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

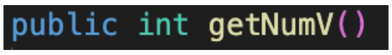
Süleyman Burak YASAR
1901042662

1. SYSTEM REQUIREMENTS

1.1. Problem Definition

Dynamic graph using graph interface and adjacency list representation

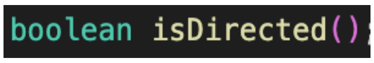
1.2. Functional Requirements



```
public int getNumV()
```

returns number of vertices

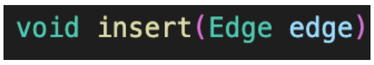
Time complexity: $\Theta(1)$



```
boolean isDirected()
```

Determine whether this is a directed graph.

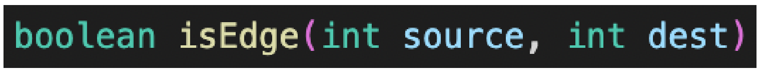
Time complexity: $\Theta(1)$



```
void insert(Edge edge)
```

Insert a new edge into the graph.

Time complexity: $\Theta(1)$



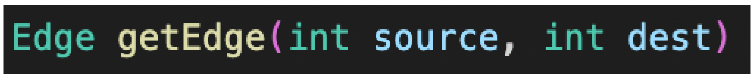
```
boolean isEdge(int source, int dest)
```

Determine whether an edge exists.

Time complexity: $\mathbf{O}(|E_u|)$

Best Case: $\Theta(1)$

Worst Case: $\Theta(V^2)$



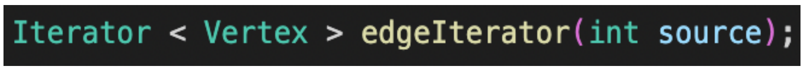
```
Edge getEdge(int source, int dest)
```

Get the edge between two vertices.

Time complexity: $\mathbf{O}(|E_u|)$

Best Case: $\Theta(1)$

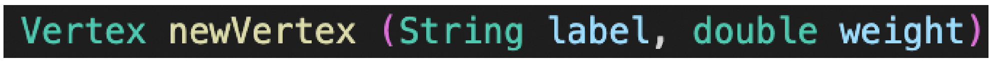
Worst Case: $\Theta(V^2)$



```
Iterator < Vertex > edgeIterator(int source);
```

Return iterator

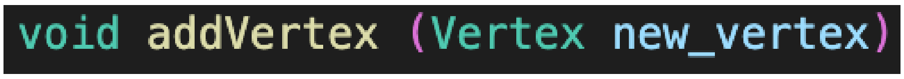
Time complexity: $\Theta(1)$



```
Vertex newVertex (String label, double weight)
```

Generate a new vertex by given parameters.

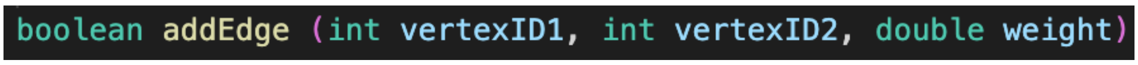
Time complexity: $\Theta(1)$



```
void addVertex (Vertex new_vertex)
```

Add the given vertex to the graph.

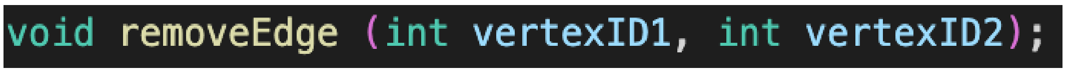
Time complexity: $\Theta(1)$



```
boolean addEdge (int vertexID1, int vertexID2, double weight)
```

Add an edge between the given two vertices in the graph.

Time complexity: $\Theta(1)$



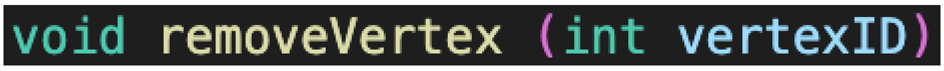
```
void removeEdge (int vertexID1, int vertexID2);
```

Remove the edge between the given two vertices.

Time complexity: $\mathbf{O}(|E_u|)$

Best Case: $\Theta(1)$

Worst Case: $\Theta(V^2)$



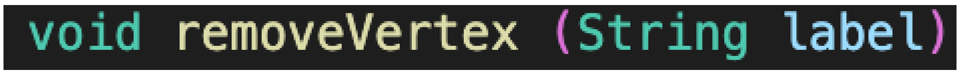
```
void removeVertex (int vertexID)
```

Remove the vertex from the graph with respect to the given vertex id.

Time complexity: $\mathbf{O}(|E_u|*V)$

Best Case: $\Theta(1)$

Worst Case: $\Theta(V^2)$



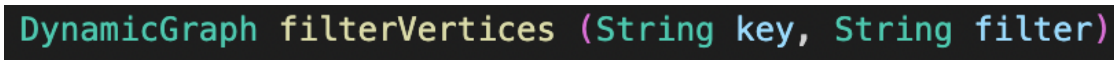
```
void removeVertex (String label)
```

Remove the vertices that have the given label from the graph.

Time complexity: $\mathbf{O}(|E_u|*V)$

Best Case: $\Theta(1)$

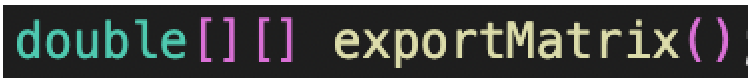
Worst Case: $\Theta(V^2)$



```
DynamicGraph filterVertices (String key, String filter)
```

Filter the vertices by the given user-defined property and returns a subgraph of the graph.

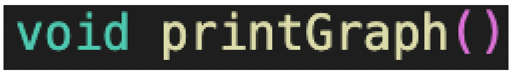
Time complexity: $\Theta(V^2)$



```
double[][] exportMatrix()
```

Generate the adjacency matrix representation of the graph and returns the matrix.

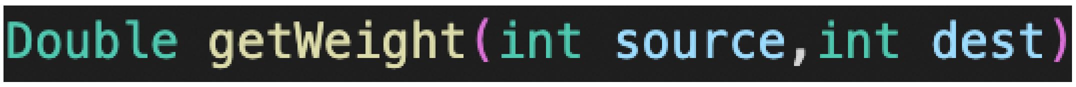
Time complexity: **O**(|E_u|*V)



```
void printGraph()
```

Print the graph in adjacency list format

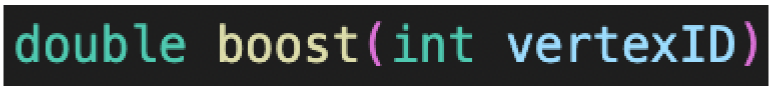
Time complexity: **O**(|E_u|*V)



```
Double getWeight(int source,int dest)
```

It returns the weight of the edge between the source and destination vertex

Time complexity: $\Theta(1)$

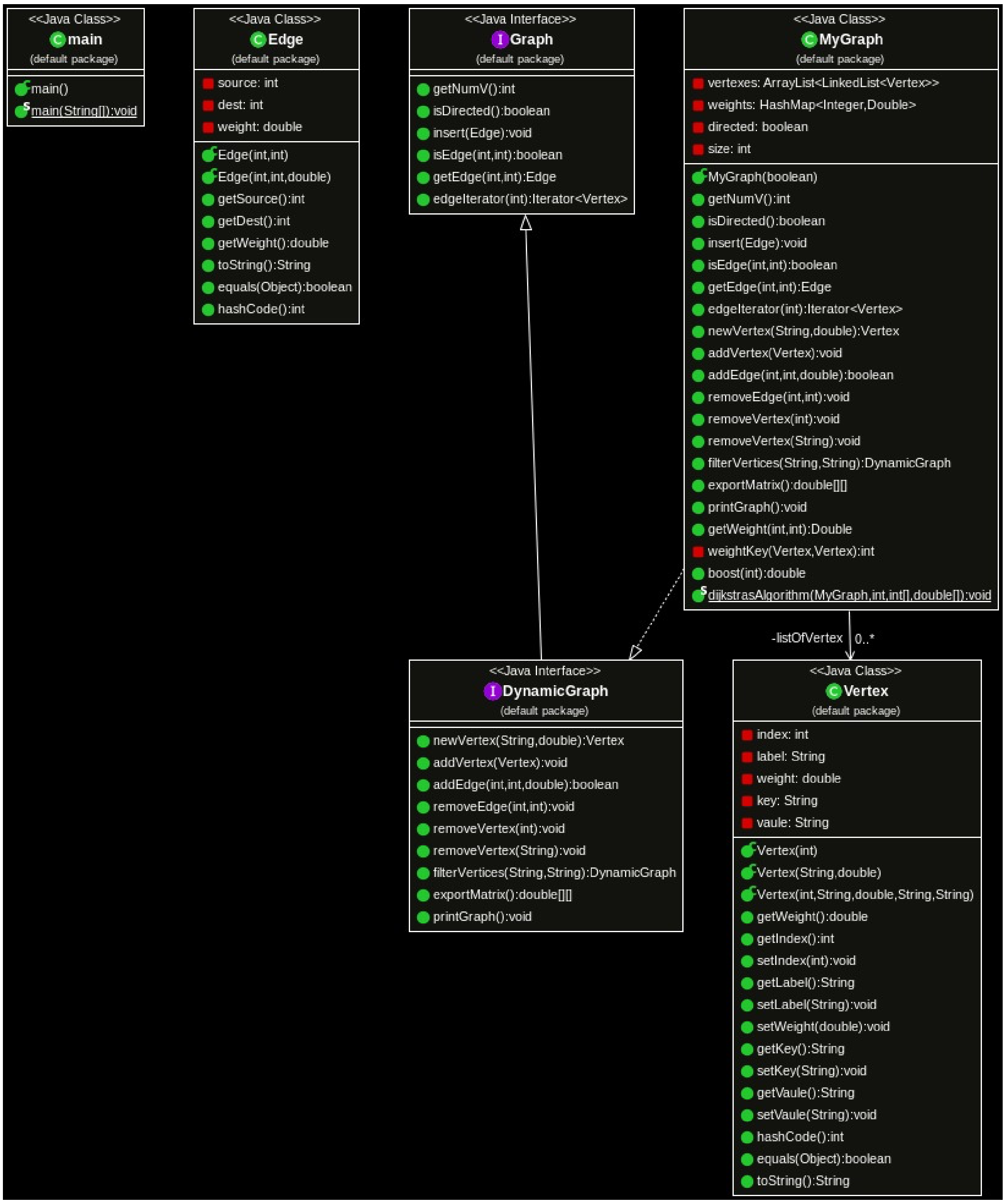


```
double boost(int vertexID)
```

Returns the boost value of a vertex

Time complexity: $\Theta(1)$

2. CLASS DIAGRAM



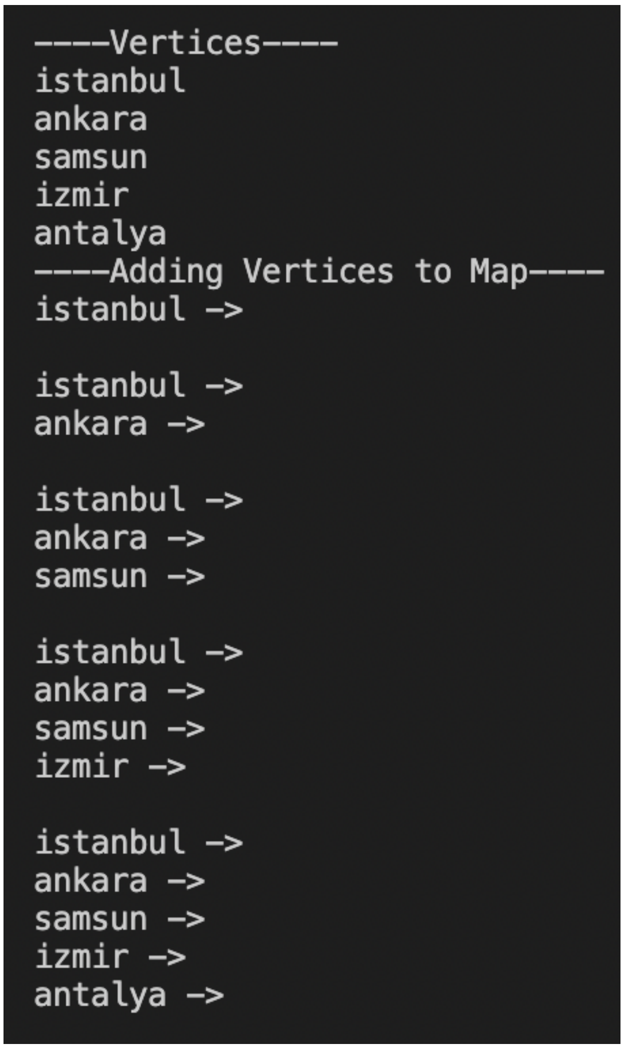
3. TEST CASES

Test Case	Test Values	Expected Result	Pass/Fail
Construct a graph	Directed = False	Construct a graph	PASS
Create a vertex (city0)	indexID = -1; Label : istanbul Weight: 34 Key: boost Value: 3	Creating a vertex with given values	PASS
Create a vertex (city1)	indexID = -1; Label : ankara Weight: 06 Key: boost Value: 2	Creating a vertex with given values	PASS
Create a vertex (city2)	indexID = -1; Label : samsun Weight: 55 Key: boost Value: 2	Creating a vertex with given values	PASS
Create a vertex (city3)	indexID = -1; Label : izmir Weight: 35 Key: boost Value: 9	Creating a vertex with given values	PASS
Create a vertex (city4)	indexID = -1; Label : antalya Weight: 05 Key: hmm Value: 24	Creating a vertex with given values	PASS
Add 5 vertex to graph	Add vertex(0-4)	Add 5 vertex to graph	PASS
Add edge from city0 to city1	Weight: 15 vertexID1: 0 vertexID2: 1	Create edge from city0 to city1	PASS

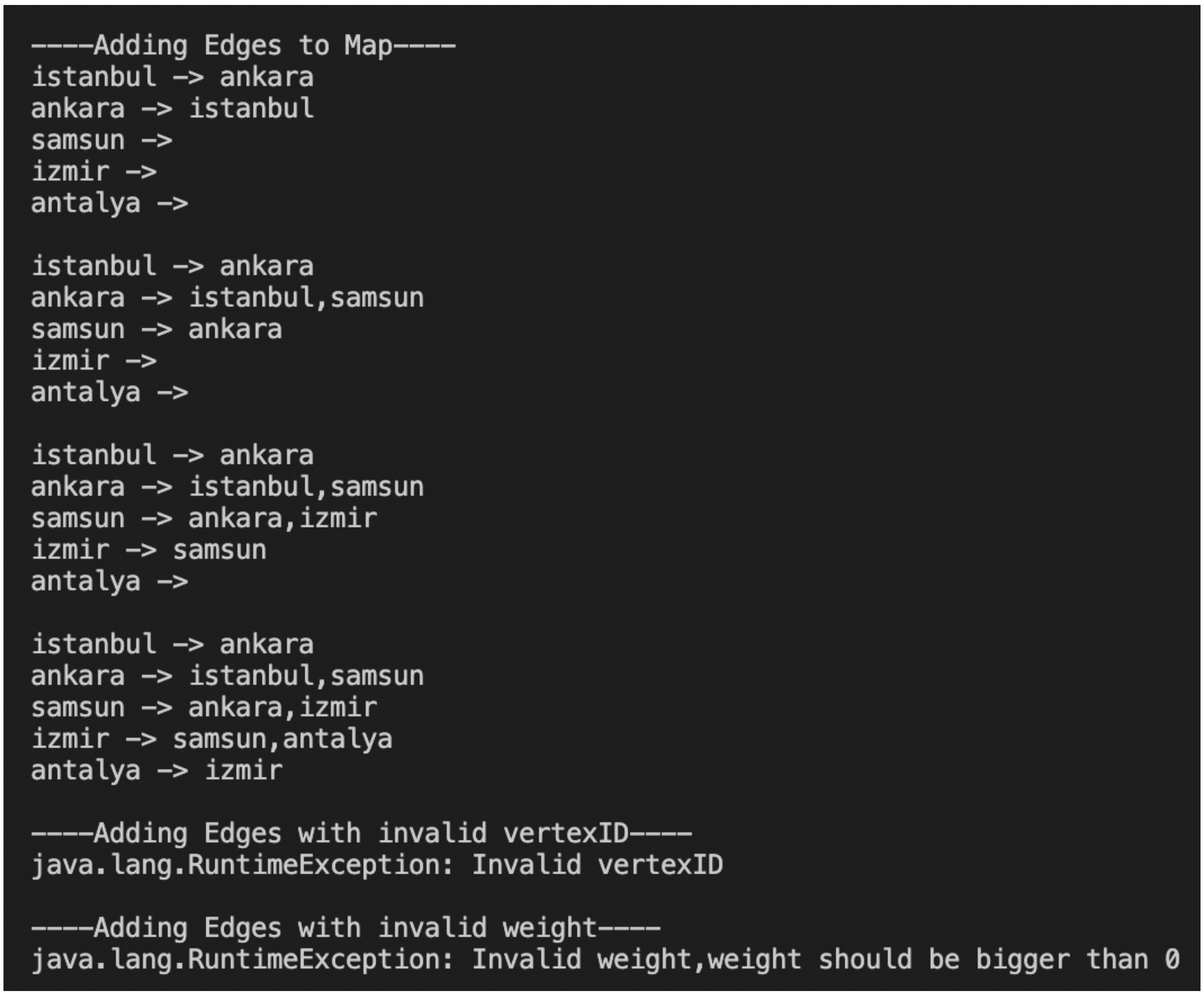
Add edge from city1 to city2	Weight: 8 vertexID1: 1 vertexID2: 2	Create edge from city1 to city2	PASS
Add edge from city2 to city3	Weight: 27 vertexID1: 2 vertexID2: 3	Create edge from city2 to city3	PASS
Add edge from city3 to city4	Weight: 45 vertexID1: 3 vertexID2: 4	Create edge from city3 to city4	PASS
Add edge with invalid index	Weight: 1 vertexID1: 9 vertexID2: 1	Exception Thrown	PASS
Add edge with invalid weight	Weight: -283 vertexID1: 1 vertexID2: 4	Exception Thrown	PASS
Add edge from vertex to same vertex	Weight: 1 vertexID1: 0 vertexID2: 0	return False	PASS
Export Adjacency Matrix of graph	-	return Adjacency Matrix of graph with two dimensional array	PASS
Create sub-graph with key and filter	key: boost filter : 2	return sub-graph of graph	PASS
Remove edge from city0 to city1	vertexID1: 0 vertexID2: 1	remove edge from city0 to city1	PASS
Remove edge from city2 to city4	vertexID1: 2 vertexID2: 4	remove edge from city2 to city4	PASS
Remove edge with invalid vertices id	vertexID1: 25 vertexID2: 4	Exception Thrown	PASS
Remove vertex with vertexID	vertexID: 0	remove city0	PASS
Remove vertex with label	label: “samsun”	remove vertex associated with “samsun”	PASS

Remove vertex with invalid vertexID	vertexID: 50	Exception Thrown	PASS
Get weight edge from “izmir”(city3) to “antalya”(city4)	vertexID1: 0 vertexID2: 2	return 45	PASS
Get weight edge with invalid vertexID	vertexID1: 50 vertexID2: 2	Exception Thrown	PASS
Get boost value	vertexID: 0	return 2	PASS
Get boost value with invalid	vertexID:50	Exception Thrown	PASS

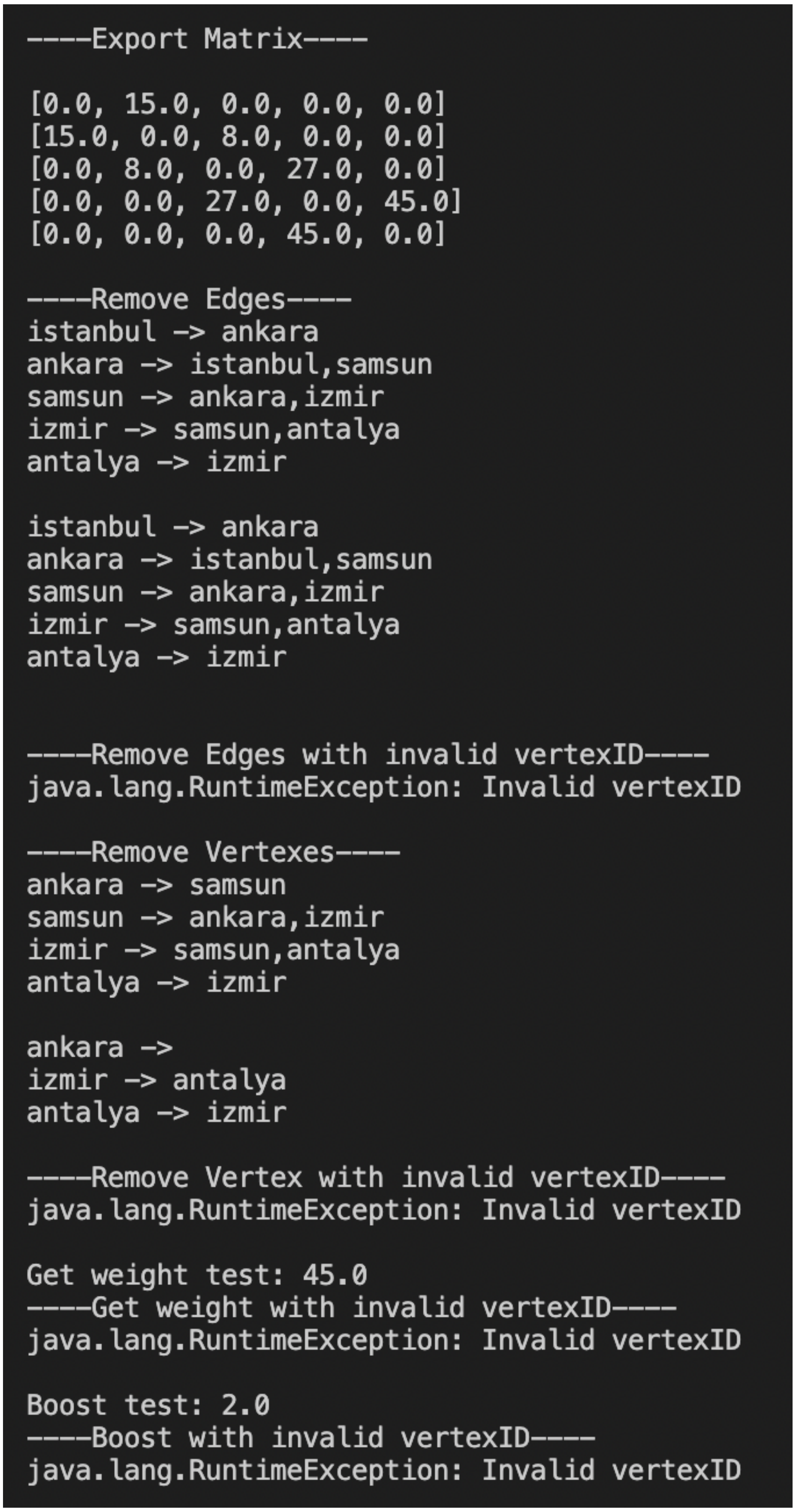
4. RUNNING AND RESULTS



```
----Vertices----
istanbul
ankara
samsun
izmir
antalya
----Adding Vertices to Map----
istanbul ->

istanbul ->
ankara ->

istanbul ->
ankara ->
samsun ->

istanbul ->
ankara ->
samsun ->
izmir ->

istanbul ->
ankara ->
samsun ->
izmir ->
antalya ->
```



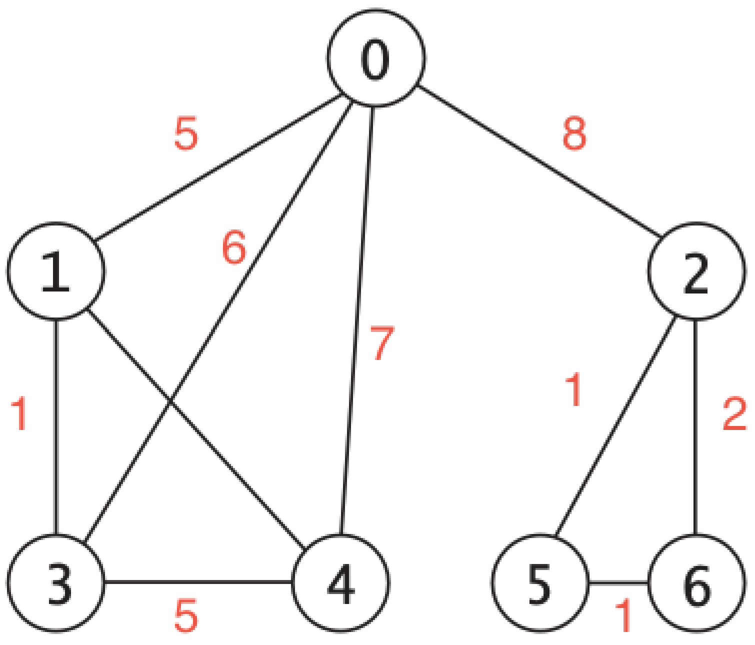
```
----Adding Edges to Map----
istanbul -> ankara
ankara -> istanbul
samsun ->
izmir ->
antalya ->

istanbul -> ankara
ankara -> istanbul,samsun
samsun -> ankara
izmir ->
antalya ->

istanbul -> ankara
ankara -> istanbul,samsun
samsun -> ankara,izmir
izmir -> samsun
antalya ->

istanbul -> ankara
ankara -> istanbul,samsun
samsun -> ankara,izmir
izmir -> samsun,antalya
antalya -> izmir

----Adding Edges with invalid vertexID----
java.lang.RuntimeException: Invalid vertexID

----Adding Edges with invalid weight----
java.lang.RuntimeException: Invalid weight,weight should be bigger than 0
```



```
----Export Matrix----

[0.0, 15.0, 0.0, 0.0, 0.0]
[15.0, 0.0, 8.0, 0.0, 0.0]
[0.0, 8.0, 0.0, 27.0, 0.0]
[0.0, 0.0, 27.0, 0.0, 45.0]
[0.0, 0.0, 0.0, 45.0, 0.0]

----Remove Edges----
istanbul -> ankara
ankara -> istanbul,samsun
samsun -> ankara,izmir
izmir -> samsun,antalya
antalya -> izmir

istanbul -> ankara
ankara -> istanbul,samsun
samsun -> ankara,izmir
izmir -> samsun,antalya
antalya -> izmir


----Remove Edges with invalid vertexID----
java.lang.RuntimeException: Invalid vertexID

----Remove Vertexes----
ankara -> samsun
samsun -> ankara,izmir
izmir -> samsun,antalya
antalya -> izmir

ankara ->
izmir -> antalya
antalya -> izmir

----Remove Vertex with invalid vertexID----
java.lang.RuntimeException: Invalid vertexID

Get weight test: 45.0
----Get weight with invalid vertexID----
java.lang.RuntimeException: Invalid vertexID

Boost test: 2.0
----Boost with invalid vertexID----
java.lang.RuntimeException: Invalid vertexID
```

Q2

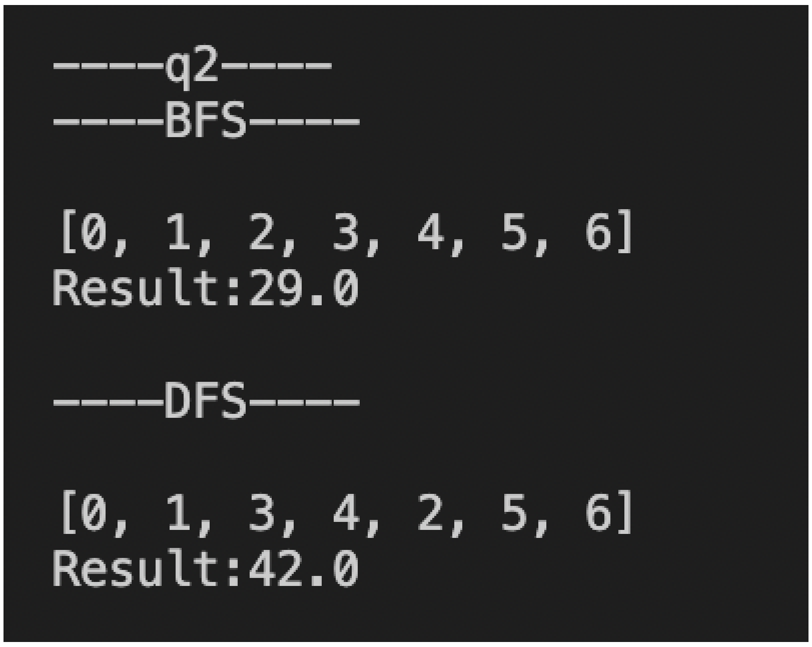
In bfs, I could not find the total length of the paths traversed, as in some cases there is a jump from one vertex to another. I only calculated the lengths when passing to adjacent vertices. However, since the calculation was made by going back from the way in dfs, it was possible to calculate all of them. So I may not have done exactly what was asked.



.

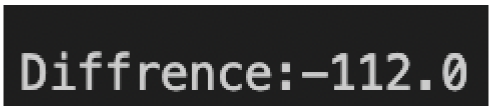
I used that graph to test BFS and DFS

When start 0



```
----q2----
----BFS----

[0, 1, 2, 3, 4, 5, 6]
Result:29.0

----DFS----

[0, 1, 3, 4, 2, 5, 6]
Result:42.0
```

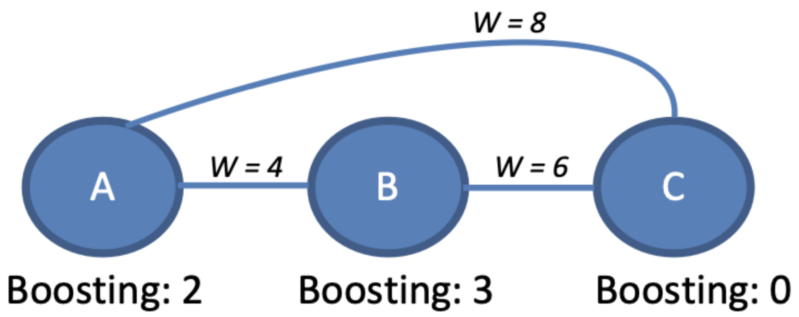
Result

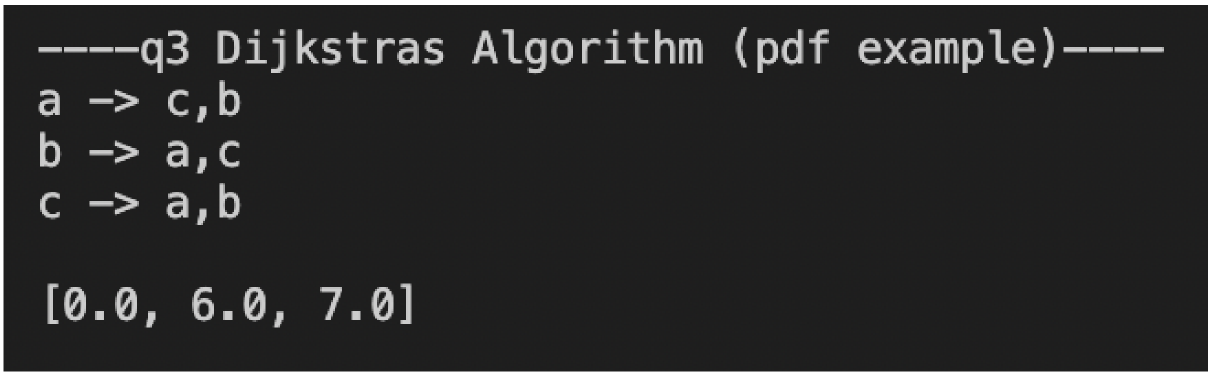


```
Diffrence:-112.0
```

Q3

Test Result with homework pdf example





```
----q3 Dijkstras Algorithm (pdf example)----
a -> c,b
b -> a,c
c -> a,b

[0.0, 6.0, 7.0]
```