (White Vertexes only)
0 -> 1 -> 3
1 -> 0 -> 3
3 -> 0 -> 1
```

- Export matrix

```
    //export matrix
    double[][] matrix = graphTest.exportMatrix();
    for (int i = 0; i < matrix.length; i++) {
        System.out.print("[");
        for (int j = 0; j < matrix[i].length; j++) {
            System.out.print(matrix[i][j]);
            if(j != matrix.length-1)
                System.out.print(", ");
        }
        System.out.println("]");
    }
```




```
[Infinity, 30.0, 40.0, 60.0, Infinity, Infinity, Infinity, Infinity, Infinity]
[30.0, Infinity, Infinity, 40.0, Infinity, Infinity, Infinity, Infinity, Infinity]
[40.0, Infinity, Infinity, Infinity, Infinity, Infinity, 50.0, Infinity, Infinity]
[60.0, 40.0, 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, 50.0, Infinity, Infinity, Infinity, Infinity, Infinity, Infinity]
[Infinity, Infinity, Infinity, Infinity, Infinity, Infinity, Infinity, Infinity, Infinity]
[Infinity, Infinity, Infinity, Infinity, Infinity, Infinity, Infinity, Infinity, Infinity]
```

- newVertex: Generating new vertex

```
        System.out.println();
        //Generate a new vertex by given parameters:
        Vertex<Integer> newV = graphTest.newVertex("newVertex", 12.0);
        System.out.println(newV.toString());
```

```
[(newVertex, 0): 12.0]
[ ( {null=null} )  ]
```

Label: newVertex - id: 0 - weight: 12.0 - key : null - value: null

Time Complexities:

```
 * @return new vertex*/
@Override
public < E extends Comparable < E >>  Vertex<E> newVertex(String label, double weight) {
    return new Vertex<E>(0,label,weight,"null","null");
}

/**
```

Θ(1)

```
 **/
@Override
public Vertex addVertex(Vertex new_vertex) {
    //Vsize++;
    vList.add(new_vertex);
    return new_vertex;
}
/**
```

O(n)

```
@Override
public void addEdge(int vertexID1, int vertexID2, double weight) {
    /* */
    boolean b1 = false;
    boolean b2 = false;
    for (Vertex<E> vertex : vList) {
        if(vertex.getId() == vertexID1)
            b1 = true;
        if(vertex.getId() == vertexID2)
            b2 = true;
        if (b1 && b2) {
            insert(new Edge(vertexID1, vertexID2, weight));
            break;
        }
    }
    if(!b1 || !b2)  System.out.println("edge : [(" + vertexID1 + ", " + vertexID2 + "): " + weight + "]");
    if(!b1)         System.out.println("Edge Couldn't be created: Vertex " + vertexID1 + " is not exist!");
    if(!b2)         System.out.println("Edge Couldn't be created: Vertex " + vertexID2 + " is not exist!");

}
```

O(n) → O(n) → O(n²)

```
@Override
public void removeEdge(int vertexID1, int vertexID2) {

    eList[vertexID1].remove(getEdge(vertexID1, vertexID2));
    if(!directed) eList[vertexID2].remove(getEdge(vertexID2, vertexID1));
}

/**
```

Θ(n) → Θ(n)

```
@Override
public void removeVertex(int vertexID) {
    boolean removed = false;

    for(int i=0; i<vList.size(); ++i) {
        if(vList.get(i).getId() == vertexID) {
            //System.out.println("removed: " +vList.get(i).toString() );
            vList.remove(i);
            removed = true;
        }
    }
    if(removed)
        for(var x : eList)
            for(var edges : x)
                this.removeEdge(vertexID, edges.getDest());

    else System.out.println("Invalid vertex id!");
}
```

O(n²) Θ(n²) → O(n²)

```java
@Override
public Vertex removeVertex(String label) {
    boolean removed = false;
    int vertexID = -1;
    Vertex temp=null;
    for(int i=0; i<vList.size(); ++i) {
        if(vList.get(i).getLabel().equals(label)) {
            temp = vList.get(i);
            vertexID = vList.get(i).getId();
            vList.remove(i);
            removed = true;
            Vsize--;
            //break;
        }
    }
    if(removed)
        for(var x : eList)
            for(var edges : x)
                this.removeEdge(vertexID, edges.getDest());

    else System.out.println("Invalid vertex id!");
    return temp;
}
```

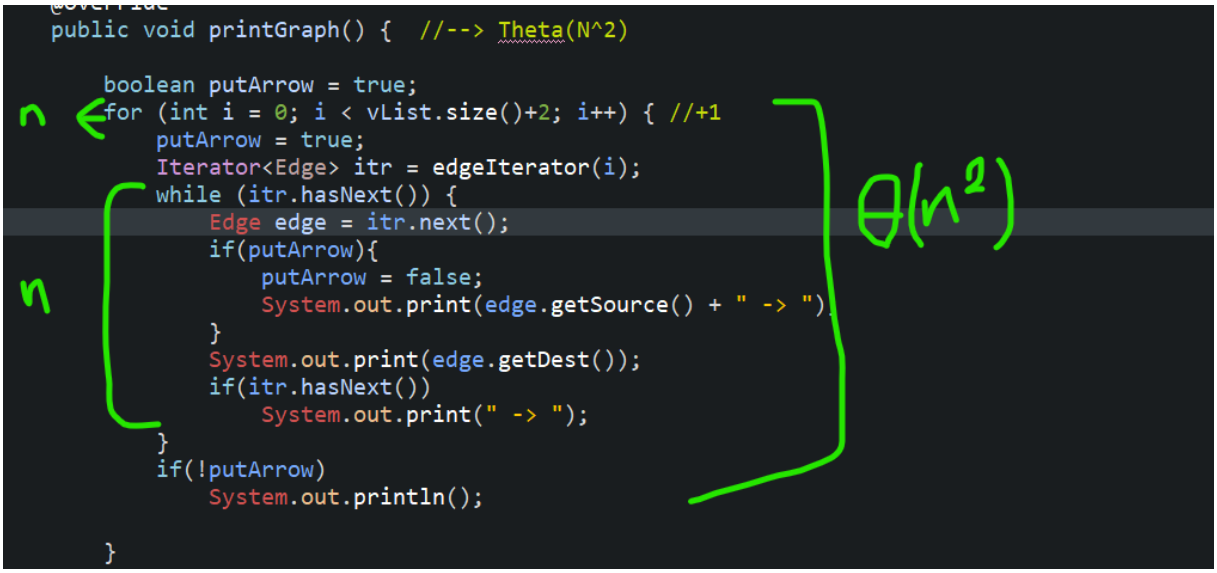
O(n²) Θ(n²) O(n²)

```java
@Override
public <E extends Comparable < E >> MyGraph<E> filterVertices(String key, String filter) {
    MyGraph<E> subGraph = new MyGraph<E>(this.getNumV(),false);
    for(var x : vList) {
        if(x.userDefinedProperity.containsKey(key) &&  x.userDefinedProperity.containsValue(filter))
            subGraph.vList.add((Vertex<E>) x);
    }
    //System.out.println(subGraph.getNumV());
    //System.out.println(i);
    for (int i = 0; i < subGraph.getNumV() - 1; i++) {
        for (int k = i + 1; k < subGraph.getNumV(); k++) {
            int source = subGraph.vList.get(i).getId();
            int dest = subGraph.vList.get(k).getId();
            if(this.isEdge(source, dest)) {
                subGraph.addEdge(source, dest, this.getEdge(source, dest).getWeight());
                //System.out.println("girdi");
            }

        }
    }
    return subGraph;
}
```

Θ(n) O(n³) O(1)

```java
/**
 * Generate the adjacency matrix representation of the graph and returns the matrix.
 * @return double [][] array*/
@Override
public double[][] exportMatrix() {

    double[][] matrix = new double[Vsize][Vsize];
    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;

}
/**
```

Θ(n²)



Q2)

BFS PART:

Problem Solution approach:

The Algorithm will check when adding a edge to que if that edge is already exist in que, it will compere existing edge's weight and new coming edge's weight. If new coming edge's weight is smaller deletes old edges from que and puts new coming edge to last of the que. In the other case does not add new coming edge to que just passes it.

Test Cases and Their Results:

We will show the process in relatively smaller graph. We choose because it is easier to show.

To be able to traverse all points in shortest path, algorithm must choose 2nd vertex to go to the 3rd

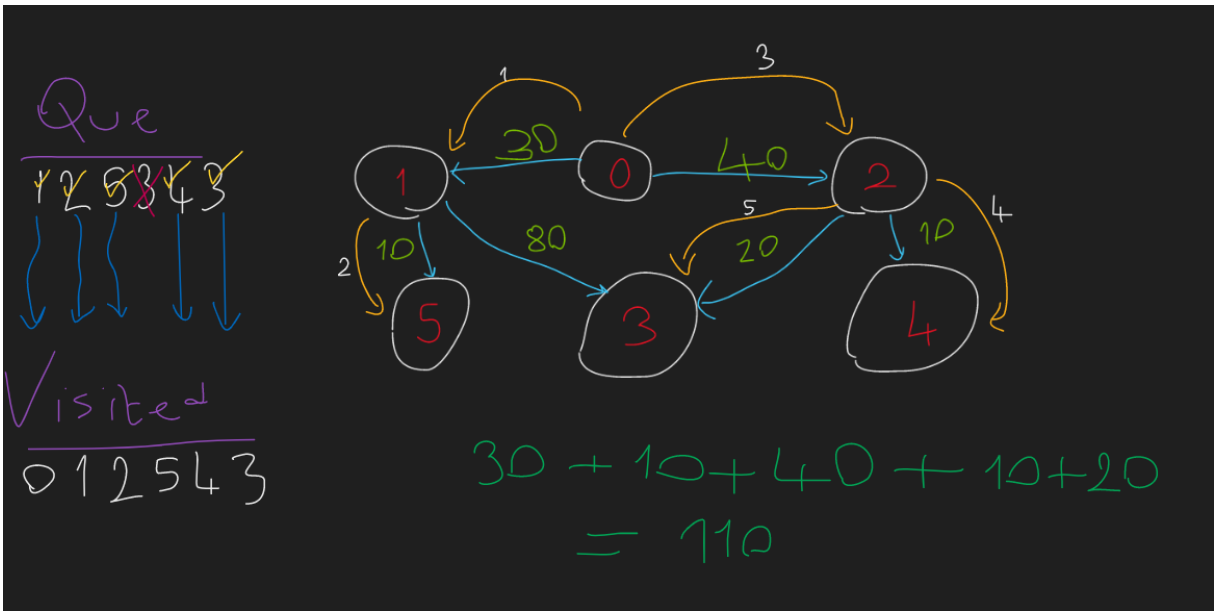
Vertex.

Starts from zero Vertex and adds neighbors to que.

continue with 1 and add its neighbors (5 and 3) to que.

Continue with 2 and add its neighbors (4 and 3) to que. But 3 is already added to que. Check weights if the edge that go through 2 to 3 has smaller weight than 1 to 3, then delete that 3 in the que and add 2's 3 this is shorter path.

Then continue in same way




```
MyGraph<Integer> gTest2 = new MyGraph<Integer>(10, true);

gTest2.addVertex(new Vertex<>(0,"v0",2.0));
gTest2.addVertex(new Vertex<>(1,"v1",1.0));
gTest2.addVertex(new Vertex<>(2,"v2",3.0));
gTest2.addVertex(new Vertex<>(3,"v3",5.0));
gTest2.addVertex(new Vertex<>(4,"v4",2.0));
gTest2.addVertex(new Vertex<>(5,"v5",4.0));

gTest2.addEdge(0, 1, 30);
gTest2.addEdge(0, 2, 40);
gTest2.addEdge(1, 3, 80);
gTest2.addEdge(1, 5, 10);
gTest2.addEdge(2, 3, 20);
gTest2.addEdge(2, 4, 10);
```

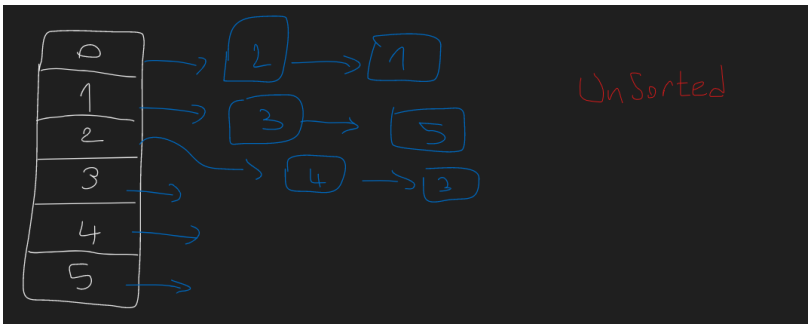
```
BreadthFirstSearch bfs = new BreadthFirstSearch();

double db = bfs.breadthFirstSearch(gTest2);
System.out.println("BFS Shortest path:");
System.out.println(db);
System.out.println();
```

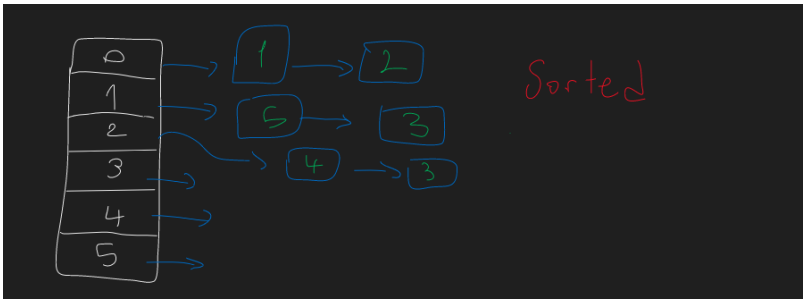
```
BFS Shortest path:
110.0
```

DFS PART:

Problem Solution approach:

I just sorted all vertex's edges from small to large. It will be enough to find the minimum path. So that algorithm will choose always smaller one.





Test Cases and Their Results:

```
MyGraph<Integer> gTest2 = new MyGraph<Integer>(20, true);

gTest2.addVertex(new Vertex<>(0,"v0",2.0));
gTest2.addVertex(new Vertex<>(1,"v1",1.0));
gTest2.addVertex(new Vertex<>(2,"v2",3.0));
gTest2.addVertex(new Vertex<>(3,"v3",5.0));
gTest2.addVertex(new Vertex<>(4,"v4",2.0));
gTest2.addVertex(new Vertex<>(5,"v5",4.0));
// gTest2.addVertex(new Vertex<>(6,"v6",4.0));

//Test1
gTest2.addEdge(0, 1, 30);
gTest2.addEdge(0, 2, 40);
gTest2.addEdge(1, 3, 80);
gTest2.addEdge(1, 5, 10);
gTest2.addEdge(2, 3, 20);
gTest2.addEdge(2, 4, 10);
```

Same graph used (that is on the bfs example)

```
DFS Shortest path:
170.0
```

|Dfs - Bfs|

```
BFS Shortest path:
110.0

DFS Shortest path:
170.0

Difference:
60.0
```

Other example:

```
[(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]
BFS Shortest path:
215.0

DFS Shortest path:
225.0

Difference:
10.0
```

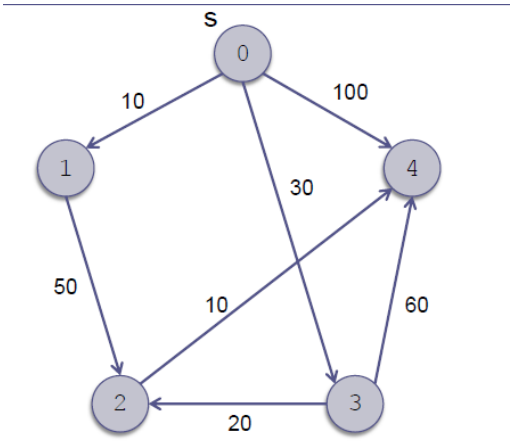
Q3)

Problem Solution Approach:

First, We will take boost value of the current vertex and subtract it from the distance + weight value. If result is smaller than distance value of v, then result is the distance of v.

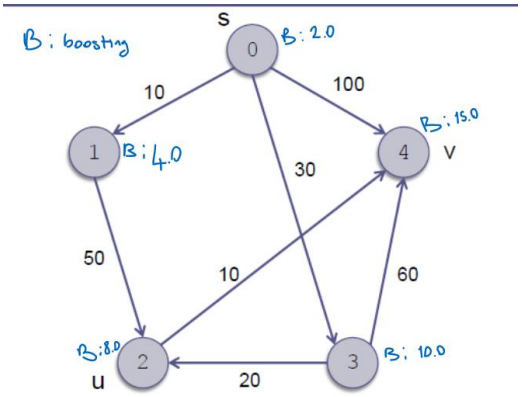
Test Cases and Their Results:

Without boosting:




```
Boost values And Result: (Without Boost)
0.0
0.0
0.0
0.0
Vertex          Shortest Distance
0               0.0
1               10.0
2               50.0
3               30.0
4               60.0
```

With boosting:




```
Boost values And Result: (With Boost)
4.0
10.0
10.0
8.0
Vertex            Shortest Distance
0                 0.0
1                 10.0
2                 40.0
3                 30.0
4                 42.0
```

HACI HASAN SAVAN

1901042704