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

Muhammed Sinan Pehlivanoğlu
1901042664

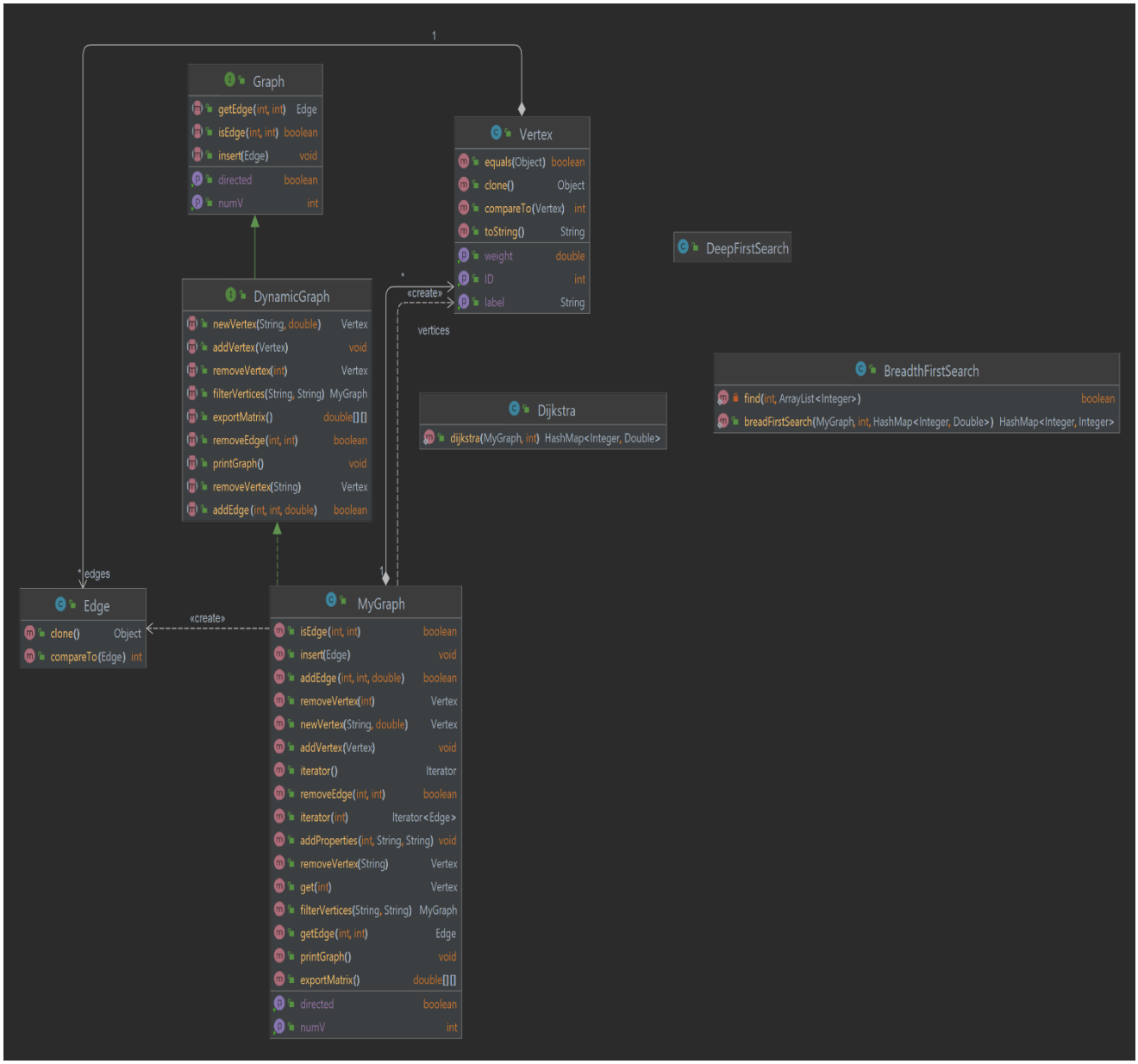
1. SYSTEM REQUIREMENTS

Operating System need to jdk and jre for start the program.
You need to enough space to store data's of program according to the how many you have objects.

2.PROBLEM SOLUTION APPROACH

- ***For first problem**, The HashMap is used to represent MyGraph.*
  HashMap has ID as key value and Vertex as value. Linked list of Edge is in the Vertex class. The Vertex is founded by its Id. Id is unique for each vertex. Id is static member variable in MyGraph. Id is increased for each created vertex.
- ***For second problem**, Only the BreadthSearch was implemented. The essence of BreadthSearch Algorithm is that the shortest path is considered as level order. The graph's edges is converted to the Priority Queue Data structure as ascending order of edge weight . Shortes edge is considered first. The traversal begin from start node. Adjacency nodes of parent nodes is added to the queue if the node is not identified. Then the nodes are removed from queue. This operation is done until the queue will be empty.*
- ***For third problem**, Dijkstra algorithm is used for finding the shortes path from started node to others. Firstly, distances to the started node from adjaceny nodes is found. Then others' that is not reachable from started node's distances set the the infinity. These distance values is stored. Then the min distance is selected from distances values and set as selected. Now started node is node with min distance. Above operations is done to the these node as weel until the all the node is selected.*

3.CLASS DIAGRAMS



4.TEST CASES

- *Create Vertex and its edges*
- *Add properties to graph's vertices.*
- *Print the graph.*
- *Get the matrix expression of graph.*

- *Remove edge from graph.*
- *Remove non Existed edge from graph.*
- *Remove vertex from graph.*
- *Remove non exited vertex from graph.*
- *Filter the graph according to the filter value.*
- *Apply BreadthSearch Algorithm.*
- *Apply the Dijkstra Algorithm.*

5.RUNNING COMMANDS AND RESULTS

Q1 RESULT

Adding Vertex

```
VERTICES
Source Node Boston,  Id: 0,  Boosted Val: 6.0-->
Source Node Istanbul,  Id: 1,  Boosted Val: 5.0-->
Source Node Madrid,  Id: 2,  Boosted Val: 4.0-->
Source Node Beijing,  Id: 3,  Boosted Val: 4.0-->
Source Node Rio,  Id: 4,  Boosted Val: 3.0-->
Source Node Cairo,  Id: 5,  Boosted Val: 7.0-->
Source Node hanoi,  Id: 6,  Boosted Val: 7.0-->
Source Node Capetown,  Id: 7,  Boosted Val: 5.0-->
Source Node Tripoli,  Id: 8,  Boosted Val: 5.0-->
Source Node Beirut,  Id: 9,  Boosted Val: 6.0-->
Source Node Moscow,  Id: 10,  Boosted Val: 2.0-->
```

Adding Edges to Vertices

```
VERTICES AND EDGES
Source Node Boston,  Id: 0,  Boosted Val: 6.0-->
    Dest Node: Istanbul,  Id: 1, Edge weight: 20.0
    Dest Node: Beijing,  Id: 3, Edge weight: 30.0
    Dest Node: Rio,  Id: 4, Edge weight: 15.0
Source Node Istanbul,  Id: 1,  Boosted Val: 5.0-->
    Dest Node: Boston,  Id: 0, Edge weight: 25.0
    Dest Node: Tripoli,  Id: 8, Edge weight: 30.0
    Dest Node: Moscow,  Id: 10, Edge weight: 25.0
    Dest Node: Cairo,  Id: 5, Edge weight: 20.0
    Dest Node: Beijing,  Id: 3, Edge weight: 20.0
Source Node Madrid,  Id: 2,  Boosted Val: 4.0-->
    Dest Node: Istanbul,  Id: 1, Edge weight: 25.0
    Dest Node: Rio,  Id: 4, Edge weight: 20.0
    Dest Node: Moscow,  Id: 10, Edge weight: 22.0
Source Node Beijing,  Id: 3,  Boosted Val: 4.0-->
    Dest Node: Beirut,  Id: 9, Edge weight: 22.0
    Dest Node: Moscow,  Id: 10, Edge weight: 22.0
    Dest Node: Boston,  Id: 0, Edge weight: 23.0
    Dest Node: Rio,  Id: 4, Edge weight: 24.0
    Dest Node: hanoi,  Id: 6, Edge weight: 25.0
    Dest Node: Istanbul,  Id: 1, Edge weight: 26.0
Source Node Rio,  Id: 4,  Boosted Val: 3.0-->
    Dest Node: Boston,  Id: 0, Edge weight: 25.0
    Dest Node: Capetown,  Id: 7, Edge weight: 25.0
    Dest Node: Madrid,  Id: 2, Edge weight: 25.0
```

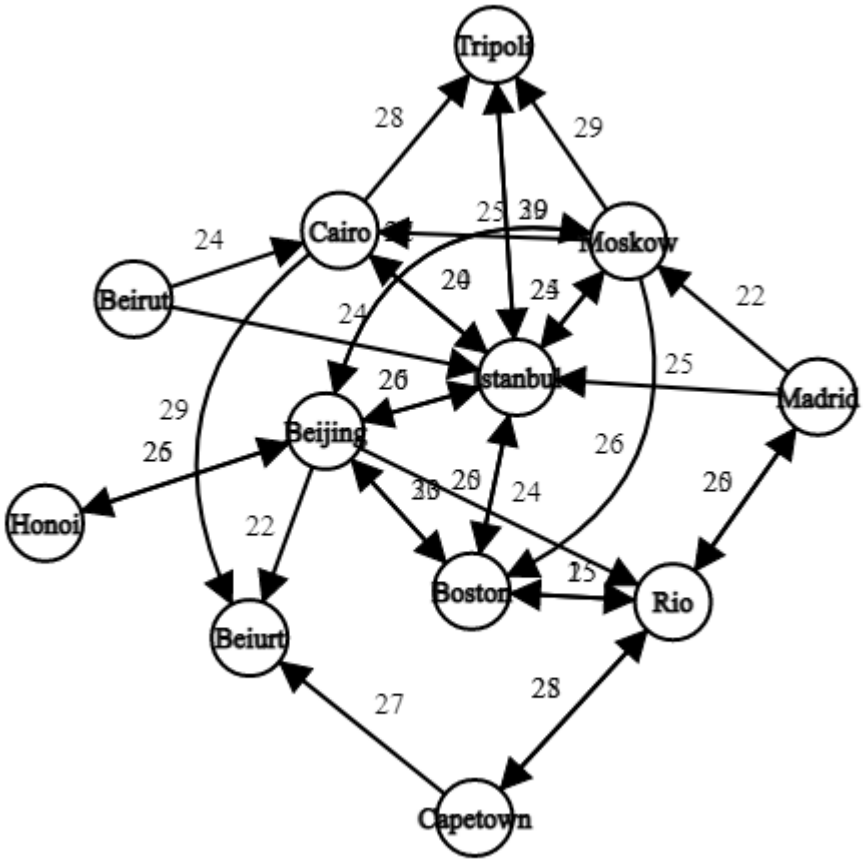
```
Source Node Cairo,  Id: 5,  Boosted Val: 7.0-->
    Dest Node: Istanbul,  Id: 1, Edge weight: 24.0
    Dest Node: Tripoli,  Id: 8, Edge weight: 28.0
    Dest Node: Beirut,  Id: 9, Edge weight: 29.0
Source Node hanoi,  Id: 6,  Boosted Val: 7.0-->
    Dest Node: Beijing,  Id: 3, Edge weight: 26.0
Source Node Capetown,  Id: 7,  Boosted Val: 5.0-->
    Dest Node: Beirut,  Id: 9, Edge weight: 27.0
    Dest Node: Rio,  Id: 4, Edge weight: 28.0
Source Node Tripoli,  Id: 8,  Boosted Val: 5.0-->
    Dest Node: Istanbul,  Id: 1, Edge weight: 29.0
Source Node Beirut,  Id: 9,  Boosted Val: 6.0-->
    Dest Node: Cairo,  Id: 5, Edge weight: 24.0
    Dest Node: Istanbul,  Id: 1, Edge weight: 24.0
Source Node Moscow,  Id: 10,  Boosted Val: 2.0-->
    Dest Node: Boston,  Id: 0, Edge weight: 26.0
    Dest Node: Beijing,  Id: 3, Edge weight: 27.0
    Dest Node: Tripoli,  Id: 8, Edge weight: 29.0
    Dest Node: Cairo,  Id: 5, Edge weight: 25.0
    Dest Node: Istanbul,  Id: 1, Edge weight: 24.0
```

THE GRAPH

ID-CITY

0:Boston - 1:Istanbul - 2:Madrid - 3:Beijing - 4:Rio – 5:Cairo – 6: Honoi – 7:Capetown

8:Tripoli – 9:Beirut – 10:Moskow



Matrix Expression of Graph

```
MATRIX EXPRESSION
0.0 20.0 0.0 30.0 15.0 0.0 0.0 0.0 0.0 0.0 0.0
25.0 0.0 0.0 20.0 0.0 20.0 0.0 0.0 30.0 0.0 25.0
0.0 25.0 0.0 0.0 20.0 0.0 0.0 0.0 0.0 0.0 22.0
23.0 26.0 0.0 0.0 24.0 0.0 25.0 0.0 0.0 22.0 22.0
25.0 0.0 25.0 0.0 0.0 0.0 0.0 25.0 0.0 0.0 0.0
0.0 24.0 0.0 0.0 0.0 0.0 0.0 0.0 28.0 29.0 0.0
0.0 0.0 0.0 26.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0 28.0 0.0 0.0 0.0 0.0 27.0 0.0
0.0 29.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
0.0 24.0 0.0 0.0 0.0 24.0 0.0 0.0 0.0 0.0 0.0
26.0 24.0 0.0 27.0 0.0 25.0 0.0 0.0 29.0 0.0 0.0
```

Removing Edge From Graph

```
REMOVE EDGE FROM GRAPH
BEIJING - ISTANBUL EDGE IS REMOVED
MADRID - MOSKOW EDGE IS REMOVED
```

Remove Non – Existed Edge

```
Remove Non-Existed Edge  false
```

The Graph After The Removing Edges

```
NEW GRAPH
Source Node Boston,  Id: 0,  Boosted Val: 6.0-->
    Dest Node: Istanbul,  Id: 1, Edge weight: 20.0
    Dest Node: Beijing,  Id: 3, Edge weight: 30.0
    Dest Node: Rio,  Id: 4, Edge weight: 15.0
Source Node Istanbul,  Id: 1,  Boosted Val: 5.0-->
    Dest Node: Boston,  Id: 0, Edge weight: 25.0
    Dest Node: Tripoli,  Id: 8, Edge weight: 30.0
    Dest Node: Moscow,  Id: 10, Edge weight: 25.0
    Dest Node: Cairo,  Id: 5, Edge weight: 20.0
    Dest Node: Beijing,  Id: 3, Edge weight: 20.0
Source Node Madrid,  Id: 2,  Boosted Val: 4.0-->
    Dest Node: Istanbul,  Id: 1, Edge weight: 25.0
    Dest Node: Rio,  Id: 4, Edge weight: 20.0
Source Node Beijing,  Id: 3,  Boosted Val: 4.0-->
    Dest Node: Beirut,  Id: 9, Edge weight: 22.0
    Dest Node: Moscow,  Id: 10, Edge weight: 22.0
    Dest Node: Boston,  Id: 0, Edge weight: 23.0
    Dest Node: Rio,  Id: 4, Edge weight: 24.0
    Dest Node: hanoi,  Id: 6, Edge weight: 25.0
Source Node Rio,  Id: 4,  Boosted Val: 3.0-->
    Dest Node: Boston,  Id: 0, Edge weight: 25.0
    Dest Node: Capetown,  Id: 7, Edge weight: 25.0
    Dest Node: Madrid,  Id: 2, Edge weight: 25.0
Source Node Cairo,  Id: 5,  Boosted Val: 7.0-->
    Dest Node: Istanbul,  Id: 1, Edge weight: 24.0
    Dest Node: Tripoli,  Id: 8, Edge weight: 28.0
    Dest Node: Beirut,  Id: 9, Edge weight: 29.0
Source Node hanoi,  Id: 6,  Boosted Val: 7.0-->
    Dest Node: Beijing,  Id: 3, Edge weight: 26.0
```

```
Source Node Capetown,  Id: 7,  Boosted Val: 5.0-->
    Dest Node: Beirut,  Id: 9, Edge weight: 27.0
    Dest Node: Rio,  Id: 4, Edge weight: 28.0
Source Node Tripoli,  Id: 8,  Boosted Val: 5.0-->
    Dest Node: Istanbul,  Id: 1, Edge weight: 29.0
Source Node Beirut,  Id: 9,  Boosted Val: 6.0-->
    Dest Node: Cairo,  Id: 5, Edge weight: 24.0
    Dest Node: Istanbul,  Id: 1, Edge weight: 24.0
Source Node Moscow,  Id: 10,  Boosted Val: 2.0-->
    Dest Node: Boston,  Id: 0, Edge weight: 26.0
    Dest Node: Beijing,  Id: 3, Edge weight: 27.0
    Dest Node: Tripoli,  Id: 8, Edge weight: 29.0
    Dest Node: Cairo,  Id: 5, Edge weight: 25.0
    Dest Node: Istanbul,  Id: 1, Edge weight: 24.0
```

Removing Vertex (EX:CAIRO)

```
REMOVE VERTEX FROM GRAPH
CAIRO IS REMOVED FORM GRAPH
```

Remove Non – Existed Vertex

```
Remove Non-Existed Vertex   :  Vertex is not found
```

The Graph After The Removing the Cairo

```
NEW GRAPH
Source Node Boston,  Id: 0,  Boosted Val: 6.0-->
    Dest Node: Istanbul,  Id: 1, Edge weight: 20.0
    Dest Node: Beijing,  Id: 3, Edge weight: 30.0
    Dest Node: Rio,  Id: 4, Edge weight: 15.0
Source Node Istanbul,  Id: 1,  Boosted Val: 5.0-->
    Dest Node: Boston,  Id: 0, Edge weight: 25.0
    Dest Node: Tripoli,  Id: 8, Edge weight: 30.0
    Dest Node: Moscow,  Id: 10, Edge weight: 25.0
    Dest Node: Beijing,  Id: 3, Edge weight: 20.0
Source Node Madrid,  Id: 2,  Boosted Val: 4.0-->
    Dest Node: Istanbul,  Id: 1, Edge weight: 25.0
    Dest Node: Rio,  Id: 4, Edge weight: 20.0
Source Node Beijing,  Id: 3,  Boosted Val: 4.0-->
    Dest Node: Beirut,  Id: 9, Edge weight: 22.0
    Dest Node: Moscow,  Id: 10, Edge weight: 22.0
    Dest Node: Boston,  Id: 0, Edge weight: 23.0
    Dest Node: Rio,  Id: 4, Edge weight: 24.0
    Dest Node: hanoi,  Id: 6, Edge weight: 25.0
Source Node Rio,  Id: 4,  Boosted Val: 3.0-->
    Dest Node: Boston,  Id: 0, Edge weight: 25.0
    Dest Node: Capetown,  Id: 7, Edge weight: 25.0
    Dest Node: Madrid,  Id: 2, Edge weight: 25.0
Source Node hanoi,  Id: 6,  Boosted Val: 7.0-->
    Dest Node: Beijing,  Id: 3, Edge weight: 26.0
Source Node Capetown,  Id: 7,  Boosted Val: 5.0-->
    Dest Node: Beirut,  Id: 9, Edge weight: 27.0
    Dest Node: Rio,  Id: 4, Edge weight: 28.0
Source Node Tripoli,  Id: 8,  Boosted Val: 5.0-->
    Dest Node: Istanbul,  Id: 1, Edge weight: 29.0
```

```
Source Node Beirut,  Id: 9,  Boosted Val: 6.0-->
    Dest Node: Istanbul,  Id: 1, Edge weight: 24.0
Source Node Moscow,  Id: 10,  Boosted Val: 2.0-->
    Dest Node: Boston,  Id: 0, Edge weight: 26.0
    Dest Node: Beijing,  Id: 3, Edge weight: 27.0
    Dest Node: Tripoli,  Id: 8, Edge weight: 29.0
    Dest Node: Istanbul,  Id: 1, Edge weight: 24.0
```

Filter the Graph

```
FILTER THE GRAPH ACCORDING TO CONTINENT OF CITY. SEARCH ASIAN CITIES
Source Node Beirut,  Id: 16,  Boosted Val: 6.0-->
Source Node Beijing,  Id: 14,  Boosted Val: 4.0-->
Source Node hanoi,  Id: 15,  Boosted Val: 7.0-->

FILTER THE EUROPE CITIES
Source Node Istanbul,  Id: 17,  Boosted Val: 5.0-->
Source Node Madrid,  Id: 18,  Boosted Val: 4.0-->
Source Node Moscow,  Id: 19,  Boosted Val: 2.0-->

FILTER THE AFRICA CITIES
Source Node Capetown,  Id: 20,  Boosted Val: 5.0-->
Source Node Tripoli,  Id: 21,  Boosted Val: 5.0-->
```

Q2 (DEEP FIRST SEARCH WAS NOT IMPLEMENTED)

BreadthSearch Algorithm

0:Boston - 1:Istanbul - 2:Madrid - 3:Beijing - 4:Rio – 6: Honoi – 7:Capetown

8:Tripoli – 9:Beirut – 10:Moskow

(5-Cairo is deleted)

```
BREADTH FIRST SEARCH ALGORITHM


DISTANCE FROM THE BOSTON TO OTHER USING BREADTH SEARCH
Source Id: 0 &  Distance : 0.0
Source Id: 1 &  Distance : 20.0
Source Id: 2 &  Distance : 40.0
Source Id: 3 &  Distance : 30.0
Source Id: 4 &  Distance : 15.0
Source Id: 6 &  Distance : 55.0
Source Id: 7 &  Distance : 40.0
Source Id: 8 &  Distance : 50.0
Source Id: 9 &  Distance : 52.0
Source Id: 10 &  Distance : 45.0
```

Parent of Each Cities According to the BreadthSearch.

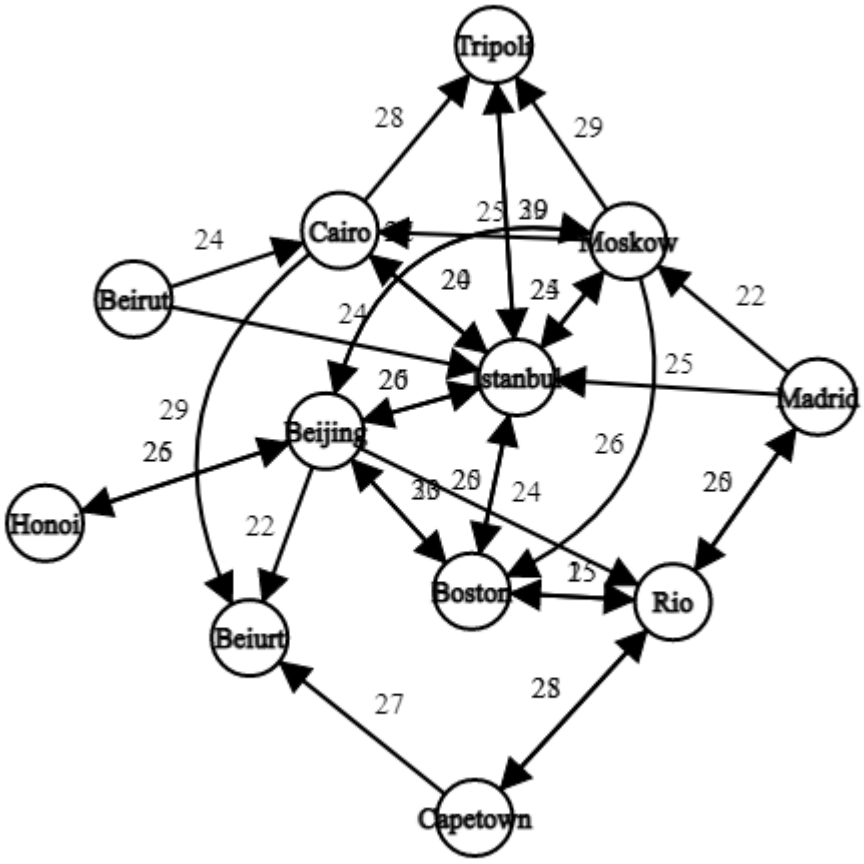
```
PARENT OF EACH CITIESA
Source Id : 1 &  Parent Id : 0
Source Id : 2 &  Parent Id : 4
Source Id : 3 &  Parent Id : 0
Source Id : 4 &  Parent Id : 0
Source Id : 6 &  Parent Id : 3
Source Id : 7 &  Parent Id : 4
Source Id : 8 &  Parent Id : 1
Source Id : 9 &  Parent Id : 3
Source Id : 10 &  Parent Id : 1
```

Q3

DIKSTRA ALGORITHM.

0:Boston - 1:Istanbul - 2:Madrid - 3:Beijing - 4:Rio – 5:Cairo – 6: Honoi – 7:Capetown

8:Tripoli – 9:Beirut – 10:Moskow



```
DIJKSTRA (DISTANCE FROM BOSTON TO OTHER CITIES)
City ID: 0 &  Distance: 0.0
City ID: 1 &  Distance: 20.0
City ID: 2 &  Distance: 37.0
City ID: 3 &  Distance: 30.0
City ID: 4 &  Distance: 15.0
City ID: 5 &  Distance: 35.0
City ID: 6 &  Distance: 51.0
City ID: 7 &  Distance: 37.0
City ID: 8 &  Distance: 45.0
City ID: 9 &  Distance: 48.0
City ID: 10 &  Distance: 40.0
```

6.TIME COMPLEXITIES

- **NewVertex Method**

```
12 usages
@Override
public Vertex newVertex(String label, double weight) {
    ++ID;
    return new Vertex(label, weight,ID);
}
```

T(n) = θ(1).

- **AddVertex Method**

```java
@Override
public void addVertex(Vertex vertex) {

    vertices.put(vertex.getID(),vertex);

}
```

Inserting the HashMap.
T(N) = O(1).

- **AddEdge Method**

```java
35 usages
@Override
public boolean addEdge(int ver1Id, int ver2Id,double weight) {
    Vertex ver1 = vertices.get(ver1Id);
    Vertex ver2  = vertices.get(ver2Id);
    boolean res = false;
    if(ver1 != null || ver2 != null /*&& ver2List !=null*/){
        //ver1 = (Vertex) getVertexById(ver1Id).clone();
        ver1.edges.add(new Edge(weight,ver2Id,ver1Id));
        if(this.directed){
            ver2.edges.add(new Edge(weight,ver1Id,ver1Id));
        }
        res = true;
    }

    return res;
}
```

HashMap get method Time Complexity = O(1).
Linked List Add method Time Complexity = θ(1).
Average Running Time Complexity is = O(1).

- **RemoveEdge**

```java
@Override
public boolean removeEdge(int ver1Id, int ver2Id) {

    Edge deleted =null,deleted2=null;
    boolean res = false;

    List<Edge> edge1List = vertices.get(ver1Id).edges;
    if(edge1List != null){
        for (Edge edge: edge1List) {
            if(edge.destId == ver2Id){
                deleted = edge;
                break;
            }
        }
        edge1List.remove(deleted);
        res = true;
    }

    if(!directed) {
        List<Edge> edgeList = vertices.get(ver2Id).edges;
        if (edge1List != null && edgeList != null) {
            for (Edge edge : edgeList) {
                if (edge.destId == ver2Id) {
                    deleted2 = edge;
                    break;
                }
            }
            edgeList.remove(deleted2);
        }
        else res = false;
    }
    return res;
}
```

HashMap get method Time Complexity = O(1).

Finding deleted edge Time Complexity (First If Block) = O(E_v).

Best Case: if the graph is directed = O(E_v).

Worst Case : if the graph is directed = O(E_v).

- **RemoveVertex**

```java
@SuppressWarnings("unchecked")
// It's removing the vertex from the graph.
@Override
public Vertex removeVertex(int verId) {
    Vertex vertex = this.get(verId);

    Set<Map.Entry<Integer,Vertex>> entrySet = this.vertices.entrySet();
    Iterator<Map.Entry<Integer,Vertex>> iter = entrySet.iterator();

    while (iter.hasNext()){
        Map.Entry<Integer,Vertex> entry =  iter.next();
        Vertex vertex1 = entry.getValue();
        Edge deleted = null;
        for (Edge edge: vertex1.edges) {
            if(edge.destId == verId){
                deleted = edge;
                break;
            }
        }
        if(deleted != null){
            vertex1.edges.remove(deleted);
        }
    }

    this.vertices.remove(verId);
    return  vertex;
}
```

Traverse All Vertex = O(V).
Traverse All Edge of Vertex = O(E_v).
Delete Edge From LinkedList = O(E_v).
Remove Vertex From HashMap = Average Time Complexity = O(1).
Total Average Time Complexity = O(V*E_v).

- **FilterVertices**

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

    MyGraph newGraph = new MyGraph( isDirected: false);
    Iterator iterator = this.iterator();
    while (iterator.hasNext()){
        Map.Entry<Integer,Vertex> entry = (Map.Entry<Integer,Vertex>) iterator.next();
        Vertex vertex = entry.getValue();
        String val = vertex.properties.get(key);
        if(val != null) {
            if (vertex.properties.get(key).equals(filter)) {
                Vertex newVertex = this.newVertex(vertex.getLabel(),vertex.getWeight());
                newGraph.addVertex(newVertex);
            }
        }

    }



    return newGraph;
}
```

Traverse All Vertex = O(V).
Creating new Vertex = O(1).
Adding new Vertex to HashMap = O(1).
So Time Complexity is = O(V).

- **ExportMatrix**

```java
@Override
public double[][] exportMatrix() {
    Set entrySet = vertices.entrySet();

    // Obtaining an iterator for the entry set
    Iterator it = entrySet.iterator();
    int size = vertices.size();
    double[][] adjacencyMatrix = new double[size][size];
    int i = 0;
    while(it.hasNext()){
        Map.Entry me = (Map.Entry)it.next();

        Vertex vertex = (Vertex) me.getValue();
        List<Edge> edges = vertex.edges;
        for (Edge edge : edges) {

            adjacencyMatrix[i][edge.destId] = edge.edgeWeight;

        }
        ++i;
    }
    return  adjacencyMatrix;
}
```

VerticesSize Time Complexity = θ(1).
Traverse All Vertex = θ(V).
Traverse All Edges Of Vertex = O(E_v).
So Time Complexity is = O(V*E_v).

- **PrintGraph**

```
/unchecked/
// It's printing the graph.
@Override
public void printGraph() {

    Set entrySet = vertices.entrySet();

    Iterator it = entrySet.iterator();

    while(it.hasNext()){

        Map.Entry<Integer,Vertex> entry =(Map.Entry<Integer,Vertex>) it.next();
        Vertex vertex = entry.getValue();
        Iterator edgeIterator = vertex.edges.iterator();
        System.out.print(vertex);
        while (edgeIterator.hasNext()){
            Edge edge = (Edge) edgeIterator.next();
            Vertex ver = get(edge.destId);
            System.out.print("\tDest Node: " + ver.getLabel() +",  Id: "+ ver.getID()+ ", Edge weight: " + edge.edgeWeight + "\n");
        }
    }

}
```

Traverse All Vertex = θ(V).
Traverse All Edges Of Vertex = O(E_v).
Get method = O(1).
So Time Complexity is = O(V*E_v).