

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

**Abdurrahman BULUT  
1901042258**

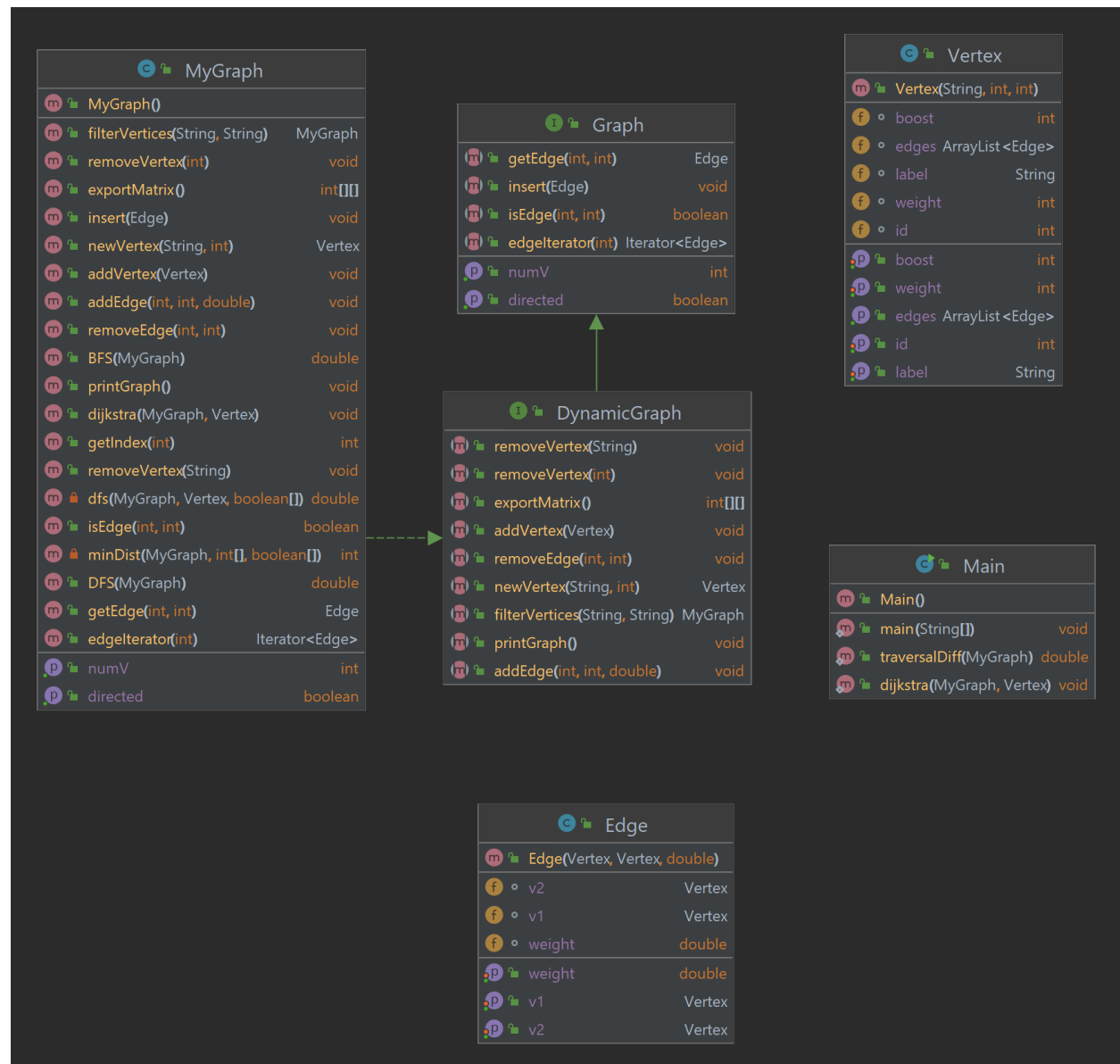
# 1. SYSTEM REQUIREMENTS

## ➤ 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

```
1  import ...
2
3
4
5  public class Vertex {
6
7      String label;
8      int id;
9      int weight;
10     int boost;
11
12     ArrayList<Edge> edges;
13
14     public Vertex(String l,int i,int w){
15         this.label = l;
16         this.id = i;
17         this.weight = w;
18         this.boost = 0;
19         this.edges = new ArrayList<>();
20     }
21
22     public String getLabel() { return label; }
23
24     public void setLabel(String label) { this.label = label; }
25
26     public int getId() { return id; }
27
28     public void setId(int id) { this.id = id; }
29
30     public int getWeight() { return weight; }
31
32     public void setWeight(int weight) { this.weight = weight; }
33
34     public int getBoost() { return boost; }
35
36     public void setBoost(int boost) { this.boost = boost; }
37
38     public ArrayList<Edge> getEdges() {
39         Collections.sort(this.edges, Comparator.comparing(Edge::getWeight));
40         return edges;
41     }
42 }
```

Handwritten annotations for time complexity:

- $\Theta(1)$  next to the field declarations (lines 7-10).
- $\Theta(1)$  next to the constructor (lines 14-19).
- $\Theta(1)$  next to the getter and setter methods (lines 22-36).
- $\Theta(n \log n)$  next to the `getEdges()` method (lines 38-40).

## Edge Class

```
1 public class Edge {
2
3     double weight;
4     Vertex v1;
5     Vertex v2;
6
7     public Edge(Vertex v1, Vertex v2, double w) {
8         this.v1 = v1;
9         this.v2 = v2;
10        this.weight = w;
11    }
12
13    public double getWeight() { return weight; }
14
15    public void setWeight(double weight) { this.weight = weight; }
16
17    public Vertex getV1() { return v1; }
18
19    public void setV1(Vertex v1) { this.v1 = v1; }
20
21    public Vertex getV2() { return v2; }
22
23    public void setV2(Vertex v2) { this.v2 = v2; }
24 }
25
26
27
28
```

Handwritten annotations in the image:

- A handwritten  $O(1)$  next to the class fields (lines 3-5).
- A handwritten  $O(1)$  next to the constructor (lines 7-11).
- A handwritten  $O(1)$  next to the getter and setter methods (lines 13-23).

## MyGraph Class

```

1  import java.util.ArrayList;
2  import java.util.Iterator;
3  import java.util.LinkedList;
4
5  public class MyGraph implements DynamicGraph{
6      ArrayList<Vertex> vertices;
7      private int ids;
8
9      public MyGraph(){
10         vertices = new ArrayList<>();
11         ids = 0;
12     }
13
14     public int getIndex(int s){
15         int res = -1;
16         for(int i = 0; i < this.vertices.size(); i++){
17             if(this.vertices.get(i).getId() == s){
18                 res = i;
19                 break;
20             }
21         }
22         return res;
23     }
24
25     @Override
26     public int getNumV() { return this.vertices.size(); }
27
28     @Override
29     public boolean isDirected() { return false; }
30
31     @Override
32     public void removeVertex(String s){
33         for(int i = 0; i < this.vertices.size(); i++){
34             if(this.vertices.get(i).getLabel().equals(s)){
35                 removeVertex(this.vertices.get(i).getId());
36             }
37         }
38     }
39 }

```

Handwritten annotations for the first image:

- A bracket on lines 5-12 is labeled  $O(1)$ .
- An arrow from line 16 to line 17 is labeled  $O(n)$ .
- An arrow from line 17 to line 18 is labeled "ArrayList, so  $O(1)$ ".
- An arrow from line 26 to line 27 is labeled  $O(1)$ .
- An arrow from line 29 to line 30 is labeled  $O(1)$ .
- An arrow from line 32 to line 33 is labeled  $O(n)$ .
- An arrow from line 33 to line 34 is labeled  $O(n)$ .
- An arrow from line 34 to line 35 is labeled  $O(1)$ .

```

44
45
46 @Override
47 public void insert(Edge edge) {
48     this.vertices.get(edge.getV1().getId()).getEdges().add(edge);
49     Edge e2 = new Edge(edge.getV2(), edge.getV1(), edge.getWeight());
50     this.vertices.get(edge.getV2().getId()).getEdges().add(e2);
51 }
52
53 @Override
54 public boolean isEdge(int s, int d) {
55     boolean res = false;
56     for(int i = 0; i < this.vertices.get(getIndex(s)).getEdges().size(); i++){
57         if(this.vertices.get(getIndex(s)).getEdges().get(i).getV2().getId() == d){
58             res = true;
59             break;
60         }
61     }
62     return res;
63 }
64
65 @Override
66 public Edge getEdge(int s, int d) {
67     Edge res = null;
68     if(isEdge(s, d)){
69         for(int i = 0; i < this.vertices.get(getIndex(s)).getEdges().size(); i++){
70             if(this.vertices.get(getIndex(s)).getEdges().get(i).getV2().getId() == d){
71                 res = this.vertices.get(getIndex(s)).getEdges().get(i);
72                 break;
73             }
74         }
75     }
76     return res;
77 }

```

Handwritten annotations for the second image:

- For the `insert` method:
  - $O(1)$  for line 46.
  - $O(n \log n)$  for line 47.
  - $O(n \log n)$  for line 48.
  - $O(n \log n)$  for line 49.
  - $O(1)$  for line 50.
  - $O(n \log n)$  for line 51.
- For the `isEdge` method:
  - $O(n \log n)$  for line 56.
  - $O(n \log n)$  for line 57.
  - $O(1)$  for line 58.
  - $O(n \log n)$  for line 59.
  - $O(n \log n)$  for line 60.
  - $O(1)$  for line 61.
  - $O(n \log n)$  for line 62.
- For the `getEdge` method:
  - $O(n^3 \log^2 n)$  for line 66.
  - $O(1)$  for line 67.
  - $O(n \log n)$  for line 68.
  - $O(n \log n)$  for line 69.
  - $O(n \log n)$  for line 70.
  - $O(n \log n)$  for line 71.
  - $O(1)$  for line 72.
  - $O(n \log n)$  for line 73.
  - $O(1)$  for line 74.
  - $O(1)$  for line 75.
  - $O(1)$  for line 76.
  - $O(1)$  for line 77.

```

Main.java x Vertex.java x MyGraph.java x DynamicGraph.java x Edge.java x
77 }
78
79 @Override
80 public Iterator<Edge> edgeIterator(int s) { —  $O(n \log n)$ 
81     Iterator<Edge> it = new Iterator<Edge>() {
82         private int currin = 0;
83         @Override
84         public boolean hasNext() {
85             return currin < vertices.get(getIndex(s)).getEdges().size()-1;
86         }
87
88         @Override
89         public Edge next() { —  $O(n \log n)$ 
90             return vertices.get(getIndex(s)).getEdges().get(currin++);
91         }
92     };
93     return it;
94 }
95
96 @Override
97 public Vertex newVertex(String l, int w) { —  $O(1)$ 
98     return new Vertex(l,ids++,w);
99 }
100
101 @Override
102 public void addVertex(Vertex v) { —  $O(1)$  Amortised Constant Time
103     this.vertices.add(v);
104 }
105
106 @Override
107 public void addEdge(int s, int d, double w) { —  $O(n \log n)$ 
108     Edge e1 = new Edge(this.vertices.get(getIndex(s)),this.vertices.get(getIndex(d)),w);
109     Edge e2 = new Edge(this.vertices.get(getIndex(d)),this.vertices.get(getIndex(s)),w);
110     this.vertices.get(getIndex(s)).getEdges().add(e1);
111     this.vertices.get(getIndex(d)).getEdges().add(e2); —  $O(n \log n)$ 
112 }
113
114

```

```

Main.java x Vertex.java x MyGraph.java x DynamicGraph.java x Edge.java x
113 }
114
115 @Override
116 public void removeEdge(int s, int d) { —  $O(n^3 \log^2 n)$ 
117     if(isEdge(s,d)){
118         Edge e1 = getEdge(s,d); —  $O(n^3 \log^2 n)$ 
119         Edge e2 = getEdge(d,s);
120         this.vertices.get(getIndex(s)).getEdges().remove(e1); —  $O(n \log n)$ 
121         this.vertices.get(getIndex(d)).getEdges().remove(e2); —  $O(n \log n)$ 
122     }
123 }
124
125 @Override
126 public void removeVertex(int s) { —  $O(n^2 \log^2 n)$ 
127     while (this.vertices.get(getIndex(s)).getEdges().size() != 0){
128         Edge e1 = this.vertices.get(getIndex(s)).getEdges().get(0); —  $O(n \log n)$ 
129         removeEdge(s,e1.getV2().getId()); —  $O(n \log n)$ 
130     }
131     this.vertices.remove(getIndex(s));
132 }
133

```



```

134 @Override
135 public MyGraph filterVertices(String key, String filter) {
136
137     if(key.equals("label")){
138         MyGraph my = new MyGraph();
139         for(int i = 0; i < this.vertices.size(); i++){
140             Vertex v = this.vertices.get(i);
141             Vertex nv = new Vertex(v.getLabel(), v.getId(), v.getWeight());
142             if(nv.getLabel().equals(filter)){
143                 my.addVertex(nv);
144             }
145         }
146
147         for(int i = 0; i < my.vertices.size(); i++){
148             Vertex nv = my.vertices.get(i);
149             Vertex v2 = null;
150             for(int j = 0; j < this.vertices.get(getIndex(nv.getId())).getEdges().size(); j++){
151                 if(this.vertices.get(getIndex(nv.getId())).getEdges().get(j).getV2().getLabel().equals(filter)){
152                     v2 = my.vertices.get(my.getIndex(this.vertices.get(getIndex(nv.getId())).getEdges().get(j).getV2().getId()));
153                     Edge e1 = new Edge(nv, v2, this.vertices.get(getIndex(nv.getId())).getEdges().get(j).getWeight());
154                     nv.getEdges().add(e1);
155                 }
156             }
157         }
158     }
159
160     return my;
161 }
162
163 else {
164     return null;
165 }
166
167 }
168
169

```

Handwritten annotations for the first code block:

- $O(n^4 \log^2 n)$  (pointing to the outer loop and the inner loop)
- $O(n)$  (pointing to the inner loop)
- $O(n)$  (pointing to the inner loop)
- $O(n)$  (pointing to the inner loop)
- $O(n^4 \log^2 n)$  (pointing to the outer loop)
- $O(n^3 \log^2 n)$  (pointing to the inner loop)
- $O(n \log n)$  (pointing to the inner loop)

```

170 @Override
171 public int[][] exportMatrix() {
172     int[][] d = new int[getNumV()][getNumV()];
173     for(int i = 0; i < getNumV(); i++){
174         for(int j = 0; j < getNumV(); j++){
175             if(i == j){
176                 d[i][j] = 0;
177             }else{
178                 if(getEdge(this.vertices.get(i).getId(), this.vertices.get(j).getId()) != null){
179                     d[i][j] = (int) getEdge(this.vertices.get(i).getId(), this.vertices.get(j).getId()).getWeight();
180                 }else{
181                     d[i][j] = 0;
182                 }
183             }
184         }
185     }
186     return d;
187 }
188
189 @Override
190 public double BFS(MyGraph my){
191     boolean[] visited = new boolean[my.vertices.size()];
192     double cost = 0;
193     visited[0] = true;
194     LinkedList<Vertex> que = new LinkedList<>();
195     que.add(my.vertices.get(0));
196
197     while (que.size() != 0){
198         Vertex v = que.poll();
199
200         for(int i = 0; i < v.getEdges().size(); i++){
201             int ind = my.getIndex(v.getEdges().get(i).getV2().getId());
202             if(!visited[ind]){
203                 visited[ind] = true;
204                 cost += v.getEdges().get(i).getWeight();
205                 que.add(my.vertices.get(ind));
206             }
207         }
208     }
209     return cost;
210 }

```

Handwritten annotations for the second code block:

- $O(n^3 \log^2 n)$  (pointing to the nested loops)
- $O(n^2 \log^2 n)$  (pointing to the BFS method)
- $O(n)$  (pointing to the BFS method)
- $O(n^2 \log^2 n)$  (pointing to the BFS method)
- $O(n \log n)$  (pointing to the BFS method)

```
210
211 @ private double dfs(MyGraph my, Vertex v, boolean[] visited){  $\Rightarrow O(n^2 \log n)$ 
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++){  $O(n \log n)$ 
217         int n = my.getIndex(v.getEdges().get(i).getV2().getId());  $O(n \log n)$ 
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){  $O(n)$ 
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 @ private int minDist(MyGraph myGraph, int[] dist, boolean[] visited){  $O(n)$ 
235     int min = Integer.MAX_VALUE, ind = -1;
236
237     for(int i = 0; i < myGraph.getNumV(); i++){  $\rightarrow O(n)$ 
238         if(!visited[i] && dist[i] < min){
239             min = dist[i];
240             ind = i;
241         }
242     }
243
244     return ind;
245 }
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

```
1 public class Main {
2
3     public static void main(String[] args){
4
5         MyGraph myGraph = new MyGraph();
6         System.out.println("Graph is created..");
7         Vertex v1 = myGraph.newVertex(k: "ada", wr: 4);
8         System.out.println("new vertex1 added , string : 'ada' with weight 4");
9         Vertex v2 = myGraph.newVertex(k: "a", wr: 8);
10        System.out.println("new vertex2 added , string : 'a' with weight 8");
11        Vertex v3 = myGraph.newVertex(k: "a", wr: 8);
12        System.out.println("new vertex3 added string : 'a' with weight 8");
13
14        v2.setBoost(3);
15        System.out.println("Set boost to 3 in vertex2");
16        v1.setBoost(2);
17        System.out.println("Set boost to 2 in vertex1");
18        v3.setBoost(3);
19        System.out.println("Set boost to 3 in vertex3");
20
21        myGraph.addVertex(v1);
22        System.out.println("Vertex1 added to graph");
23        myGraph.addVertex(v2);
24        System.out.println("Vertex2 added to graph");
25        myGraph.addVertex(v3);
26        System.out.println("Vertex3 added to graph");
27
28        myGraph.addEdge(v1.getId(), v2.getId(), wr: 4);
29        System.out.println("Edge added between vertex1 and vertex2 with weight 4");
30        myGraph.addEdge(v1.getId(), v3.getId(), wr: 8);
31        System.out.println("Edge added between vertex1 and vertex3 with weight 8");
32        myGraph.addEdge(v2.getId(), v3.getId(), wr: 6);
33        System.out.println("Edge added between vertex2 and vertex3 with weight 6");
34
35        myGraph.printGraph();
36        System.out.println("Graph is printed");
37        System.out.println("\n\n");
38
39        MyGraph m2 = myGraph.filterVertices(key: "label", filter: "a");
40
41        m2.printGraph();
```

```
36     System.out.println("Graph is printed");
37     System.out.println("\n\n");
38
39     MyGraph m2 = myGraph.filterVertices( key: "label", filter: "a");
40
41     m2.printGraph();
42     System.out.println("\n");
43     myGraph.printGraph();
44
45     System.out.println(traversalDiff(myGraph));
46
47     myGraph.exportMatrix();
48     dijkstra(myGraph,v1);
49
50     myGraph.removeVertex( s: "ada");
51     myGraph.printGraph();
52     myGraph.addVertex(v1);
53     myGraph.addEdge(v1.getId(), v2.getId(), w: 4);
54     myGraph.addEdge(v1.getId(),v3.getId(), w: 6);
55     myGraph.printGraph();
56
57     dijkstra(myGraph,v1);
58
59 }
60
61
62 @ public static double traversalDiff(MyGraph my){
63     double c1 = my.BFS(my);
64     double c2 = my.DFS(my);
65     return c1-c2;
66 }
67
68
69 @ public static void dijkstra(MyGraph myGraph,Vertex v) { myGraph.dijkstra(myGraph,v); }
70
71 }
72
73
```

```
src \MyGraph\MyGraph
Run: Main
C:\Program Files\Amazon Corretto\jdk17.0.2_8\bin\java.exe -javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2022.1\lib\idea_rt.jar=49490:C:\Program Files\JetBrains\IntelliJ IDEA 2022.1\bin -Dfile.encoding=UTF-8 -classpath
Graph is created.
new vertex1 added , string : 'ada' with weight 4
new vertex2 added , string : 'a' with weight 8
new vertex3 added string : 'a' with weight 8
Set boost to 3 in vertex2
Set boost to 2 in vertex1
Set boost to 3 in vertex3
Vertex added to graph
Vertex2 added to graph
Vertex3 added to graph
Edge added between vertex1 and vertex2 with weight 4
Edge added between vertex1 and vertex3 with weight 8
Edge added between vertex2 and vertex3 with weight 0
0 =>
-> 1 -> 2
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
```

```
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 (no commits yet)

9/10 CRLF UTF-8 4 spaces