GTU Department of Computer Engineering
CSE 222 - Homework 8 - Report

Doğukan Taştan 1901042627

Complexity Analysis

BFS Algorithm

It is defined in terms of the number of nodes (V) and edges (E) in the graph.
Time Complexity: O(V + E)
The algorithm visits each vertex once and moves through all adjacent vertices via the edges. Here V represents the number of vertices and E represents the number of edges in the graph.

It takes O(V) time to visit each vertex
It takes O(E) time to check each edge
So the total time complexity of the BFS algorithm is O(V + E).
Space Complexity: O(V)
The algorithm maintains a queue data structure, the cluster visited, and a hashmap.

In the worst-case scenario, all vertices are added to the queue and the visited cluster, so this takes O(V) space. Each entry in the HashMap prev takes up fixed space and in the worst case it takes up O(V) space as it will grab all the vertices.Therefore, the total area complexity is O(V).

Dijkstra Algorithm

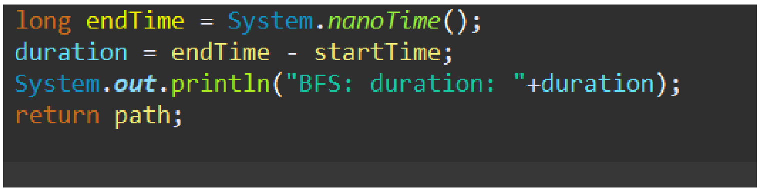
Checking Initialization and Starting Point: The complexity here is fixed, i.e. O(1), because we are just creating a few data structures and performing a check on the start and end nodes.
Main Loop: This loop runs until the priority queue is empty. Each operation inside the loop has a different complexity:
Querying from the queue: This operation takes O(log V) time because polling on a binary stack takes logarithmic time.
Checking and adding visited cluster: This takes O(1) time.
Checking Neighbors: Checking neighbors is actually going around all the edges of the node and collectively takes O(E) time as it is done for each node.
Updating and queuing distances: These operations take O(log V) time because they include operations in the priority queue.
Path reconfiguration: This process takes O(V) time in the worst case scenario when the path includes all nodes.
Writing the path to a file: This takes O(V) time as we iterate over every node in the path.

*Summing all this up, **the time complexity of Dijkstras Algorithm is O((V + E) log V),** assuming the priority queue uses a binary heap data structure. The E term results from visiting all edges of each vertex, and the V term originates from operations in the priority queue.*

space complexity:

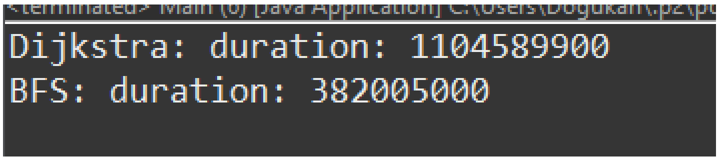
The visited cluster, distance map, and previous node map can each store all vertices, so each takes up O(V) space.The queue can also grab all the vertices in the worst-case scenario, which takes up O(V) space.
The final path list holds at most all vertices, which is O(V).
*Hence, **the space complexity of Dijkstra's Algorithm is O(V).***

Running time performance



```
long endTime = System.nanoTime();
duration = endTime - startTime;
System.out.println("BFS: duration: "+duration);
return path;
```

Working times were measured with time variables added to the code.



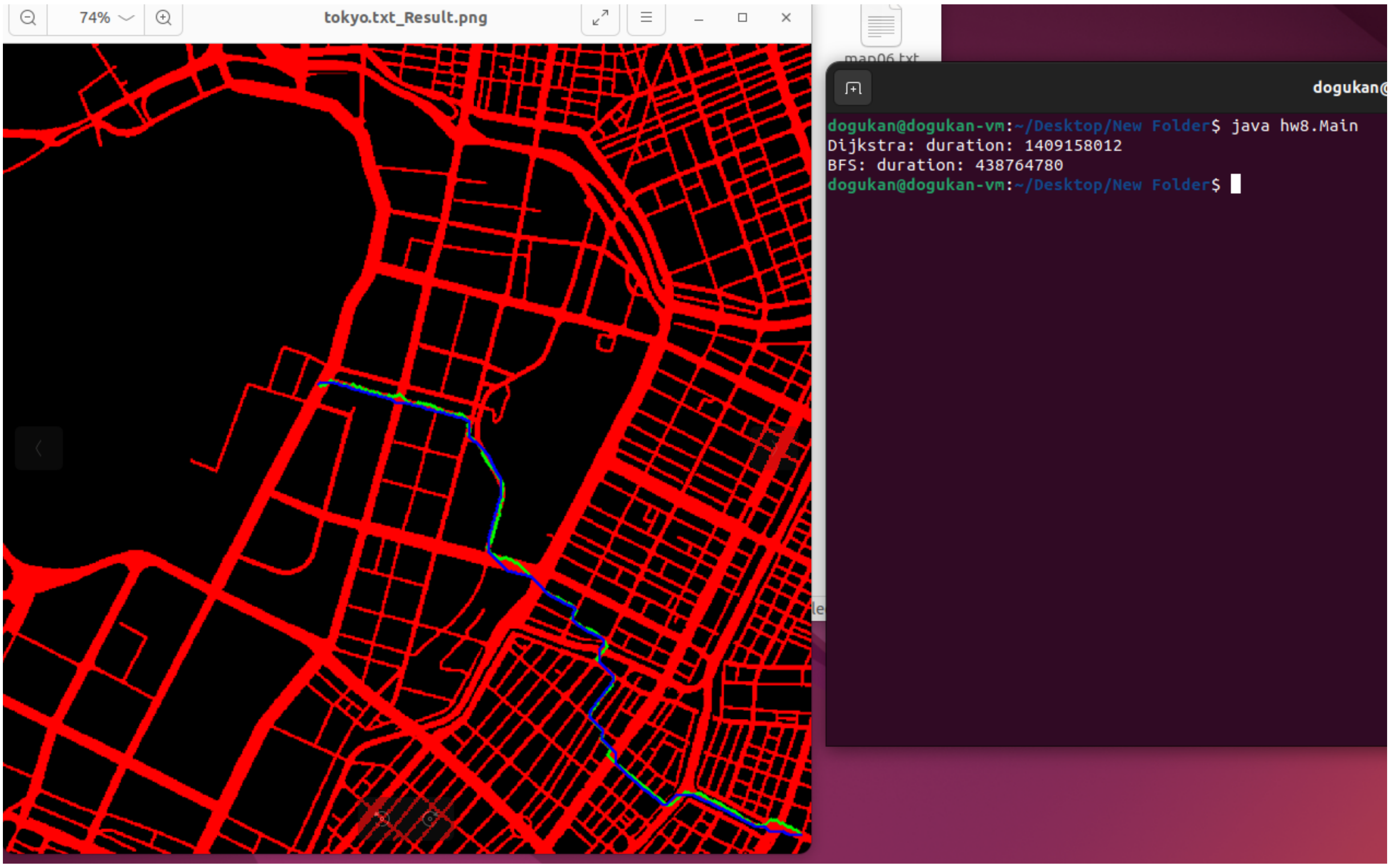
```
<terminated> Main (0) [Java Application] C:\Users\Dogukan\.p2\pl
Dijkstra: duration: 1104589900
BFS: duration: 382005000
```

(nanosecond)

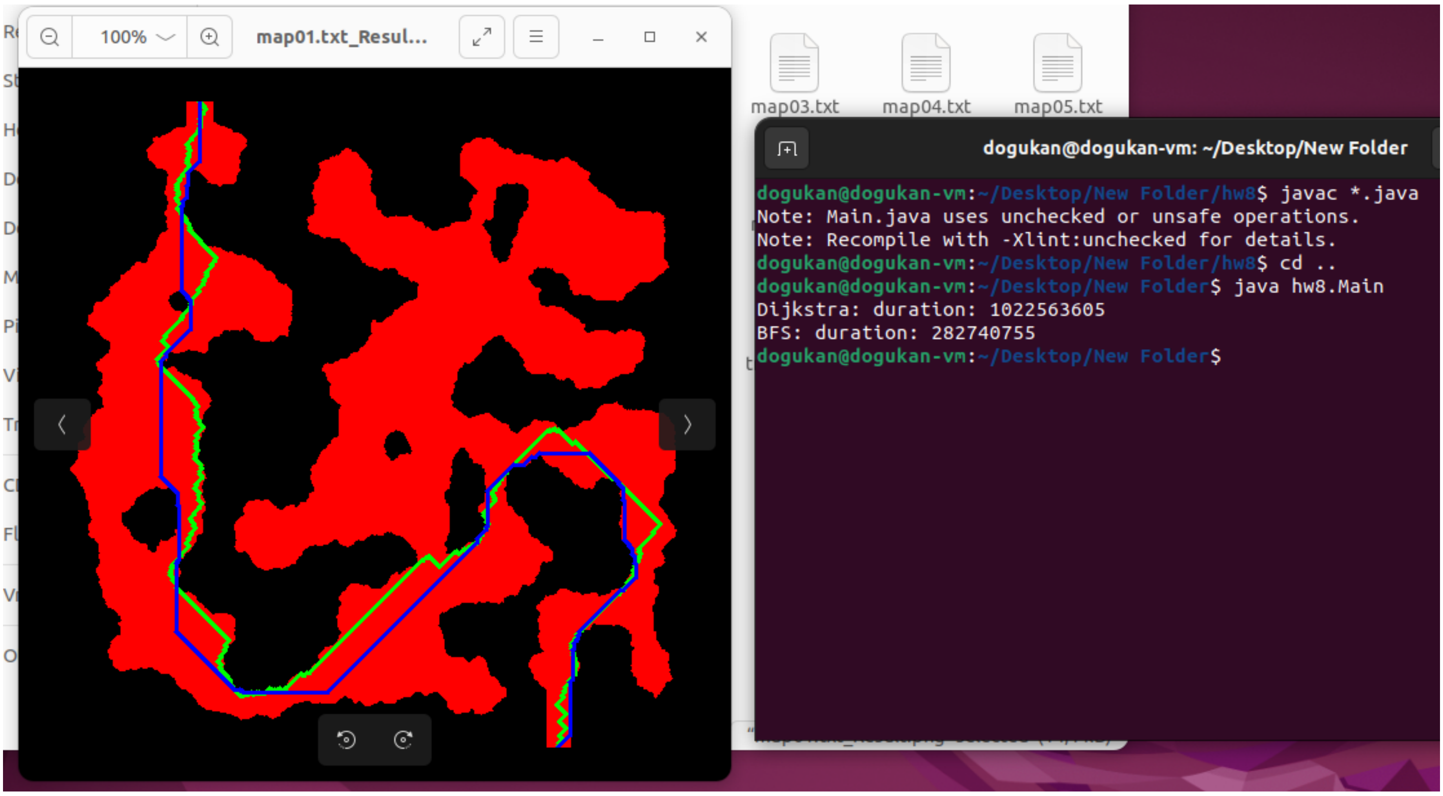
As it can be seen, it provides a faster result in accordance with the theoretical calculation.

Outputs:

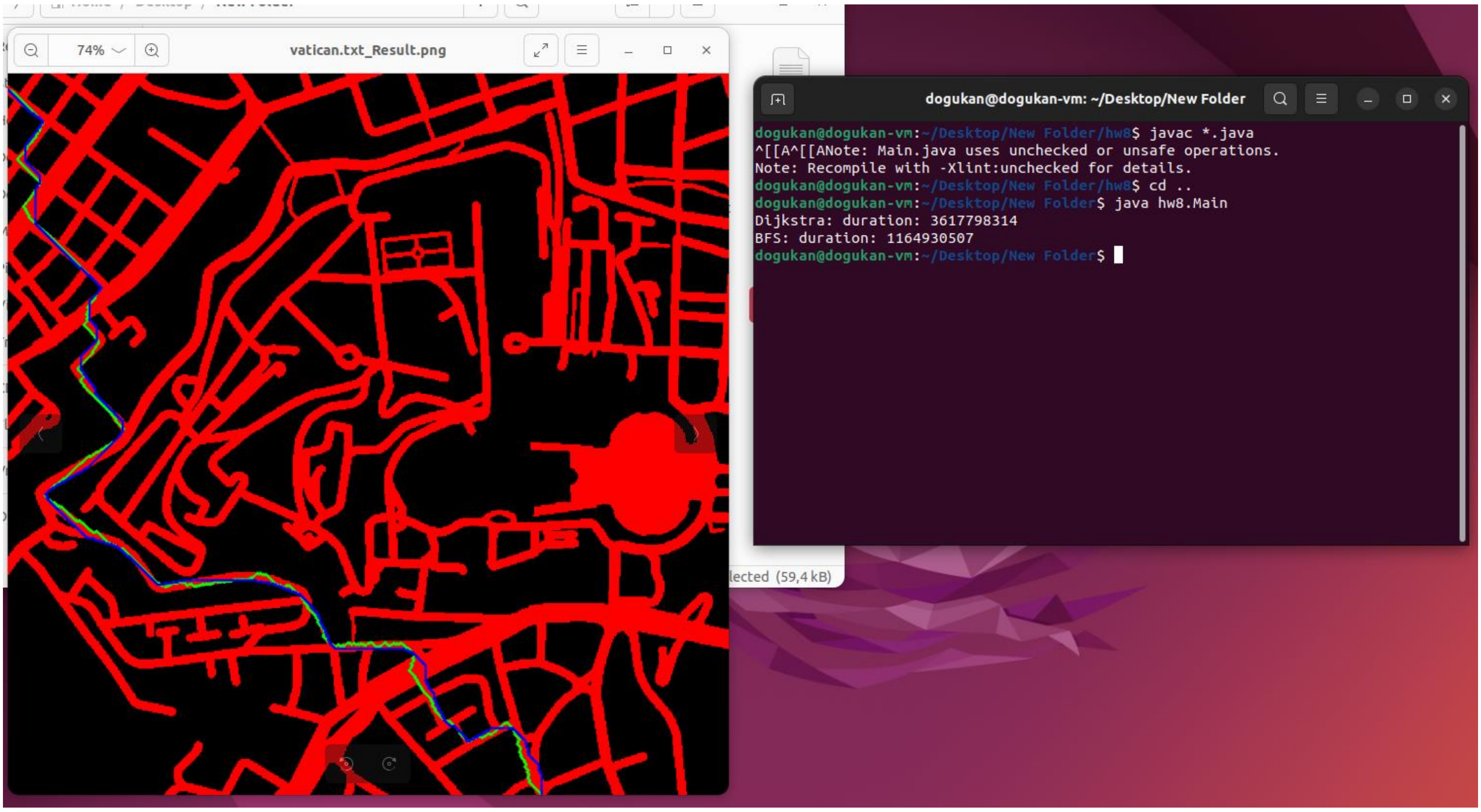
Green Dijkstra and Blue BFS



```
dogukan@dogukan-vm:~/Desktop/New Folder$ java hw8.Main
Dijkstra: duration: 1409158012
BFS: duration: 438764780
dogukan@dogukan-vm:~/Desktop/New Folder$
```



```
dogukan@dogukan-vm:~/Desktop/New Folder/hw8$ javac *.java
Note: Main.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
dogukan@dogukan-vm:~/Desktop/New Folder/hw8$ cd ..
dogukan@dogukan-vm:~/Desktop/New Folder$ java hw8.Main
Dijkstra: duration: 1022563605
BFS: duration: 282740755
dogukan@dogukan-vm:~/Desktop/New Folder$
```



```
dogukan@dogukan-vm:~/Desktop/New Folder/hw8$ javac *.java
^[[A^[[ANote: Main.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
dogukan@dogukan-vm:~/Desktop/New Folder/hw8$ cd ..
dogukan@dogukan-vm:~/Desktop/New Folder$ java hw8.Main
Dijkstra: duration: 3617798314
BFS: duration: 1164930507
dogukan@dogukan-vm:~/Desktop/New Folder$ 
```