GTU Department of Computer Engineering

CSE 222/505 - Spring 2022

Homework #8 Report

Serhat SARI
200104004028

1- System Requirements

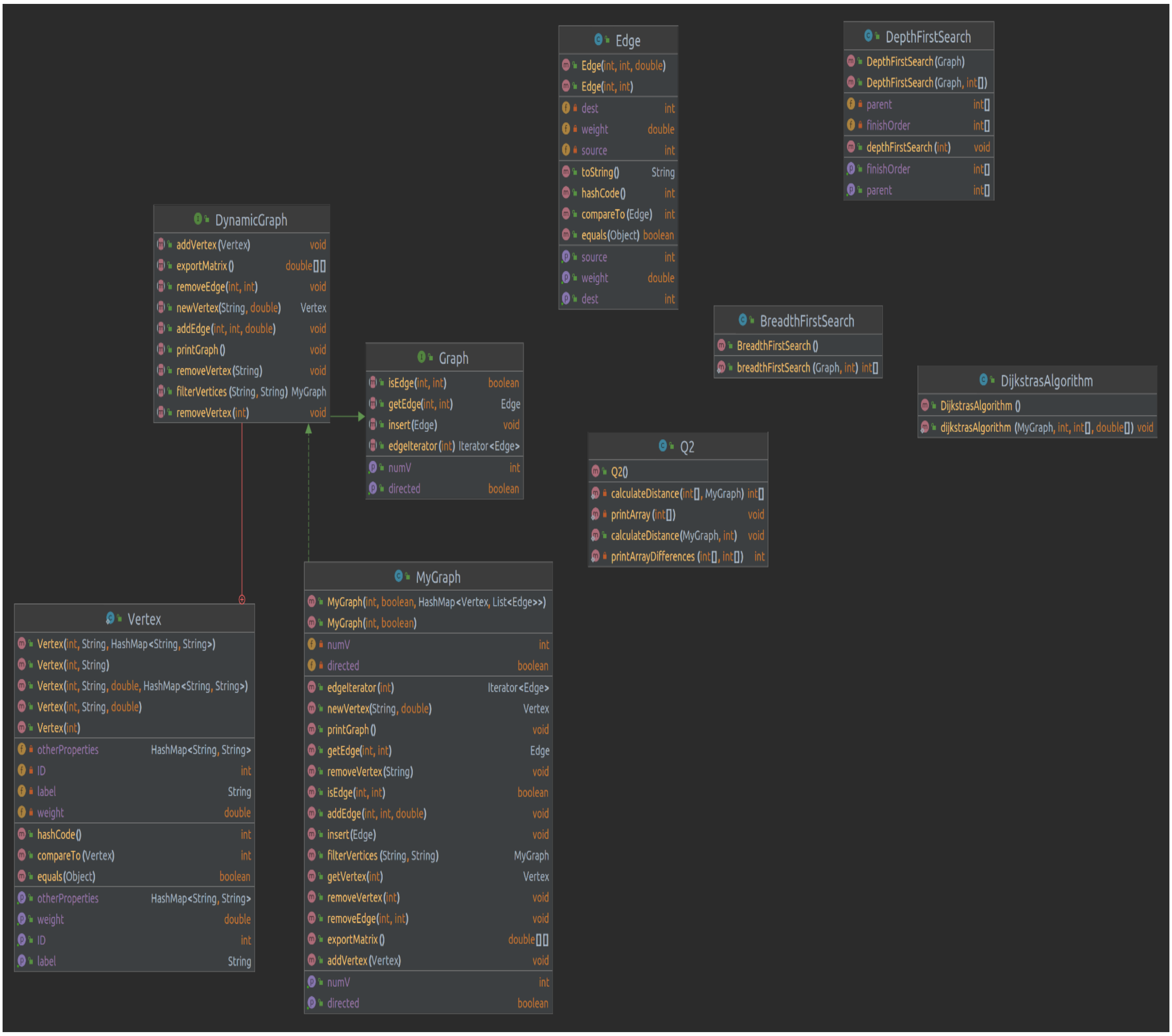
a- Non-Functional System Requirements

1- Back-end Software : Java 11
2- Software should be able to compile with “javac” on a linux distribution.

b- Functional System Requirements

1- You can use MyGraph to add vertex and edge to the graph.
2- You can use MyGraph to remove vertex and edge from the graph.
3- You can use all methods of Graph interface with MyGraph class.
4- You can traverse graph with the BFS and DFS implementation of mine.
5- You can use Dijkstras Algorithm for calculating the shortest path with my implementation, also there is a boosting value.

2- Class Diagram



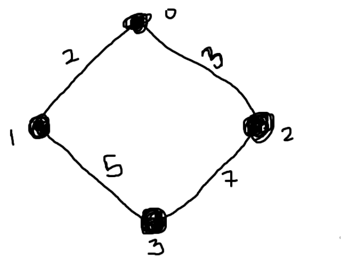
3- Problem Solution Approach

Q1:

For the first problem, i have implemented MyGraph class and DynamicGraph interface. Our problem was designing a class that doesn’t represent verteces as numbers (indexes).
We should have represented them as objects.
So I used HashMap to represent verteces as objects.
HashMap<Vertex, List<Edge>> : this is my data structure to store the graph.
In this implementation, i stored verteces as objects. To use HashMap, i implemented vertex equals and hashCode methods.
After this implementation, task became easy to accomplish. I implemented all methods that is wanted from us.

Q2:

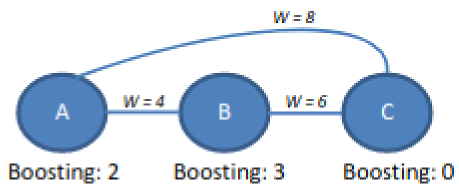
For the second problem, i redesigned the algorithms of BFS and DFS.
For BFS, When i go the level 2 from level 1. I used the less weighted edge.
For DFS , When we recursively go the any up level, we choose the less weighted edge.



For example, for BFS, going 0 from to 3 is done with the path: 0 - 1 - 3
because we chose the shorter alternative.

Q3:

For the third problem, i have changed a little bit of Dijkstra’s Algorithm.
I also used the boosting values of the middle verteces.
And i used the example in the PDF as my example.
It works correctly.
To hold the boosting values, i have HashMap<String, String> data structure in the Vertex class. When i want to add a boosting value, i should add “boosting” as key, and the boosting value as value.



For example, going from A to C is done with A - B - C because of the boosting value of B.

Complexity Analysis

```java
public Vertex newVertex(String label, double weight) {
  return new Vertex(numV - 1, label, weight);
}
```

Complexity: theta(1)
Constant time method.

```java
public void addVertex(DynamicGraph.Vertex new_vertex) {
    numV++;
    edges.put(new_vertex, new LinkedList<>());
 }
```

Complexity: theta(1)
put() method in HashMap is constant time.

```java
public void addEdge(int vertexID1, int vertexID2, double weight) {
    edges
      .get(new Vertex(vertexID1))
      .add(new Edge(vertexID1, vertexID2, weight));
    if (!directed) {
      edges
        .get(new Vertex(vertexID2))
        .add(new Edge(vertexID2, vertexID1, weight));
    }
 }
```

Complexity: theta(1)
get() method of HashMap is constant time,
add method of LinkedList is constant time.

```java
public void removeEdge(int vertexID1, int vertexID2) {
  edges.get(new Vertex(vertexID1)).remove(new Edge(vertexID1, vertexID2));

  if (!directed) {
    edges.get(new Vertex(vertexID2)).remove(new Edge(vertexID1, vertexID2));
  }
}
```

Complexity: O(n)
get() method of HashMap is constant time.
remove method of LinkedList is O(n)

```java
public void removeVertex(int vertexID) {
  numV--;
  edges.remove(new Vertex(vertexID));
  for (List<Edge> entry : edges.values()) {
    for (Edge e : entry) {
      if (e.getSource() == vertexID || e.getDest() == vertexID) {
        entry.remove(e);
      }
    }
  }
}
```

Complexity: theta(E)
it will traverse all edges, so complexity is theta(EdgeNumber)

```java
public MyGraph filterVertices(String key, String filter) {
  HashMap<Vertex, List<Edge>> vertices = new HashMap<>();
  int vertNum = 0;
  for (Entry<Vertex, List<Edge>> entry : edges.entrySet()) {
    if (entry.getKey().getOtherProperties().containsKey(key)) {
      String value = entry.getKey().getOtherProperties().get(key);
      if (value.equals(filter)) {
        vertices.put(entry.getKey(), entry.getValue());
      }
    }
  }

  return new MyGraph(vertNum, directed, vertices);
}
```

Complexity: theta(V), because it will traverse as much as the vertex number.

```java
public double[][] exportMatrix() {
  double matrixEdges[][] = new double[numV][numV];
  for (int i = 0; i < numV; i++) {
    for (int k = 0; k < numV; k++) {
      if (edges.get(new Vertex(i)).contains(new Edge(i, k))) {
        matrixEdges[i][k] = 1.0;
      } else {
        matrixEdges[i][k] = 0;
      }
    }
  }
  return matrixEdges;
}
```

Complexity: theta(V ^2)
it will traverse the square of
vertex number.

```java
public void printGraph() {
  System.out.println("Graph: ");
  for (Entry<Vertex, List<Edge>> entry : edges.entrySet()) {
    System.out.print(entry.getKey().getID() + ": ");
    for (Edge edge : entry.getValue()) {
      System.out.print(edge);
    }
    System.out.println();
  }
}
```

Complexity: theta(E)
it will traverse all edges, so complexity is theta(EdgeNumber)

```java
public void removeVertex(String label) {
  int deletedID;
  for (Entry<Vertex, List<Edge>> entry : edges.entrySet()) {
    if (entry.getKey().getLabel().equals(label)) {
      deletedID = entry.getKey().getID();
      edges.remove(entry);
      for (List<Edge> edg : edges.values()) {
        for (Edge e : edg) {
          if (e.getSource() == deletedID || e.getDest() == deletedID) {
            edg.remove(e);
          }
        }
      }
    }
  }
}
```

Complexity: theta(E)
it will traverse all edges, so complexity is theta(EdgeNumber)

4- Test Cases

TESTING Q1

```java
MyGraph graph = new MyGraph(0, false);
System.out.println("addVertex() TEST");
graph.addVertex(new Vertex(0, "A"));
graph.addVertex(new Vertex(1, "B"));
graph.addVertex(new Vertex(2, "C"));
System.out.println("getNumV() TEST");
System.out.println("Number of vertices in the graph: " + graph.getNumV());
System.out.println("isDirected() TEST");
System.out.println("Is graph directed: " + graph.isDirected());
System.out.println("printGraph() TEST");
graph.printGraph();
System.out.println("addEdge() TEST");
graph.addEdge(0, 2, 2);
graph.addEdge(1, 2, 3);
graph.printGraph();
System.out.println("insert() TEST");
graph.insert(new Edge(0, 1, 1));
graph.printGraph();
System.out.println("isEdge() TEST");
System.out.println("isEdge 0 - 2 : " + graph.isEdge(0, 2));
System.out.println("getEdge() TEST");
System.out.println("getEdge 0 - 2: " + graph.getEdge(0, 2));
System.out.println("newVertex() TEST");
System.out.println("newVertex method: " + graph.newVertex("t", 3));
System.out.println("exportMatrix() TEST");
System.out.println("exportMatrix: ");
print2DArray(graph.exportMatrix());
System.out.println("removeVertex() TEST");
graph.removeVertex(2);
graph.printGraph();
System.out.println("removeEdge() TEST");
graph.removeEdge(0, 1);
graph.printGraph();
System.out.println("filterVertices() TEST");
MyGraph graph1 = new MyGraph(0, false);
HashMap<String, String> test = new HashMap<>();
test.put("ky", "0A");
HashMap<String, String> test1 = new HashMap<>();
test1.put("ky", "0A");
graph1.addVertex(new Vertex(0, "a", test));
graph1.addVertex(new Vertex(1, "b", test1));
graph1.addEdge(0, 1, 3);
graph1.filterVertices("ky", "0A").printGraph();
System.out.println("AS YOU CAN SEE ALL METHODS WORKS CORRECTLY!");
```

TEST Q2

```
MyGraph graph = new MyGraph(5, false);
graph.addEdge(0, 1, 2);
graph.addEdge(0, 4, 2);
graph.addEdge(1, 2, 3);
graph.addEdge(2, 3, 6);
graph.addEdge(3, 4, 4);
graph.addEdge(2, 4, 7);
graph.addEdge(0, 2, 2);
Q2.calculateDistance(graph, 0);
```

TEST Q3

```java
MyGraph graph = new MyGraph(0, false);
System.out.println("For this problem, ");
System.out.println("I used the exact example in the PDF");
HashMap<String, String> firstAdditional = new HashMap<>();
firstAdditional.put("boosting", "2.0");
HashMap<String, String> secondAdditional = new HashMap<>();
secondAdditional.put("boosting", "3.0");
HashMap<String, String> thirdAdditional = new HashMap<>();
int[] pred = new int[3];
double[] dist = new double[3];
thirdAdditional.put("boosting", "0.0");
graph.addVertex(new Vertex(0, "A", firstAdditional));
graph.addVertex(new Vertex(1, "B", secondAdditional));
graph.addVertex(new Vertex(2, "C", thirdAdditional));

graph.addEdge(0, 1, 4);
graph.addEdge(0, 2, 8);
graph.addEdge(1, 2, 6);
DijkstrasAlgorithm.dijkstrasAlgorithm(graph, 0, pred, dist);
System.out.println("PRED ARRAY:");
printArray(pred);
System.out.println("DIST ARRAY:");
printArray(dist);
```

5- Running Command and Results

```
TESTING Q1
addVertex() TEST
getNumV() TEST
Number of vertices in the graph: 3
isDirected() TEST
Is graph directed: false
printGraph() TEST
Graph:
0:
1:
2:
addEdge() TEST
Graph:
0: [(0, 2): 2.0]
1: [(1, 2): 3.0]
2: [(2, 0): 2.0][(2, 1): 3.0]
insert() TEST
Graph:
0: [(0, 2): 2.0][(0, 1): 1.0]
1: [(1, 2): 3.0][(1, 0): 1.0]
2: [(2, 0): 2.0][(2, 1): 3.0]
isEdge() TEST
isEdge 0 - 2 : true
getEdge() TEST
getEdge 0 - 2: [(0, 2): 2.0]
newVertex() TEST
newVertex method: src.DynamicGraph$Vertex@2
exportMatrix() TEST
exportMatrix:
0.0 1.0 1.0
1.0 0.0 1.0
1.0 1.0 0.0
removeVertex() TEST
Graph:
0: [(0, 1): 1.0]
1: [(1, 0): 1.0]
```

```
removeEdge() TEST
Graph:
0:
1:
filterVertices() TEST
Graph:
0: [(0, 1): 3.0]
1: [(1, 0): 3.0]
AS YOU CAN SEE ALL METHODS WORKS CORRECTLY!


TESTING Q2
DFS PARENT ARRAY:
-1 0 1 2 3
BFS PARENT ARRAY:
-1 0 0 4 0
DFS Total: 26
BFS Total: 10
Total Difference 16


TESTING Q3
For this problem,
I used the exact example in the PDF
PRED ARRAY:
0 0 1
DIST ARRAY:
0.0 4.0 7.0
```