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

Ufukcan Erdem
1901042686

1. SYSTEM REQUIREMENTS

Functional Requirements

1-Add Operations (addVertex() etc.)

2-Remove Operations (removeVertex etc.)

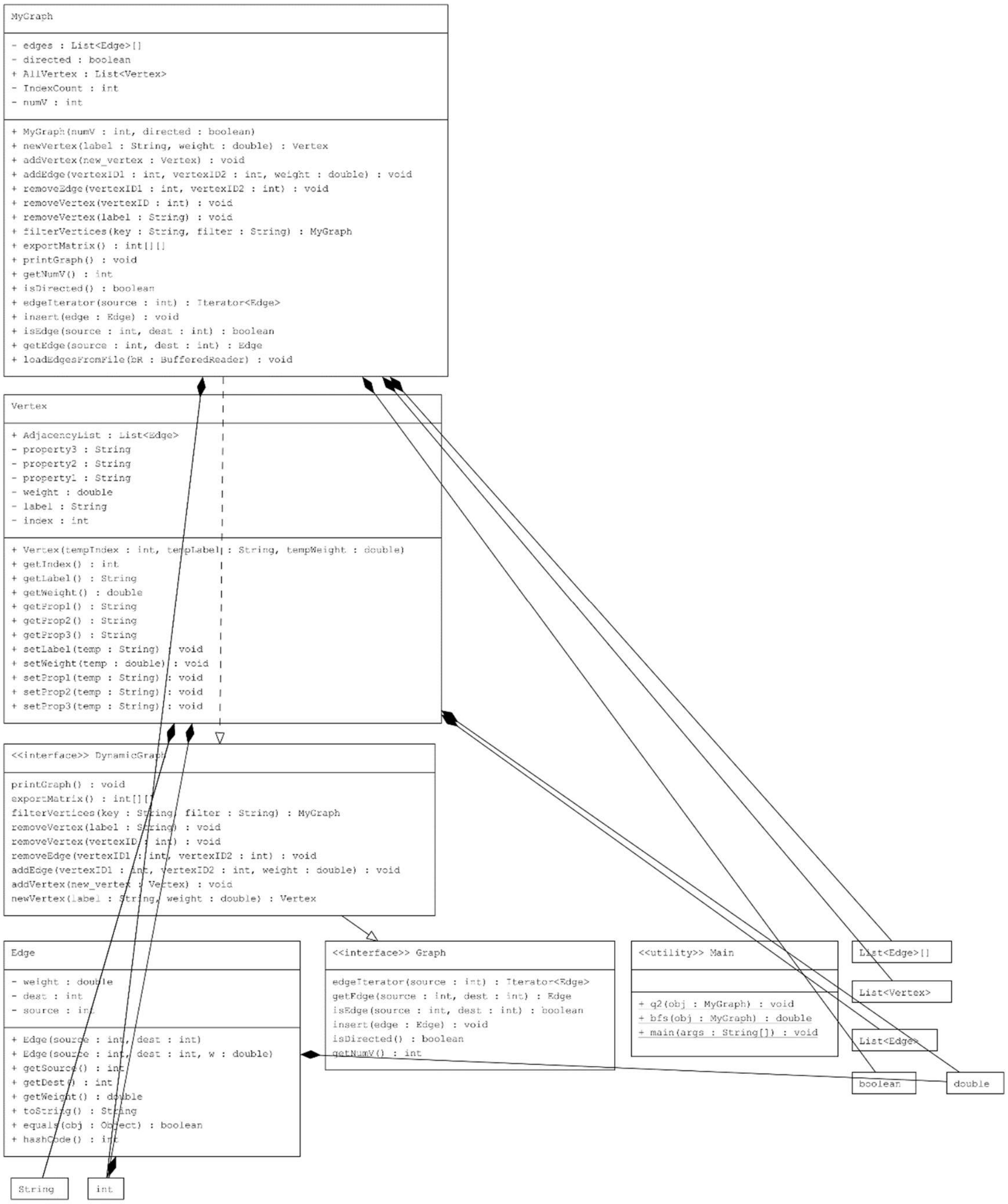
3-Search Operations (AdjacencyList.get(k) etc.)

Unfunctional Requirements

1-Hardware should be able to run at least JAVA-SE17.

2. USE CASE AND CLASS DIAGRAMS

I Separately added .png version of ClassDiagram in Folder.


MyGraph
- edges : List<Edge>[]
- directed : boolean
+ AllVertex : List<Vertex>
- IndexCount : int
- numV : int
+ MyGraph(numV : int, directed : boolean)
+ newVertex(label : String, weight : double) : Vertex
+ addVertex(new_vertex : Vertex) : void
+ addEdge(vertexID1 : int, vertexID2 : int, weight : double) : void
+ removeEdge(vertexID1 : int, vertexID2 : int) : void
+ removeVertex(vertexID : int) : void
+ removeVertex(label : String) : void
+ filterVertices(key : String, filter : String) : MyGraph
+ exportMatrix() : int[][]
+ printGraph() : void
+ getNumV() : int
+ isDirected() : boolean
+ edgeIterator(source : int) : Iterator<Edge>
+ insert(edge : Edge) : void
+ isEdge(source : int, dest : int) : boolean
+ getEdge(source : int, dest : int) : Edge
+ loadEdgesFromFile(bR : BufferedReader) : void
Vertex
+ AdjacencyList : List<Edge>
- property3 : String
- property2 : String
- property1 : String
- weight : double
- label : String
- index : int
+ Vertex(tempIndex : int, tempLabel : String, tempWeight : double)
+ getIndex() : int
+ getLabel() : String
+ getWeight() : double
+ getProp1() : String
+ getProp2() : String
+ getProp3() : String
+ setLabel(temp : String) : void
+ setWeight(temp : double) : void
+ setProp1(temp : String) : void
+ setProp2(temp : String) : void
+ setProp3(temp : String) : void
<<interface>> DynamicGraph
printGraph() : void
exportMatrix() : int[][]
filterVertices(key : String, filter : String) : MyGraph
removeVertex(label : String) : void
removeVertex(vertexID : int) : void
removeEdge(vertexID1 : int, vertexID2 : int) : void
addEdge(vertexID1 : int, vertexID2 : int, weight : double) : void
addVertex(new_vertex : Vertex) : void
newVertex(label : String, weight : double) : Vertex
Edge
- weight : double
- dest : int
- source : int
+ Edge(source : int, dest : int)
+ Edge(source : int, dest : int, w : double)
+ getSource() : int
+ getDest() : int
+ getWeight() : double
+ toString() : String
+ equals(obj : Object) : boolean
+ hashCode() : int
<<interface>> Graph
edgeIterator(source : int) : Iterator<Edge>
getFdge(source : int, dest : int) : Edge
isEdge(source : int, dest : int) : boolean
insert(edge : Edge) : void
isDirected() : boolean
getNumV() : int
<<utility>> Main
+ q2(obj : MyGraph) : void
+ bfs(obj : MyGraph) : double
+ main(args : String[]) : void
List<Edge>[]
List<Vertex>
List<Edge>
boolean
double
String
int

3. PROBLEM SOLUTION APPROACH

For Part 1;

Part 1 was clearly explained in “Homework8.pdf”. I created “MyGraph” class which implements “DynamicGraph” interface. Then I wrote the required methods according to homework pdf. I used edge class on my problem solution approach. While using edge class, I respectfully created Adjacency List with its rules.

I added 3 template property which are Strings and user can use it as he/she wants and can meet the requirements of the method used.

For Part 2;

I only managed to calculate distance of graph with Bfs. My q2 method only prints the distance of BFS. In bfs method I visit first vertices in the query and add their Adjacency vertices into query(query->’qIDs’).

```
List<Integer> vIds = new ArrayList<Integer>();            //Visited
List<Integer> qIds = new LinkedList<Integer>();           //In queue
```

Then I add distance for every visit into ‘TotalDistance’ variable, returned it and add visited vertices in the ‘vIds’ list which keeps IDs of visited vertices.

Time Complexity of “MyGraph” class methods;

-public Vertex newVertex (String label, double weight); -> O(1), creates a vertex.

-public void addVertex(Vertex new_vertex); -> O(1), adds Vertex to List.

-public void addEdge (int vertexID1, int vertexID2, double weight); -> O(n), checks all Vertices in the graph and according to Ids, creates an edge between that Ids.

-public void removeEdge (int vertexID1, int vertexID2); -> O(n), checks all Vertices in the graph and removes the edge between vertices according to parameters.

-public void removeVertex (int vertexID); -> O(n), checks all the Vertices in the graph and if a ID of an vertex same with the parameter, deletes that vertex.

-public void removeVertex (String label); -> O(n), Same algorithm with above method. Just try to find same label name vertex with parameter.

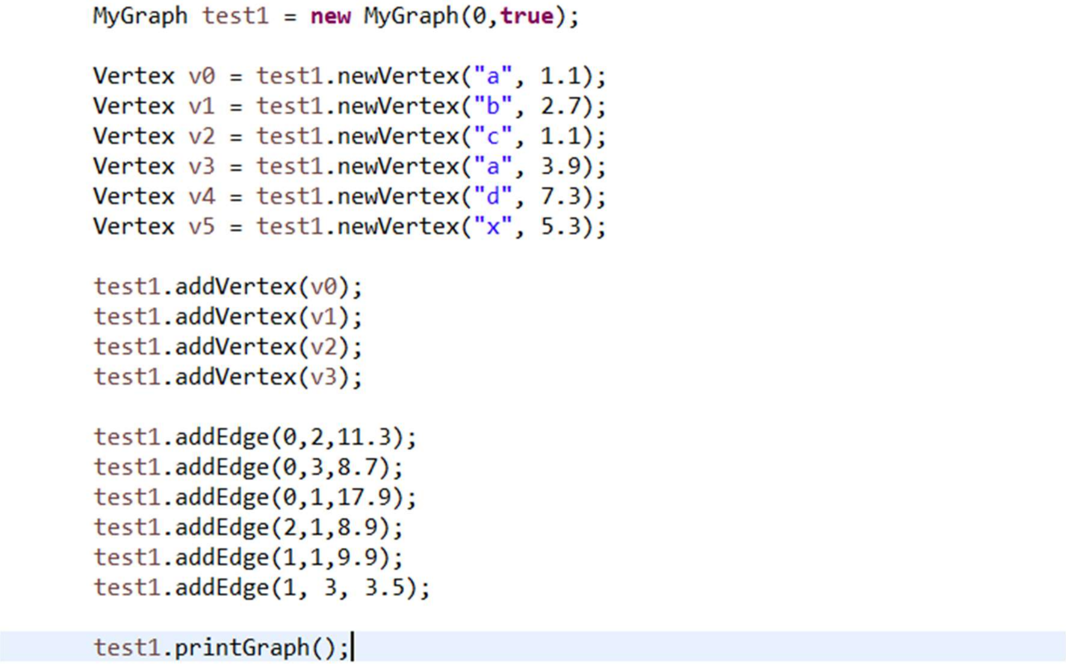
-public MyGraph filterVertices (String key, String filter); -> O(n), checks all vertices in the graph according to key number and filter. It creates temp MyGraph object and fills it with valid filter vertices. Returns temp MyGraph object.

-public int[][] exportMatrix(); -> O(n^3), Checks all vertices and their Adjacency Lists. Puts 1 if there is an edge between that vertices(Ex: For Vertices m and n, puts 1 to Matrix[m][n] if there is edge). Otherwise puts 0.

-public void printGraph(); -> O(n^2), prints all vertices in the adjacency list of every vertices.

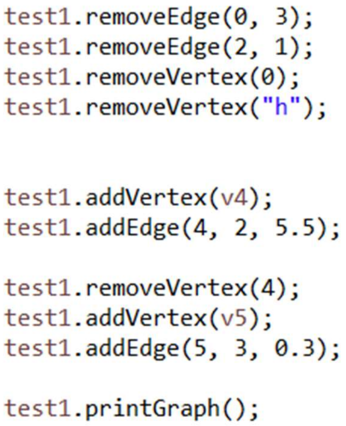
4. TEST CASES

-Test1, tests newVertex(), addVertex() and addEdge() methods. Results are in results part with Result1 name.



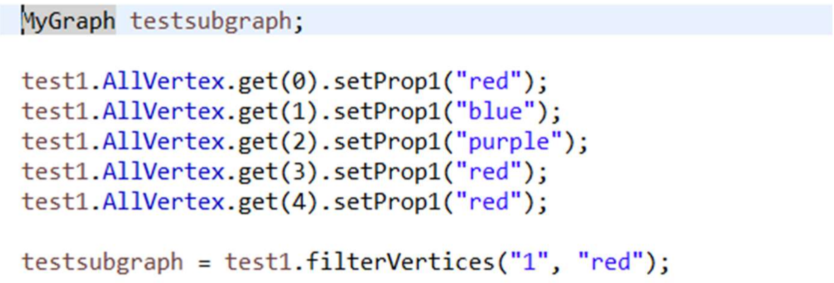
```
MyGraph test1 = new MyGraph(0,true);

Vertex v0 = test1.newVertex("a", 1.1);
Vertex v1 = test1.newVertex("b", 2.7);
Vertex v2 = test1.newVertex("c", 1.1);
Vertex v3 = test1.newVertex("a", 3.9);
Vertex v4 = test1.newVertex("d", 7.3);
Vertex v5 = test1.newVertex("x", 5.3);

test1.addVertex(v0);
test1.addVertex(v1);
test1.addVertex(v2);
test1.addVertex(v3);

test1.addEdge(0,2,11.3);
test1.addEdge(0,3,8.7);
test1.addEdge(0,1,17.9);
test1.addEdge(2,1,8.9);
test1.addEdge(1,1,9.9);
test1.addEdge(1, 3, 3.5);

test1.printGraph();
```

- Test2, tests removeEdge(), removeVertex(label or ID) methods. Uses printGraph method for result. Results are in the result part with Result2 name.



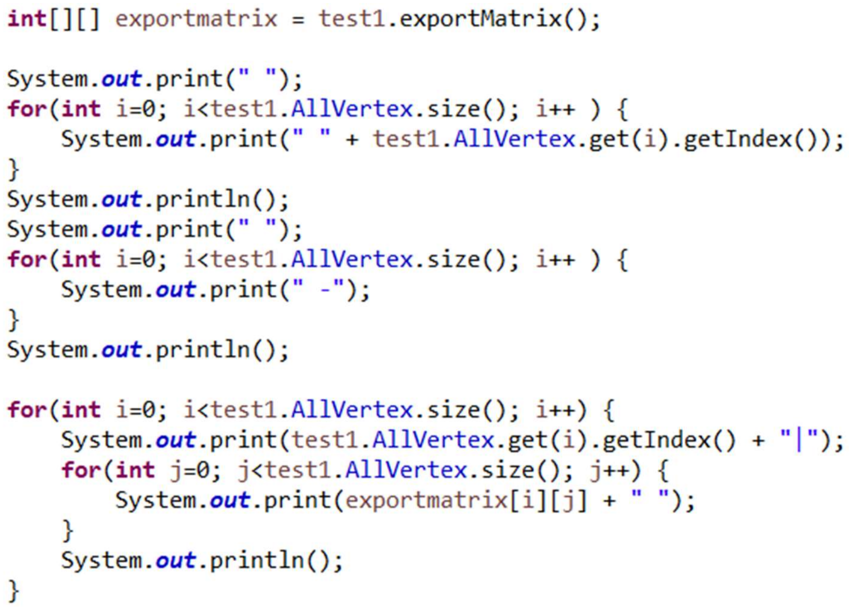
```
test1.removeEdge(0, 3);
test1.removeEdge(2, 1);
test1.removeVertex(0);
test1.removeVertex("h");


test1.addVertex(v4);
test1.addEdge(4, 2, 5.5);

test1.removeVertex(4);
test1.addVertex(v5);
test1.addEdge(5, 3, 0.3);

test1.printGraph();
```

- Test3, tests filterVertices() method. Results are in the result part with Result3 name.



```
MyGraph testsubgraph;

test1.AllVertex.get(0).setProp1("red");
test1.AllVertex.get(1).setProp1("blue");
test1.AllVertex.get(2).setProp1("purple");
test1.AllVertex.get(3).setProp1("red");
test1.AllVertex.get(4).setProp1("red");

testsubgraph = test1.filterVertices("1", "red");
```

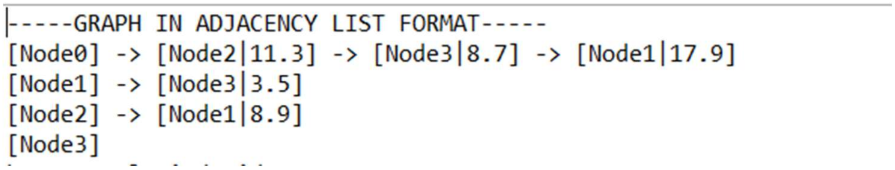
- Test4, tests exportMatrix() method. Results are in the result part with Result4 name.



```
int[][] exportmatrix = test1.exportMatrix();

System.out.print(" ");
for(int i=0; i<test1.AllVertex.size(); i++ ) {
    System.out.print(" " + test1.AllVertex.get(i).getIndex());
}
System.out.println();
System.out.print(" ");
for(int i=0; i<test1.AllVertex.size(); i++ ) {
    System.out.print(" -");
}
System.out.println();

for(int i=0; i<test1.AllVertex.size(); i++) {
    System.out.print(test1.AllVertex.get(i).getIndex() + "|");
    for(int j=0; j<test1.AllVertex.size(); j++) {
        System.out.print(exportmatrix[i][j] + " ");
    }
    System.out.println();
}
```

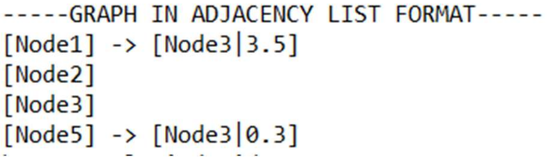
5. RUNNING AND RESULTS

-RESULT1



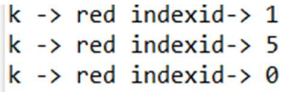
```
-----GRAPH IN ADJACENCY LIST FORMAT-----
[Node0] -> [Node2|11.3] -> [Node3|8.7] -> [Node1|17.9]
[Node1] -> [Node3|3.5]
[Node2] -> [Node1|8.9]
[Node3]
```

-RESULT2



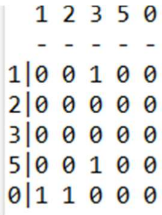
```
-----GRAPH IN ADJACENCY LIST FORMAT-----
[Node1] -> [Node3|3.5]
[Node2]
[Node3]
[Node5] -> [Node3|0.3]
```

-RESULT3



```
k -> red indexid-> 1
k -> red indexid-> 5
k -> red indexid-> 0
```

-RESULT4



```
  1 2 3 5 0
  - - - - -
1|0 0 1 0 0
2|0 0 0 0 0
3|0 0 0 0 0
5|0 0 1 0 0
0|1 1 0 0 0
```