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

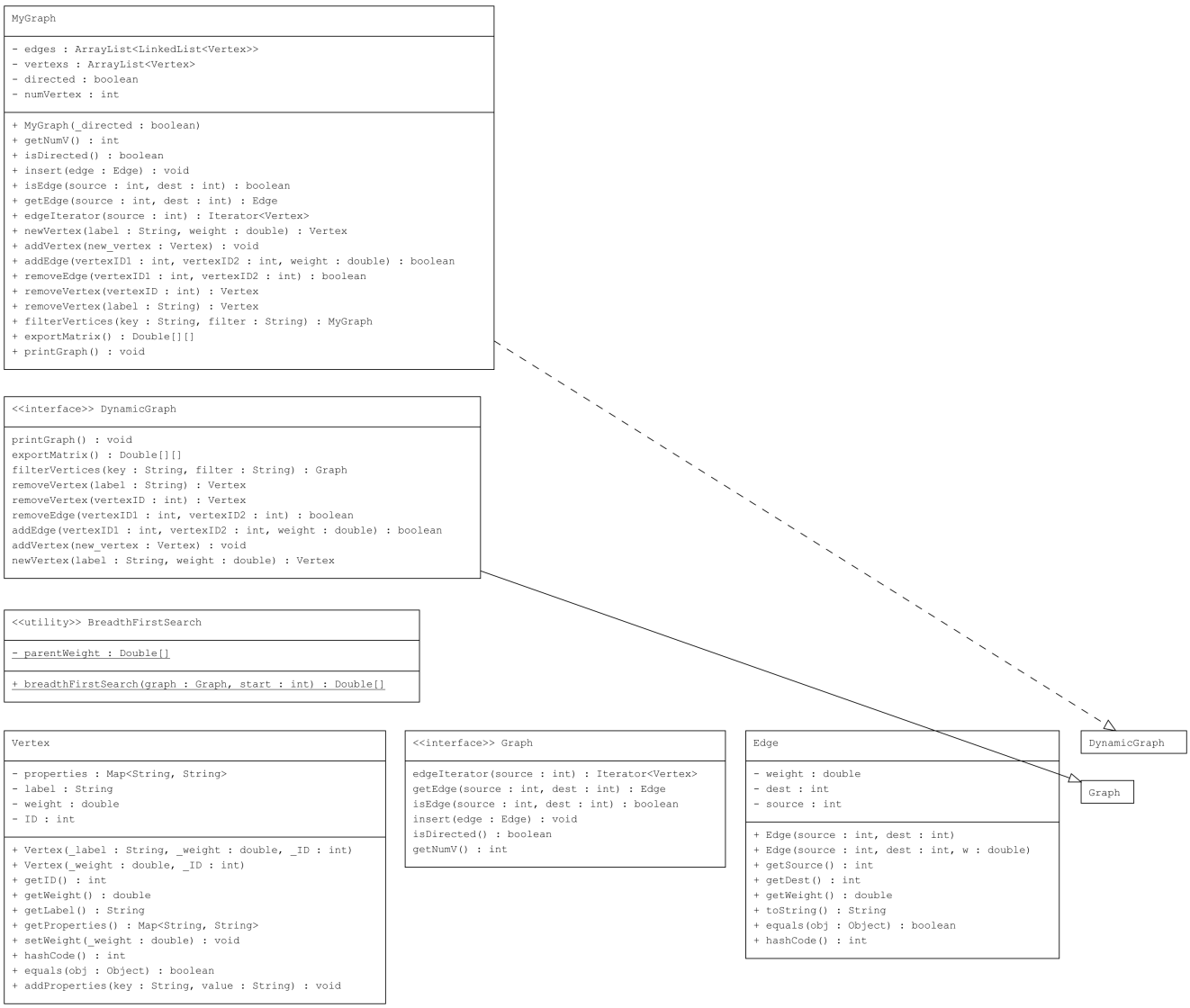
Atacan Başaran
200104004008008

1. SYSTEM REQUIREMENTS

Question 1 -> In this question we are expected to implement a graph and extend it with new features as creating new vertex,adding it into graph,remove vertex,adding and removing edges(I thought edges as a connected 2 vertex).Also program need a filtervertice method to create a subgraph with certain features.Lastly program need an exportmatrix method which create a edges matrix.

Question 2 -> In question 2,we are expected to write a method that calculates shortest path for each vertex during two different traversal methodology(BFS DFS).Method need to calculate distance during the certain sequence.

2. CLASS DIAGRAM



For higher resolution you can check the folder

3. PROBLEM SOLUTION APPROACH

Question 1 -> Graph interface methods mostly implemented by book,DynamicGraph extended by Graph and MyGraph implements the DynamicGraph.In MyGraph i store vertexs in Arrraylist and store edges in Arraylist of LinkedList.I use LinkedList for faster remove and add.I create vertex class for representation of Edges.newVertex() create a vertex with constructor.addVertex() and removeVertex() doing operation on Arraylist of Vertexs also controls the edges and do necessary delete operations.addEdge() adding right element into the right place on LinkedList.filterVertices() create a new MyGraph object with certain vertex and edges by using above methods.exportMatrix() create a double 2D matrix representation of edge weights by using Edges.printGraph() print the edges of graph as mentioned by pdf

Question 2 -> During BreadthFirstSearch i created parentWeight array which stores the edge weights.I calculate total weights(distances) by using this values for each vertex considering the traverse order.I could not do second part(DepthFirstSearch)

4. TEST CASES

Question 1

Create a Graph

```
System.out.println(x: "-----Question 1-----");
MyGraph aGraph = new MyGraph(_directed: true);
```

Create new Vertex,adding them into the graph and add properties on vertexs

```
Vertex target = aGraph.newVertex(label: "zero", weight: 10)
aGraph.addVertex(target);
target.addProperties(key: "drink", value: "coffee");
target.addProperties(key: "food", value: "burger");
```

```
target = aGraph.newVertex(label: "one", weight: 20);
aGraph.addVertex(target);
target.addProperties(key: "drink", value: "milk");
target.addProperties(key: "food", value: "sausage");

target = aGraph.newVertex(label: "two", weight: 50);
aGraph.addVertex(target);
target.addProperties(key: "drink", value: "tea");
target.addProperties(key: "food", value: "fish");

target = aGraph.newVertex(label: "three", weight: 100);
aGraph.addVertex(target);
target.addProperties(key: "drink", value: "tea");
target.addProperties(key: "food", value: "lahmacun");

target = aGraph.newVertex(label: "four", weight: 500);
aGraph.addVertex(target);
target.addProperties(key: "drink", value: "tea");
target.addProperties(key: "food", value: "meatball");

target = aGraph.newVertex(label: "five",weight: 1);
aGraph.addVertex(target);
target.addProperties(key: "drink", value: "tea");
target.addProperties(key: "food", value: "bread");

target = aGraph.newVertex(label: "six", weight: 30);
aGraph.addVertex(target);
```

Adding edges to the Graph

```
aGraph.addEdge(vertexID1: 0, vertexID2: 3, weight: 5);
aGraph.addEdge(vertexID1: 0, vertexID2: 1, weight: 3);
aGraph.addEdge(vertexID1: 3, vertexID2: 2, weight: 1);
aGraph.addEdge(vertexID1: 1, vertexID2: 2, weight: 7);
aGraph.addEdge(vertexID1: 2, vertexID2: 4, weight: 4);
aGraph.addEdge(vertexID1: 2, vertexID2: 5, weight: 10);
aGraph.addEdge(vertexID1: 5, vertexID2: 6, weight: 5);
aGraph.addEdge(vertexID1: 1, vertexID2: 5, weight: 20);
```

Remove vertex and edge in Graph

```
aGraph.removeEdge(vertexID1: 1, vertexID2: 5);
aGraph.removeVertex(label: "seven");
```

Create Matrix version of Graph and print it.Also print the original Graph

```
System.out.println(x: "\nExport matrix edges printing : \n");
for (Double[] matrixEdges : aGraph.exportMatrix())
 System.out.print(Arrays.deepToString(matrixEdges) + "\n");

System.out.println(x: "\nOriginal graph printing: \n");
aGraph.printGraph();
```

Create filteredGraph by using filterVertices() and print the filteredGraph

```
MyGraph filteredGraph = aGraph.filterVertices(key: "drink", filter: "tea");

System.out.println(x: "\nFiltered graph printing: \n");
filteredGraph.printGraph();
```

Do BreadthFirstSearch and print the calculated total distance results

```
System.out.println(x: "\n-----Question 2-----");
Double[] result = BreadthFirstSearch.breadthFirstSearch(aGraph,start: 0);

System.out.println(x: "-----------BFS----- DFS");
for(int i=0;i<result.length;i++)
    System.out.printf(format: "vertex  %d :  %3.1f       ?\n",i,result[i]);
```

5. RUNNING AND RESULTS

```
-----Question 1-----

Export matrix edges printing :

[null, 3.0, null, 5.0, null, null, null]
[null, null, 7.0, null, null, null, null]
[null, null, null, null, 4.0, 10.0, null]
[null, null, 1.0, null, null, null, null]
[null, null, null, null, null, null, null]
[null, null, null, null, null, null, 5.0]
[null, null, null, null, null, null, null]

Original graph printing:

Source_vertex -> Destination_vertex  (Weight of edge)

0 -> 3  (5.0)
0 -> 1  (3.0)
1 -> 2  (7.0)
2 -> 4  (4.0)
2 -> 5  (10.0)
3 -> 2  (1.0)
5 -> 6  (5.0)



Filtered graph printing:

Source_vertex -> Destination_vertex  (Weight of edge)

2 -> 4  (4.0)
2 -> 5  (10.0)
3 -> 2  (1.0)



-----Question 2-----
----------BFS----- DFS
vertex  0 :  0.0       ?
vertex  1 :  3.0       ?
vertex  2 :  6.0       ?
vertex  3 :  5.0       ?
vertex  4 :  10.0       ?
vertex  5 :  16.0       ?
vertex  6 :  21.0       ?
```

6. TIME COMPLEXITY ANALYSIS

newVertex() -> θ(1)

addVertex() -> θ(1)

addEdge() -> O(n) n is the number of edges of a vertex

removeEdge() -> O(n) n is the number of edges of a vertex

removeVertex() -> O(m) m is the number of all edges in graph

filterVertices() -> O(n*m) n is the number of edges of a vertex
m is the number of all edges in graph

exportMatrix() -> θ(m) m is the number of all edges in graph

printGraph() -> θ(m) m is the number of all edges in graph