GTU Department Of Computer Engineering

CSE 222 / 505 – Spring 2023

Homework 8

Ahmet CEBECİ

1901042708

The provided code consists of several classes that work together to create and analyze a graph representing a map. The main goal is to find paths between nodes using different algorithms like BFS (Breadth-First Search) and Dijkstra. In this report, we will explain how the code works and provide instructions on how to run it.

Classes and Their Functions:

CSE222Map: This class handles reading the map image and associated data from a file. It also provides methods to convert the image format and draw paths on the map.

Node: The Node class represents a node in the graph. It holds the x and y coordinates of the node and provides methods for comparison and retrieval of coordinates.

CSE222Graph: The CSE222Graph class constructs the graph from the map data by analyzing the matrix of values. It creates a map of nodes and their neighbors based on the connectivity defined in the matrix.

CSE222BFS: This class implements the Breadth-First Search algorithm to find the shortest path between two nodes in the graph. It utilizes the adjacency list from the CSE222Graph class and performs a breadth-first traversal to find the path.

CSE222Dijkstra: The CSE222Dijkstra class implements Dijkstra's algorithm to find the shortest path between two nodes in the graph. It uses the adjacency list from the CSE222Graph class and calculates the distances from the start node to all other nodes.

The process starts by creating an instance of the CSE222Map class and providing the input files (image and text file) containing the map data. The CSE222Map class reads and processes this data, creating a matrix representation of the map and identifying the start and end nodes.

Next, an instance of the CSE222Graph class is created, passing the CSE222Map object. The CSE222Graph class constructs the graph by analyzing the matrix and creating a map of nodes and their neighbors.

To find the path between the start and end nodes, you can choose between the CSE222BFS or CSE222Dijkstra class. Both classes take an instance of CSE222Graph as a parameter in their constructors. The CSE222BFS class uses the Breadth-First Search algorithm, while the CSE222Dijkstra class utilizes Dijkstra's algorithm.

In this report, we have discussed the classes and their functions, explained the code flow, and provided instructions on how to run the code. By following these steps, you can utilize the provided classes to construct a graph from a map, find paths using BFS or Dijkstra's algorithm, and visualize the results on the map image.

Time Complexity and Performance Analysis

Breadth-First Search (BFS):

Time Complexity: O(V + E), where V is the number of nodes (vertices) in the graph and E is the number of edges. In the worst case, BFS visits all nodes and edges in the graph.

Space Complexity: O(V), as BFS requires space to store the visited nodes and the queue for traversal. In the worst case, all nodes can be in the queue.

Dijkstra's Algorithm:

Time Complexity: O((V + E) * log V), where V is the number of nodes (vertices) in the graph and E is the number of edges. The time complexity is dominated by the priority queue operations.

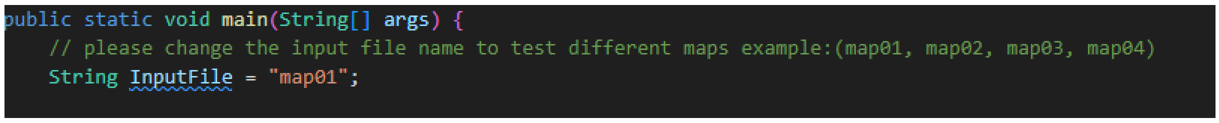
Space Complexity: O(V), as Dijkstra's algorithm requires space to store the distances and previous nodes for each node in the graph.

The performance of the BFS and Dijkstra algorithms can vary depending on the size of the graph and the complexity of the paths. BFS guarantees finding the shortest path in an unweighted graph, while Dijkstra's algorithm handles weighted graphs.

To improve the performance, you can consider optimizing the code or using more advanced algorithms like A* search if you have additional information about the problem domain.

NOTE:

If the InputFile name is not paid attention to in the main section, the code gives an error. Please enter the file name when typing don't write the .txt part

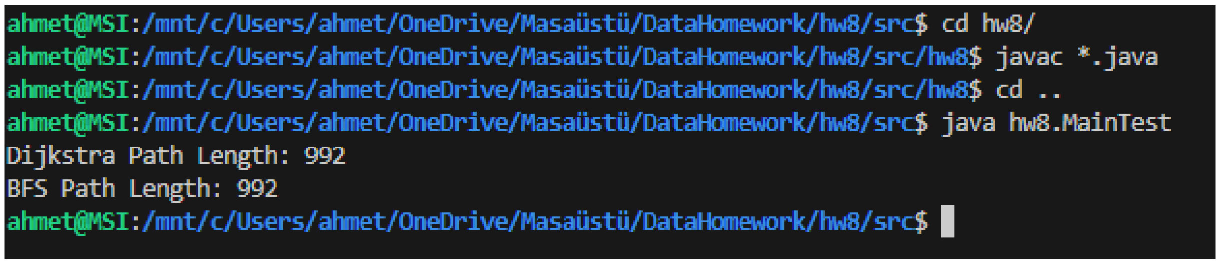


```
public static void main(String[] args) {
    // please change the input file name to test different maps example:(map01, map02, map03, map04)
    String InputFile = "map01";
```

If you want to test it with other files, if you have the file, just change the name of the file from the main part.

OUTPUT

For map01:



```
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/DataHomework/hw8/src$ cd hw8/
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/DataHomework/hw8/src/hw8$ javac *.java
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/DataHomework/hw8/src/hw8$ cd ..
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/DataHomework/hw8/src$ java hw8.MainTest
Dijkstra Path Length: 992
BFS Path Length: 992
ahmet@MSI:/mnt/c/Users/ahmet/OneDrive/Masaüstü/DataHomework/hw8/src$
```