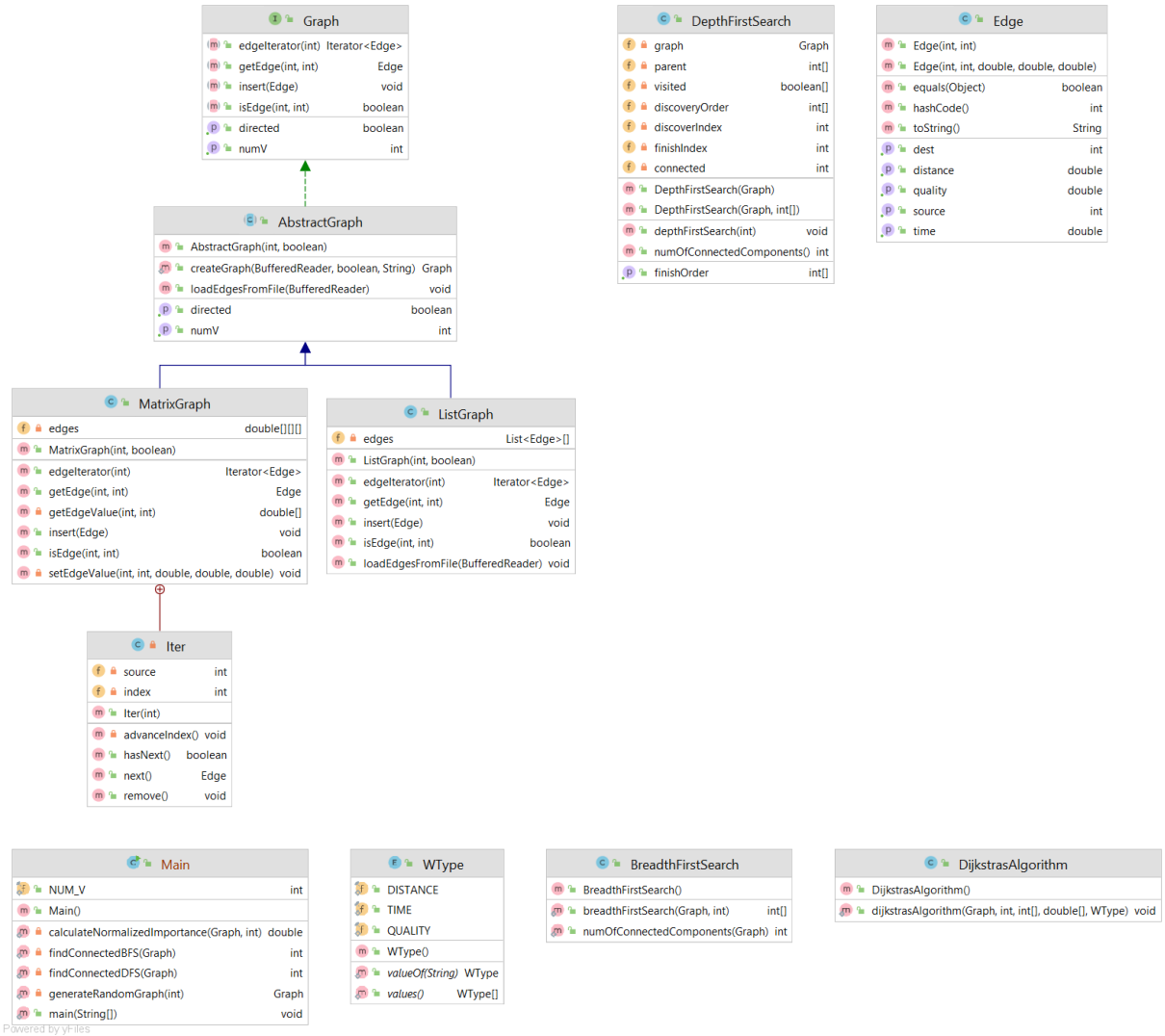
GTU Department of Computer Engineering
CSE 222/505 - Spring 2021
Homework #8 Report

Ömer Faruk Bitikçioğlu
161044010

1. SYSTEM REQUIREMENTS

Requirements are the same with homework pdf.
The homework is developed on IntelliJ IDE.
Java version 13.

2. CLASS DIAGRAMS



3. PROBLEM SOLUTION APPROACH

I used the Graph interface, AbstractGraph class, and implementations of ListGraph MatrixGraph and other needed graph related classes from the book directly.

Part1

Since the DijkstrasAlgorithm uses Graph interface as an input, the algorithm works fine for both implementations of ListGraph and MatrixGraph already. So, the problem 1.1 is solved.

To achieve edges to have several properties, I deleted the weight and instead added three different quantities: time, distance, and quality.

I rearranged the MatrixGraph and ListGraph implementations of Graph to work efficiently with these three kinds of weights.

Rearranging MatrixGraph and Edge classes was enough since ListGraph just uses linked list to hold the edges. On the other hand, MatrixGraph was holding the weights on a 2d array. It becomes 3d array to hold the other different weights.

I added a new parameter to the Dijkstras's Algorithm which is WType (short form of weight type). WType can be DISTANCE, TIME, or QUALITY. Algorithm works fine for each weight type and Graph implementation.

The algorithm works fine with associative operators. I didn't get the idea exactly but when I changed + with * it worked fine.

Part2

I took the breath first search algorithm from the book. Breath first search returns an array of parent indices. If one vertex has no parent, it means it is a separated component. It may have children but counting only -1 values (no parents) will give the result of number of connected components. But every time we start breath first search with the vertex has a value of -1 it removes the older parent values. I checked these differences and corrected the parent values comparing with older parent values.

I also took the depth first search algorithm from the book. I counted every time it encounters an unvisited vertex while doing depth first search. I hold the counted value as connected in the class and when needed I returned this value.

Part3

I tried to use breath first search algorithm to find the shortest path for each pair in a graph and tried to calculate importance of vertex v but it doesn't count. There is something wrong that I did here.

```java
/**
 * Calculates and returns the importance of vertex v
 * in the given graph.
 * @param graph The graph to be traversed
 * @param v The vertex to find its importance
 * @return The importance of vertex v
 */
private static double calculateNormalizedImportance(Graph graph, int v) {
    if (v < 0 || v > graph.getNumV()) {
        throw new IndexOutOfBoundsException();
    }
    int sigmaUW = 0;
    int sigmaUWV = 0;
    // Pick an u and w vertex and find if there is a shortest path between them.
    for (int i = 0; i < graph.getNumV(); ++i) {
        int[] parent = BreadthFirstSearch.breadthFirstSearch(graph, i);
        for (int j = 0; j < graph.getNumV() && i != j; ++j) {
            // Construct the path.
            Stack thePath = new Stack();
            while (parent[j] != -1) {
                thePath.push(j);
                j = parent[j];
                // Count the paths
                if (j == i) {
                    sigmaUW++;
                    break;
                }
            }
            if (thePath.contains(v)) {
                sigmaUWV++;
            }
        }
    }
    return (sigmaUWV / sigmaUW) / Math.pow(graph.getNumV(),2);
}
```

4. TEST CASES

- Create a ListGraph and MatrixGraph with same vertices and weights and use Dijktra's Algorithm to see if it works for both implementations.

- Give different values of distance, time, and quality properties and calculate the shortest distance using the algorithm for each.

- Replace addition operation in the algorithm when calculating paths with multiplication.

- Generate 10 random graphs for each graph size of 1000, 2000, 5000 and 10000 vertices and various sparsity, run your methods and collect the running times.

- Show results of the BFS and DFS counting of connected parts of the graph.

5. RUNNING AND RESULTS

```
Use Dijktra's algorithm for ListGraph
d[0]=0.0 p[0]=0
d[1]=5.0 p[1]=0
d[2]=15.0 p[2]=1
d[3]=35.0 p[3]=2
d[4]=50.0 p[4]=2

Use Dijktra's algorithm for MatrixGraph
d[0]=0.0 p[0]=0
d[1]=5.0 p[1]=0
d[2]=15.0 p[2]=1
d[3]=35.0 p[3]=2
d[4]=50.0 p[4]=2
```

```
Use Dijktra's algorithm for ListGraph
Calculate for distance
d[0]=0.0 p[0]=0
d[1]=5.0 p[1]=0
d[2]=15.0 p[2]=1
d[3]=35.0 p[3]=2
d[4]=50.0 p[4]=2

Calculate for time
d[0]=0.0 p[0]=0
d[1]=15.0 p[1]=0
d[2]=20.0 p[2]=0
d[3]=25.0 p[3]=1
d[4]=69.0 p[4]=3

Calculate for quality
d[0]=0.0 p[0]=0
d[1]=13.0 p[1]=2
d[2]=2.0 p[2]=0
d[3]=69.0 p[3]=1
d[4]=115.0 p[4]=3
```

```
Use Dijktra's algorithm for MatrixGraph
Calculate for distance
d[0]=0.0 p[0]=0
d[1]=5.0 p[1]=0
d[2]=15.0 p[2]=1
d[3]=35.0 p[3]=2
d[4]=50.0 p[4]=2

Calculate for time
d[0]=0.0 p[0]=0
d[1]=15.0 p[1]=0
d[2]=20.0 p[2]=0
d[3]=25.0 p[3]=1
d[4]=69.0 p[4]=3

Calculate for quality
d[0]=0.0 p[0]=0
d[1]=13.0 p[1]=2
d[2]=2.0 p[2]=0
d[3]=69.0 p[3]=1
d[4]=115.0 p[4]=3
```

```
Use Dijktra's algorithm for MatrixGraph
Calculate for distance
d[0]=0.0 p[0]=0
d[1]=5.0 p[1]=0
d[2]=50.0 p[2]=1
d[3]=1000.0 p[3]=2
d[4]=50000.0 p[4]=3

Calculate for time
d[0]=0.0 p[0]=0
d[1]=15.0 p[1]=0
d[2]=20.0 p[2]=0
d[3]=460.0 p[3]=2
d[4]=20240.0 p[4]=3

Calculate for quality
d[0]=0.0 p[0]=0
d[1]=22.0 p[1]=2
d[2]=2.0 p[2]=0
d[3]=23552.0 p[3]=4
d[4]=512.0 p[4]=2
```

```
Use Dijktra's algorithm for ListGraph
Calculate for distance
d[0]=0.0 p[0]=0
d[1]=5.0 p[1]=0
d[2]=50.0 p[2]=1
d[3]=1000.0 p[3]=2
d[4]=50000.0 p[4]=3

Calculate for time
d[0]=0.0 p[0]=0
d[1]=15.0 p[1]=0
d[2]=20.0 p[2]=0
d[3]=460.0 p[3]=2
d[4]=20240.0 p[4]=3

Calculate for quality
d[0]=0.0 p[0]=0
d[1]=22.0 p[1]=2
d[2]=2.0 p[2]=0
d[3]=23552.0 p[3]=4
d[4]=512.0 p[4]=2
```

```
Testing BFS
Testing with size 1000
Case 0 connected: 4 time passed: 5307300
Case 1 connected: 3 time passed: 6644300
Case 2 connected: 5 time passed: 7657400
Case 3 connected: 5 time passed: 9404200
Case 4 connected: 2 time passed: 10718800
Case 5 connected: 8 time passed: 13430000
Case 6 connected: 5 time passed: 15437800
Case 7 connected: 1 time passed: 17387200
Case 8 connected: 7 time passed: 22916300
Case 9 connected: 5 time passed: 23747300

Testing with size 2000
Case 0 connected: 5 time passed: 25815300
Case 1 connected: 7 time passed: 34841500
Case 2 connected: 8 time passed: 39325700
Case 3 connected: 4 time passed: 50312700
Case 4 connected: 3 time passed: 51850800
Case 5 connected: 7 time passed: 60501900
Case 6 connected: 2 time passed: 62132800
Case 7 connected: 5 time passed: 65230000
Case 8 connected: 4 time passed: 71712300
Case 9 connected: 5 time passed: 74878000

Testing with size 5000
Case 0 connected: 3 time passed: 78743200
Case 1 connected: 7 time passed: 84115400
Case 2 connected: 3 time passed: 97091900
Case 3 connected: 6 time passed: 99968300
Case 4 connected: 4 time passed: 104067100
Case 5 connected: 8 time passed: 108736400
Case 6 connected: 5 time passed: 112029600
Case 7 connected: 4 time passed: 116592200
Case 8 connected: 3 time passed: 119134000
Case 9 connected: 6 time passed: 121336400
```

```
Testing with size 10000
Case 0 connected: 5 time passed: 125896900
Case 1 connected: 6 time passed: 128920200
Case 2 connected: 4 time passed: 131809400
Case 3 connected: 6 time passed: 141348500
Case 4 connected: 5 time passed: 144414100
Case 5 connected: 6 time passed: 147782700
Case 6 connected: 5 time passed: 150688400
Case 7 connected: 7 time passed: 153277300
Case 8 connected: 7 time passed: 156449300
Case 9 connected: 3 time passed: 158546800

Testing DFS
Testing with size 1000
Case 0 connected: 2 time passed: 160486100
Case 1 connected: 7 time passed: 160756500
Case 2 connected: 1 time passed: 160898400
Case 3 connected: 4 time passed: 161041200
Case 4 connected: 3 time passed: 161176600
Case 5 connected: 1 time passed: 161391900
Case 6 connected: 2 time passed: 161614100
Case 7 connected: 5 time passed: 161856700
Case 8 connected: 6 time passed: 162095200
Case 9 connected: 4 time passed: 162325800

Testing with size 2000
Case 0 connected: 6 time passed: 162855100
Case 1 connected: 7 time passed: 163373700
Case 2 connected: 4 time passed: 163917700
Case 3 connected: 2 time passed: 164430200
Case 4 connected: 6 time passed: 164997700
Case 5 connected: 5 time passed: 165851800
Case 6 connected: 2 time passed: 166432100
Case 7 connected: 6 time passed: 166960500
Case 8 connected: 3 time passed: 167512900
Case 9 connected: 2 time passed: 168163800
```

-

```
Testing with size 5000
Case 0 connected: 7 time passed: 170163400
Case 1 connected: 8 time passed: 173328100
Case 2 connected: 3 time passed: 175232200
Case 3 connected: 8 time passed: 178509300
Case 4 connected: 4 time passed: 183514100
Case 5 connected: 4 time passed: 187399700
Case 6 connected: 6 time passed: 188488100
Case 7 connected: 4 time passed: 191975400
Case 8 connected: 5 time passed: 193339200
Case 9 connected: 6 time passed: 194455700

Testing with size 10000
Case 0 connected: 8 time passed: 199299500
Case 1 connected: 5 time passed: 201707800
Case 2 connected: 5 time passed: 204447300
Case 3 connected: 4 time passed: 206328400
Case 4 connected: 5 time passed: 208067400
Case 5 connected: 3 time passed: 209965100
Case 6 connected: 5 time passed: 211757800
Case 7 connected: 4 time passed: 214023000
Case 8 connected: 6 time passed: 216281800
Case 9 connected: 4 time passed: 218145800
```