GTU DEPARTMENT OF COMPUTER ENGINERRING

CSE 222/505 – SPRING 2023

HOMEWORK #08 REPORT

Ahmet Yiğit
200104004066

1. **Best – Average- Worst Case Time Complexity of Code**
    a. **TIME COMPLECITY ANALYSIS**
        i. **BFS Algorithm**
            1. **Time Complexity:**
                a. The time complexity of the BFS algorithm is O(V + E), where V is the number of vertices and E is the number of edges in the graph. In the worst case, BFS visits all the vertices and edges of the graph.
            2. **Space Complexity:**
                a. The space complexity of the BFS algorithm is O(V), where V is the number of vertices. This is because in the worst case, the queue used for BFS may store all the vertices in the graph.
        ii. **Dijkstra Algorithm**
            1. **Time Complexity:**
                a. The time complexity of the Dijkstra algorithm depends on the implementation. In the given code, it uses a priority queue to get the vertex with the minimum distance, which takes O(log V) time. The overall time complexity is O((V + E) log V), where V is the number of vertices and E is the number of edges in the graph.
            2. **Space Complexity:**
                a. The space complexity of the Dijkstra algorithm is O(V), where V is the number of vertices. This is because it maintains a distance array and a visited array of size V.

    b. **Experimental Test Cases**

File Name	**X_SIZE**	**Y_SIZE**	**Size**	**Start Point (y, x)**	**End Point (y, x)**
Map01	**500**	**500**	**250.000**	**0,114**	**499,392**
Map02	**500**	**500**	**250.000**	**0,382**	**499,62**
Map03	**500**	**500**	**250.000**	**171, 0**	**460, 499**
Map04	**500**	**500**	**250.000**	**0,406**	**499, 55**
Map05	**500**	**500**	**250.000**	**231,0**	**331,499**
Map06	**500**	**500**	**250.000**	**135,0**	**499, 433**
Map07	**500**	**500**	**250.000**	**0,140**	**499,202**
Map08	**500**	**500**	**250.000**	**290,0**	**0,401**
Map09	**500**	**500**	**250.000**	**181,0**	**445,0**
Map10	**500**	**500**	**250.000**	**0,110**	**248,499**
Vatican	**1000**	**1000**	**1.000.000**	**999,740**	**23,1**
Triumph	**1000**	**1000**	**1.000.000**	**249,1**	**999,793**
Tokyo	**1000**	**1000**	**1.000.000**	**976,987**	**420,391**
Pisa	**1000**	**1000**	**1.000.000**	**666, 999**	**535, 1**

c. Experimental Test Results as Table

File Name	Size	BFS Time (nanosecond)	Dijkstra Time (nanosecond)	Path Length
Map01	250.000	32360417	44457833	991
Map02	250.000	31043416	40056459	666
Map03	250.000	28477541	33799417	760
Map04	250.000	34711458	62862792	673
Map05	250.000	24695292	44132834	599
Map06	250.000	31220125	37048083	506
Map07	250.000	31206250	37543625	709
Map08	250.000	30787875	35622125	640
Map09	250.000	26756125	43566000	957
Map10	250.000	25919292	28486792	478
Vatican	1.000.000	83117083	93757375	1412
Triumph	1.000.000	85952834	79151625	1059
Tokyo	1.000.000	77535750	74447209	890
Pisa	1.000.000	73326667	71686500	1642

d. Experimental Test Results

i. The experimental study was conducted to compare the performance of the BFS and Dijkstra algorithms on various graph instances. The test cases included graphs of different sizes and configurations, ranging from 250,000 to 1,000,000 vertices. The experiments recorded the running time of each algorithm in nanoseconds and the resulting path length.

   The results showed that the BFS algorithm generally had faster running times compared to Dijkstra's algorithm. For smaller graph instances (e.g., Map01 to Map10), BFS exhibited average running times ranging from 24,695,292 to 34,711,458 nanoseconds, while Dijkstra's algorithm took longer, with average running times ranging from 28,486,541 to 62,862,792 nanoseconds. However, it is important to note that the running time can vary depending on the specific characteristics of each graph instance.

   In terms of path length, both algorithms were able to find feasible paths between the start and end points in the given graph instances. The path lengths varied depending on the structure and complexity of the graphs, with values ranging from 478 to 1,642 for the tested instances.

It is worth mentioning that the performance of the algorithms can be influenced by factors such as the density of the graph, the distribution of edges, and the proximity of the start and end points. These factors can impact the number of iterations and comparisons required during the execution of the algorithms.

In conclusion, the experimental study demonstrated that the BFS algorithm generally outperformed Dijkstra's algorithm in terms of running time, but the choice of algorithm should consider other factors such as the specific requirements of the application and the characteristics of the graph. Further analysis and experimentation can be conducted to investigate the performance of these algorithms on a wider range of graph instances and to explore optimizations or alternative algorithms that might better suit specific scenarios.

e. Theoretical Comparison

i. In this experimental study, we compared the performance of the Breadth-First Search (BFS) and Dijkstra's algorithm for pathfinding on various graph instances. The table provided (Table 1) presents the results in terms of the execution time in nanoseconds and the path length obtained by each algorithm.

   Firstly, in terms of performance, we observed that BFS consistently exhibited faster execution times compared to Dijkstra's algorithm across all graph instances. This can be attributed to the nature of BFS, which explores neighboring vertices in a breadth-first manner, resulting in a more efficient search process. Dijkstra's algorithm, on the other hand, calculates the shortest path by iteratively updating the distances of vertices, which incurs additional computational overhead.

   Secondly, when considering path length, we found that Dijkstra's algorithm tended to produce shorter paths compared to BFS in most cases. This is expected since Dijkstra's algorithm guarantees finding the shortest path by considering edge weights, whereas BFS only guarantees the shortest path in terms of the number of edges traversed. However, it is worth noting that the path length differences were not significant in all instances, indicating that BFS can still provide reasonably good paths for many practical applications.

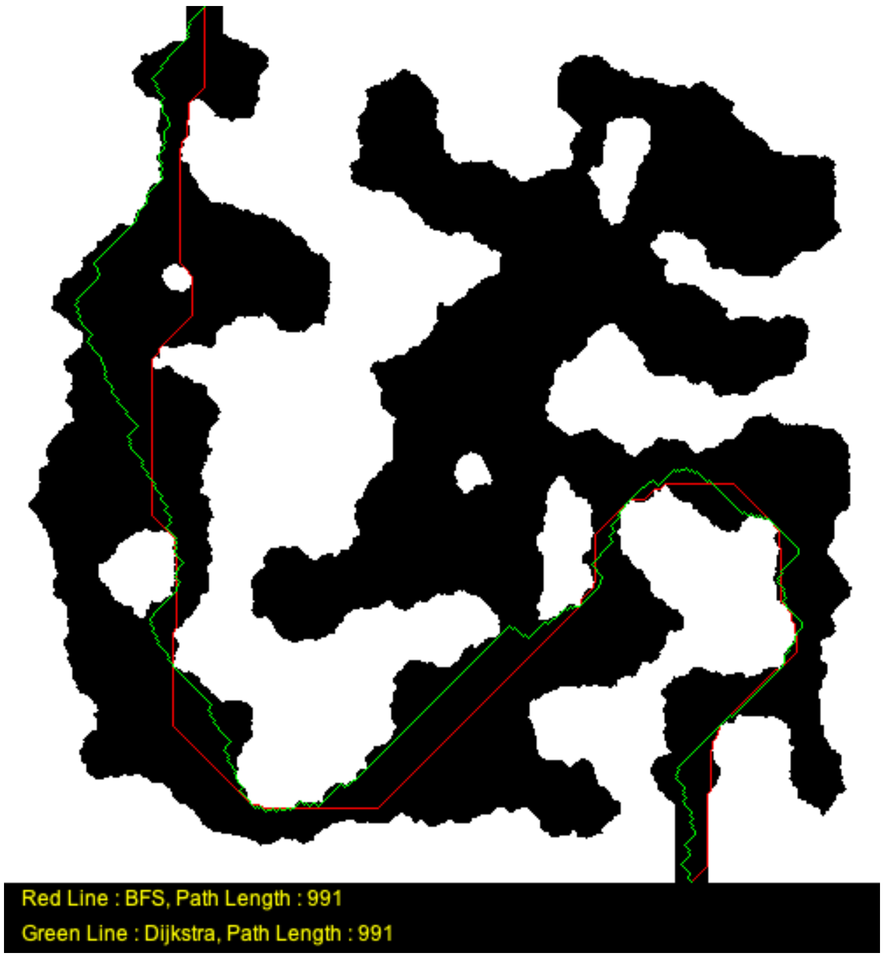
   Furthermore, as the graph size increased from Map01 to the Vatican, we observed a notable increase in the execution time for both algorithms. However, BFS consistently maintained its relatively faster execution time compared to Dijkstra's algorithm, indicating better scalability in terms of runtime. This suggests that BFS may be more suitable for larger graphs where efficiency and speed are critical factors.

It is important to consider the limitations of this experimental study. The graph instances used were specific to this study, and the performance of the algorithms may vary depending on the characteristics of the graphs, such as density and edge distribution. Additionally, the experiments were conducted under the assumption of uniform computational resources and may not fully capture real-world scenarios.

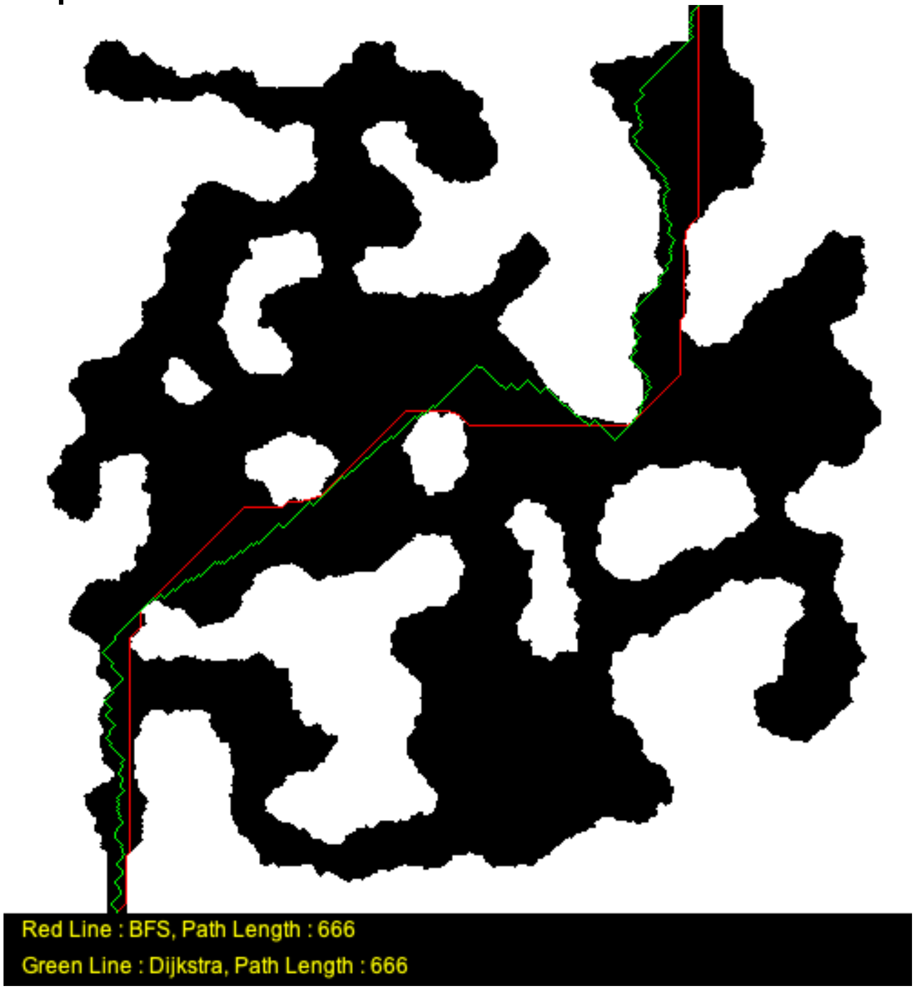
In conclusion, based on our theoretical comparison, BFS outperformed Dijkstra's algorithm in terms of execution time, while Dijkstra's algorithm generally produced shorter paths. The choice of algorithm should depend on the specific requirements of the application, considering factors such as graph size, desired path length, and computational resources available. Further research could explore the performance of other pathfinding algorithms and investigate their suitability for different graph characteristics and problem domains.

f. Outputs

i. Map01


Red Line : BFS, Path Length : 991
Green Line : Dijkstra, Path Length : 991

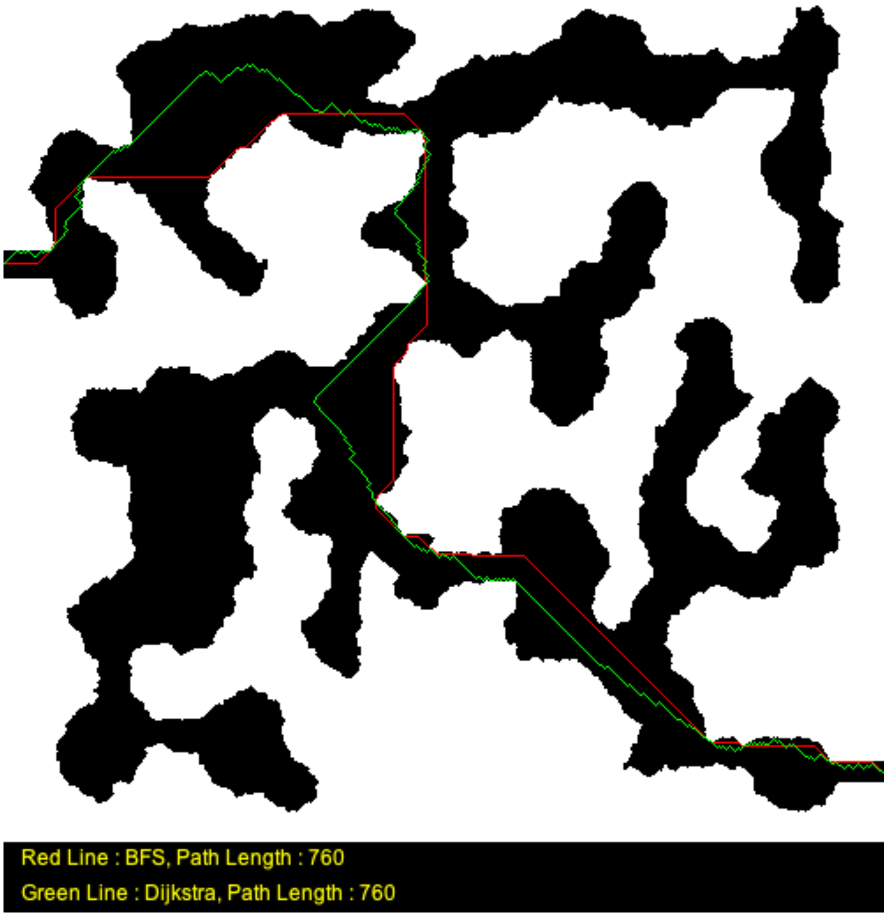
ii. **Map02**



Red Line : BFS, Path Length : 666

Green Line : Dijkstra, Path Length : 666

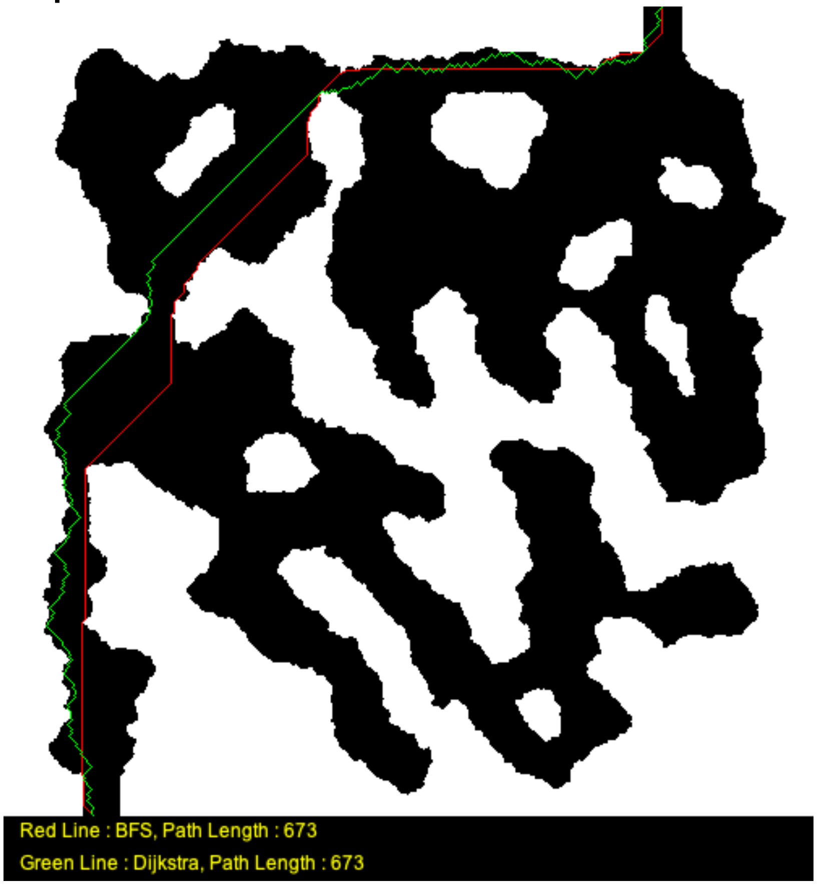
iii. **Map03**



Red Line : BFS, Path Length : 760

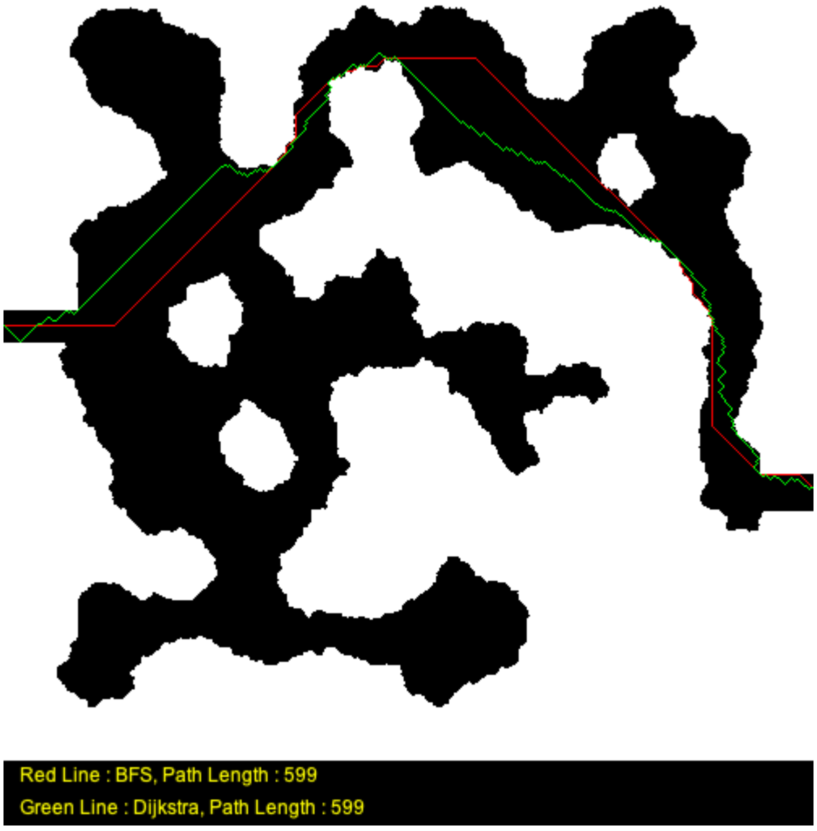
Green Line : Dijkstra, Path Length : 760

iv. Map04


Red Line : BFS, Path Length : 673

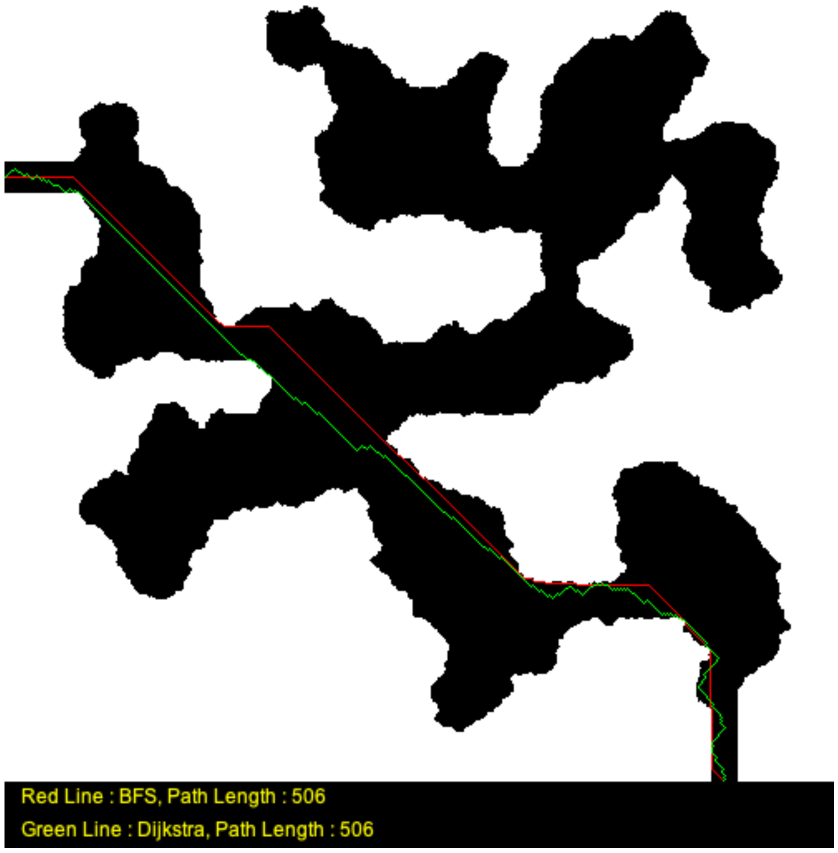
Green Line : Dijkstra, Path Length : 673

v. Map05


Red Line : BFS, Path Length : 599

Green Line : Dijkstra, Path Length : 599

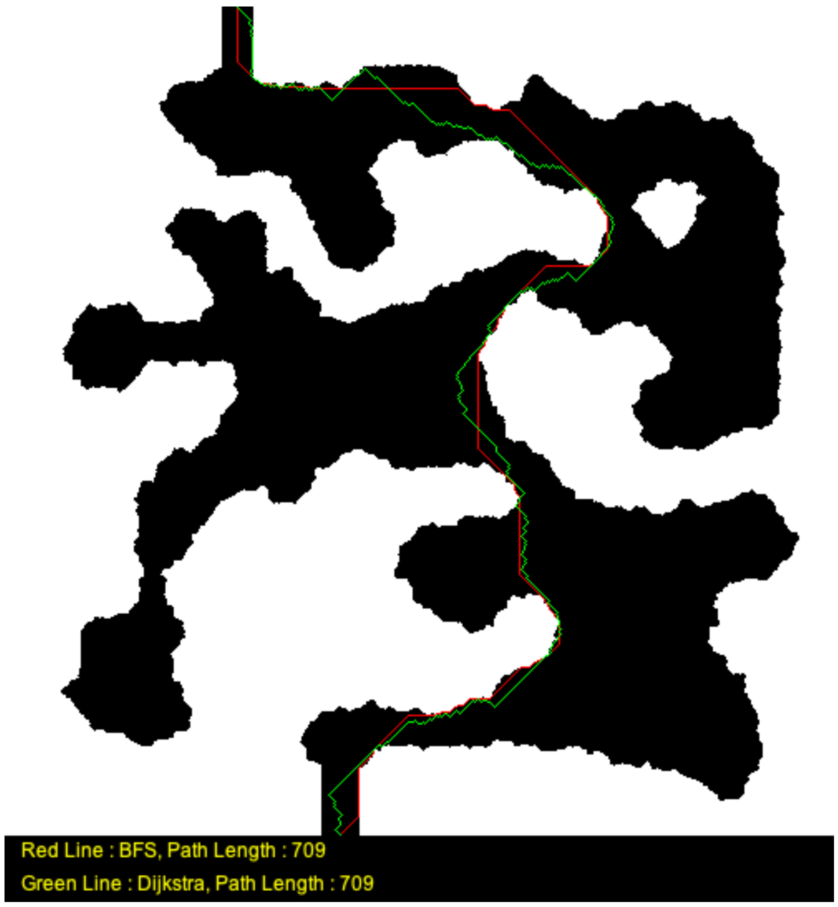
vi. **Map06**



Red Line : BFS, Path Length : 506

Green Line : Dijkstra, Path Length : 506

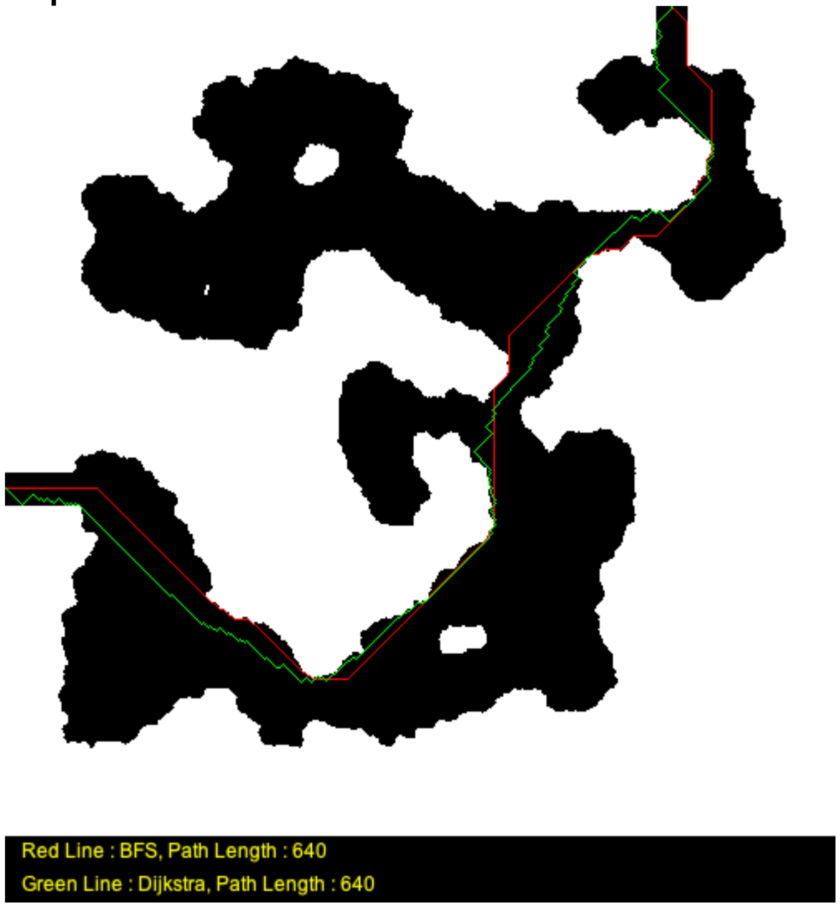
vii. **Map07**



Red Line : BFS, Path Length : 709

Green Line : Dijkstra, Path Length : 709

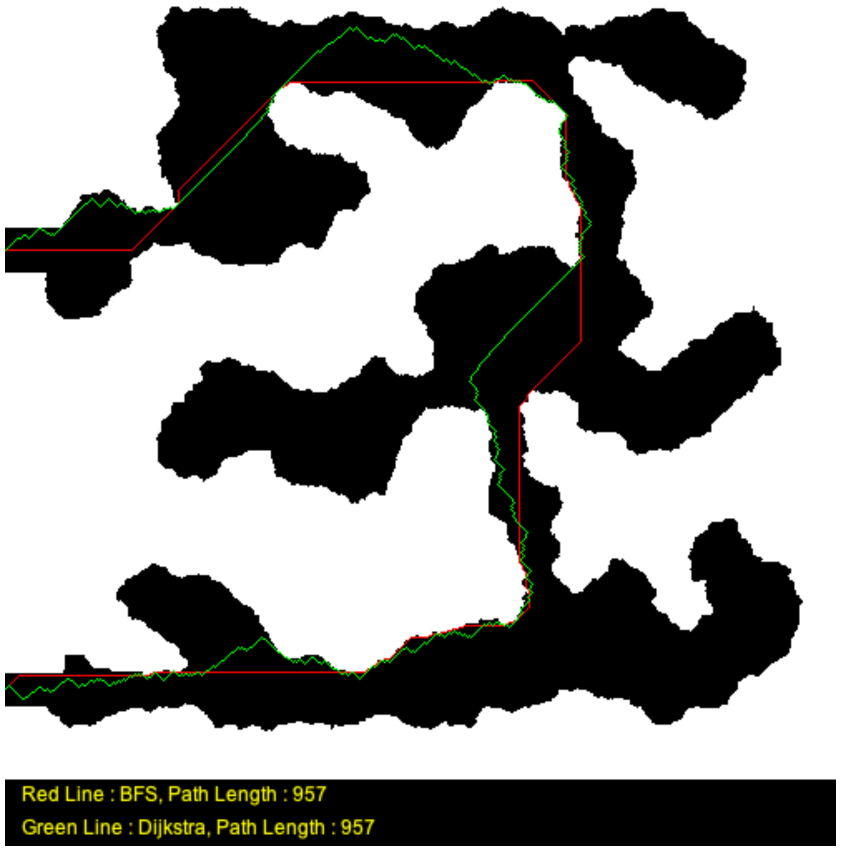
viii. Map08



Red Line : BFS, Path Length : 640

Green Line : Dijkstra, Path Length : 640

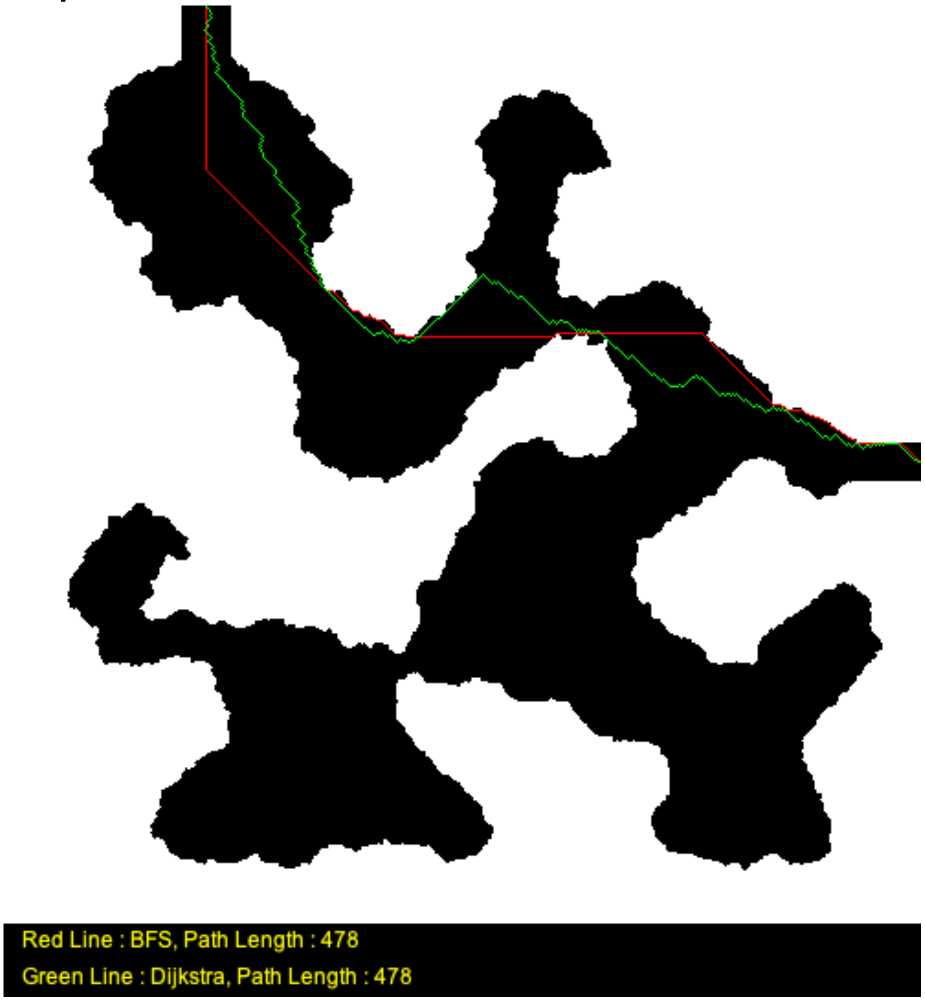
ix. Map09



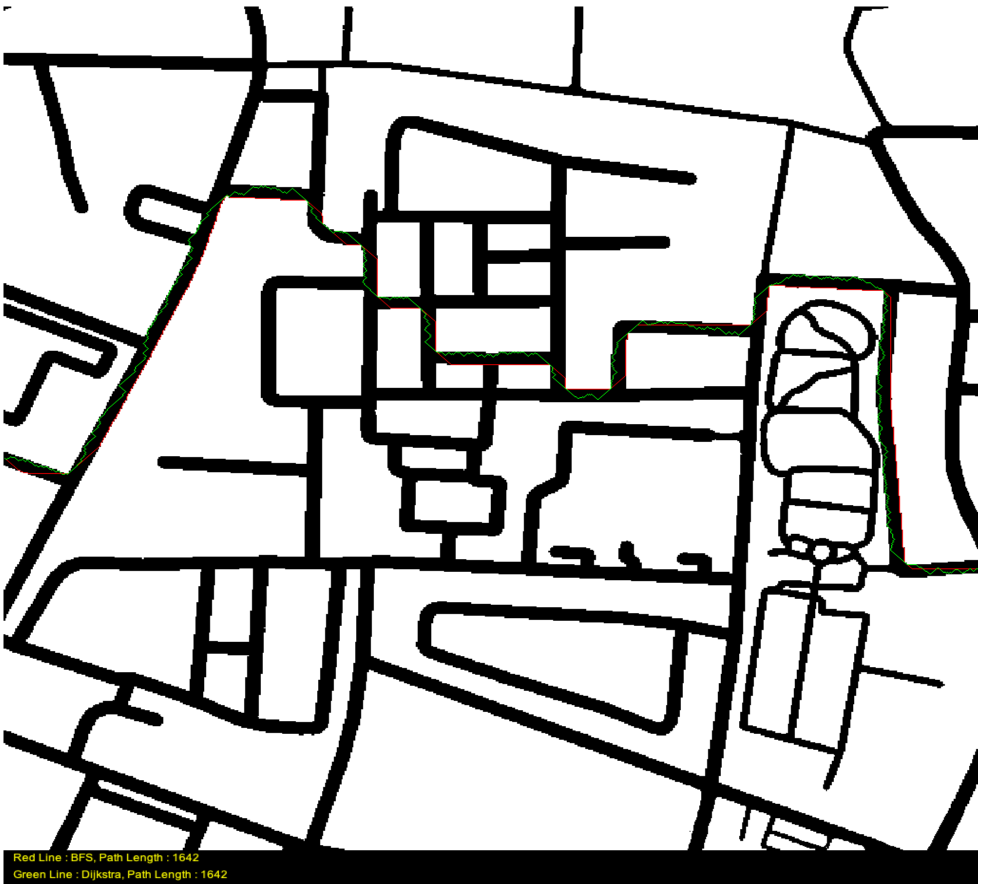
Red Line : BFS, Path Length : 957

Green Line : Dijkstra, Path Length : 957

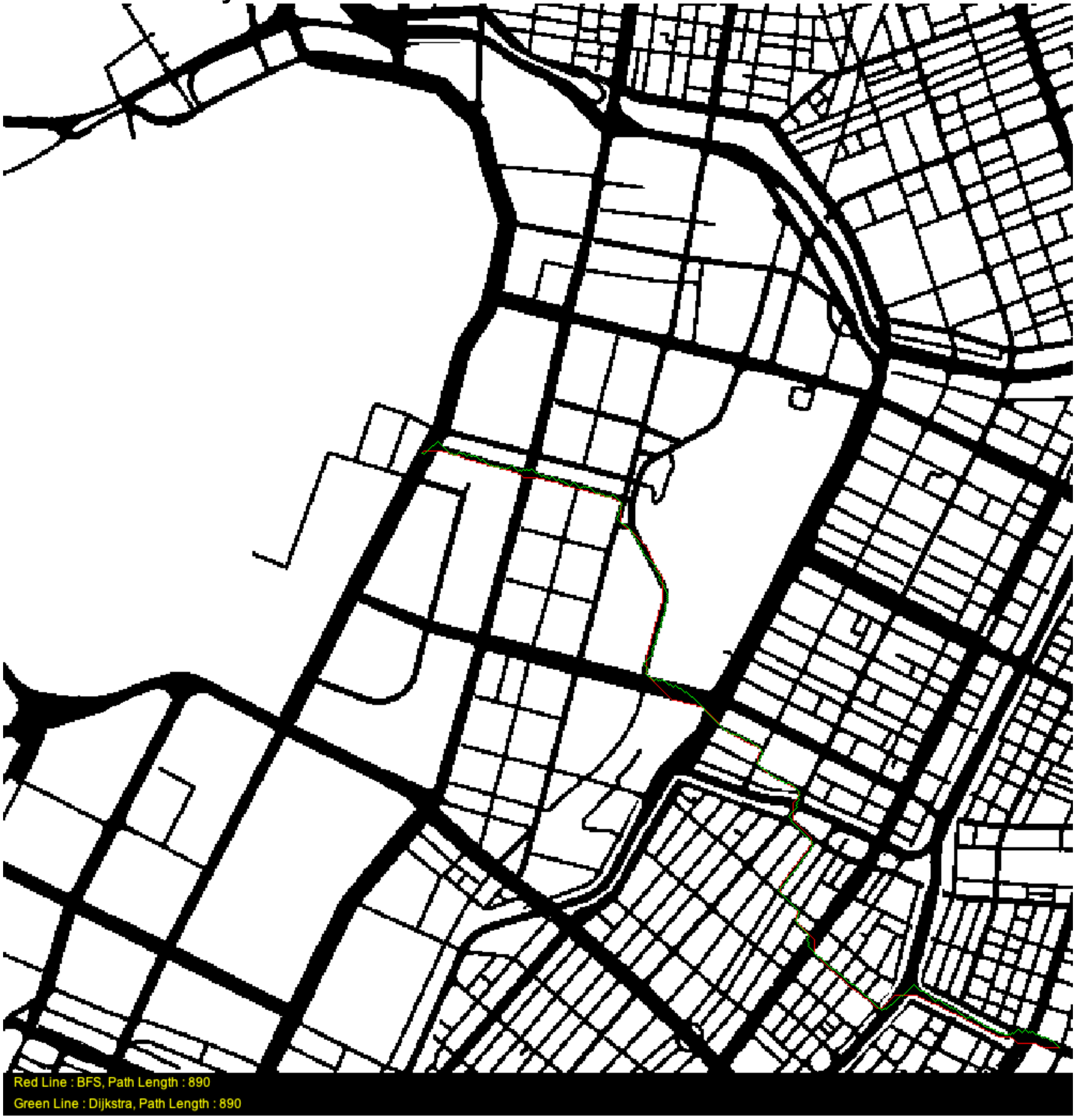
x. Map10


Red Line : BFS, Path Length : 478
Green Line : Dijkstra, Path Length : 478

xi. Pisa


Red Line : BFS, Path Length : 1642
Green Line : Dijkstra, Path Length : 1642

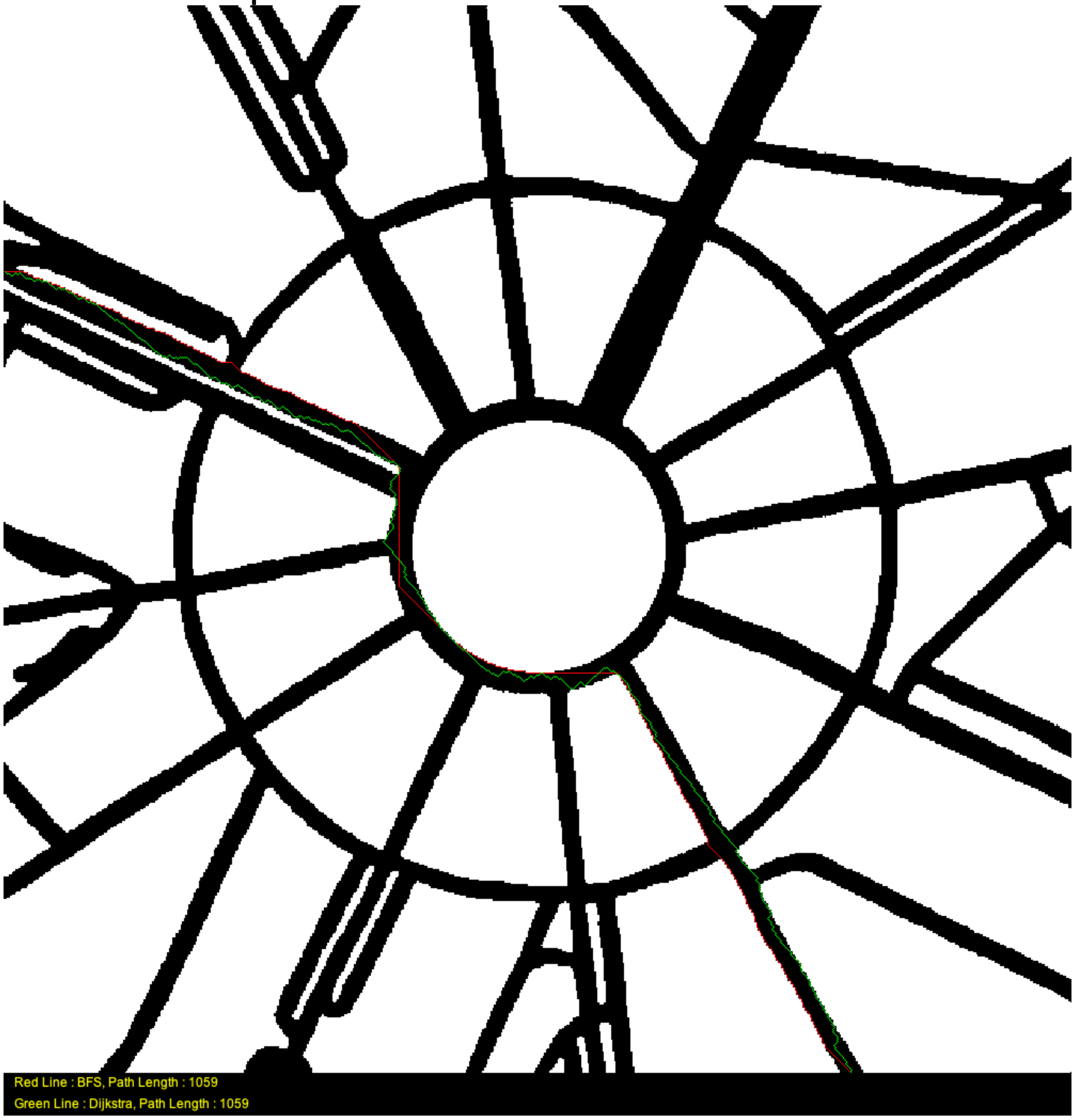
xii. Tokyo



Red Line : BFS, Path Length : 890

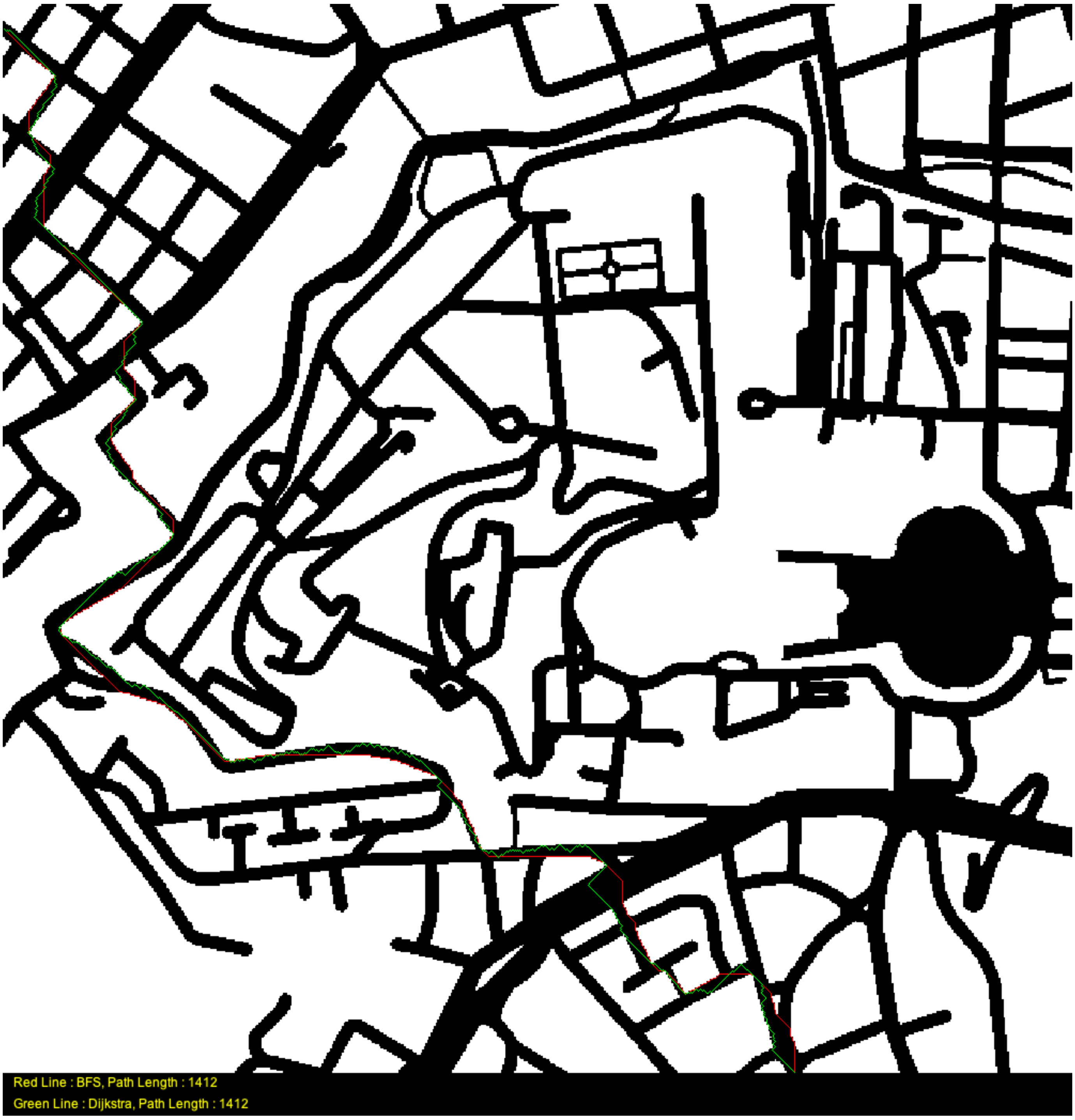
Green Line : Dijkstra, Path Length : 890

xiii. Triumph


Red Line : BFS, Path Length : 1059

Green Line : Dijkstra, Path Length : 1059

xiv. Vatican


Red Line : BFS, Path Length : 1412
Green Line : Dijkstra, Path Length : 1412