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

Abdurrahman BULUT
1901042258

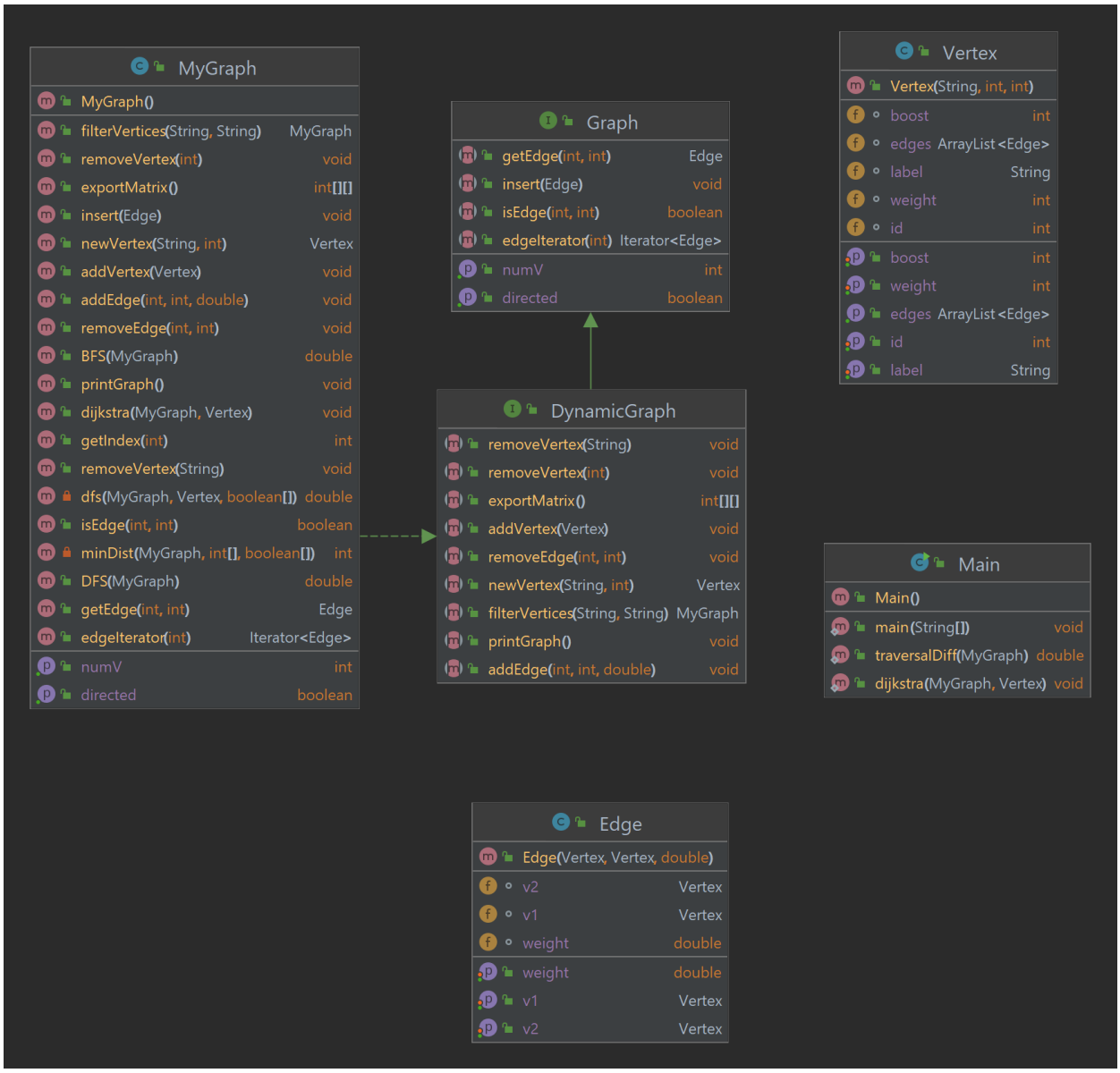
1. SYSTEM REQUIREMENTS

➢ Functional Requirements

♦ System

- openjdk 17.0.2 2022-01-18 LTS
- OpenJDK Runtime Environment Corretto-17.0.2.8.1 (build 17.0.2+8-LTS)
- OpenJDK 64-Bit Server VM Corretto-17.0.2.8.1 (build 17.0.2+8-LTS, mixed mode, sharing)

2. CLASS DIAGRAM

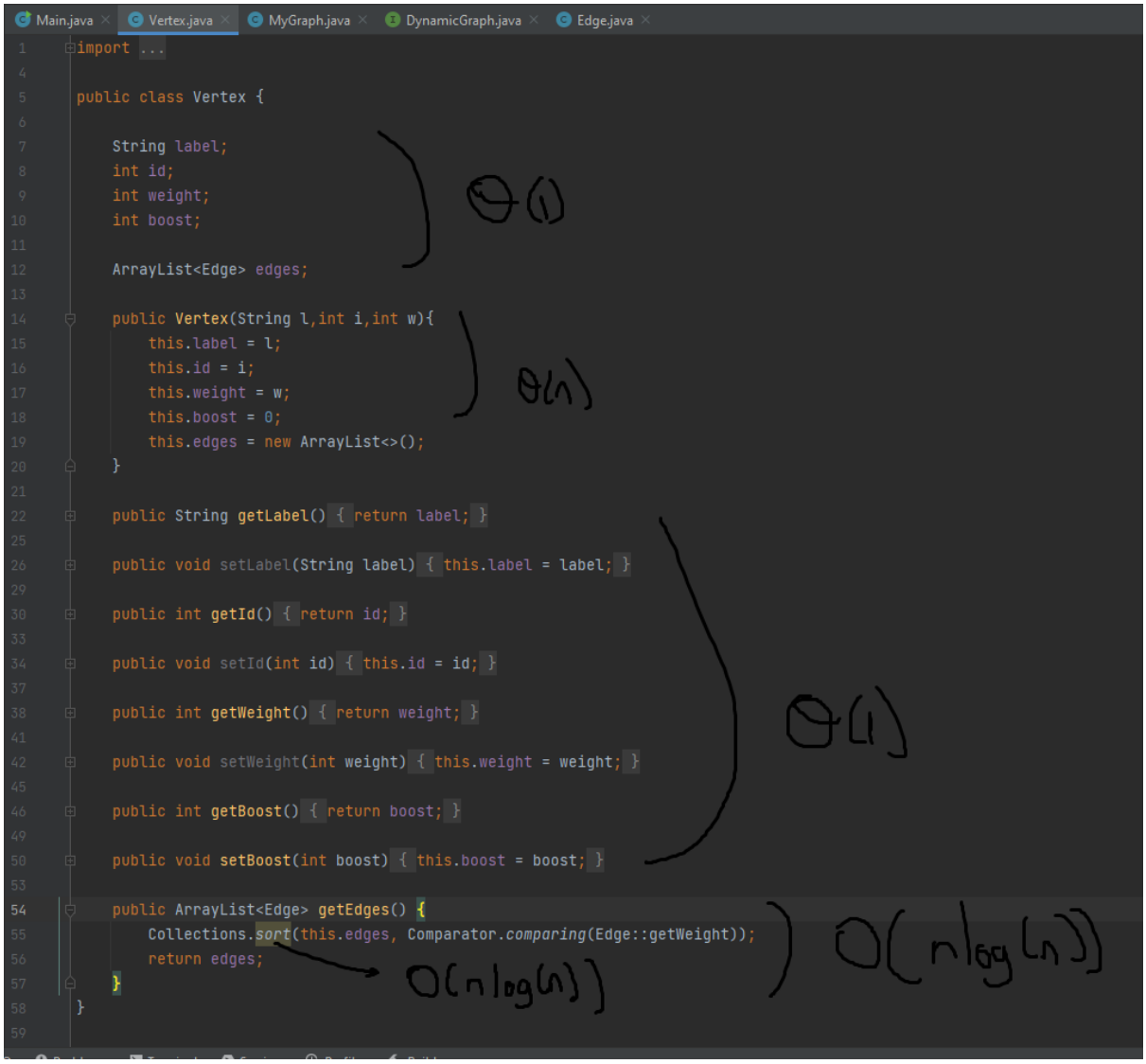


3. PROBLEM SOLUTION APPROACH

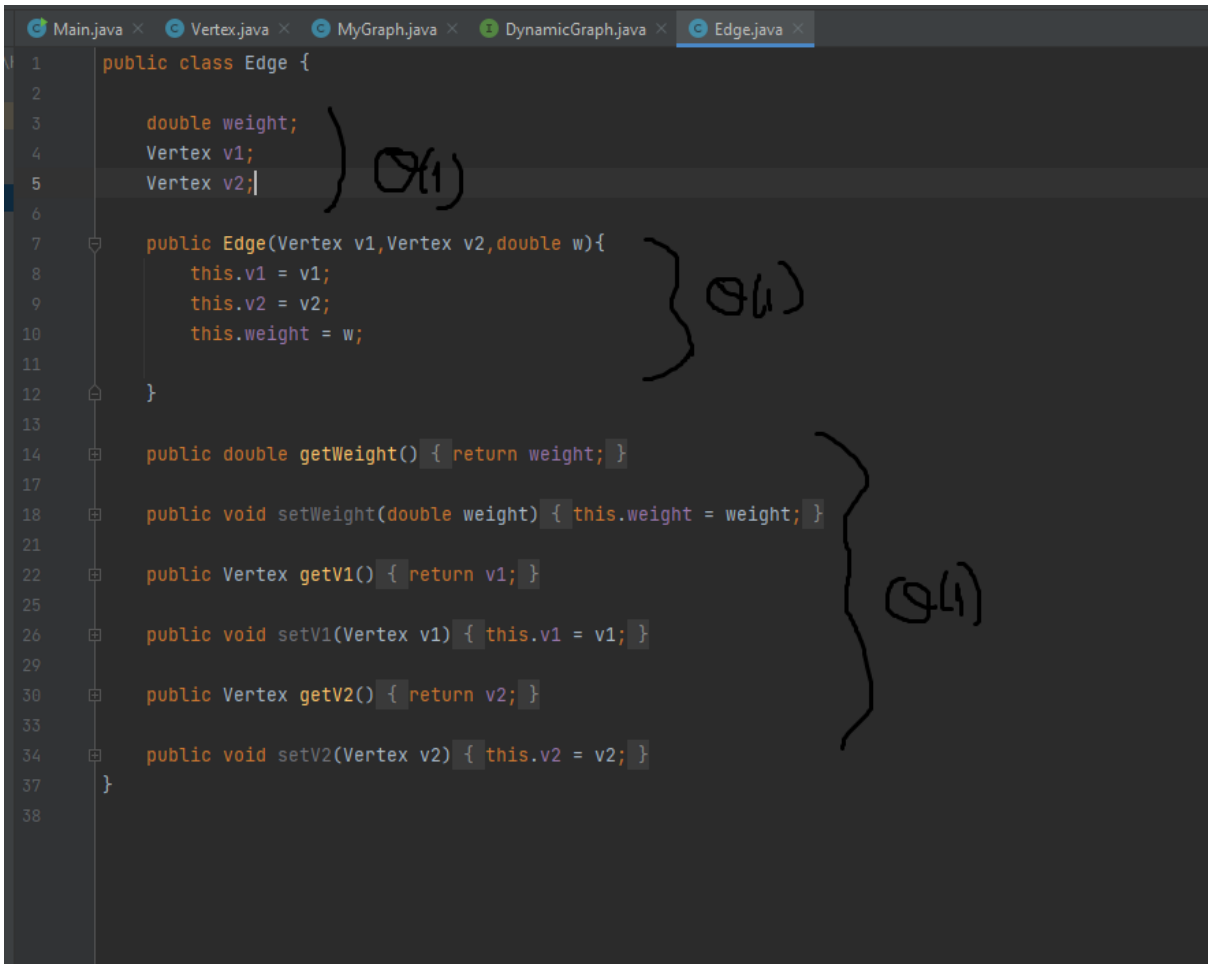
For vertexes I defined label, id, weight and boost. To store edges, I used Array List. An edge has a weight and 2 vertexes. I got Graph interface and created DynamicGraph interface by extend Graph interface. I created MyGraph class and implemented the functions which are in interfaces. Time complexity analyses are on the other page. For question 2, I simply wrote BFS and DFS functions. I used a visited Boolean array. For vertices, I created a linkedList. The function for question 2 is in Main.java file. It subtracts dfs from bfs and returns the result. For question 3, I implemented dijsktra() function.

4. TIME COMPLEXITY

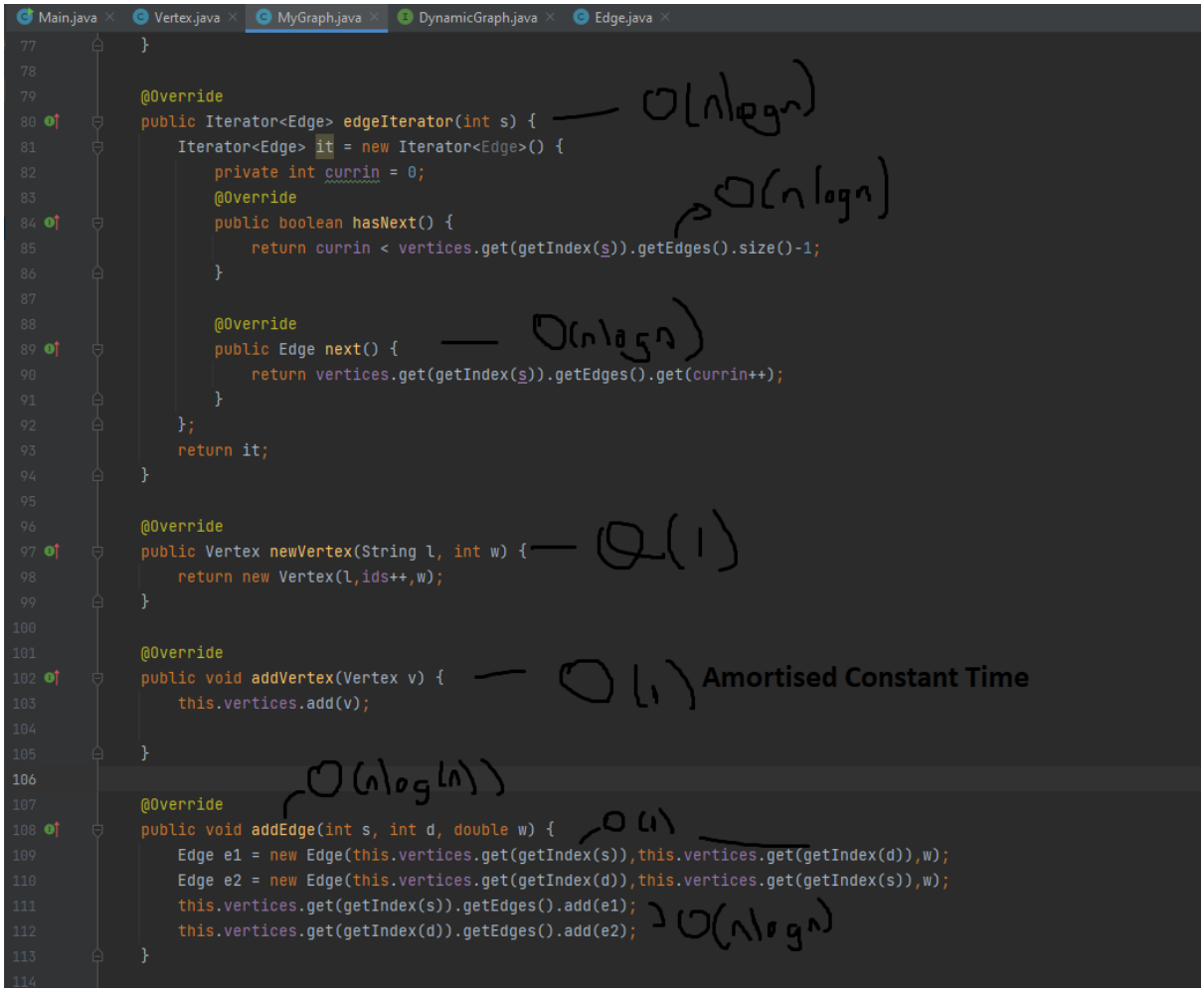
Vertex Class

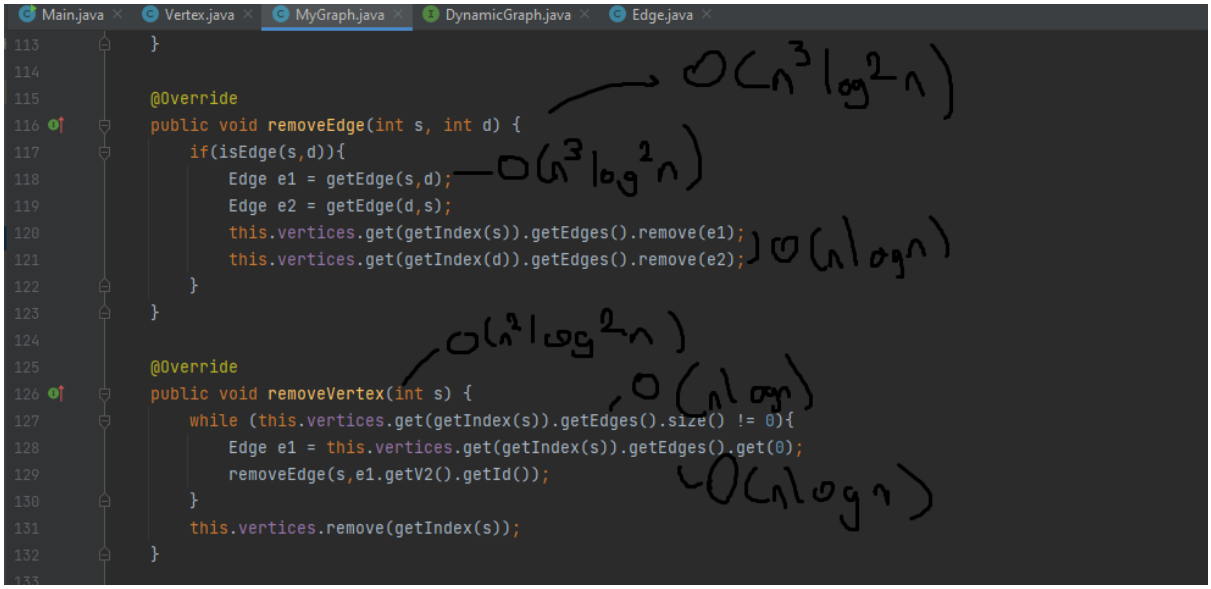


Edge Class


Main.java
Vertex.java
MyGraph.java
DynamicGraph.java
Edge.java
public class Edge {
double weight;
Vertex v1;
Vertex v2;
O(1)
public Edge(Vertex v1,Vertex v2,double w){
this.v1 = v1;
this.v2 = v2;
this.weight = w;
}
O(1)
public double getWeight() { return weight; }
public void setWeight(double weight) { this.weight = weight; }
public Vertex getV1() { return v1; }
public void setV1(Vertex v1) { this.v1 = v1; }
public Vertex getV2() { return v2; }
public void setV2(Vertex v2) { this.v2 = v2; }
O(1)
}

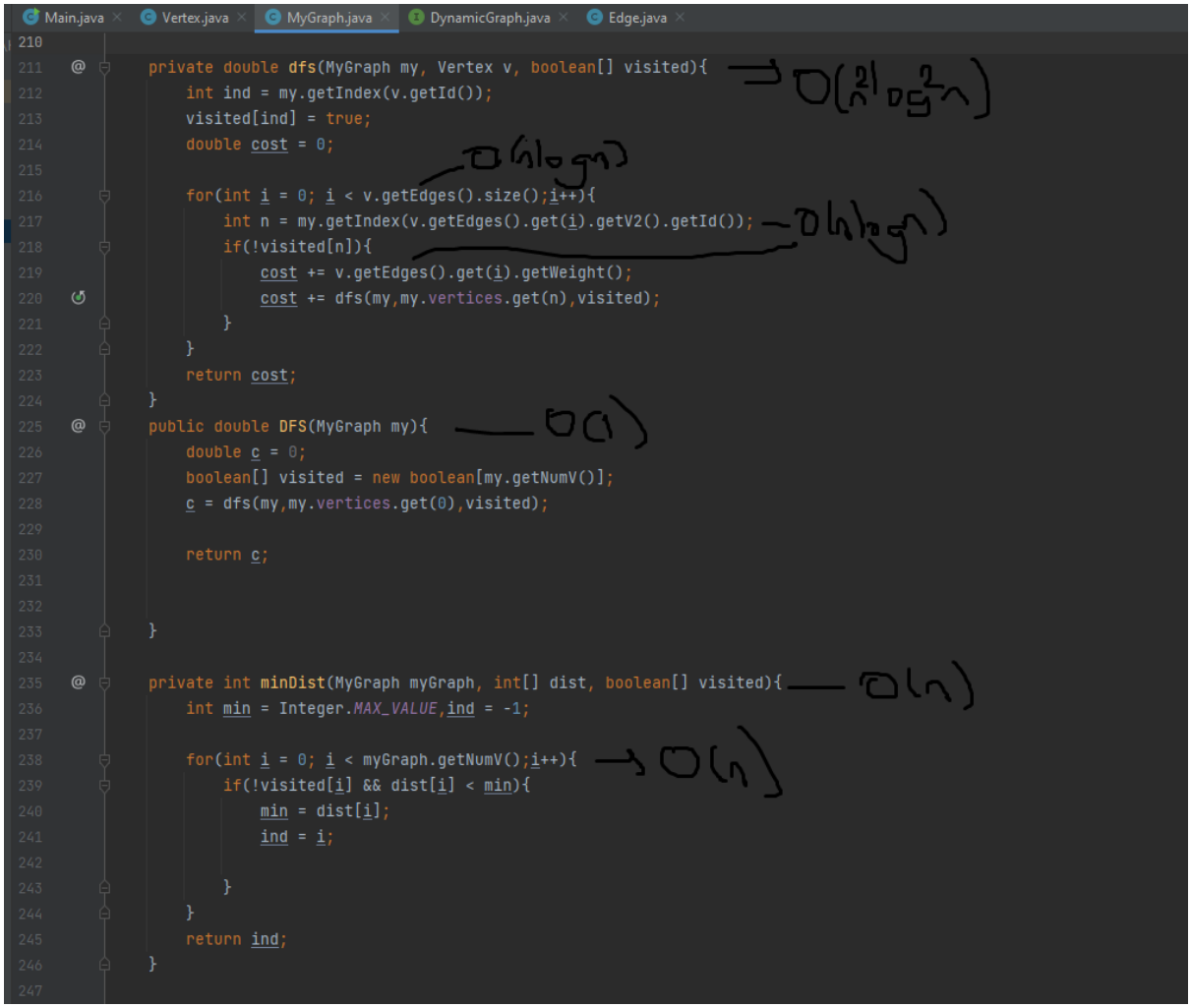
MyGraph Class


Main.java
Vertex.java
MyGraph.java
DynamicGraph.java
Edge.java
}
@Override
public Iterator<Edge> edgeIterator(int s) {
Iterator<Edge> it = new Iterator<Edge>() {
private int currin = 0;
@Override
public boolean hasNext() {
return currin < vertices.get(getIndex(s)).getEdges().size()-1;
}
@Override
public Edge next() {
return vertices.get(getIndex(s)).getEdges().get(currin++);
}
};
return it;
}
@Override
public Vertex newVertex(String l, int w) {
return new Vertex(l,ids++,w);
}
@Override
public void addVertex(Vertex v) {
this.vertices.add(v);
}
@Override
public void addEdge(int s, int d, double w) {
Edge e1 = new Edge(this.vertices.get(getIndex(s)),this.vertices.get(getIndex(d)),w);
Edge e2 = new Edge(this.vertices.get(getIndex(d)),this.vertices.get(getIndex(s)),w);
this.vertices.get(getIndex(s)).getEdges().add(e1);
this.vertices.get(getIndex(d)).getEdges().add(e2);
}
O(n log n)
O(n log n)
O(n log n)
O(1)
O(1) Amortised Constant Time
O(1)
O(n log n)


Main.java
Vertex.java
MyGraph.java
DynamicGraph.java
Edge.java
}
@Override
public void removeEdge(int s, int d) {
if(isEdge(s,d)){
Edge e1 = getEdge(s,d);
Edge e2 = getEdge(d,s);
this.vertices.get(getIndex(s)).getEdges().remove(e1);
this.vertices.get(getIndex(d)).getEdges().remove(e2);
}
}
@Override
public void removeVertex(int s) {
while (this.vertices.get(getIndex(s)).getEdges().size() != 0){
Edge e1 = this.vertices.get(getIndex(s)).getEdges().get(0);
removeEdge(s,e1.getV2().getId());
}
this.vertices.remove(getIndex(s));
}
O(n^3 log^2 n)
O(n^3 log^2 n)
O(n log n)
O(n^2 log^2 n)
O(n log n)
O(n log n)

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

    if(key.equals("label")){
        MyGraph my = new MyGraph();
        for(int i = 0; i< this.vertices.size();i++){
            Vertex v = this.vertices.get(i);
            Vertex nv = new Vertex(v.getLabel(),v.getId(),v.getWeight());
            if(nv.getLabel().equals(filter)){
                my.addVertex(nv);
            }

        }

        for(int i = 0; i < my.vertices.size();i++){
            Vertex nv = my.vertices.get(i);
            Vertex v2 = null;
            for(int j = 0; j < this.vertices.get(getIndex(nv.getId())).getEdges().size();j++){
                if(this.vertices.get(getIndex(nv.getId())).getEdges().get(j).getV2().getLabel().equals(filter)){
                    v2 = my.vertices.get(my.getIndex(this.vertices.get(getIndex(nv.getId())).getEdges().get(j).getV2().getId()));
                    Edge e1 = new Edge(nv,v2,this.vertices.get(getIndex(nv.getId())).getEdges().get(j).getWeight());

                    nv.getEdges().add(e1);

                }
            }

        }

        return my;
    }
    else {
        return null;
    }
}
```

```java
@Override
public int[][] exportMatrix() {
    int[][] d = new int[getNumV()][getNumV()];
    for(int i = 0; i < getNumV();i++){
        for(int j = 0; j < getNumV();j++){
            if(i == j){
                d[i][j] = 0;
            }else{
                if(getEdge(this.vertices.get(i).getId(),this.vertices.get(j).getId()) != null){
                    d[i][j] = (int)getEdge(this.vertices.get(i).getId(),this.vertices.get(j).getId()).getWeight();
                }else{
                    d[i][j]  = 0;
                }
            }
        }

    }
    return d;
}
public double BFS(MyGraph my){
    boolean[] visited = new boolean[my.vertices.size()];
    double cost = 0;
    visited[0] = true;
    LinkedList<Vertex> que = new LinkedList<>();
    que.add(my.vertices.get(0));

    while (que.size() != 0){
        Vertex v = que.poll();

        for(int i = 0; i < v.getEdges().size();i++){
            int ind = my.getIndex(v.getEdges().get(i).getV2().getId());
            if(!visited[ind]){
                visited[ind] = true;
                cost += v.getEdges().get(i).getWeight();
                que.add(my.vertices.get(ind));
            }
        }
    }
    return cost;
}
```



```
Main.java | Vertex.java | MyGraph.java | DynamicGraph.java | Edge.java

210
211 @    private double dfs(MyGraph my, Vertex v, boolean[] visited){
212         int ind = my.getIndex(v.getId());
213         visited[ind] = true;
214         double cost = 0;
215
216         for(int i = 0; i < v.getEdges().size();i++){
217             int n = my.getIndex(v.getEdges().get(i).getV2().getId());
218             if(!visited[n]){
219                 cost += v.getEdges().get(i).getWeight();
220 ↺               cost += dfs(my,my.vertices.get(n),visited);
221             }
222         }
223         return cost;
224     }
225 @   public double DFS(MyGraph my){
226         double c = 0;
227         boolean[] visited = new boolean[my.getNumV()];
228         c = dfs(my,my.vertices.get(0),visited);
229
230         return c;
231
232
233     }
234
235 @   private int minDist(MyGraph myGraph, int[] dist, boolean[] visited){
236         int min = Integer.MAX_VALUE,ind = -1;
237
238         for(int i = 0; i < myGraph.getNumV();i++){
239             if(!visited[i] && dist[i] < min){
240                 min = dist[i];
241                 ind = i;
242
243             }
244         }
245         return ind;
246     }
247
```

5. TEST CASES

```
Test 1: Create myGraph object

      MyGraph myGraph = new MyGraph();
      Pass


Test 2: Create vertex

      Vertex v1 = myGraph.newVertex("ada",4);
      Vertex v2 = myGraph.newVertex("a",8);
      Vertex v3 = myGraph.newVertex("a",8);
      Pass

Test 3: Set boost

      v2.setBoost(3);
      v1.setBoost(2);
      v3.setBoost(3);
      Pass

Test 4: Add vertex

      myGraph.addVertex(v1);
      myGraph.addVertex(v2);
      myGraph.addVertex(v3);
      Pass

Test 5: Add edge

      myGraph.addEdge(v1.getId(), v2.getId(), 4);
      myGraph.addEdge(v1.getId(), v3.getId(), 8);
      myGraph.addEdge(v2.getId(), v3.getId(), 6);
      Pass

Test 6: Print graph

       myGraph.printGraph();
       Pass

Test 7: Filter Vertices

      MyGraph m2 = myGraph.filterVertices("label","a");
      Pass
```

Test 8: BFS, DFS difference

```
System.out.println(traversalDiff(myGraph));
```

Pass

Test 9: Generate the adjacency matrix

```
myGraph.exportMatrix();
```

Pass

Test 10: Dijkstra algorithm

```
dijkstra(myGraph,v1);
```

Pass

Test 11: Remove vertex

```
myGraph.removeVertex("ada");
```

Pass

6. RUNNING AND RESULTS

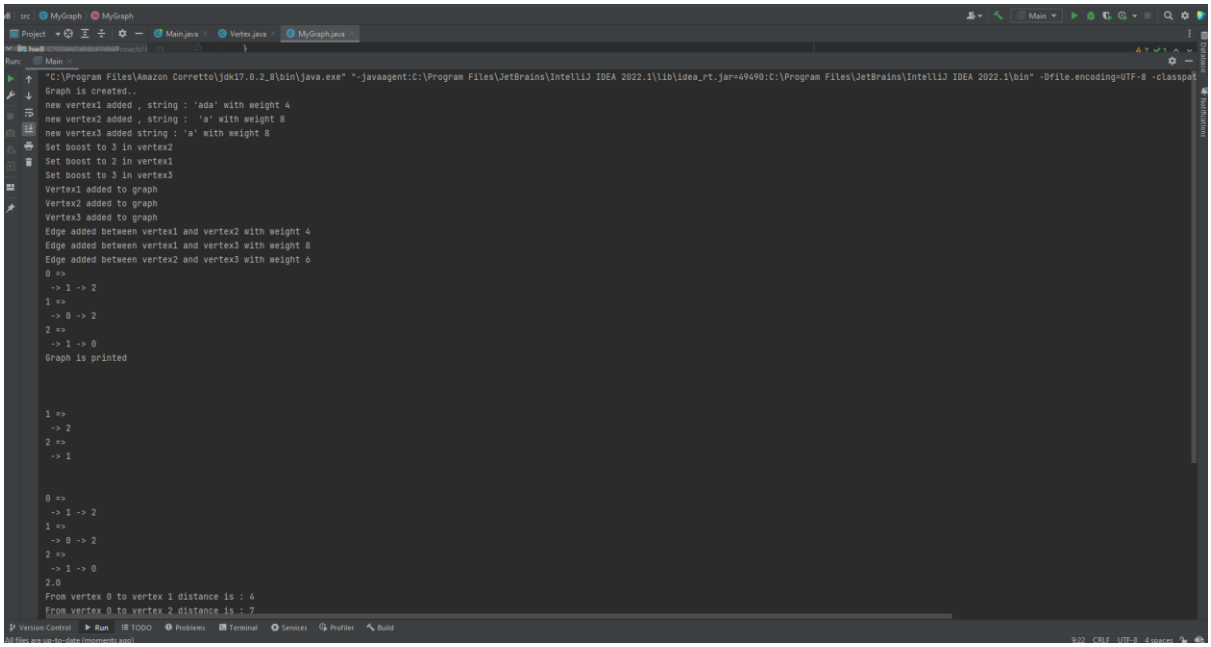
Main.java Vertex.java MyGraph.java

```java
public class Main {

    public static void main(String[] args){

        MyGraph myGraph = new MyGraph();
        System.out.println("Graph is created..");
        Vertex v1 = myGraph.newVertex( k: "ada", w: 4);
        System.out.println("new vertex1 added , string : 'ada' with weight 4");
        Vertex v2 = myGraph.newVertex( k: "a", w: 8);
        System.out.println("new vertex2 added , string :  'a' with weight 8");
        Vertex v3 = myGraph.newVertex( k: "a", w: 8);
        System.out.println("new vertex3 added string : 'a' with weight 8");

        v2.setBoost(3);
        System.out.println("Set boost to 3 in vertex2");
        v1.setBoost(2);
        System.out.println("Set boost to 2 in vertex1");
        v3.setBoost(3);
        System.out.println("Set boost to 3 in vertex3");

        myGraph.addVertex(v1);
        System.out.println("Vertex1 added to graph");
        myGraph.addVertex(v2);
        System.out.println("Vertex2 added to graph");
        myGraph.addVertex(v3);
        System.out.println("Vertex3 added to graph");

        myGraph.addEdge(v1.getId(), v2.getId(),  w: 4);
        System.out.println("Edge added between vertex1 and vertex2 with weight 4");
        myGraph.addEdge(v1.getId(), v3.getId(),  w: 8);
        System.out.println("Edge added between vertex1 and vertex3 with weight 8");
        myGraph.addEdge(v2.getId(), v3.getId(),  w: 6);
        System.out.println("Edge added between vertex2 and vertex3 with weight 6");

        myGraph.printGraph();
        System.out.println("Graph is printed");
        System.out.println("\n\n");

        MyGraph m2 = myGraph.filterVertices( key: "label", filter: "a");

        m2.printGraph();
```

Main.java | Vertex.java | MyGraph.java

```
        System.out.println("Graph is printed");
        System.out.println("\n\n");

        MyGraph m2 = myGraph.filterVertices( key: "label", filter: "a");

        m2.printGraph();
        System.out.println("\n");
        myGraph.printGraph();

        System.out.println(traversalDiff(myGraph));

        myGraph.exportMatrix();
        dijkstra(myGraph,v1);

        myGraph.removeVertex( s: "ada");
        myGraph.printGraph();
        myGraph.addVertex(v1);
        myGraph.addEdge(v1.getId(), v2.getId(), w: 4);
        myGraph.addEdge(v1.getId(),v3.getId(), w: 6);
        myGraph.printGraph();

        dijkstra(myGraph,v1);


    }

    public static double traversalDiff(MyGraph my){
        double c1 = my.BFS(my);
        double c2 = my.DFS(my);
        return c1-c2;

    }

    public static void dijkstra(MyGraph myGraph,Vertex v) { myGraph.dijkstra(myGraph,v); }
}
```

Problems | Terminal | Services | Profiler | Build




Run: Main
1 =>
-> 0 -> 2
2 =>
-> 1 -> 0
Graph is printed
1 =>
-> 2
2 =>
-> 1
0 =>
-> 1 -> 2
1 =>
-> 0 -> 2
2 =>
-> 1 -> 0
2.0
From vertex 0 to vertex 1 distance is : 4
From vertex 0 to vertex 2 distance is : 7
1 =>
-> 2
2 =>
-> 1
1 =>
-> 0 -> 2
2 =>
-> 1 -> 0
0 =>
-> 1 -> 2
From vertex 0 to vertex 1 distance is : 4
From vertex 0 to vertex 2 distance is : 6
Process finished with exit code 0
Version Control
Run
TODO
Problems
Terminal
Services
Profiler
Build
All files are up-to-date (moments ago)
9:22
CRLF
UTF-8
4 spaces