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

Emircan Demirel
1901042674

1. SYSTEM REQUIREMENTS

Functional Requirements:

Define a Vertex class for representing the vertices in the graph. A vertex must have an index, a label and a weight.

The vertices may have user-defined additional properties.

Use adjacency list representation to handle the edges between vertices.

Implement methods to add/remove edges and vertices.

Implement method to generate Vertex class object from MyGraph class.

Implement method to filter vertices by the given user-defined property and returns subgraph of the graph

Implement method to convert adjacency representation to matrix representation.

Non-Functional Requirements:

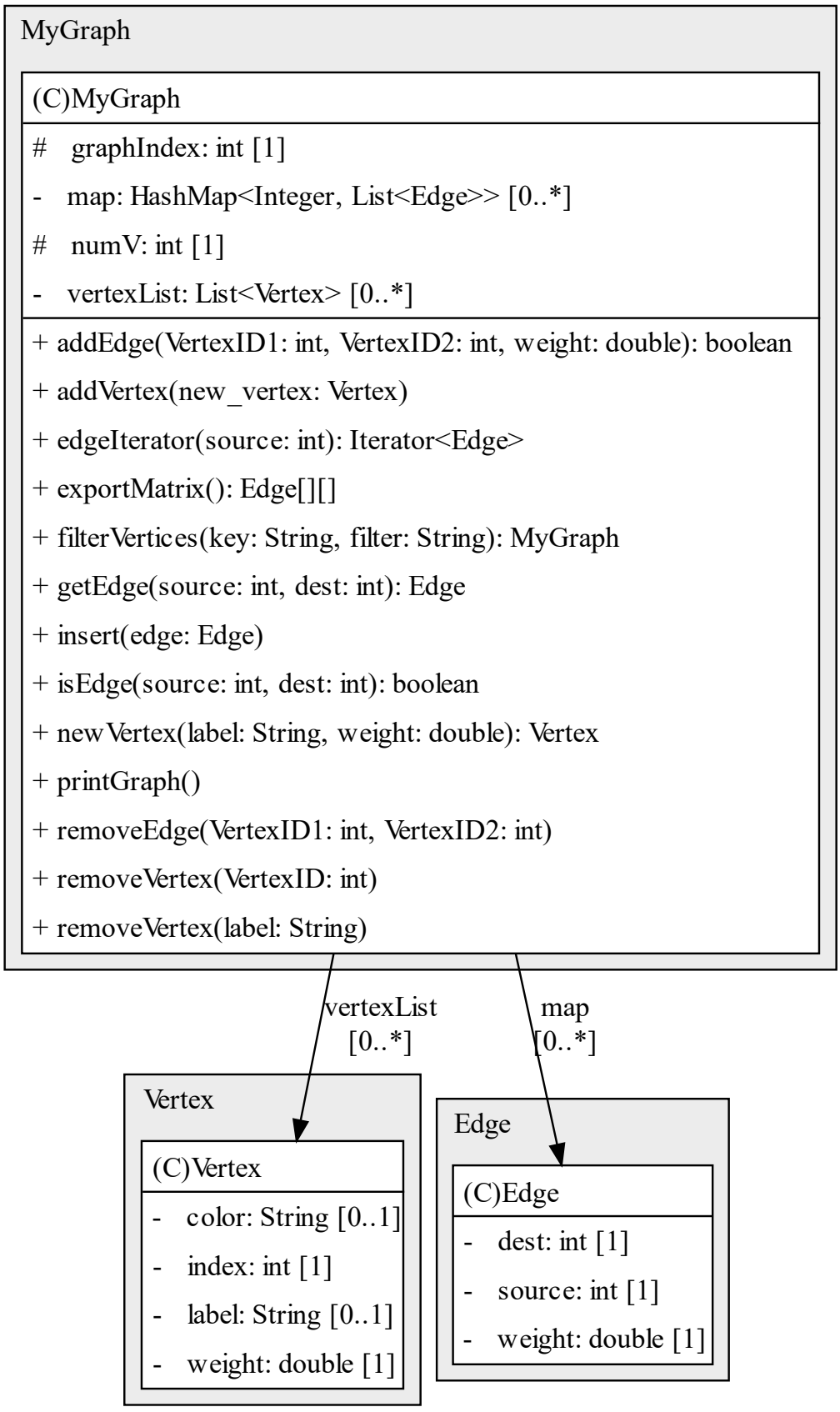
Program must handle errors and colliding items.

Edges couldn't have same source and destination.

Index value for every Vertex should be unique.

Java-SE13 or higher should be installed and running properly.

2. USE CASE AND CLASS DIAGRAMS



3. PROBLEM SOLUTION APPROACH

Q1)

In this part of the assignment, We have to implement MyGraph class which implements Dynamic Graph (extended by Graph class). With the help of Hashmap (map), I keep Vertex indices as key, Edges in linked list as values. I create a list to keep Vertices (vertexList).

newVertex: To generate Vertex via MyGraph's newVertex , Vertex class's constructor method called and in each call graphIndex parameter increases.

addVertex: A Vertex object passes to addVertex method as a parameter to added to 'map' and 'vertexList'.

addEdge method creates a new Edge and assign it to the valuesSet of given source. If linked list of edges does not contain same edge and source and destination indices are different then the method call insert method of Graph class.

Removal Methods: I use HashMap's and LinkedList's removal methods to delete an Edge/ a Vertex.

FilterVertices: I create new MyGraph object called as subgraph. Then, filtered ones added subgraph. After that, all edges of filtered vertices compared and compatible ones will be added subgraph too.

ExportMatrix: this method convert adjacency list format to matrix by creating two dimensional array in sizes of map.

4. **Analysis of Time Complexities**

```java
public Vertex newVertex(String label, double weight){
    Vertex newObj = new Vertex(label, weight, graphIndex++);
    return newObj;
}
```

Constant Time: θ(1)

```java
public void addVertex(Vertex new_vertex){
    numV++;
    vertexList.add(new_vertex);
    map.put(new_vertex.getIndex(), new LinkedList<Edge>());
}
```

Linear Time because of the add method of List: O(n)

```java
public boolean addEdge(int VertexID1, int VertexID2, double weight){
    if(VertexID1 == VertexID2){
        System.err.println("same source and dest values");
        return false;
    }
    if(!map.containsKey(VertexID1)){
        System.err.println("graph does not contain given VertexID1");
        return false;
    }
    if(!map.containsKey(VertexID2)){
        System.err.println("graph does not contain given VertexID2");
        return false;
    }
    Edge newEdge = new Edge(VertexID1, VertexID2, weight);

    for (int i = 0; i < map.get(VertexID1).size(); i++) {
        if(map.get(VertexID1).get(i).equals(newEdge)){
            System.out.println("this edge has already added to graph");
            return false;
        }
    }
    insert(newEdge);
    return true;
}
```

Insert method's time complexity: Linear time θ(n)

addEdge method's time complexity: Linear time θ(n)

```java
public void removeEdge(int VertexID1, int VertexID2){
    Edge newEdge = new Edge(VertexID1, VertexID2);
    map.get(VertexID1).remove(newEdge);
}
```

Linked list's removal method: Linear time θ(n)

```java
public void removeVertex(int VertexID){
    for (var element:
         vertexList) {
        if(element.getIndex() == VertexID){
            for (int i = 0; i < vertexList.size(); i++) {
                for (int j = 0; j < map.get(i).size(); j++) {
                    if(map.get(i).get(j).getSource() == element.getIndex() ||
                            map.get(i).get(j).getDest() == element.getIndex()){
                        map.get(i).remove(j);
                    }
                }
            }
            vertexList.remove(element);
            map.remove(VertexID);
            numV--;
            System.out.println("Node " + VertexID + " Successfully removed");
            return;
        }
    }
    System.out.println(VertexID + " index couldn't found on the list");
}
```

HashMap's removal method by using loop: $O(n^2)$

```java
public void removeVertex(String label){
    for (var element:
            vertexList) {
        if(element.getLabel().equals(label)){
            for (int i = 0; i < vertexList.size(); i++) {
                for (int j = 0; j < map.get(i).size(); j++) {
                    if(map.get(i).get(j).getSource() == element.getIndex() ||
                            map.get(i).get(j).getDest() == element.getIndex()){
                        map.get(i).remove(j);
                    }
                }
            }
            vertexList.remove(element);
            map.remove(element);
            numV--;
            System.out.println("Label " + label + " Successfully removed");
            return;
        }
    }
    System.out.println(label + " label couldn't find on the list");
}
```

HashMap's removal method by using loop: $O(n^2)$

```java
public MyGraph filterVertices(String key, String filter){
    MyGraph subGraph = new MyGraph();
    for(var element: vertexList){
        if(element.getColor().equals(key)){
            subGraph.addVertex(element);
        }
    }
    for(var element: subGraph.vertexList){
        for(Edge item: map.get(element.getIndex())){
            if(subGraph.map.containsKey(item.getSource()) && subGraph.map.containsKey(item.getDest())){
                subGraph.addEdge(item.getSource(), item.getDest(), item.getWeight());
            }
        }
    }
    System.out.println("----------subgraph----------");
    return subGraph;
}
```

time complexity: $\theta(n^2)$ quadratic

```java
public Edge[][] exportMatrix(){
    Edge[][] matrix = new Edge[map.size()][];
    for (int i = 0; i < matrix.length; i++) {
        matrix[i] = new Edge[map.get(i).size()];
        for (int j = 0; j < map.get(i).size(); j++) {
            matrix[i][j] = map.get(i).get(j);
        }
    }
    /*for (int i = 0; i < matrix.length; i++) {
        System.out.print("[");
        for (int j = 0; j < matrix[i].length; j++) {
            System.out.print(" " + matrix[i][j] + " ");
        }
        System.out.println("]");
    }*/
    return matrix;
}
```

time complexity: $\theta(n^2)$ quadratic

5. TEST CASES

Q1)

1. Compile -> Test newVertex Method -> create new Vertex object
2. Compile -> Test addVertex Method -> Add new Vertex to graph -> Test printGraph Method
3. Compile -> Test addEdge Method -> Valid Parameters
4. Compile -> Test addEdge Method -> vertexID which does not exist on map
5. Compile -> Test addEdge Method -> same source and dest indices
6. Compile -> Test addEdge Method -> try to add an existing edge
7. Compile -> Test removeEdge Method -> valid input
8. Compile -> Test removeEdge Method -> try non-existing edge object
9. Compile -> Test removeVertex Method -> enter valid index value
10. Compile -> Test removeVertex Method -> enter invalid index value
11. Compile -> Test removeVertex Method -> enter valid label string
12. Compile -> Test removeVertex Method -> enter invalid label string
13. Compile -> Test filterVertices Method
14. Compile -> Test exportMatrix Method

6. RUNNING AND RESULTS

Q1)

1. **Compile -> Test newVertex Method -> create new Vertex object**

```
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe"
[Node 0] white

Process finished with exit code 0
```

2. **Compile -> Test addVertex Method -> Add new Vertex to graph -> Test printGraph Method**

Input set:

```
MyGraph myGraph = new MyGraph();
Vertex obj_0 = myGraph.newVertex( label: "milk",  weight: 1.0);
obj_0.setColor("white");
Vertex obj_1 = myGraph.newVertex( label: "tea",  weight: 1.0);
obj_1.setColor("red");
Vertex obj_2 = myGraph.newVertex( label: "coffee",  weight: 1.0);
obj_2.setColor("brown");
myGraph.addVertex(obj_0);
myGraph.addVertex(obj_1);
myGraph.addVertex(obj_2);
myGraph.printGraph();
```

```
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe"
[Node 0] white
[Node 1] red
[Node 2] brown

Process finished with exit code 0
```

3. **Compile -> Test addEdge Method -> Valid Parameters**

   Input set:

```
myGraph.addEdge( VertexID1: 0,  VertexID2: 1,  weight: 5.0);
myGraph.addEdge( VertexID1: 1,  VertexID2: 0,  weight: 3);
myGraph.addEdge( VertexID1: 0, VertexID2: 2, weight: 1);
myGraph.addEdge( VertexID1: 2, VertexID2: 0,  weight: 4);
```

```
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe"
[Node 0] white-> [(0, 1): 5.0] -> [(0, 2): 1.0]
[Node 1] red-> [(1, 0): 3.0]
[Node 2] brown-> [(2, 0): 4.0]

Process finished with exit code 0
```

4. **Compile -> Test addEdge Method -> vertexID which does not exist on map**

```
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe"
graph does not contain given VertexID2

Process finished with exit code 0
```

5. **Compile -> Test addEdge Method -> same source and dest indices**

```
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe"
same source and dest values

Process finished with exit code 0
```

6. **Compile -> Test addEdge Method -> try to add an existing edge**

```
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe"
this edge has already added to graph

Process finished with exit code 0
```

7. **Compile -> Test removeEdge Method -> valid input**

```
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe"
[Node 0] white-> [(0, 1): 5.0] -> [(0, 2): 1.0]
[Node 1] red-> [(1, 0): 3.0]
[Node 2] brown-> [(2, 0): 4.0]

Node 0 Successfully removed
[Node 1] red
[Node 2] brown

Process finished with exit code 0
```

8. **Compile -> Test removeEdge Method -> try non-existing edge object**

```
myGraph.addEdge( VertexID1: 0,  VertexID2: 1,  weight: 5.0);
myGraph.addEdge( VertexID1: 1,  VertexID2: 0,  weight: 3);
myGraph.addEdge( VertexID1: 0,  VertexID2: 2,  weight: 1);
myGraph.addEdge( VertexID1: 2,  VertexID2: 0,  weight: 4);
myGraph.printGraph();
myGraph.removeEdge(1,2);
```

```
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe"
[Node 0] white-> [(0, 1): 5.0] -> [(0, 2): 1.0]
[Node 1] red-> [(1, 0): 3.0]
[Node 2] brown-> [(2, 0): 4.0]

edge could not found
```

9. **Compile -> Test removeVertex Method -> enter valid index value**

```
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe"
[Node 0] white-> [(0, 1): 5.0] -> [(0, 2): 1.0]
[Node 1] red-> [(1, 0): 3.0]
[Node 2] brown-> [(2, 0): 4.0]

Node 0 Successfully removed
[Node 1] red-> [(1, 0): 3.0]
[Node 2] brown-> [(2, 0): 4.0]
```

10. **Compile -> Test removeVertex Method -> enter invalid index value**

```
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe"
[Node 0] white-> [(0, 1): 5.0] -> [(0, 2): 1.0]
[Node 1] red-> [(1, 0): 3.0]
[Node 2] brown-> [(2, 0): 4.0]

6 index couldn't found on the list
```

11. **Compile -> Test removeVertex Method -> enter valid label string**

```
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe"
[Node 0] white-> [(0, 1): 5.0] -> [(0, 2): 1.0]
[Node 1] red-> [(1, 0): 3.0]
[Node 2] brown-> [(2, 0): 4.0]

Label tea Successfully removed
[Node 0] white-> [(0, 2): 1.0]
[Node 2] brown-> [(2, 0): 4.0]
```

12. **Compile -> Test removeVertex Method -> enter invalid label string**

```
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe"
[Node 0] white-> [(0, 1): 5.0] -> [(0, 2): 1.0]
[Node 1] red-> [(1, 0): 3.0]
[Node 2] brown-> [(2, 0): 4.0]

water label couldn't find on the list
```

13. **Compile -> Test filterVertices Method**

Input set:

```
Vertex obj_0 = myGraph.newVertex( label: "milk",  weight: 1.0);
obj_0.setColor("white");
Vertex obj_1 = myGraph.newVertex( label: "tea",  weight: 1.0);
obj_1.setColor("red");
Vertex obj_2 = myGraph.newVertex( label: "coffee",  weight: 1.0);
obj_2.setColor("brown");
Vertex obj_3 = myGraph.newVertex( label: "water",  weight: 1.0);
obj_3.setColor("white");
myGraph.addVertex(obj_0);
myGraph.addVertex(obj_1);
myGraph.addVertex(obj_2);
myGraph.addVertex(obj_3);

myGraph.addEdge( VertexID1: 0,  VertexID2: 3,  weight: 5.0);
myGraph.addEdge( VertexID1: 3,  VertexID2: 0,  weight: 5.0);
myGraph.addEdge( VertexID1: 1,  VertexID2: 0,  weight: 3);
myGraph.addEdge( VertexID1: 3, VertexID2: 2, weight: 1);
myGraph.addEdge( VertexID1: 2, VertexID2: 3,  weight: 4);
```

```
"C:\Program Files\Java\jdk-17.0.1\bin\java.exe"
[Node 0] white-> [(0, 3): 5.0]
[Node 1] red-> [(1, 0): 3.0]
[Node 2] brown-> [(2, 3): 4.0]
[Node 3] white-> [(3, 0): 5.0] -> [(3, 2): 1.0]

----------subgraph-----------
[Node 0] white-> [(0, 3): 5.0]
[Node 3] white-> [(3, 0): 5.0]

Process finished with exit code 0
```

14. Compile -> Test exportMatrix Method