GIT Department of Computer Engineering
CSE 222/505 - Spring 2022
Homework #08 Report

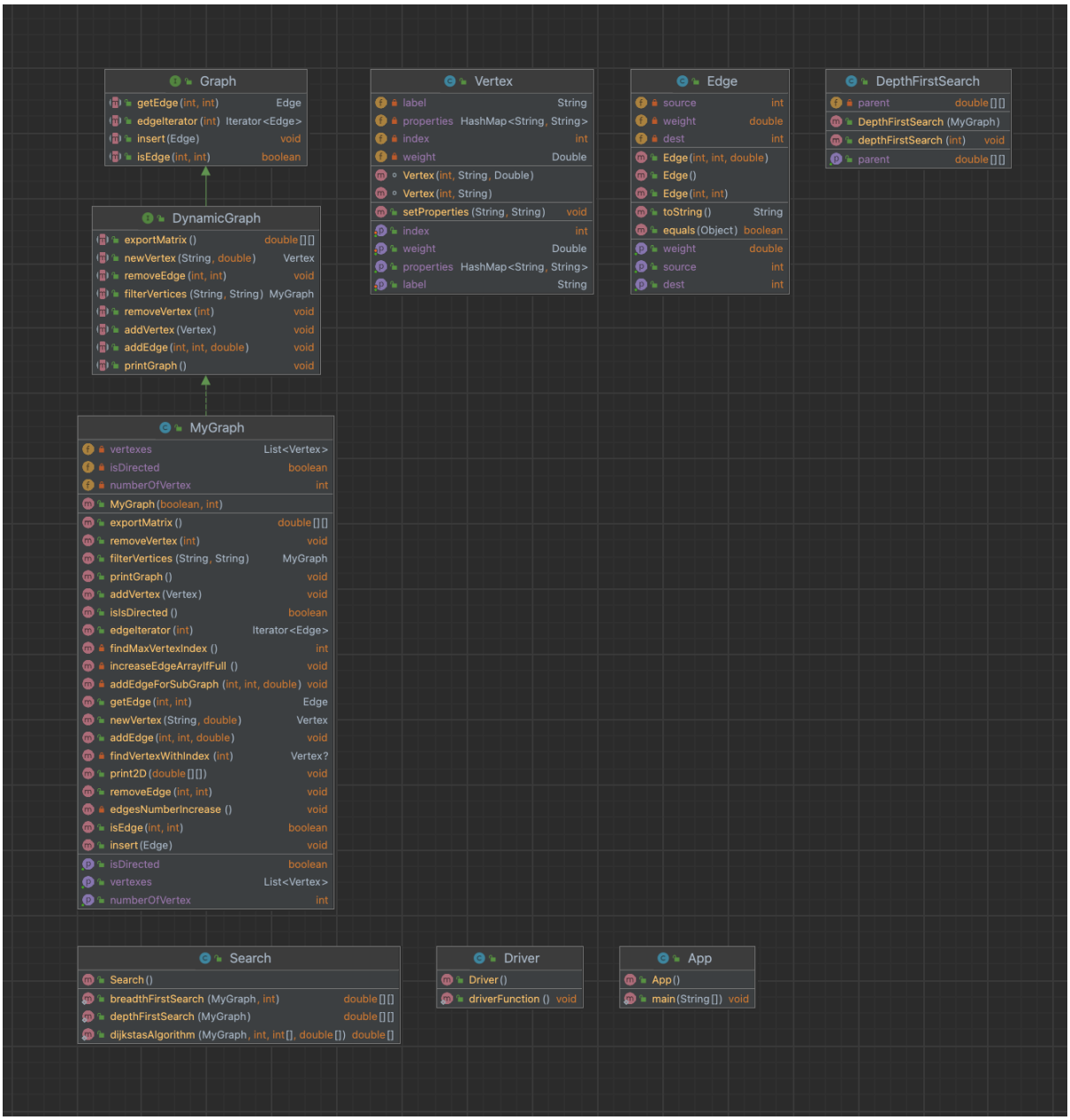
Muhammed Sefa Cahyir
1801042686

1. SYSTEM REQUIREMENTS

No need expensive system for this project. If don't use with big datas.
It is enough to have a Java and a machine to run JVM
The minimum system requirements for Java Virtual Machine are as follows:

Windows 8/7/Vista/XP/2000.
Windows Server 2008/2003.
Intel and 100% compatible processors are supported.
Pentium 166 MHz or faster processor with at least 64 MB of physical RAM. 98 MB of free disk space.

2. USE CASE AND CLASS DIAGRAMS



3. OTHER DIAGRAMS

No other diagram

4. PROBLEM SOLUTION APPROACH

In this homework Q1 problem was the create a new Graph with these methods.

```
// TO GENERATE NEW VERTEX WITH PARAMATERS
public Vertex newVertex(String label, double weight);

// ADD THE GIVEN VERTEX TO GRAPH
public void addVertex(Vertex newVertex);

// ADD AN EDGE BETWEEN THE GIVEN TWO VERTEX TO GRAPH
public void addEdge(int vertexId1, int vertexId2, double weight);

// REMOVE THE EDGE BETWEEN TWO GIVEN VERTEX
public void removeEdge(int vertexId1, int vertexId2);

// REMOVE THE VERTEX WITH GIVEN BY ID
public void removeVertex(int vertexId);

// FILTER THE VERTICES BY GIVEN USER DEFINED PROPERTY AND RETURNS A SUBGRAPH OF GRAPH
public MyGraph filterVertices(String key, String filter);

// GENERATE THE ADJACENCY MATRIX REPRESANTATION OF THE GRAPH AND RETURNS THE MATRIX
public double[][] exportMatrix();

// PRINT THE GRAPH IN ADJACENCY LIST FORMAT
public void printGraph();
```

In books example we already make the Grahp with just Edges but in this problem we need to use a Vertex with properties, I was thinking to use Mapping but it can be more complex for this project. Because of it I used the ArrayList for generally using get options O(1).

```
private List<Vertex> vertexes = new ArrayList<Vertex>();
```

For newVertex index should be uniqe because of it I create a method for the find max Vertex index and increase one and create new Vertex with it.

```
public Vertex newVertex(String label, double weight) {
    // Find the max vertex index and new vertex id is more than one
    int max = findMaxVertexIndex() + 1;
    return new Vertex(max, label, weight);
}
```

For this method time complexity is ø(n) for each case because it's traverse all Vertex in all case to find max.

For addVertex we need to check Vertex index, in some case we increase automatically but if user want to add this manually we need to check the id is unique or not. For this one I create a method with findVertexWithIndex and if it's null we can create new vertex.

```
public void addVertex(Vertex newVertex) {
    // Check if vertex index is already using. (Should be uniuqe)
    if (findVertexWithIndex(newVertex.getIndex()) == null) {
        vertexes.add(newVertex);
        numberOfVertex++;
    } else
        throw new RuntimeErrorException(null, "The index is already using please try with another index.");
}
```

For this method time complexity ø(n) also, findVertexWithIndex traverse in all case.

In this method for adding an edge need to check vertexId is available or not, for this case also used findVertexWithIndex and after add edge is ø(1).

```
public void addEdge(int vertexId1, int vertexId2, double weight) {
    // If Edge array is full increase the edge array size
    increaseEdgeArrayIfFull();

    // Check there is have a vertexes with these id's
    if (findVertexWithIndex(vertexId1) != null && findVertexWithIndex(vertexId1) != null) {
        edges[vertexId1].add(new Edge(vertexId1, vertexId2, weight));
        if (isDirected)
            edges[vertexId2].add(new Edge(vertexId2, vertexId1, weight));
    } else
        throw new RuntimeErrorException(null,
                "There is no vertex with these id, please try with using vertex id.");

}
```

For this method time complexity is O(n) because findVertexIndex traverse all cases, this is ø(n) but in Edge i used the ArrayList and add method some times it can be O(n).

In this method we need to check it's directed or not, if directed we need to delete also from the other side. And others is same with Add Edge

```
public void removeEdge(int vertexId1, int vertexId2) {
    for (Edge item : edges[vertexId1]) {
        if (item.getDest() == vertexId2)
            edges[vertexId1].remove(item);
    }
    if (isDirected) {
        for (Edge item : edges[vertexId2]) {
            if (item.getDest() == vertexId1)
                edges[vertexId2].remove(item);
        }
```

```
    }
  }
```

For this method time complexity is O(n). Because it is not to traverse all edges some some case and it's have Best case and worts case.

In this method we need to check when delete a Vertex it can be have edges with connected with it. And we need to also delete these vertex

```
  public void removeVertex(int vertexId) {
    if (!edges[vertexId].isEmpty())
      edges[vertexId].clear();
    vertexes.remove(findVertexWithIndex(vertexId));
  }
```

For this method time complexity is O(n). Because in some case we maybe have edges but some case we don't have edges because of it we have worst and best case for it.

In this method first we need to traverse vertex to find the filtered method. Later we need to traverse filtered vertex to find edges and later add these and new sub graph.

```
  public MyGraph filterVertices(String key, String filter) {
    MyGraph subGraph = new MyGraph(isDirected, numberOfVertex);

    // Find vertex and add this on mg (Sub graph )
    for (Vertex item : vertexes) {
      if (!item.getProperties().isEmpty() && item.getProperties().get(key).equals(filter))
        subGraph.addVertex(item);

    }

    // Find edges and add this to on mg (Sub graph )
    for (Vertex item : subGraph.vertexes) {
      if (edges[item.getIndex()] != null) {
        for (Edge edge : edges[item.getIndex()]) {
          if (subGraph.findVertexWithIndex(edge.getDest()) != null)
            subGraph.addEdgeForSubGraph(item.getIndex(),
subGraph.findVertexWithIndex(edge.getDest()).getIndex(), edge.getWeight());
        }
      }
    }
    return subGraph;
  }
```

For this method time complexity is worst case O(m*n) and best case ø(n). Because in worst case we need to traverse all Vertex and later traverse all edges. In best case it's enough to traverse just Vertexes.

5. TEST CASES

For test cases tried;

- **Add Vertex and Edge with 100, 1000, 10000 size (Undirected)**
- **Add Vertex and Edge with 100, 1000, 10000 size (Directed)**
- **Filter Vertexes**
- **Create Sub Graph**
- **Create Matrix with Sub Graph**
- **Remove Vertex and Edge with 100, 1000, 10000 size (Undirected)**
- **Remove Vertex and Edge with 100, 1000, 10000 size (Directed)**
- **Set properties (Boost and Color)**
- **Search with BST**
- **Search with DST**
- **Search with Djikstra (include Booster)**

6. RUNNING AND RESULTS

```
 ------------------- ADD EDGE AND VERTICES (UNDIRECTED) -------------------

Create 100 Vertex size Graph undirected
Create and add 100 Vertex to Graph
Create and add 100 Vertex to Graph = 35 milisecond

Create and add 100 Edge to Graph
Create and add 100 Edge to Graph = 51 milisecond


Create 1000 Vertex size Graph undirected
Create and add 1000 Vertex to Graph
Create and add 1000 Vertex to Graph = 149 milisecond

Create and add 1000 Edge to Graph
Create and add 1000 Edge to Graph = 135 milisecond


Create 10000 Vertex size Graph undirected
Create and add 10000 Vertex to Graph
Create and add 10000 Vertex to Graph = 1191 milisecond

Create and add 10000 Edge to Graph
Create and add 10000 Edge to Graph = 1608 milisecond



 ------------------- ADD EDGE AND VERTICES (DIRECTED) -------------------

Create 100 Vertex size Graph directed
Create and add 100 Vertex to Graph
Create and add 100 Vertex to Graph = 0 milisecond

Create and add 100 Edge to Graph
Create and add 100 Edge to Graph = 1 milisecond


Create 1000 Vertex size Graph directed
Create and add 1000 Vertex to Graph
Create and add 1000 Vertex to Graph = 5 milisecond

Create and add 1000 Edge to Graph
Create and add 1000 Edge to Graph = 15 milisecond


Create 1000 Vertex size Graph directed
Create and add 10000 Vertex to Graph
Create and add 10000 Vertex to Graph = 191553 milisecond

Create and add 10000 Edge to Graph
Create and add 10000 Edge to Graph = 1801 milisecond



 ------------------- FILTER VERTICES, PRINT GRAPH AND EXPORT MATRIX -------------------

Set properties to 100 Vertex and filter between 45-60 and print graph
[(46, 47): 98.0]
[(47, 48): 100.0]
[(48, 49): 102.0]
[(49, 50): 104.0]
[(50, 51): 106.0]
[(51, 52): 108.0]
[(52, 53): 110.0]
[(53, 54): 112.0]
[(54, 55): 114.0]
[(55, 56): 116.0]
[(56, 57): 118.0]
[(57, 58): 120.0]
[(58, 59): 122.0]
```

```
 ------------------- FILTER VERTICES, PRINT GRAPH AND EXPORT MATRIX --------------------

Set properties to 100 Vertex and filter between 45-60 and print graph
[(46, 47): 98.0]
[(47, 48): 100.0]
[(48, 49): 102.0]
[(49, 50): 104.0]
[(50, 51): 106.0]
[(51, 52): 108.0]
[(52, 53): 110.0]
[(53, 54): 112.0]
[(54, 55): 114.0]
[(55, 56): 116.0]
[(56, 57): 118.0]
[(57, 58): 120.0]
[(58, 59): 122.0]

 Export the matrix filtered

        46.0      47.0      48.0      49.0      50.0      51.0      52.0      53.0      54.0      55.0      56.0      57.0      58.0      59.0
46      Infinity  98.0      Infinity  94.0      Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity
47      Infinity  Infinity  100.0     Infinity  Infinity  Infinity  Infinity  Infinity  96.0      Infinity  Infinity  Infinity  Infinity  Infinity
48      Infinity  Infinity  Infinity  102.0     Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  98.0      Infinity  Infinity  Infinity
49      Infinity  Infinity  Infinity  Infinity  104.0     Infinity  Infinity  Infinity  100.0     Infinity  Infinity  Infinity  Infinity  Infinity
50      Infinity  Infinity  Infinity  Infinity  Infinity  106.0     Infinity  Infinity  Infinity  102.0     Infinity  Infinity  Infinity  Infinity
51      104.0     Infinity  Infinity  Infinity  Infinity  Infinity  108.0     Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity
52      Infinity  Infinity  106.0     Infinity  Infinity  Infinity  Infinity  110.0     Infinity  Infinity  Infinity  Infinity  Infinity  Infinity
53      Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  112.0     Infinity  108.0     Infinity  Infinity  Infinity
54      Infinity  110.0     Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  114.0     Infinity  Infinity  Infinity  Infinity
55      Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  116.0     Infinity  Infinity  Infinity
56      Infinity  Infinity  Infinity  114.0     Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  118.0     Infinity  Infinity
57      Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  116.0     120.0     Infinity
58      Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  118.0     Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  122.0
59      120.0     Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity  Infinity

 Set properties to 100 Vertex and filter between 45-60, print graph and print export matrix duration time  = 45 milisecond


Remove 1000/2 Vertex to Graph (If Vertex is deleting related edge is also deleting)
Remove 1000/2 Vertex to Graph = 48 milisecond


Remove 10000/2 Vertex to Graph (If Vertex is deleting related edge is also deleting)
Remove 10000/2 Vertex to Graph = 250876 milisecond



 ------------------- REMOVE EDGE AND VERTICES --------------------

Remove 100/2 Vertex to Graph (If Vertex is deleting related edge is also deleting)
Remove 100/2 Vertex to Graph = 0 milisecond


Remove 1000/2 Vertex to Graph (If Vertex is deleting related edge is also deleting)
Remove 1000/2 Vertex to Graph = 5 milisecond


Remove 10000/2 Vertex to Graph (If Vertex is deleting related edge is also deleting)
Remove 10000/2 Vertex to Graph = 462337 milisecond

Diffirences between BST - DFS duration weight = 0.0 milisecond

Diffirences between BST - DFS duration time = -96 milisecond

Shortest path Djikstra with Booster = 132.0

Duration time Djikstra with Booster = 12 milisecond

msc@Sefas-MacBook-Air Hw8 %
```