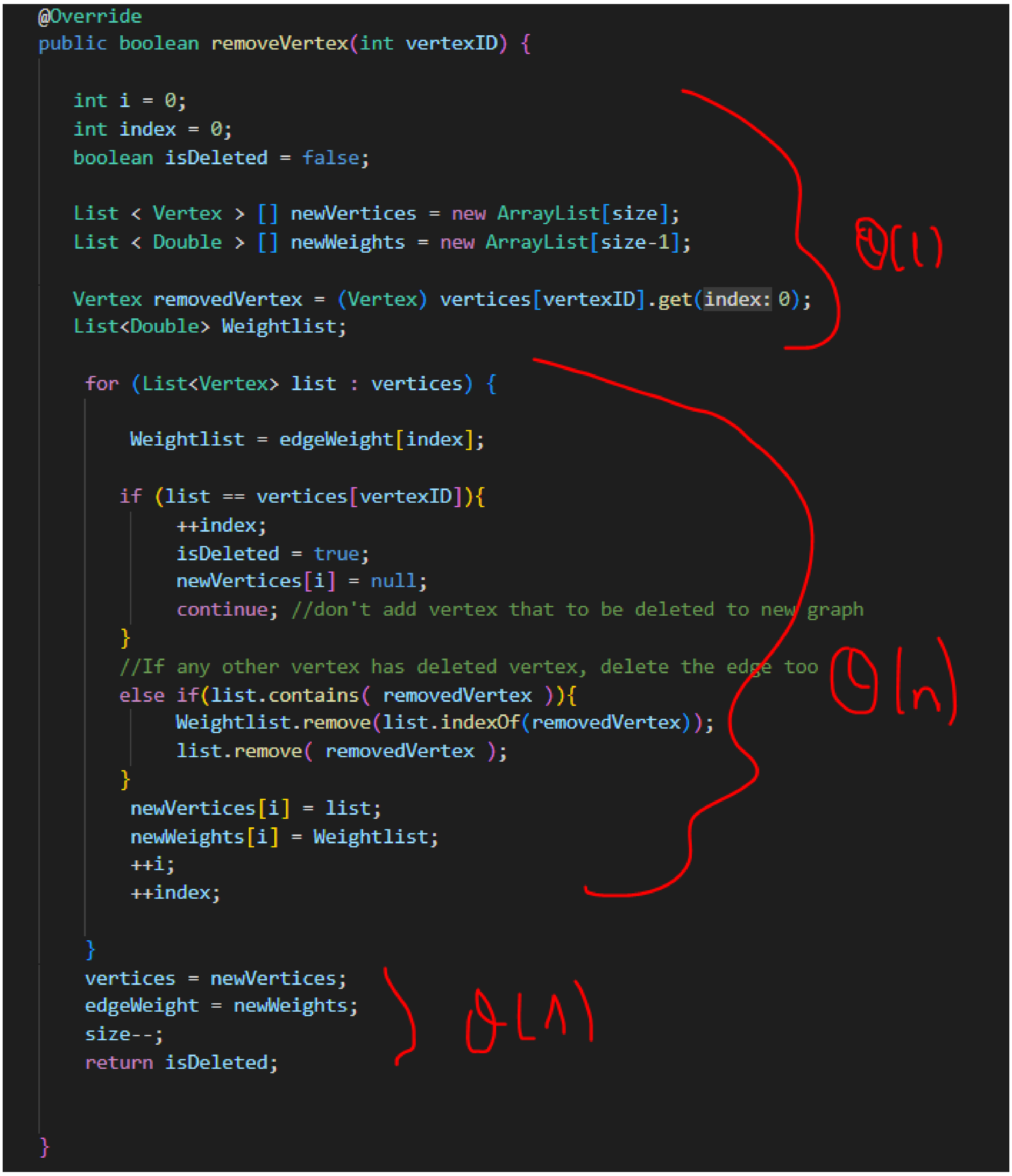
GIT Department of Computer Engineering
CSE 222/505 - Spring 2022
Homework 8 Report

ABDULSAMED ASLAN
200104004098

Theoretical Run Time Complexity Analysis

- The methods which are not in the report have theta(1) running time.

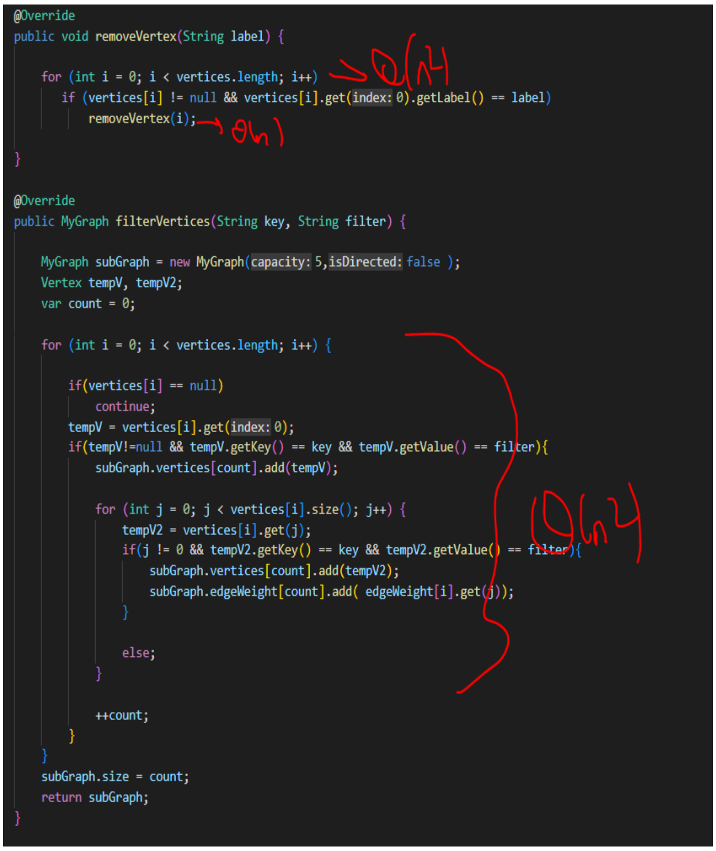
removeVertex: Theta(n)



```
@Override
public boolean removeVertex(int vertexID) {

   int i = 0;
   int index = 0;
   boolean isDeleted = false;

   List < Vertex > [] newVertices = new ArrayList[size];
   List < Double > [] newWeights = new ArrayList[size-1];

   Vertex removedVertex = (Vertex) vertices[vertexID].get(index: 0);
   List<Double> Weightlist;

    for (List<Vertex> list : vertices) {

        Weightlist = edgeWeight[index];

       if (list == vertices[vertexID]){
            ++index;
            isDeleted = true;
            newVertices[i] = null;
            continue; //don't add vertex that to be deleted to new graph
       }
       //If any other vertex has deleted vertex, delete the edge too
       else if(list.contains( removedVertex )){
            Weightlist.remove(list.indexOf(removedVertex));
            list.remove( removedVertex );
       }
        newVertices[i] = list;
        newWeights[i] = Weightlist;
        ++i;
        ++index;

    }
    vertices = newVertices;
    edgeWeight = newWeights;
    size--;
    return isDeleted;


}
```

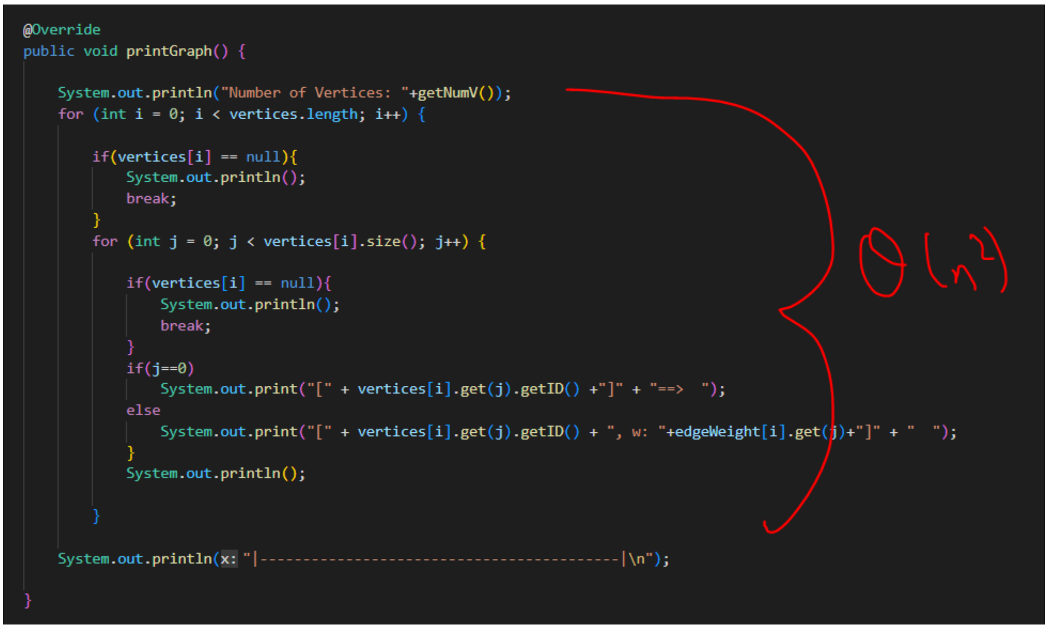
removeVertex: Theta(n^2)

filterVertices: Theta(n^2)



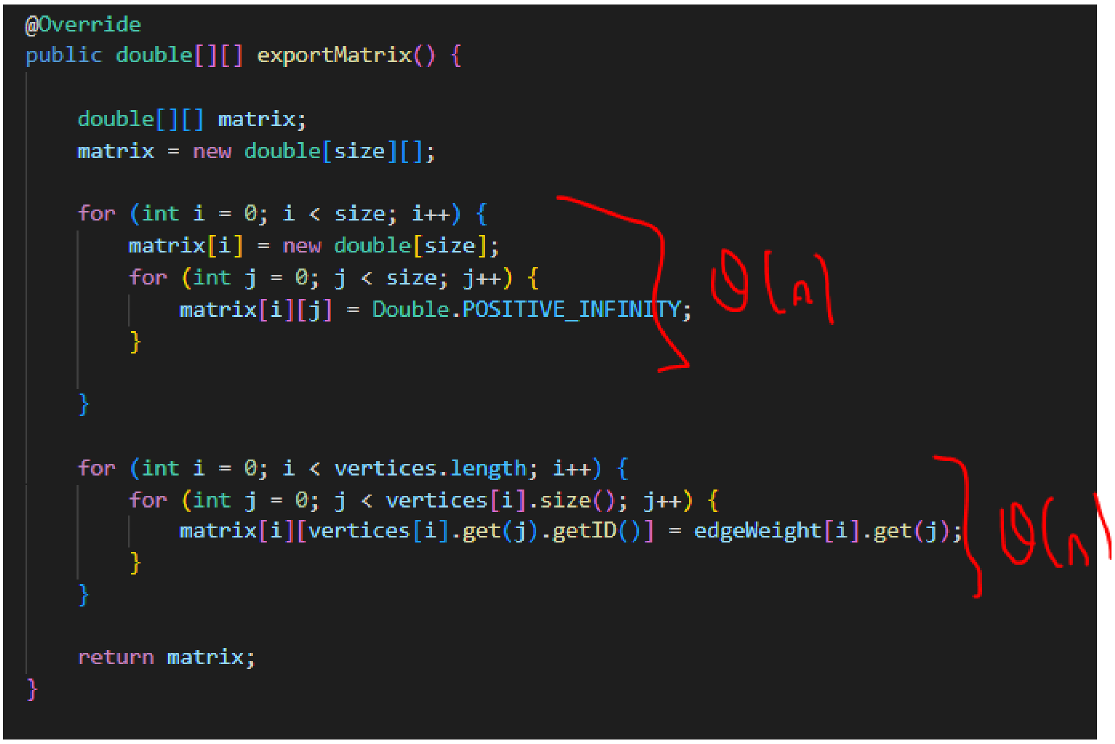
```
@Override
public void removeVertex(String label) {

    for (int i = 0; i < vertices.length; i++)
       if (vertices[i] != null && vertices[i].get(index: 0).getLabel() == label)
           removeVertex(i);

}

@Override
public MyGraph filterVertices(String key, String filter) {

    MyGraph subGraph = new MyGraph(capacity: 5,isDirected: false );
    Vertex tempV, tempV2;
    var count = 0;

    for (int i = 0; i < vertices.length; i++) {

        if(vertices[i] == null)
            continue;
        tempV = vertices[i].get(index: 0);
        if(tempV!=null && tempV.getKey() == key && tempV.getValue() == filter){
            subGraph.vertices[count].add(tempV);

            for (int j = 0; j < vertices[i].size(); j++) {
                tempV2 = vertices[i].get(j);
                if(j != 0 && tempV2.getKey() == key && tempV2.getValue() == filter){
                    subGraph.vertices[count].add(tempV2);
                    subGraph.edgeWeight[count].add( edgeWeight[i].get(j));
                }

                else;
            }

            ++count;
        }
    }
    subGraph.size = count;
    return subGraph;
}
```

printGraph(): Totally Theta(n^2)



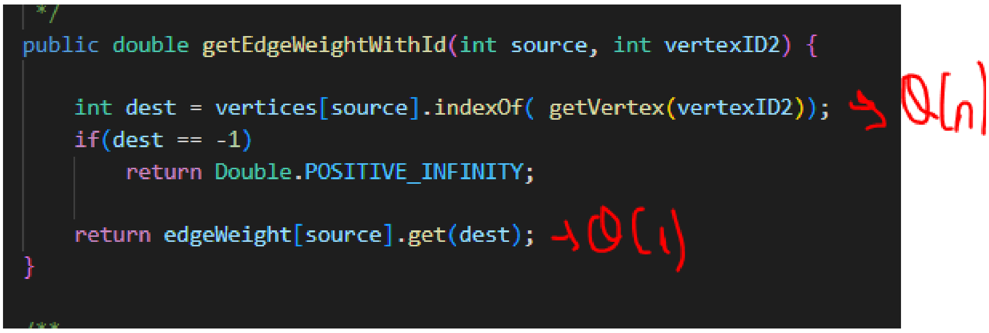
```
@Override
public void printGraph() {

    System.out.println("Number of Vertices: "+getNumV());
    for (int i = 0; i < vertices.length; i++) {

        if(vertices[i] == null){
            System.out.println();
            break;
        }
        for (int j = 0; j < vertices[i].size(); j++) {

            if(vertices[i] == null){
                System.out.println();
                break;
            }
            if(j==0)
                System.out.print("[" + vertices[i].get(j).getID() +"]" + "==>  ");
            else
                System.out.print("[" + vertices[i].get(j).getID() + ", w: "+edgeWeight[i].get(j)+"]" + "  ");
            }
            System.out.println();

        }

    System.out.println(x: "|------------------------------------------|\n");

}
```

exportMatrix(): Totally theta(n^2)



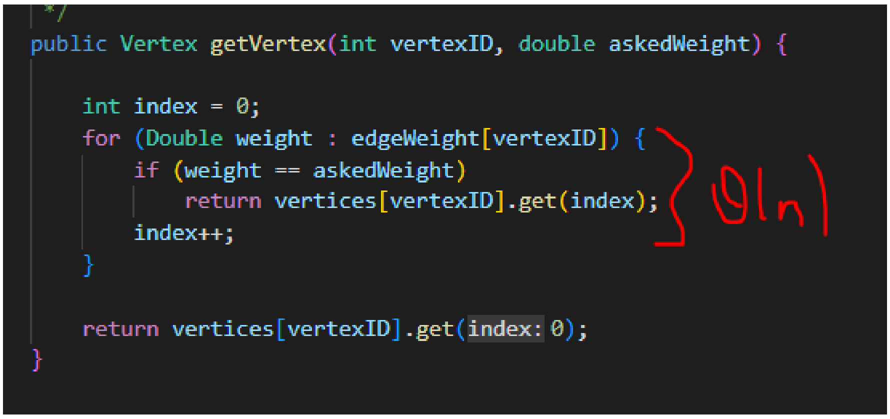
```java
@Override
public double[][] exportMatrix() {

    double[][] matrix;
    matrix = new double[size][];

    for (int i = 0; i < size; i++) {
        matrix[i] = new double[size];
        for (int j = 0; j < size; j++) {
            matrix[i][j] = Double.POSITIVE_INFINITY;
        }

    }

    for (int i = 0; i < vertices.length; i++) {
        for (int j = 0; j < vertices[i].size(); j++) {
            matrix[i][vertices[i].get(j).getID()] = edgeWeight[i].get(j);
        }
    }

    return matrix;
}
```

getEdgeWeightWithId(): Totally Theta(n)



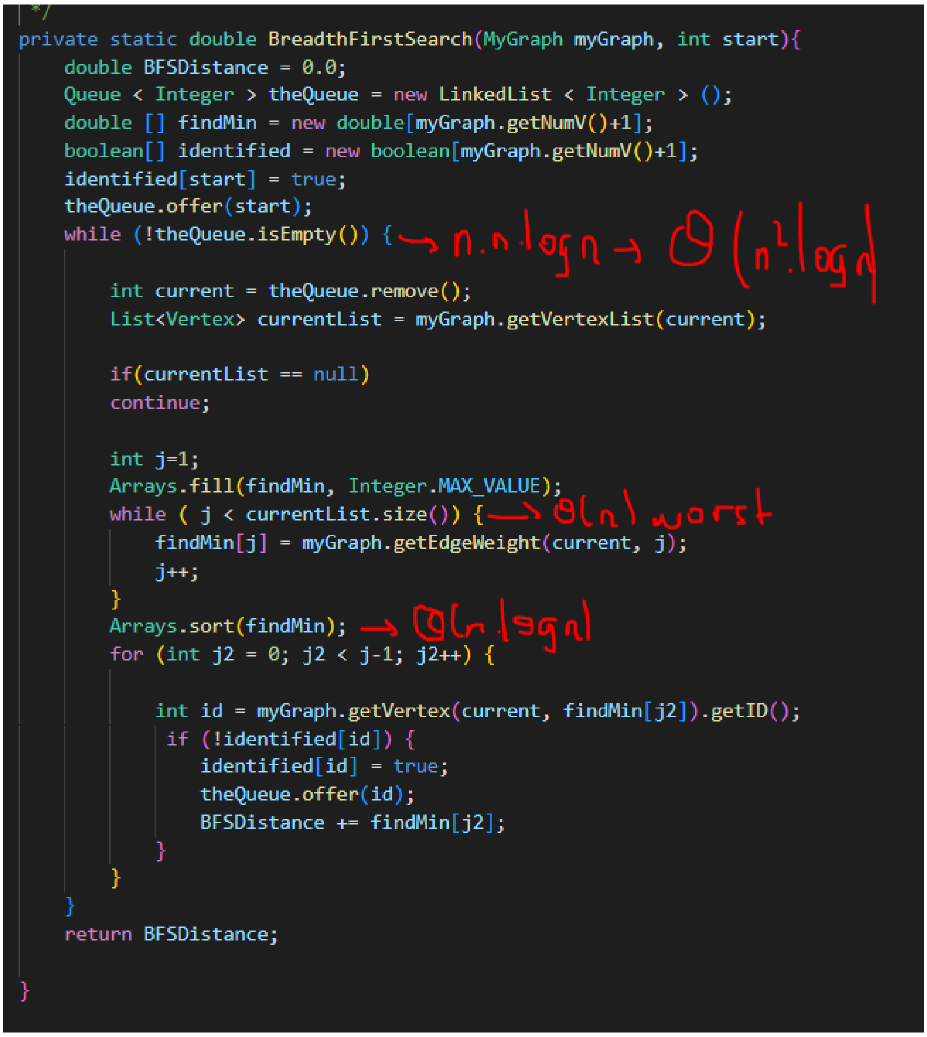
```java
 */
public double getEdgeWeightWithId(int source, int vertexID2) {

    int dest = vertices[source].indexOf( getVertex(vertexID2));
    if(dest == -1)
        return Double.POSITIVE_INFINITY;

    return edgeWeight[source].get(dest);
}

/**
```

getVertex: Totally Theta(n)



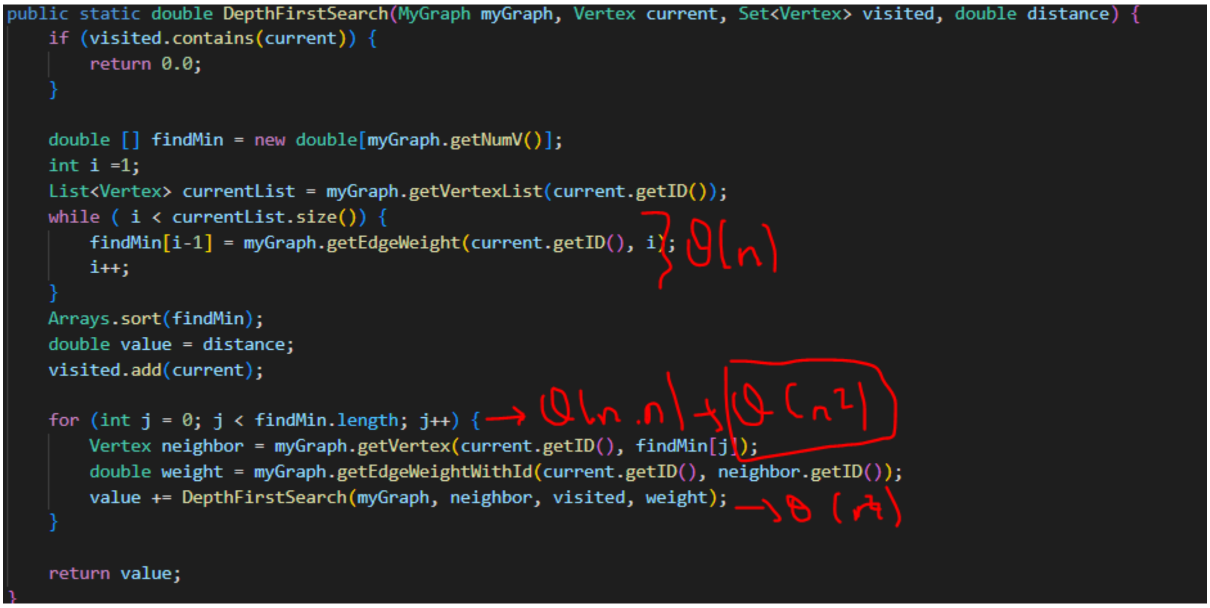
```java
 */
public Vertex getVertex(int vertexID, double askedWeight) {

    int index = 0;
    for (Double weight : edgeWeight[vertexID]) {
        if (weight == askedWeight)
            return vertices[vertexID].get(index);
        index++;
    }

    return vertices[vertexID].get(index: 0);
}
```

BreadthFirstSearch() = Totally Theta(n^2logn)



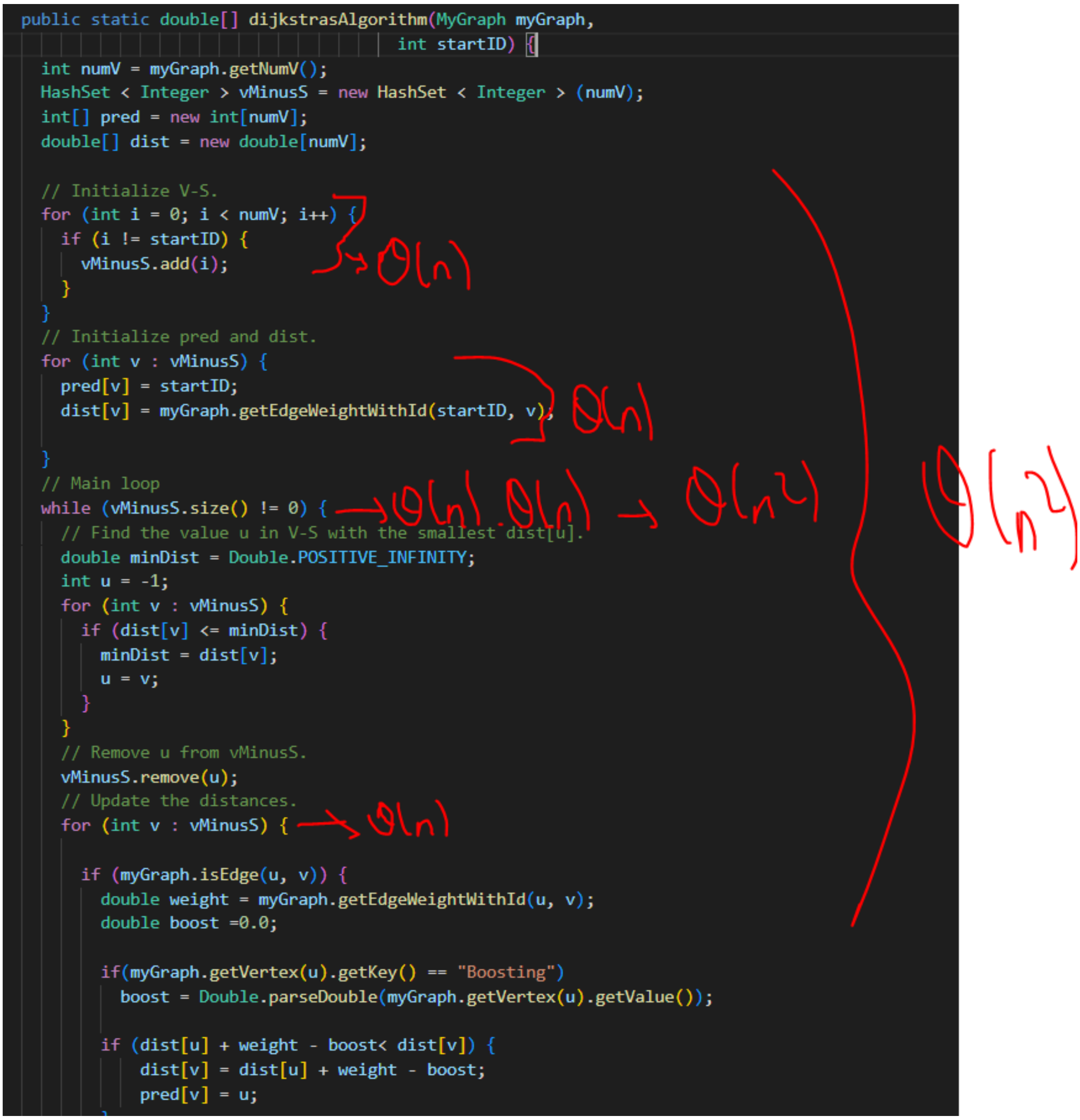
```
 */
private static double BreadthFirstSearch(MyGraph myGraph, int start){
    double BFSDistance = 0.0;
    Queue < Integer > theQueue = new LinkedList < Integer > ();
    double [] findMin = new double[myGraph.getNumV()+1];
    boolean[] identified = new boolean[myGraph.getNumV()+1];
    identified[start] = true;
    theQueue.offer(start);
    while (!theQueue.isEmpty()) {

        int current = theQueue.remove();
        List<Vertex> currentList = myGraph.getVertexList(current);

        if(currentList == null)
        continue;

        int j=1;
        Arrays.fill(findMin, Integer.MAX_VALUE);
        while ( j < currentList.size()) {
            findMin[j] = myGraph.getEdgeWeight(current, j);
            j++;
        }
        Arrays.sort(findMin);
        for (int j2 = 0; j2 < j-1; j2++) {

            int id = myGraph.getVertex(current, findMin[j2]).getID();
             if (!identified[id]) {
                identified[id] = true;
                theQueue.offer(id);
                BFSDistance += findMin[j2];
            }
        }
    }
    return BFSDistance;

}
```

DepthFirstSearch(): Totally Theta(n^2)



```
public static double DepthFirstSearch(MyGraph myGraph, Vertex current, Set<Vertex> visited, double distance) {
    if (visited.contains(current)) {
        return 0.0;
    }

    double [] findMin = new double[myGraph.getNumV()];
    int i =1;
    List<Vertex> currentList = myGraph.getVertexList(current.getID());
    while ( i < currentList.size()) {
        findMin[i-1] = myGraph.getEdgeWeight(current.getID(), i);
        i++;
    }
    Arrays.sort(findMin);
    double value = distance;
    visited.add(current);

    for (int j = 0; j < findMin.length; j++) {
        Vertex neighbor = myGraph.getVertex(current.getID(), findMin[j]);
        double weight = myGraph.getEdgeWeightWithId(current.getID(), neighbor.getID());
        value += DepthFirstSearch(myGraph, neighbor, visited, weight);
    }

    return value;
}
```

DijkstraAlgorithm: Totally Theta(n^2)



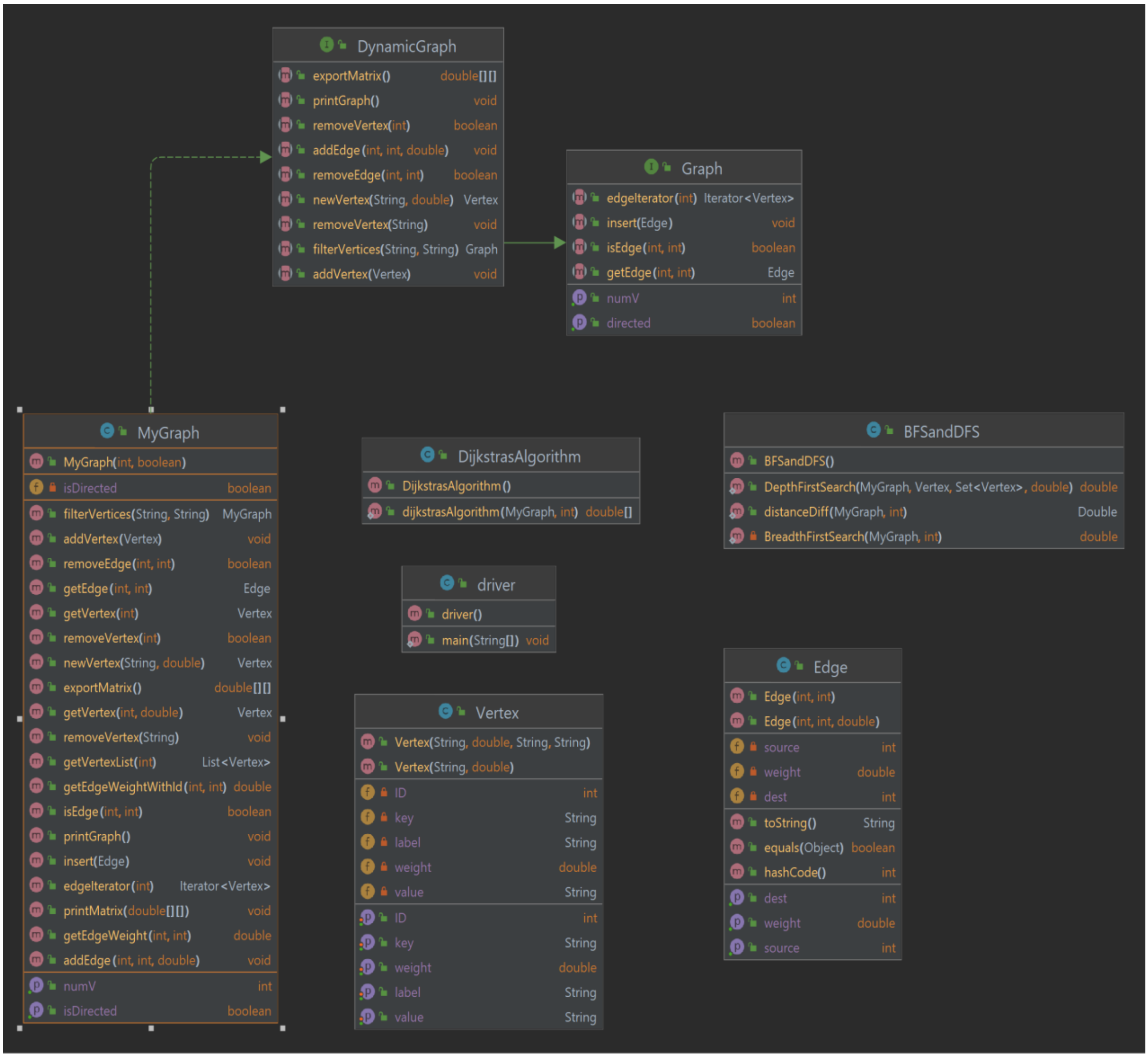
```
public static double[] dijkstrasAlgorithm(MyGraph myGraph,
                                          int startID) {
  int numV = myGraph.getNumV();
  HashSet < Integer > vMinusS = new HashSet < Integer > (numV);
  int[] pred = new int[numV];
  double[] dist = new double[numV];

  // Initialize V-S.
  for (int i = 0; i < numV; i++) {
    if (i != startID) {
      vMinusS.add(i);
    }
  }
  // Initialize pred and dist.
  for (int v : vMinusS) {
    pred[v] = startID;
    dist[v] = myGraph.getEdgeWeightWithId(startID, v);

  }
  // Main loop
  while (vMinusS.size() != 0) {
    // Find the value u in V-S with the smallest dist[u].
    double minDist = Double.POSITIVE_INFINITY;
    int u = -1;
    for (int v : vMinusS) {
      if (dist[v] <= minDist) {
        minDist = dist[v];
        u = v;
      }
    }
    // Remove u from vMinusS.
    vMinusS.remove(u);
    // Update the distances.
    for (int v : vMinusS) {

      if (myGraph.isEdge(u, v)) {
        double weight = myGraph.getEdgeWeightWithId(u, v);
        double boost =0.0;

        if(myGraph.getVertex(u).getKey() == "Boosting")
          boost = Double.parseDouble(myGraph.getVertex(u).getValue());

        if (dist[u] + weight - boost< dist[v]) {
          dist[v] = dist[u] + weight - boost;
          pred[v] = u;
```

1. SYSTEM REQUIREMENTS

Create a new MyGraph object with capacity and choose whether directed. Then create and add new vertices to graph by using addVertex method. IDs of vertices given by counter automatically. So Graph is ready.

2. USE CASE AND CLASS DIAGRAMS

2.1 UML CLASS DIAGRAM


DynamicGraph
exportMatrix() double[][]
printGraph() void
removeVertex(int) boolean
addEdge(int, int, double) void
removeEdge(int, int) boolean
newVertex(String, double) Vertex
removeVertex(String) void
filterVertices(String, String) Graph
addVertex(Vertex) void
Graph
edgeIterator(int) Iterator<Vertex>
insert(Edge) void
isEdge(int, int) boolean
getEdge(int, int) Edge
numV int
directed boolean
MyGraph
MyGraph(int, boolean)
isDirected boolean
filterVertices(String, String) MyGraph
addVertex(Vertex) void
removeEdge(int, int) boolean
getEdge(int, int) Edge
getVertex(int) Vertex
removeVertex(int) boolean
newVertex(String, double) Vertex
exportMatrix() double[][]
getVertex(int, double) Vertex
removeVertex(String) void
getVertexList(int) List<Vertex>
getEdgeWeightWithId(int, int) double
isEdge(int, int) boolean
printGraph() void
insert(Edge) void
edgeIterator(int) Iterator<Vertex>
printMatrix(double[][]) void
getEdgeWeight(int, int) double
addEdge(int, int, double) void
numV int
isDirected boolean
DijkstrasAlgorithm
DijkstrasAlgorithm()
dijkstrasAlgorithm(MyGraph, int) double[]
BFSandDFS
BFSandDFS()
DepthFirstSearch(MyGraph, Vertex, Set<Vertex>, double) double
distanceDiff(MyGraph, int) Double
BreadthFirstSearch(MyGraph, int) double
driver
driver()
main(String[]) void
Vertex
Vertex(String, double, String, String)
Vertex(String, double)
ID int
key String
label String
weight double
value String
ID int
key String
weight double
label String
value String
Edge
Edge(int, int)
Edge(int, int, double)
source int
weight double
dest int
toString() String
equals(Object) boolean
hashCode() int
dest int
weight double
source int

3. PROBLEM SOLUTION APPROACH

At first, I learned how graph mechanism works. Then I've started generating vertex class, each vertex has an id, label, key, value, and weight but ids are unique, so they are given by counter automatically. After that, I generated MyGraph class. MyGraph has an Array of ArrayList as data structure. Array holds the source vertex and source ArrayList holds the neighbors of this. I got some help from book to implement Dijkstra, Breadth First Search and Depth First Search algorithm. I added new properties to this algorithm. Each algorithm calculates the shortest way to access neighbor of the source vertex.

TEST CASES

PART 1-)

Create a Graph

```
MyGraph myGraph = new MyGraph(4, false);
```

Add new vertex to the graph with key and value

```
var a1 = new Vertex("A", 2,"red","blue");
myGraph.addVertex(a1);
myGraph.addVertex(new Vertex("B", 2,"Boosting","2.0"));
myGraph.addVertex(new Vertex("C", 2,"Boosting","3.0"));
myGraph.addVertex(new Vertex("D", 2,"red","blue"));
myGraph.addVertex(new Vertex("E", 2,"red","blue"));
myGraph.addVertex(new Vertex("F", 2,"red","blue"));
```

Add edges and print the current graph

```
myGraph.addEdge(0, 1, 7);
myGraph.addEdge(0, 2, 9);
myGraph.addEdge(0, 5, 14);

myGraph.addEdge(1, 2, 10);
myGraph.addEdge(1, 3, 15);
myGraph.addEdge(2, 3, 11);
myGraph.addEdge(2, 5, 2);

myGraph.addEdge(3, 4, 6);
myGraph.addEdge(4, 5, 9);
myGraph.printGraph();
```

Filter the graph with specific key and value and print the subgraph which is only including the vertices that are have given key and value

```
System.out.println("|--------------Filter Vertices--------------|");
MyGraph subGraph = myGraph.filterVertices("red", "blue");
subGraph.printGraph();
```

Call the exportMatrix and print the matrix representation of the graph

```
System.out.println("|---------------Export Matrix---------------|");
myGraph.printMatrix( myGraph.exportMatrix());
```

Call the distanceDiff to calculate difference between BFS and DFS algorithms values.

```
System.out.println("\n|--Difference Distance Between BFS and DFS--|");
System.out.println("Distance Difference: " +
BFSandDFS.distanceDiff(myGraph,0));
System.out.println();
```

Call the dijsktraAlgoritm to calculate shortest way with boosting values

```
System.out.println("|----Dijkstras Algo with Boosting Vertex----|");
double[] distances = DijkstrasAlgorithm.dijkstrasAlgorithm(myGraph, 0);
for (int i = 0; i < distances.length; i++)
   System.out.println("StartID to " + i + " -> " + distances[i]);
System.out.println();
```

Delete an Edge from the graph

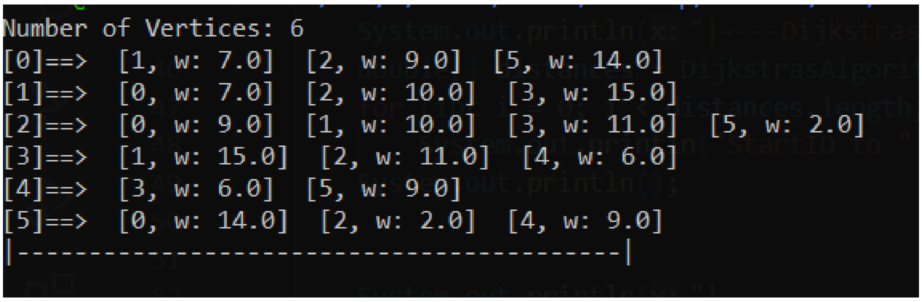
```
System.out.println("|---------------Delete Edge--------------|");
myGraph.removeEdge(5, 0);
System.out.println("|---------------[5,0] deleted------------|");
myGraph.printGraph();
```

Delete a Vertex from the graph

```
System.out.println("|---------------Delete Vertex--------------|");
myGraph.removeVertex("B");
System.out.println("|------------1. index was deleted----------|");
myGraph.printGraph();
```

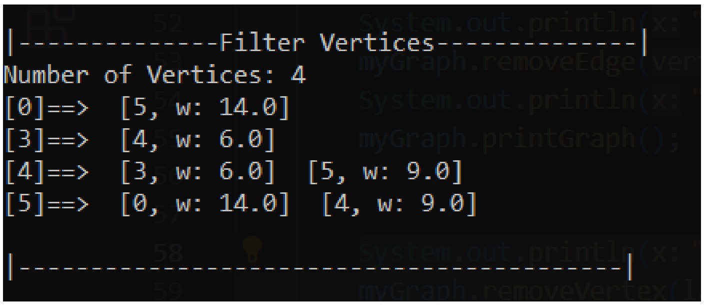
5.RUNNING AND RESULTS

Initial Graph



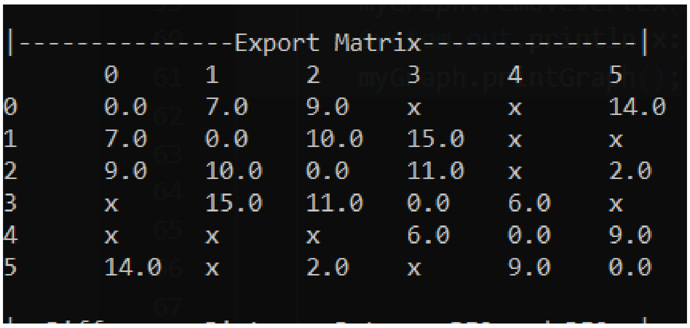
```
Number of Vertices: 6
[0]==>  [1, w: 7.0]  [2, w: 9.0]  [5, w: 14.0]
[1]==>  [0, w: 7.0]  [2, w: 10.0]  [3, w: 15.0]
[2]==>  [0, w: 9.0]  [1, w: 10.0]  [3, w: 11.0]  [5, w: 2.0]
[3]==>  [1, w: 15.0]  [2, w: 11.0]  [4, w: 6.0]
[4]==>  [3, w: 6.0]  [5, w: 9.0]
[5]==>  [0, w: 14.0]  [2, w: 2.0]  [4, w: 9.0]
|-------------------------------------------|
```

Subgraph after filtering



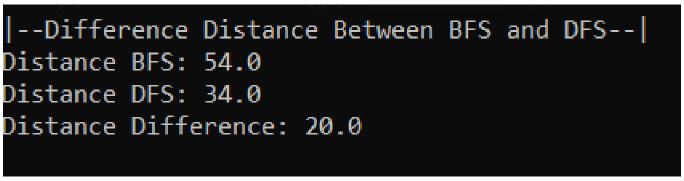
```
|--------------Filter Vertices--------------|
Number of Vertices: 4
[0]==>  [5, w: 14.0]
[3]==>  [4, w: 6.0]
[4]==>  [3, w: 6.0]  [5, w: 9.0]
[5]==>  [0, w: 14.0]  [4, w: 9.0]

|-------------------------------------------|
```

Export matrix representation



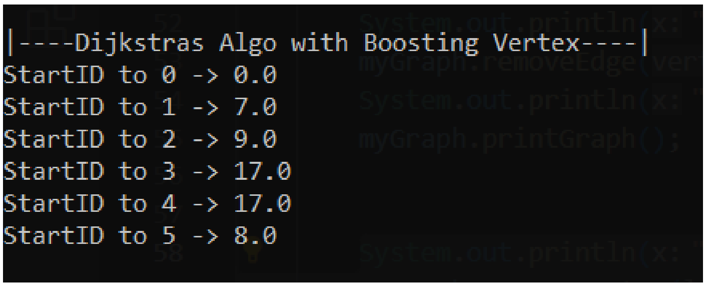
```
|---------------Export Matrix---------------|
       0      1      2      3      4      5
0      0.0    7.0    9.0    x      x      14.0
1      7.0    0.0    10.0   15.0   x      x
2      9.0    10.0   0.0    11.0   x      2.0
3      x      15.0   11.0   0.0    6.0    x
4      x      x      x      6.0    0.0    9.0
5      14.0   x      2.0    x      9.0    0.0
```

BFS and DFS distances difference



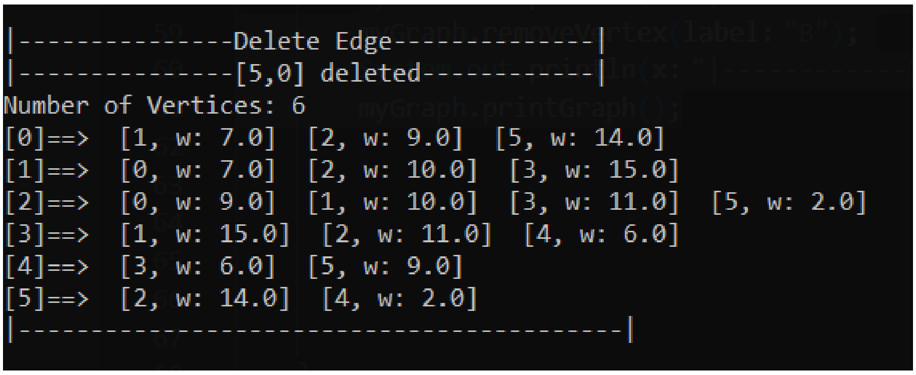
```
|--Difference Distance Between BFS and DFS--|
Distance BFS: 54.0
Distance DFS: 34.0
Distance Difference: 20.0
```

Dijsktras algorithm with boosting value



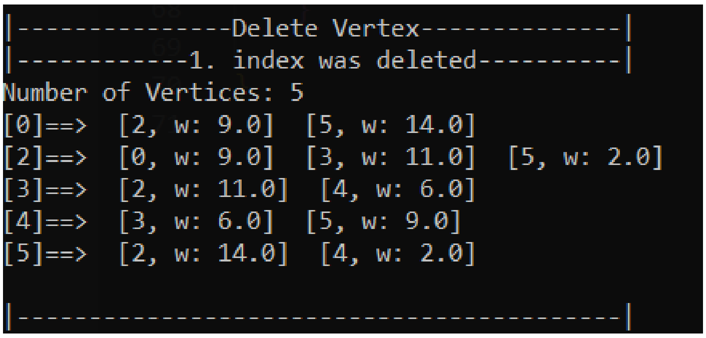
```
|----Dijkstras Algo with Boosting Vertex----|
StartID to 0 -> 0.0
StartID to 1 -> 7.0
StartID to 2 -> 9.0
StartID to 3 -> 17.0
StartID to 4 -> 17.0
StartID to 5 -> 8.0
```

After deleting an edge



```
|---------------Delete Edge--------------|
|---------------[5,0] deleted-------------|
Number of Vertices: 6
[0]==>  [1, w: 7.0]  [2, w: 9.0]  [5, w: 14.0]
[1]==>  [0, w: 7.0]  [2, w: 10.0]  [3, w: 15.0]
[2]==>  [0, w: 9.0]  [1, w: 10.0]  [3, w: 11.0]  [5, w: 2.0]
[3]==>  [1, w: 15.0]  [2, w: 11.0]  [4, w: 6.0]
[4]==>  [3, w: 6.0]  [5, w: 9.0]
[5]==>  [2, w: 14.0]  [4, w: 2.0]
|-------------------------------------------|
```

After deleting a vertex



```
|---------------Delete Vertex--------------|
|------------1. index was deleted----------|
Number of Vertices: 5
[0]==>  [2, w: 9.0]  [5, w: 14.0]
[2]==>  [0, w: 9.0]  [3, w: 11.0]  [5, w: 2.0]
[3]==>  [2, w: 11.0]  [4, w: 6.0]
[4]==>  [3, w: 6.0]  [5, w: 9.0]
[5]==>  [2, w: 14.0]  [4, w: 2.0]

|------------------------------------------|
```