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

Çağrı Çaycı
1901042629

1. SYSTEM REQUIREMENTS

a. Graph Interface

The Graph interface is required to implement MyGraph class. The Graph interface must be implemented as in the book.

b. DynamicGraph Interface

The DynamicGraph interface is required to implement MyGraph class. The DynamicGraph interface must be implemented as in homework document.

c. MyGraph Class

To implement MyGraph class five data fields are required. Three integers to keep capacity, number of deleted vertex and size. A vertex array is required to keep vertices. A Boolean value is required to set the Graph as directed or undirected.

MyGraph class implements both Graph and DynamicGraph interface. For this reason, MyGraph class must have implementations of methods of both Graph and DynamicGraph interfaces. Besides these methods, MyGraph class also need the following methods: reallocate method (to reallocate the vertex array when it is full), addPairs method (to add some key-value pairs to some vertex).

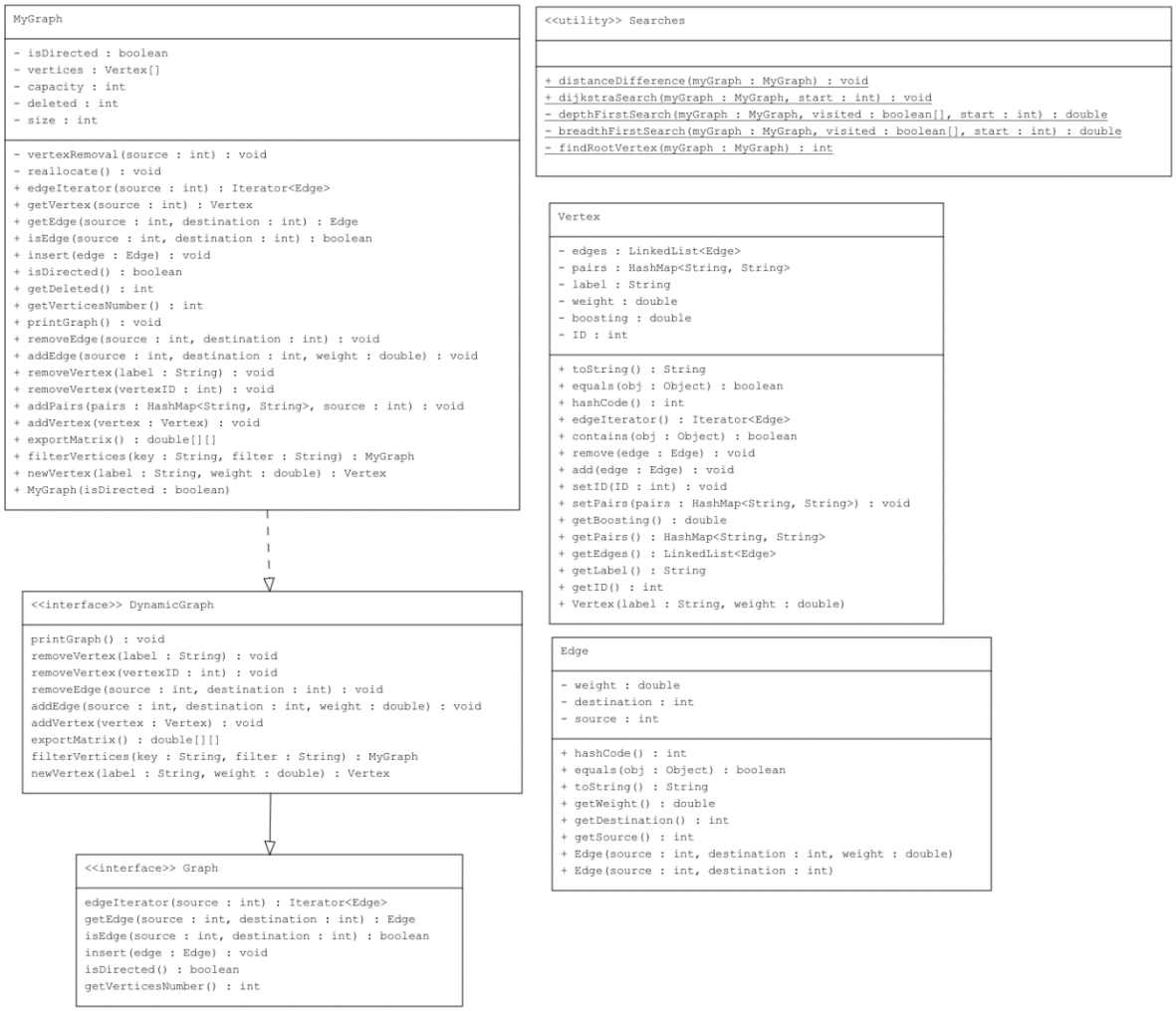
d. Vertex Class

To implement Vertex class six data fields are required. One integer to keep ID, two doubles to keep weight and boosting value, one String to keep label, one LinkedList to keep edges, one HashMap to keep key-value pairs. Following methods are required to implement Vertex class: toString, equals, hashCode, edgeIterator (creates an iterator to traverse Edge LinkedList), contains (to check whether the vertex has the edge or not), remove (to remove an edge from the vertex), add (to add an edge to the vertex). A constructor which gets String and double as parameter and some getter and setter methods are also required.

e. Edge Class

To implement Edge class 3 data fields are required. Two integers to keep source and destination, one double to keep weight. Following methods are required to implement Vertex class: toString, equals, hashCode. Two constructors are required, one gets source and destination of the edge, the other one gets source, destination, and weight of the edge. Some getter methods are also required.

2. USE CASE AND CLASS DIAGRAMS



3. PROBLEM SOLUTION APPROACH

a. Adding a Vertex to the Graph

MyGraph class keeps vertices as vertex array. So, if there is an empty place on vertex array, adds the new vertex to the empty place on vertex array. If there is no empty place on vertex array, adds the new vertex after reallocating the vertex array.

b. Removing a Vertex from the Graph

MyGraph class keeps vertices as vertex array. To remove a vertex from the graph, all the edges which points at the vertex must be removed from the graph. To remove the edges, if the graph is undirected, the edges which points at the vertex can be found with edges of the vertex. Otherwise, the edges must be found by checking all the edges on the graph. After all edges which points at the vertex removed, the position of the vertex on the vertex array sets as null.

c. Adding an Edge to the Graph

To add an edge to the graph, both source and destination vertex should be valid. If the graph is undirected, adds the edge both source vertex and destination vertex, otherwise adds it only source vertex. Vertex class keeps edges as LinkedList so, add method is enough to add an edge to the vertex.

d. Removing an Edge from the Graph

To remove an edge to the graph, both source and destination vertex should be valid. If the graph is undirected, finds the edge and removes it both source vertex and destination vertex, otherwise removes it only source vertex. Vertex class keeps edges as LinkedList so, remove method is enough to remove an edge from the vertex.

e. Filtering Vertices on the Graph

To create a subgraph which consists of the vertex which has a unique key-value pair, firstly all vertices and edges must be added to the subgraph. After that, all vertices should be checked, if the vertex does not contain the unique key-value pair, removes the vertex from the graph.

f. Converting the Graph into a Matrix

A 2D matrix must be generated and it must be filled with infinite value. After that weight of all edges must be placed to the correct position according to their source vertex ID and destination vertex ID.

g. Finding Distance by Breadth First Search

To find total distance by Breadth First Search, all vertices must be visited. The algorithm starts from first non-empty vertex, and it continues with its neighbors and marks the vertex as visited. Adds the distance between the vertex and its neighbors to the total distance. The algorithm continues until there are no unvisited vertex.

h. Finding Distance by Depth First Search

To find total distance by Depth First Search, all vertices must be visited. The algorithm starts from first non-empty vertex, and it continues with its neighbor which has the minimum distance from the vertex and marks the vertex as visited. Adds the distance between the vertex and its neighbor to the total distance. The algorithm continues until there are no unvisited vertex.

i. Finding Shortest Path with Dijkstra Algorithm

The algorithm finds the shortest path to the other vertices from the given vertex. It starts from the given vertex, the distance the vertex to itself sets as 0 and the vertex sets as visited. If the distance of a vertex is bigger than sum of the distance of the source vertex and weight of the edge and boosting value, the distance of the vertex sets as the sum of the distance of the source vertex and weight of the edge and boosting value. The algorithm continues until there are no unvisited vertex.

4. TEST CASES

The following test cases are for both directed and undirected graphs.

a. Creating a Vertex.
b. Adding a Vertex to the Graph.
c. Adding an Edge which is unconnected to graph.
d. Adding an Edge which is connected to graph.
e. Removing an Edge which is not on the graph.
f. Removing an Edge which is already on the graph.
g. Removing a Vertex with its ID which is not on the graph.
h. Removing a Vertex with its label which is not on the graph.
i. Removing a Vertex with its ID which is already on the graph.
j. Removing a Vertex with is label which is already on the graph.
k. Converting the graph into matrix.
l. Creating a subgraph which has a unique key-value pair.
m. Finding total distance of the path for accessing each vertex by Breadth First Search Algorithm.
n. Finding the total distance of the path for accessing each vertex by Depth First Search Algorithm.
o. Finding the total distance of each vertex to the start vertex by Dijkstra Algorithm.

5. RUNNING AND RESULTS

Results are added to the homework folder.