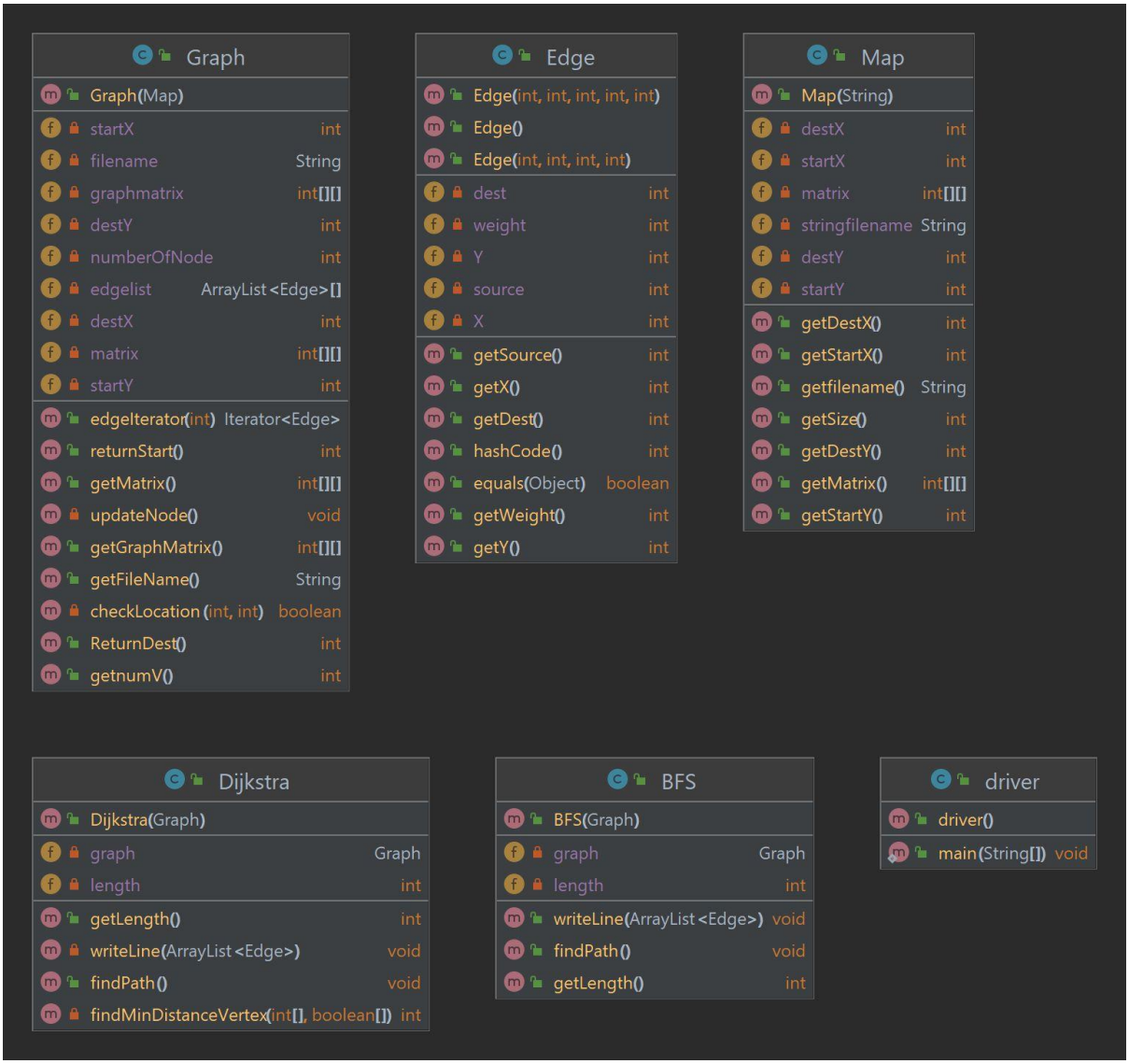
GIT Department of Computer Engineering
CSE 222/505 - Spring 2023
Homework # Report 8

Hamza KONAÇ
210104004202

1. **SYSTEM REQUIREMENTS**

   In this assignment we have to find shortest path for given map.Maps are given the txt file.By using txt file and create map , I have deaclare Map class.Then I have declare Grap class . this class use map class and create graph.In orfer to determine shortest path for given destination I have create 2 class named BFS and Disjkstra.Edge class decleared for strore nodes informaiton

2. **USE CASE AND CLASS DIAGRAMS**



3. **OTHER DIAGRAMS**

Not necessary in this assignment

4. **PROBLEM SOLUTION APPROACH**

   Create MAP:

   I have read file and create a matrix with content of the txt file.

Create graph:

In order to create graph , I simulate each 0 in the matrix a node . With this approach I create a matrix and give node number each zero. Before create node I have declare edge class and this class store node information. Node informations are node's source ,destinations, weight and nodes coordinate.Then I traversal in the matrix and when specified index equal to zero , This node inserted in to adjency list.I perform this operation all matrix

BFS algorithm:

I have apply BFS algorithm on graph.This algorihm determine the shortest path and strore path locations

Dijkstra Algorithm:

I have apply BFS algorithm on graph.This algorihm determine the shortest path and strore path locations

TIME Complexity of the BFS:

```
Queue<Integer> theQueue = new LinkedList<Integer>();
       int[] parent = new int[graph.getnumV()];
       for (int i = 0; i < graph.getnumV(); i++) {
           parent[i] = -1;
       }
     1-Complexity of the this 3 line is Θ(n)
       // Declare array identified and
       // initialize its elements to false.
       boolean[] identified = new boolean[graph.getnumV()];
       identified[graph.returnStart()] = true;
       theQueue.offer(graph.returnStart());
     2-while loops complexity is O(n)
       while (!theQueue.isEmpty()) {

           3-complexity of the remove method is O(n)
           int current = theQueue.remove();

           Iterator<Edge> itr = graph.edgeIterator(current);
           4-Complexity of the this while is O(|E|)
           while (itr.hasNext()) {
               Edge edge = itr.next();
               int neighbor = edge.getDest();
               // If neighbor has not been identified
               if (!identified[neighbor]) {
                   // Mark it identified.
                   identified[neighbor] = true;
                  5-Complexity of the this method is O(n)

                   theQueue.offer(neighbor);
                   parent[neighbor] = current;
               }
           }
       // Finished visiting current.
       }
```

1-this loop assign each element of the array to -1

2-this queue strore each node one time so that it will be O(n)

3-java queue use linked list interface so that it complexity will be O(n)

4-adjency list contains only adjence so that it iterate just adjence element.Therefore it complexity will be O(|E|)

5- java queue use linked list interface so that it complexity will be O(n)

→After this steps we obtain time complexity of the BFS is (n+|E|)

Time complexity of the DIJKSTRA algorithm:

1-Complexitty of the this loop is Θ(n)

```
for (int i = 0; i < graph.getnumV() - 1; i++) {
            3-findMinDistanceVertex complexity is O(n)
            int minVertex = findMinDistanceVertex(distances, visited);
            visited[minVertex] = true;

            Iterator<Edge> itr = graph.edgeIterator(minVertex);
2-complexity of the this loop is O(|E|)
            while (itr.hasNext()) {
                Edge edge=itr.next();
                int adjacentVertex = edge.getDest();
                int weight = 1;

                if (!visited[adjacentVertex] && distances[minVertex] !=
Integer.MAX_VALUE &&
                        distances[minVertex] + weight <
distances[adjacentVertex]) {
                    distances[adjacentVertex] = distances[minVertex] + weight;
                    parent[adjacentVertex] = minVertex;
                }
            }
        }
```

```
private int findMinDistanceVertex(int[] distances, boolean[] visited) {
        int minDistance = Integer.MAX_VALUE;
        int minVertex = -1;
    3-complexity of the this loop is O(n)
        for (int i = 0; i < graph.getnumV(); i++) {
            if (!visited[i] && distances[i] <= minDistance) {
                minDistance = distances[i];
                minVertex = i;
            }
        }

        return minVertex;
    }
```

1-first loop run for each node in the graph

2-complexity of the this loop will be O(|E|) because of iteration

3-complexity of the this method will be O(n) because of it check all node in in the grap

→ After this steps we obtain time complexity of the Dijkstra is (n^2+|E|)

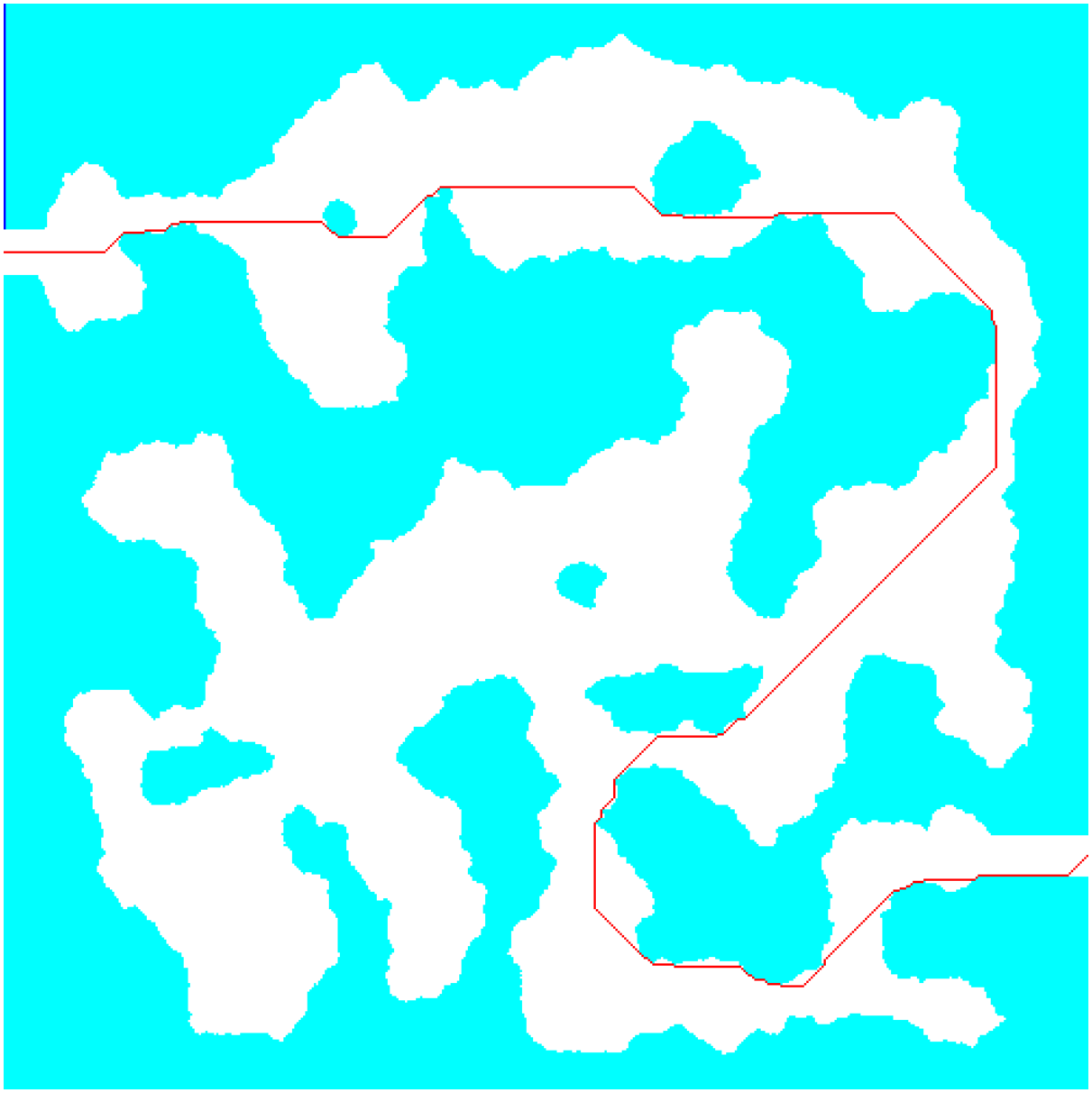
5. TEST CASES

Map01.txt
Tokyo.txt

6. RUNNING AND RESULTS

Map01.txt:

BFS:

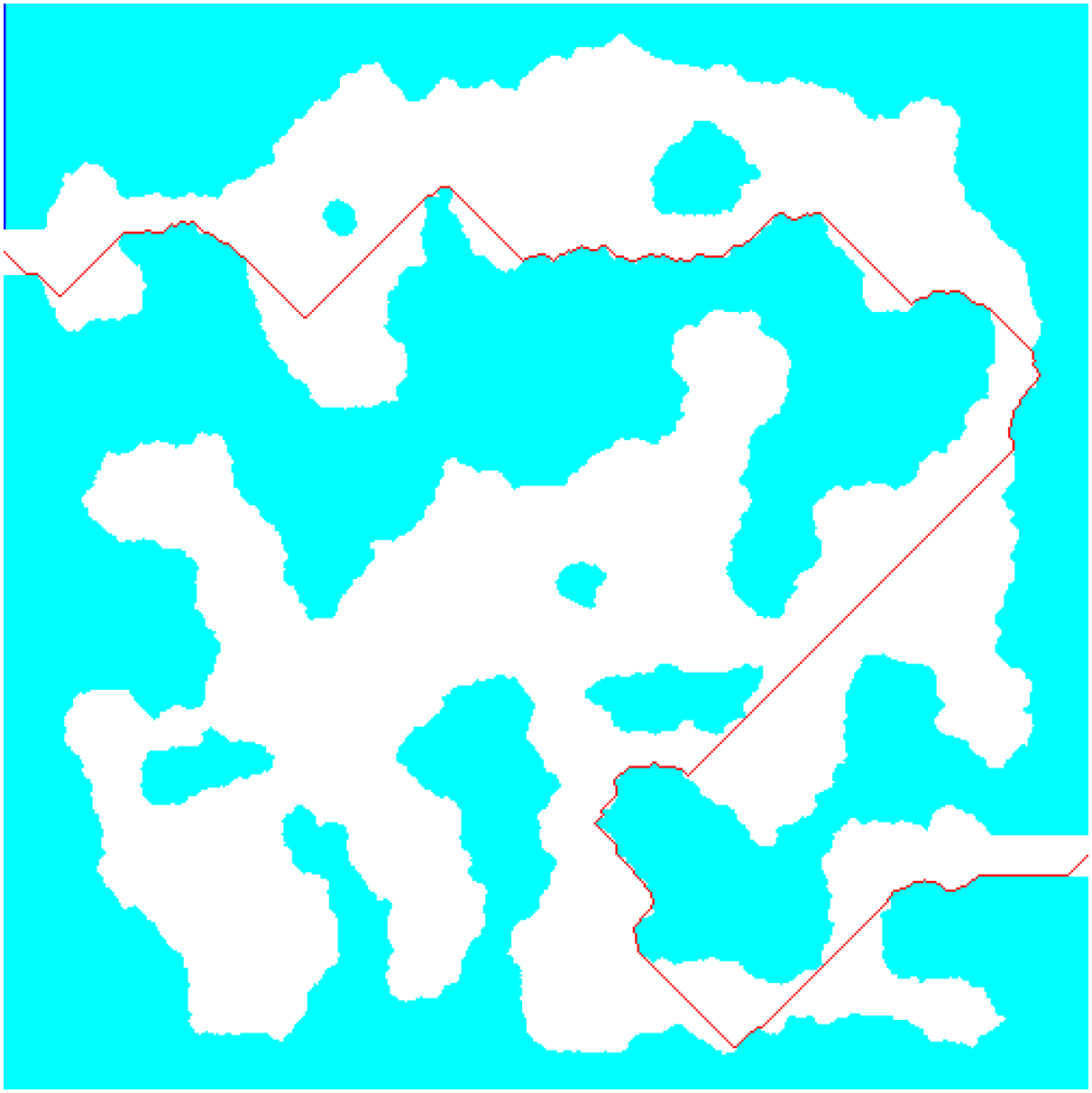


```
FOR map01.txt optimum path coordinate has been written on map01_path.txt

Map image saved successfully on map01.png

Lengh of BST for map01.txt is 992
```

Dijkstra:



```
FOR map01.txt optimum path coordinate has been written on map01_path.txt

Map image saved successfully on map01.png

Lengh of Dijkstra for map01.txt is 992
```

Tokyo.txt

BFS:

```
FOR tokyo.txt optimum path coordinate has been written on tokyo_path.txt

Map image saved successfully on tokyo.png

Lengh of BST for tokyo.txt is 891
```

Dijkstra:

```
FOR tokyo.txt optimum path coordinate has been written on tokyo_path.txt

Map image saved successfully on tokyo.png

Lengh of Dijkstra for tokyo.txt is 891
```

!!!

Output show us algorithms are run correctly. Their sortest path lengths are same

!!!