CSE 222 DATA STRUCTURES AND ALGORITHMS

HOMEWORK 8

REPORT 8

MERVE HORUZ
1801042651

1) Detailed system requirements

1.1) Non-functional requirements

- Adaptability
- Efficiency
- Elasticity
- Integrability
- Maintainability

openjdk 17.0.3 2022-04-19
OpenJDK Runtime Environment (build 17.0.3+7-Ubuntu-0ubuntu0.20.04.1)
OpenJDK 64-Bit Server VM (build 17.0.3+7-Ubuntu-0ubuntu0.20.04.1, mixed mode, sharing)

1.2) Functional requirements

public MyGraph(int numV, boolean directed)
Constructs MyGraph object taking number of vertex and directed value.

public boolean isEdge(Vertex source, Vertex dest)
Returns true, if there is edge between source and destination vertex and called with these parameters.

public void insert(Edge edge)
It takes Edge object as a parameter and inserts to edges array.

public Iterator<Edge> edgeIterator(Vertex source)
It takes source vertex and returns iterator over graph by starting source vertex.

public Edge getEdge(Vertex source, Vertex dest)
Returns the edge between source and dest vertices it takes as a parameter if exist such an edge, otherwise null.

public Vertex newVertex (String label, double weight)
Generates new vertex by taking label and weight of the vertex.

public boolean addVertex (Vertex new_vertex)
Adds the vertex that parameter to graph.

public boolean addEdge (int vertexID1, int vertexID2, double weight)
Adds edge to between vertexID1 and vertexID2 and sets weight.

public boolean removeEdge (int vertexID1, int vertexID2)
Removes edge between vertexID1 and vertexID2. It takes ids as parameter.

public boolean removeVertex (int vertexID)
It takes vertexID1 to remove specified vertex.

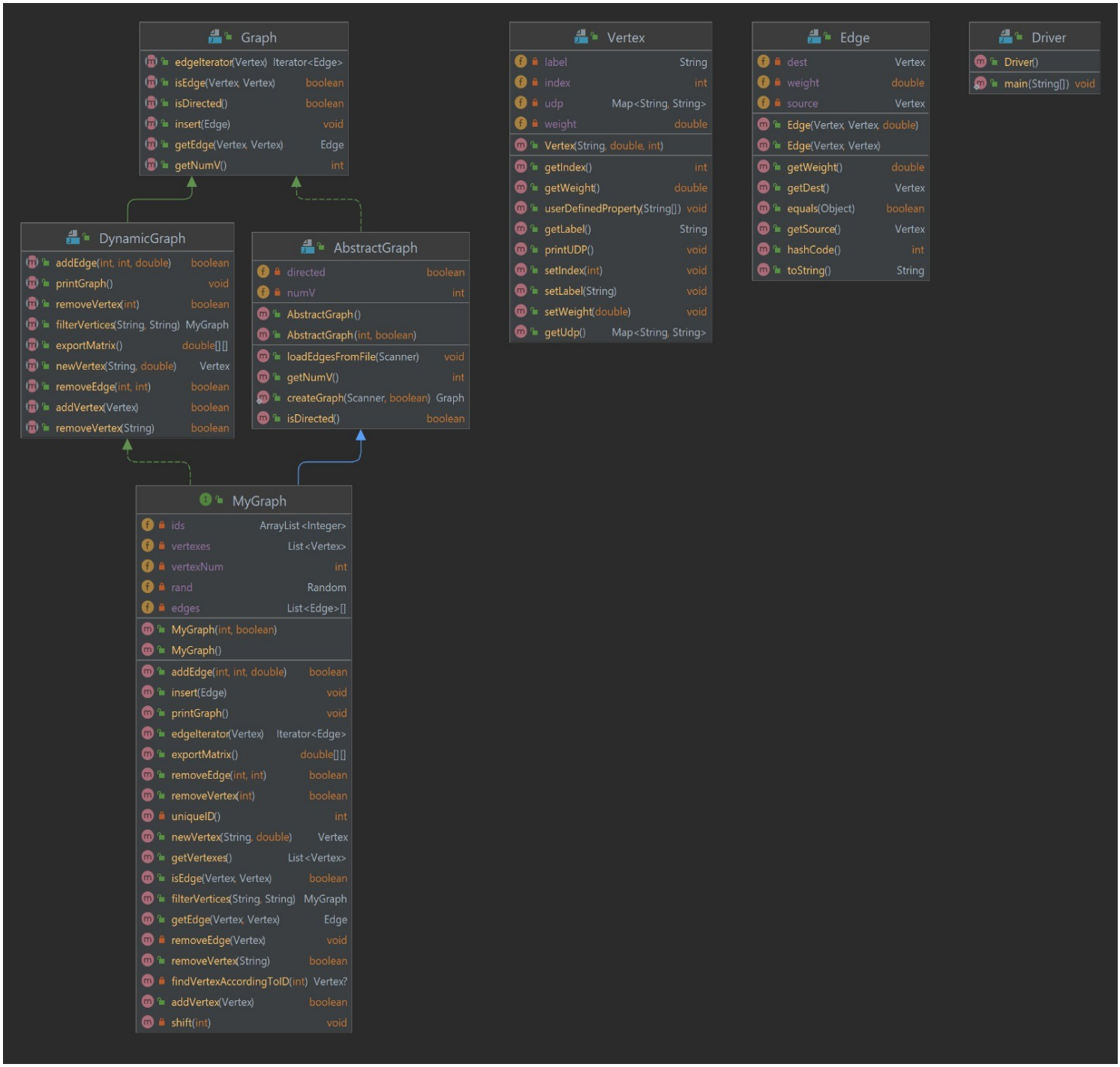
public boolean removeVertex (String label)
It takes label to remove vertexes.

public MyGraph filterVertices (String key, String filter)
It takes key and filter to filter vertices. Key is the key of the map, and filter is the value of map.

public double[][] exportMatrix()
Exports graph to matrix, and returns it.

public void userDefinedProperty(String... args)
It takes user defined properties of vertex.

2) Class diagram



3) Problem solution approach

First, extended DynamicGraph interface from Graph interface. Edge class is generated to represent an edge and contains source vertex, destination vertex and weight of edge. Vertex class is coded to represent each vertex and contains label, weight, index of vertex and user defined property in map data structure. In vertex class userDefinedProperty(String... args) function takes properties as a parameter and for example first string key second string is value and so on. MyGraph class implements DynamicGraph and extends AbstractGraph. In this class, edges are kept as array of lists, vertexes are kept in arraylist. While adding new vertex to graph, unique id is generated using random library and vertex is added to vertexes arraylist. While adding new edge to graph, first vertexes are searched according to their ids and if there is such vertex, edge is inserted to edges array. While removing an edge, first vertexes are searched according to their ids and if there is such edge, then removed from edges array (from edges[first parameter]). While removing vertex, vertex is searched according to their id and if there is such vertex, removed all edges connected to this vertex from edges array, shifts edges array toward up from last index to index of removed vertex,, removed vertex from vertexes arraylist, and vertex number is decreased. While exporting matrix, all edges are exported to 2D array.

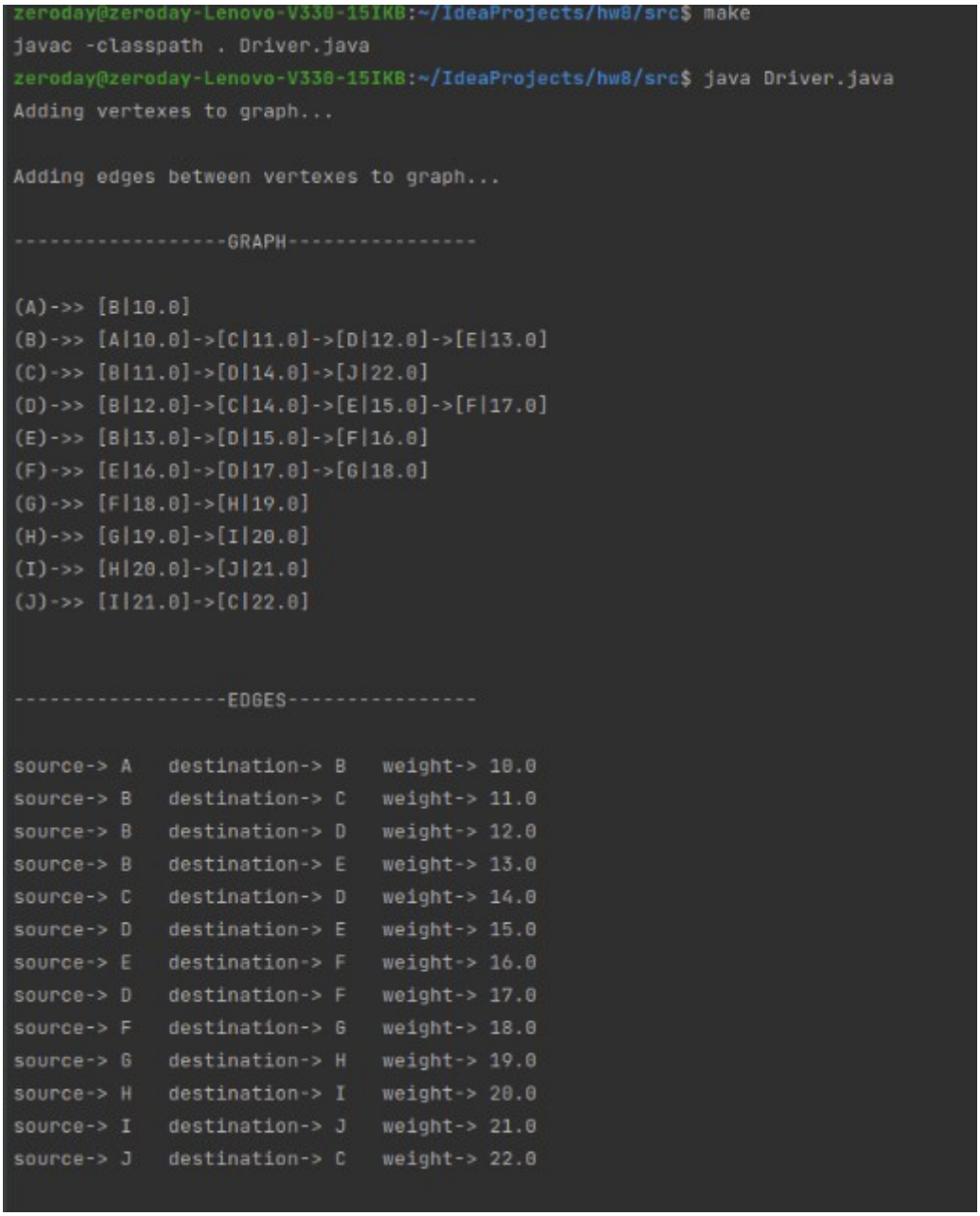
3) Test cases

Test case no	Test case	Input	Expected outputs	Test data	Resulting outputs	Situation	Error no	Error
1	User can add new vertex	Vertex new_vertex	True	Label = E Weight=76 id=4	Vertex is added	Successful	-	-
2	User can add new edge	int vertexID1, int vertexID2, double weight	True	Id1=4 id2 = 5 weight = 16	Edge is added	Successful	-	-
3	User can remove edge	int vertexID1, int vertexID2	True	Id1=4 id2 = 5	Edge is removed	Successful	-	-
4	User can remove vertex according to vertex id	int vertexID	True	Id1 = 4	Vertex is removed	Successful	-	-
5	User can remove vertex according to label	string label	True	label=E	Vertex is removed	Successful	-	-
6	User can filter vertices	string key, string filter	MyGraph object contains filtered vertices	Key=color filter=red	Vertexes are filtred	Successful	-	-
7	User can export graph to matrix	-	2D array contain graph	-	Matrix representation of graph	Successful	-	-
8	User can print graph	-	-	-	Prints graph	Successful	-	-

4) Running commands

- make
- java Driver.java
- if you would like to run one more time remove .class file -rm *.class

5) Running results



```
Exporting matrix...
0.0   10.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
10.0   0.0   11.0   12.0   13.0   0.0   0.0   0.0   0.0   0.0
0.0   11.0   0.0   14.0   0.0   0.0   0.0   0.0   0.0   22.0
0.0   12.0   14.0   0.0   15.0   17.0   0.0   0.0   0.0   0.0
0.0   13.0   0.0   15.0   0.0   16.0   0.0   0.0   0.0   0.0
0.0   0.0   0.0   17.0   16.0   0.0   18.0   0.0   0.0   0.0
0.0   0.0   0.0   0.0   0.0   18.0   0.0   19.0   0.0   0.0
0.0   0.0   0.0   0.0   0.0   0.0   19.0   0.0   20.0   0.0
0.0   0.0   0.0   0.0   0.0   0.0   0.0   20.0   0.0   21.0
0.0   0.0   22.0   0.0   0.0   0.0   0.0   0.0   21.0   0.0
```

```
---------User defined property of vertex0-----------

Key is: color value is: red
Key is: shape value is: circle
Key is: size value is: 5
Key is: boosting value is: 2.5


Key is: color value is: red
Key is: shape value is: square
Key is: size value is: 7
Key is: boosting value is: 3.5


Key is: color value is: green
Key is: shape value is: ellipse
Key is: size value is: 9
Key is: boosting value is: 4.5


Vertices are filtering according to color(red)...
(A)->> [B|10.0]
(B)->> [A|10.0]->[C|11.0]->[D|12.0]->[E|13.0]
```

```
Removing edge between vertex 2(C) and 3(D)...

(A)->> [B|10.0]
(B)->> [A|10.0]->[C|11.0]->[D|12.0]->[E|13.0]
(C)->> [B|11.0]->[J|22.0]
(D)->> [B|12.0]->[E|15.0]->[F|17.0]
(E)->> [B|13.0]->[D|15.0]->[F|16.0]
(F)->> [E|16.0]->[D|17.0]->[G|18.0]
(G)->> [F|18.0]->[H|19.0]
(H)->> [G|19.0]->[I|20.0]
(I)->> [H|20.0]->[J|21.0]
(J)->> [I|21.0]->[C|22.0]

Removing vertex 4(E)...


(A)->> [B|10.0]
(B)->> [A|10.0]->[C|11.0]->[D|12.0]
(C)->> [B|11.0]->[J|22.0]
(D)->> [B|12.0]->[F|17.0]
(F)->> [D|17.0]->[G|18.0]
(G)->> [F|18.0]->[H|19.0]
(H)->> [G|19.0]->[I|20.0]
(I)->> [H|20.0]->[J|21.0]
(J)->> [I|21.0]->[C|22.0]

Removing label A...

(B)->> [C|11.0]->[D|12.0]
(C)->> [B|11.0]->[J|22.0]
(D)->> [B|12.0]->[F|17.0]
(F)->> [D|17.0]->[G|18.0]
(G)->> [F|18.0]->[H|19.0]
(H)->> [G|19.0]->[I|20.0]
(I)->> [H|20.0]->[J|21.0]
(J)->> [I|21.0]->[C|22.0]
zeroday@zeroday-Lenovo-V330-15IKB:~/IdeaProjects/hw8/src$
```

Execution time analysis

```
public Vertex newVertex (String label, double weight){
    return new Vertex(label, weight, uniqueID());
} → // O(1)


public boolean addVertex (Vertex new_vertex){
    vertexes.add(new_vertex); //O(n)
    ids.add(new_vertex.getIndex(), new_vertex.getIndex()); //O(n)
    return true;
} → //O(n)


public boolean addEdge (int vertexID1, int vertexID2, double
weight){

    Vertex temp0 = findVertexAccordingToID(vertexID1); //O(n)
    Vertex temp1 = findVertexAccordingToID(vertexID2); //O(n)

    if(temp0 != null && temp1 != null)
        insert(new Edge(temp0, temp1, weight)); //O(1)
    return true;
} → //O(n)
```

```
public boolean removeEdge (int vertexID1, int vertexID2){

    Vertex temp0 = findVertexAccordingToID(vertexID1); //O(n)
    Vertex temp1 = findVertexAccordingToID(vertexID2); //O(n)

    if(temp0 != null && temp1 != null) {
        edges[vertexID1].remove(new Edge(temp0, temp1)); //O(n)
        if (!isDirected()) edges[vertexID2].remove(new Edge(temp1,
temp0)); //O(n)
        return true;
    }
    return false;
} → //O(n)
```

```
public boolean removeVertex (int vertexID){

    Vertex temp0 = findVertexAccordingToID(vertexID); //O(n)
    if(temp0 != null){
        removeEdge(temp0);  //O(n^2)
        shift(vertexID);  //O(n)
        vertexes.remove(temp0); //O(n)
        ids.remove(vertexID); //O(n)
        vertexNum--;
        return true;
    }    return false;
} → //O(n^2)
```

```
public MyGraph filterVertices (String key, String filter){

    MyGraph subGraph = new MyGraph();

    for(Vertex v : vertexes) {  //O(n)
        Set entrySet = v.getUdp().entrySet();
        Iterator it = entrySet.iterator();

        while(it.hasNext()){ //O(m)
            Map.Entry me = (Map.Entry)it.next();
            if(me.getKey().equals(key) &&
me.getValue().equals(filter)){
                subGraph.addVertex(v); //O(n)
                subGraph.vertexNum += 1;
            }
        }
    }
    subGraph.edges = new List[subGraph.vertexNum];
    for (int i = 0; i < subGraph.vertexNum; i++) { //O(n)
        subGraph.edges[i] = new LinkedList<>();
    }
    for (Vertex temp : subGraph.getVertexes()) { //O(n)
        for (Edge edge : edges[temp.getIndex()]) //O(n)
            subGraph.edges[temp.getIndex()].add(edge); //O(n)
    }
    return subGraph;
} → //O(n^2)
```

```
public double[][] exportMatrix(){

    double[][] matrix = new double[vertexes.size()]
[vertexes.size()];

    for(List<Edge> el : edges){ //O(m)
        for(Edge e : el) { //O(n)
            matrix[e.getSource().getIndex()][e.getDest().getIndex()] =
e.getWeight(); //O(1)
            if(!isDirected())
                matrix[e.getDest().getIndex()][e.getSource().getIndex()]
= e.getWeight(); //O(1)
        }
    }
    return matrix;
} //O(m*n)


public void printGraph(){

    for(int i=0;i<vertexNum;i++){ //O(n)
        List<Edge> el = edges[i];
        System.out.print("(" + vertexes.get(i).getLabel() + ")->> ");
        //O(1)
        int k=0;
        for(Edge e : el) { //O(m)
            System.out.print("[" + e.getDest().getLabel() + "|" +
e.getWeight() + "]");
            k++;
            if(k<el.size()) System.out.print("->");
        }
        System.out.print("\n");
    }
} → //O(m*n)
```