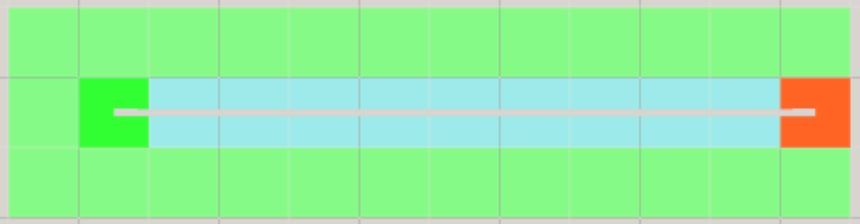
TESTING OF

Algorithms

In a Path-Finding Grid,
with, and without Hurdles

BY, MUHAMMAD AMMAR (M63)

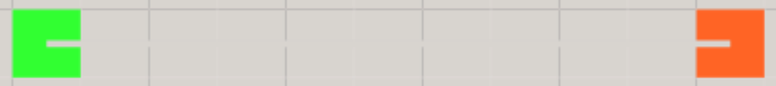


A*

Hurdles: No

Complexities

Length: 10 Units
Time: 4.30ms
Space: O(b^d)
Operations: 47

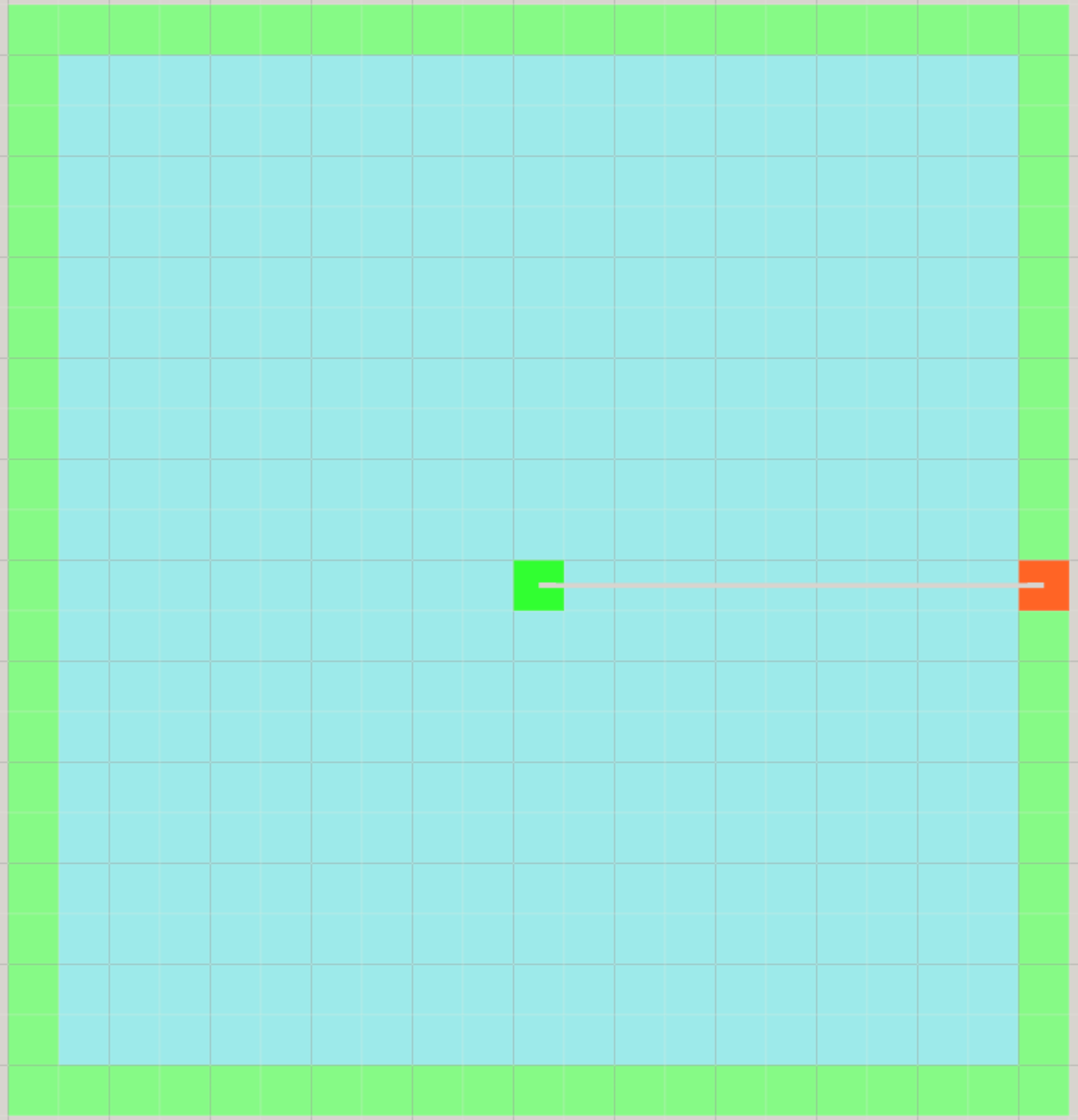


IDA*

Hurdles: No

Complexities

Length: 10 Units
Time: 2.20ms
Space: O(b^d)
Operations: 30

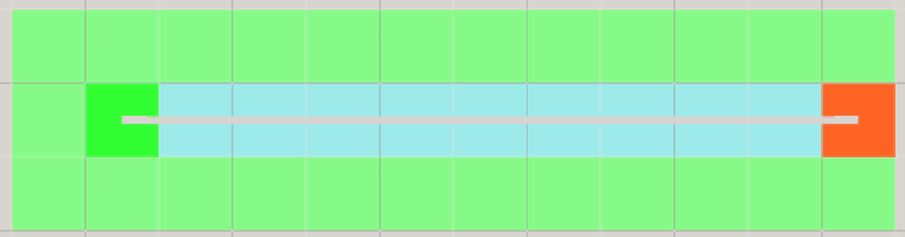


Breath First

Hurdles: No

Complexities

Length: 10 Units
Time: 3.70ms
Space: O(b^d)
Operations: 843



Best First

Hurdles: No

Complexities

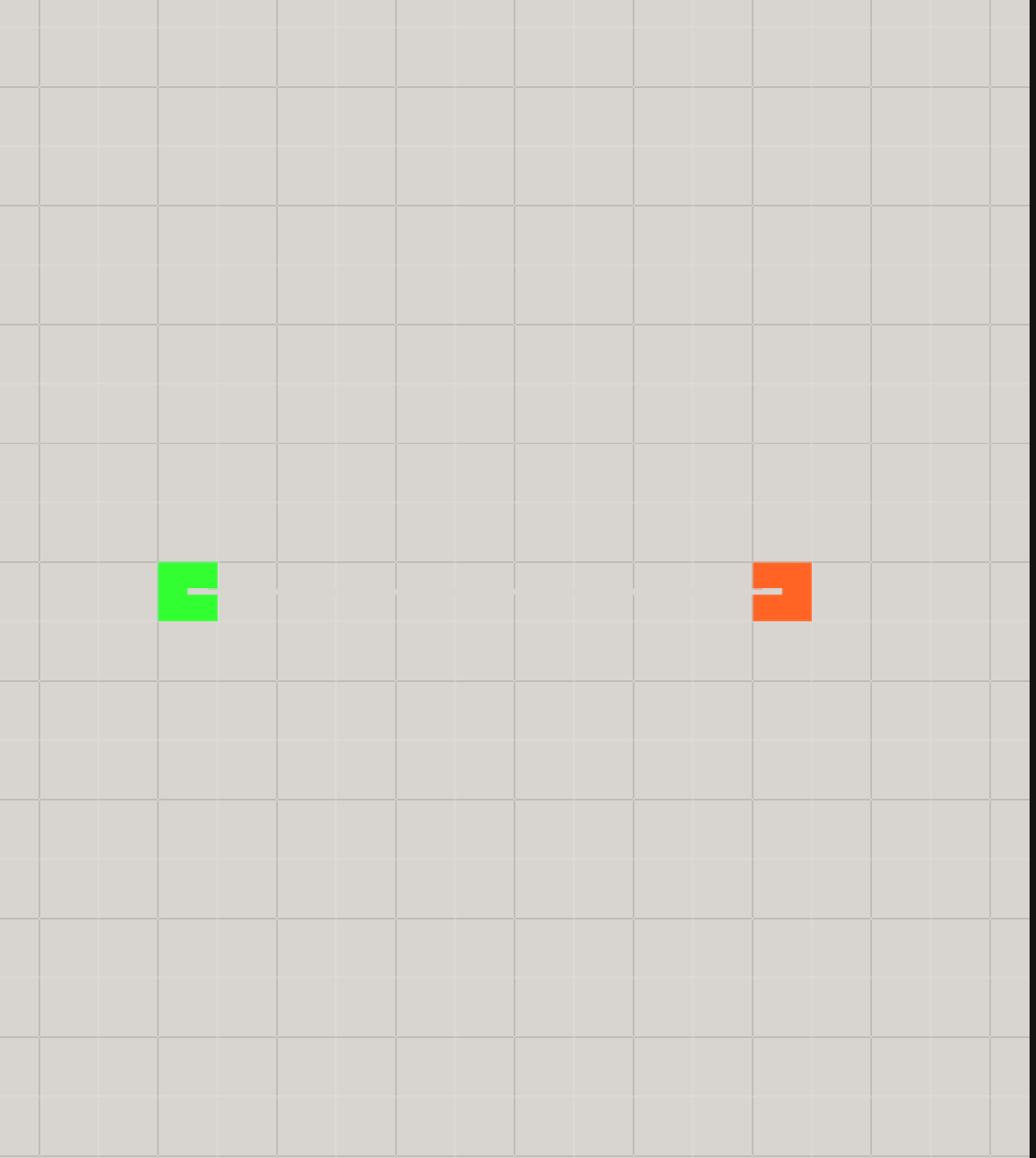
Length: 10 Units
Time: 1.50ms
Space: O(b^d)
Operations: 47

Dijkstra

Hurdles: No

Complexities

Length: 10 Units
Time: 6.80ms
Space: O(Vertices)
Operations: 643

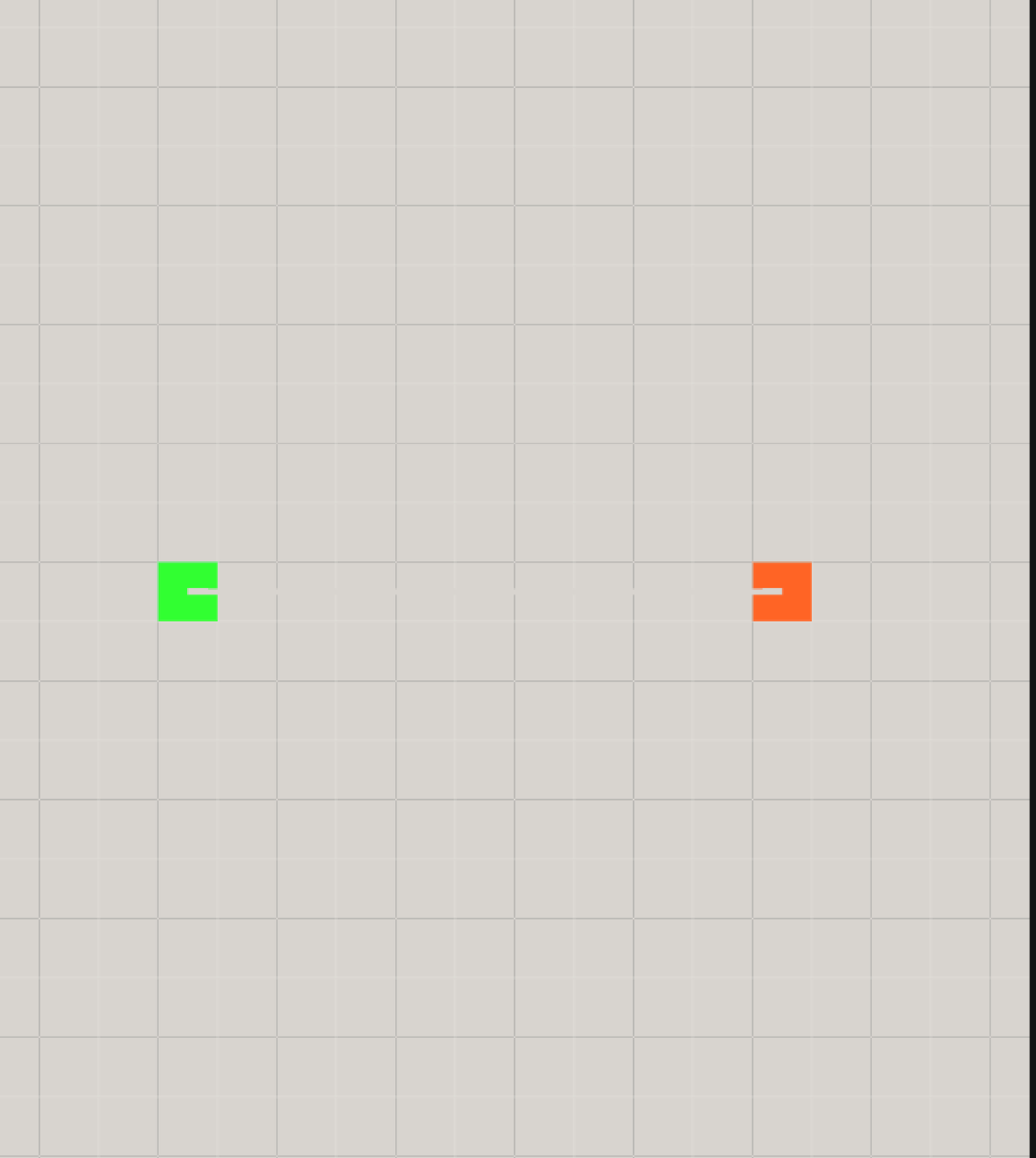


Jump Point

Hurdles: No

Complexities

Length: 10 Units
Time: 3.30ms
Space: O(1)
Operations: 2271

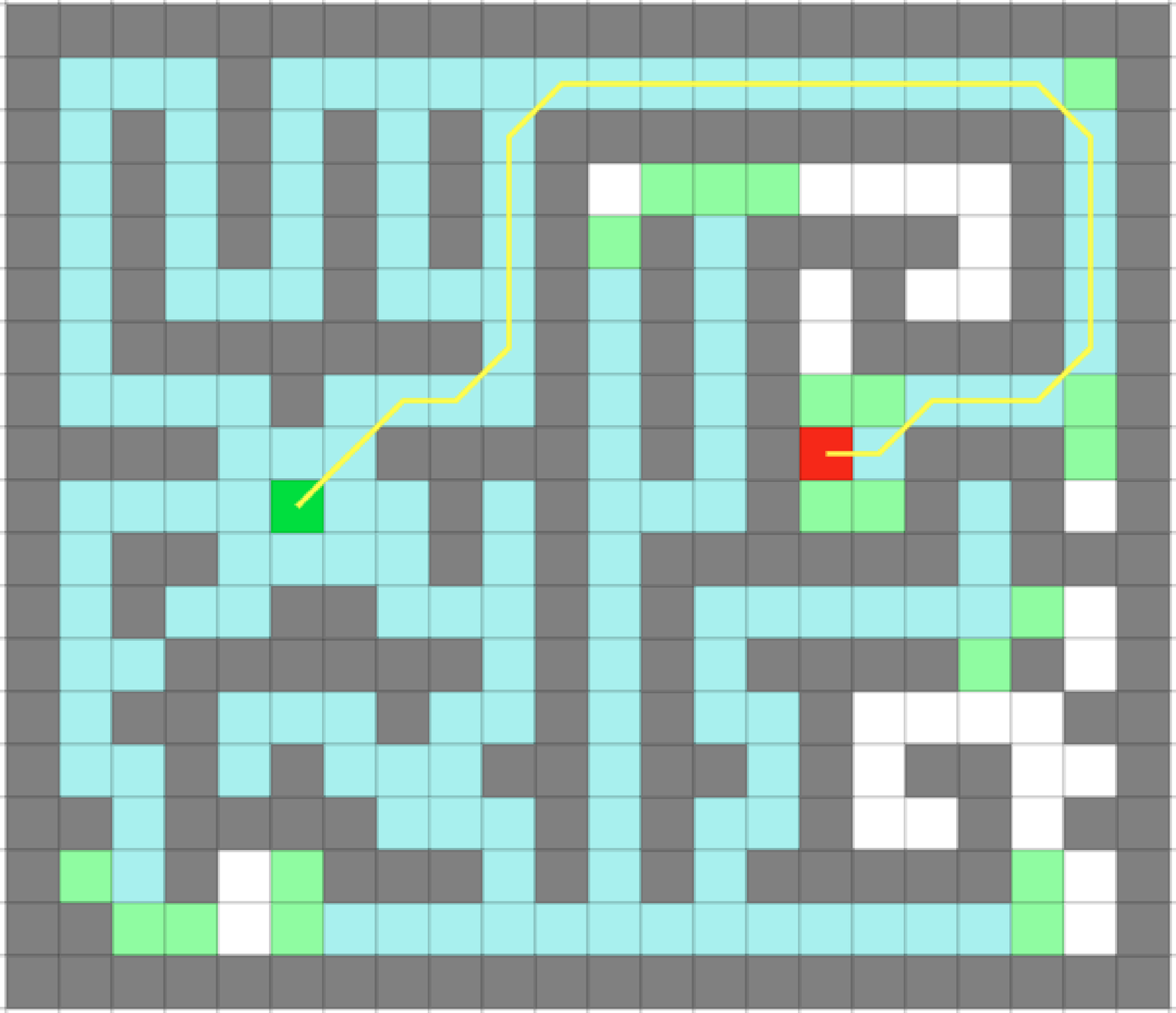


Orthogonal Jump Point

Hurdles: No

Complexities

Length: 10 Units
Time: 3.20ms
Space: O(1)
Operations: 2271

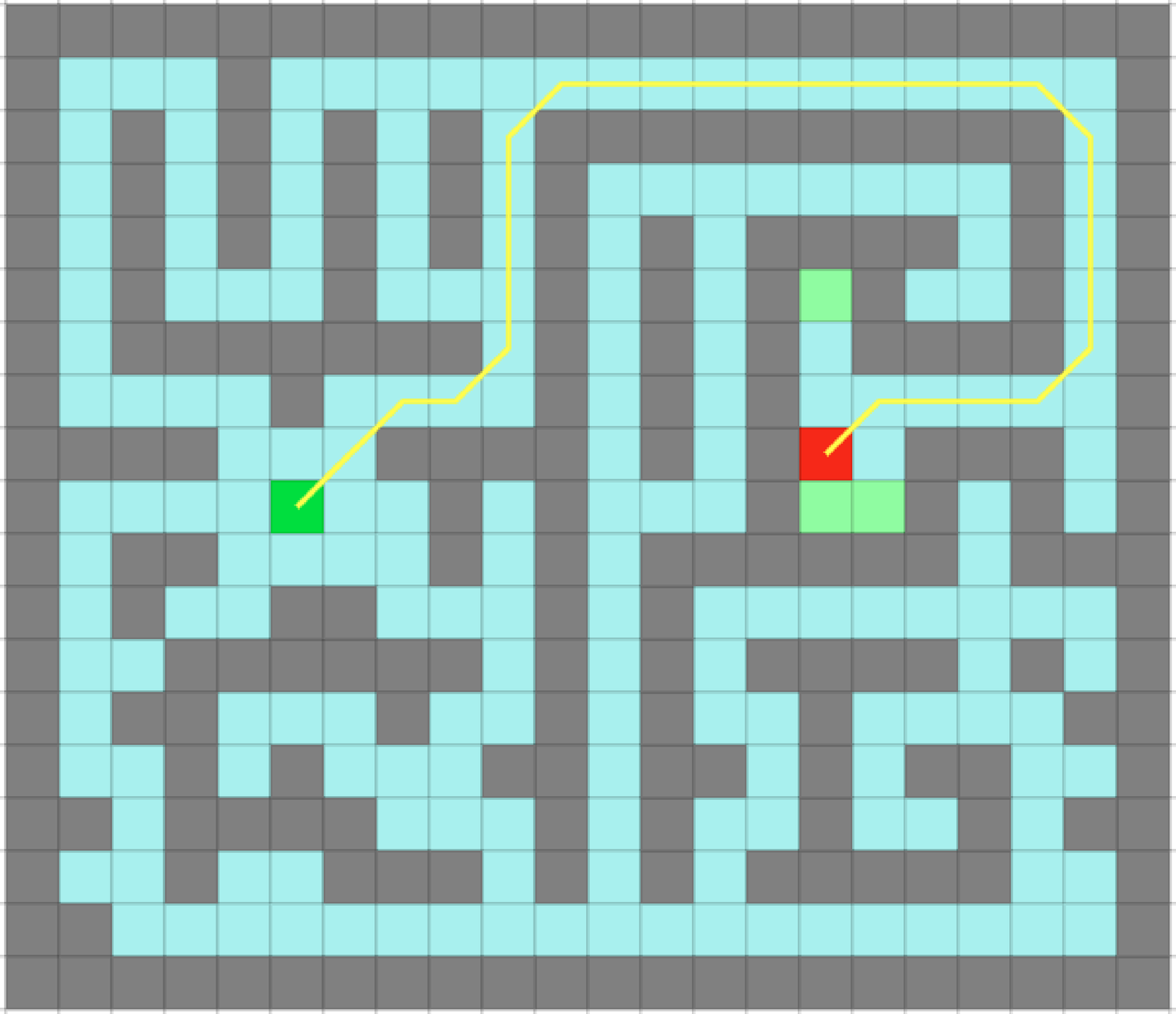


A*

Hurdles: Yes (Maze)

Complexities

Length: 30.9 Units
Time: 0.70ms
Space: O(b^d)
Operations: 322

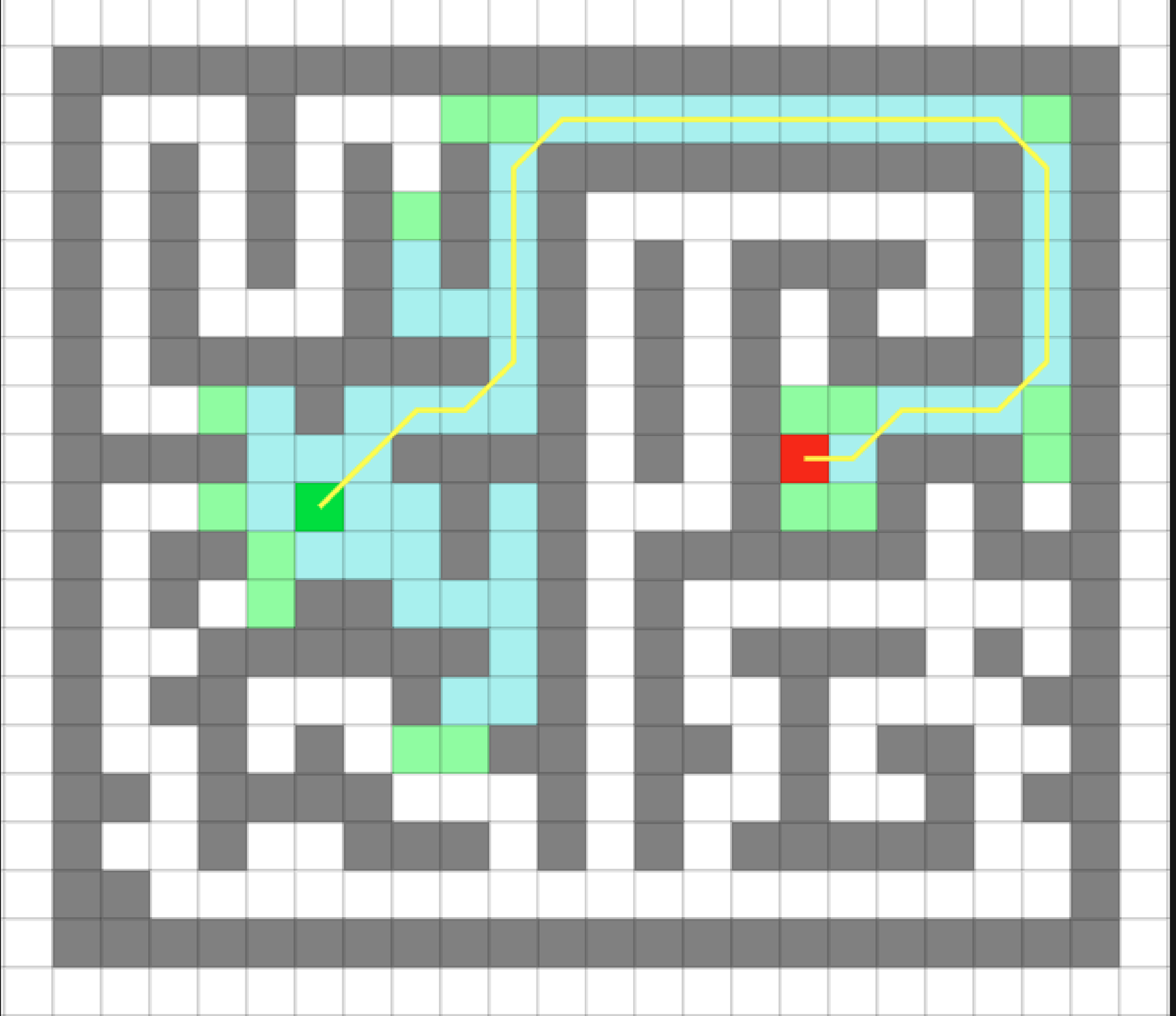


Breath First

Hurdles: Yes (Maze)

Complexities

Length: 30.9 Units
Time: 0.50ms
Space: O(b^d)
Operations: 393

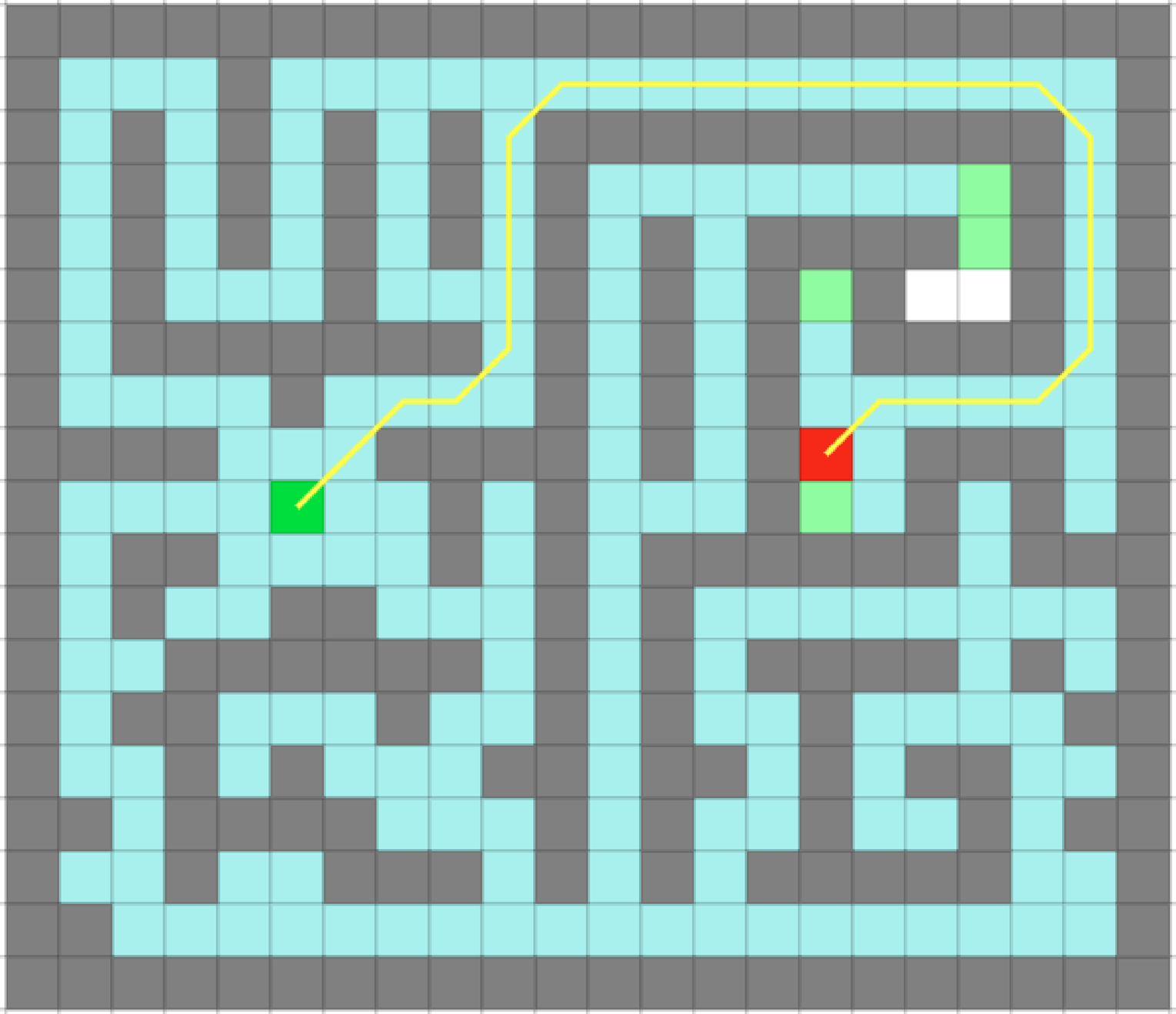


Best First

Hurdles: Yes (Maze)

Complexities

Length: 30.9 Units
Time: 1.90ms
Space: O(b^d)
Operations: 118

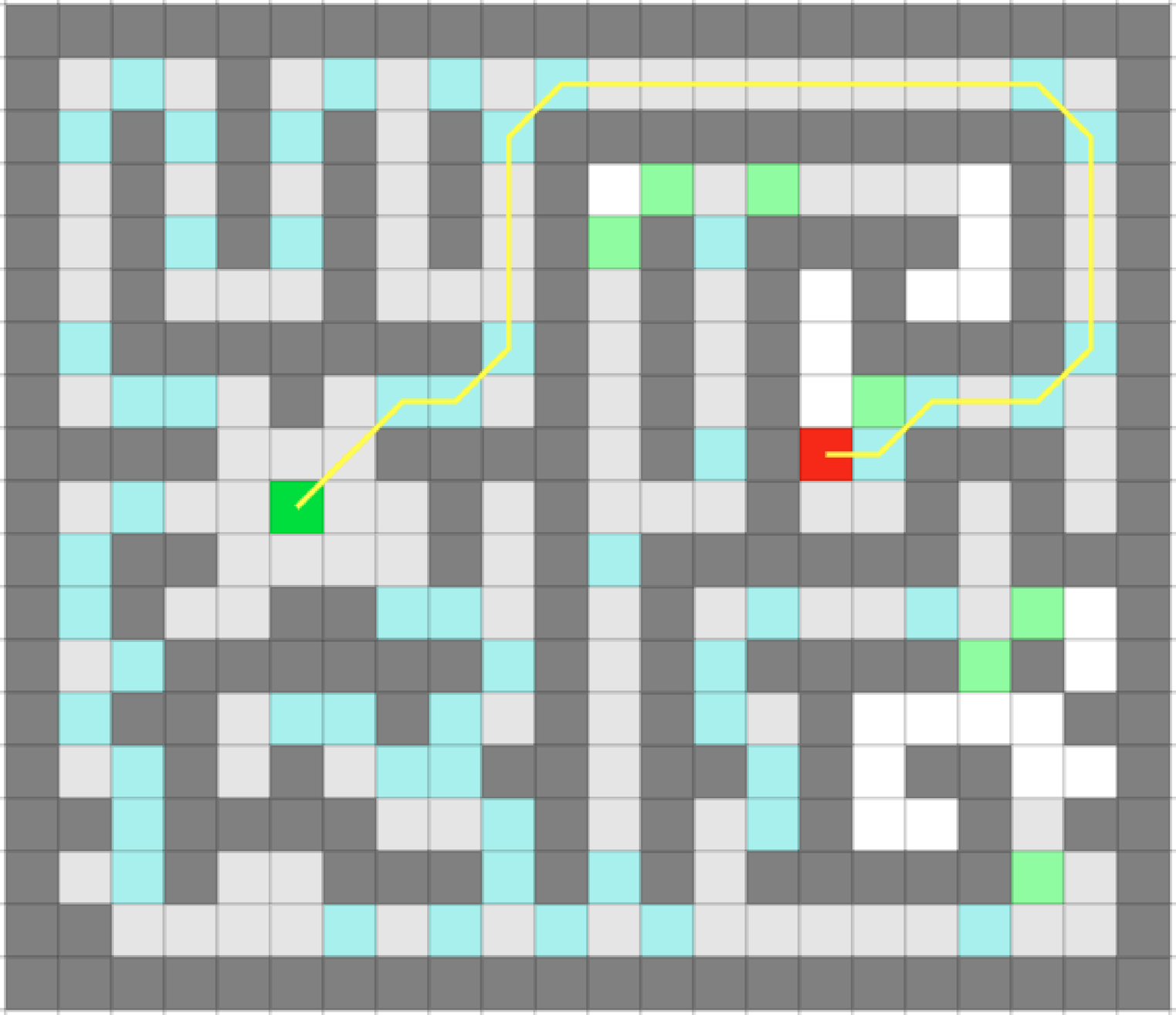


Dijkstra

Hurdles: Yes (Maze)

Complexities

Length: 30.9 Units
Time: 1.90ms
Space: O(Vertices)
Operations: 388

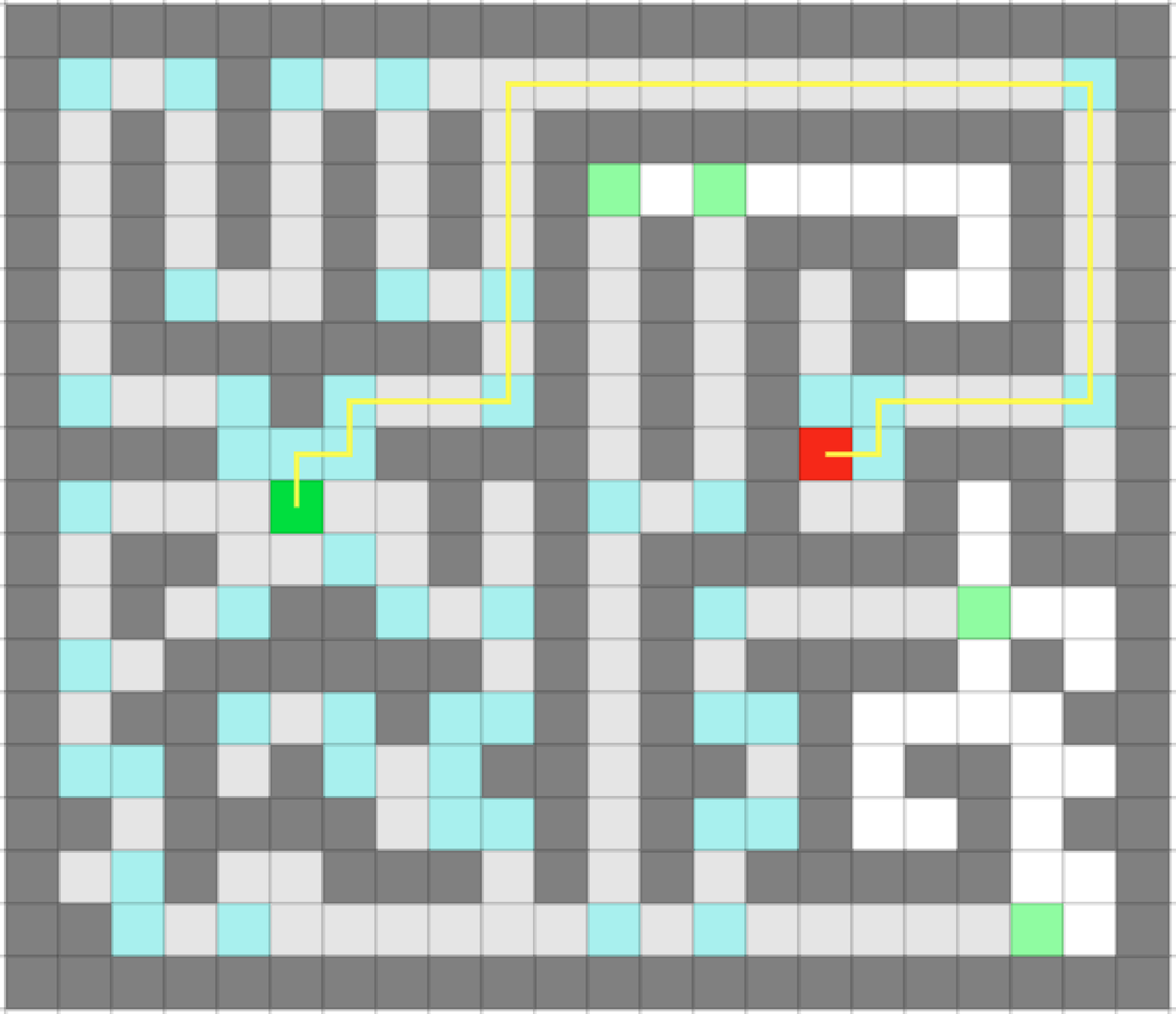


Jump Point

Hurdles: Yes (Maze)

Complexities

Length: 30.9 Units
Time: 3.40ms
Space: O(1)
Operations: 369



Orthogonal Jump Point

Hurdles: Yes (Maze)

Complexities

Length: 30.9 Units
Time: 0.40ms
Space: O(1)
Operations: 333

Performance

Without Hurdles

The “Best-First Search” has been the most efficient algorithm, in-terms of Performance. The Statistics are:

Length of the Path: 10 Units
Time taken: 1.50ms
Computations: 47

With Hurdles

The “Orthogonal Jump Point” has been the most efficient algorithm, in-terms of Performance. The Statistics are:

Length of the Path: 30.9 Units
Time taken: 0.40ms
Computations: 333

Computations

Without Hurdles

The “Iterative Deepening A*” has been the most efficient algorithm, in-terms of Computations. The Statistics are:

Length of the Path: 10 Units
Time taken: 2.20ms
Computations: 730

With Hurdles

The “Best-First Search” has been the most efficient algorithm, in-terms of Computations. The Statistics are:

Length of the Path: 30.9 Units
Time taken: 1.90ms
Computations: 118

Optimal Result

Without Hurdles

The “Best-First Search” has been the most efficient algorithm, in-terms of Computations, and Performance. The Statistics are:

Length of the Path: 10 Units
Time taken: 1.50ms
Computations: 47

With Hurdles

The “Orthogonal Jump Point” has been the most efficient algorithm, in-terms of Computations, and Performance. The Statistics are:

Length of the Path: 30.9 Units
Time taken: 0.40ms
Computations: 333

Written Questions

Question 02

Distance Measuring Algorithms

Question 03

Dijkstra's Algorithm

29th May 22

Artificial Intelligence // Assignment

The distance measures play an important role in machine learning. Most commonly used machine learning algorithms are ;

L, Hamming Distance

The Hamming distance is the number of bits, that differ between two binary strings. It's named after Richard Hamming, who introduced the concept in his Research Paper.

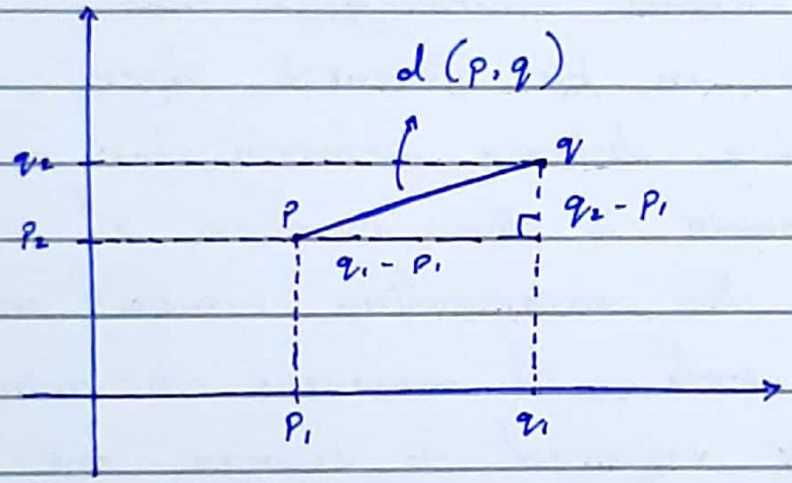
The Hamming distance between two strings of equal length is the number of positions at which, the corresponding symbols are different. In other words, it measures the minimum number of substitutions required to change one string into another. For example, some Hamming distances are ;

- "Karolin" and "Katherin" is 3
- "1011101" and "1001001" is 2
- "2173896" and "2233796" is 3

↳ Euclidean Distance

The Euclean Distance between two points in Euclidean Space is the length of a Line Segment between two points.

It can be calculated from cartesian coordinates of the points using Pythagoras Theorem.



$$d(p,q)^2 = (q_1 - p_1)^2 + (q_2 - p_2)^2 \quad // \text{ 2D}$$

or, $$d(p,q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_1)^2}$$

For 1-Dimensional space,

$$d(p,q) = |p - q|$$

↳ Manhattan Distance

In Manhattan Distance, the distance between two points is the sum of the absolute differences of their Cartesian Coordinates.

Manhattan Distance is considered better than Euclidian, if the distance for two vectors is in Integer Feature space.

For Example, in a plane, with $p(x_1, y_1)$ and $q(x_2, y_2)$ is : $|x_1 - x_2| + |y_1 - y_2|$

Example : the Manhattan Distance between between points (1,1) and (2,2) is ;

$= |1-2| + |1-2|$

$= |-1| + |-1|$

$= 1 + 1$

$= 2$ units

↳ Minkowski Distance

It is the generalization of Euclidean, and Manhattan Distance. It measures and adds a parameter, called order (p). This allows different distance measures to be calculated.

Minkowski distance is a metric used to calculate the distance between two points in a space, with more than 3-Dimensions.

For Example, the Minkowski distance between x and y is;

$$d(x,y) = \|x-y\| = \left(|x_1-y_1|^p + |x_2-y_2|^p + \dots + |x_n-y_n|^p\right)^{1/p},$$

where, $\|x-y\|$ is the Euclidean distance between x and y.

In general, it's used to calculate the distance between 2-points in a space, more than 3-Dimensions.

The Minkowski Distance is named after Hermann Minkowski, who first defined it in 1907 Research Paper: ("On the Geometry of Numbers")

Q. Explain Dijkstra's Algorithm, with properties.

↳ Definition

Dijkstra's Algorithm allows us to find the shortest path between any two vertices of a graph. It differs from the minimum spanning tree, as two vertices might not include all the vertices of the graph.

↳ Working

Dijkstra's Algorithm works on the basis ~~of~~ that any sub-path { B → D } fo the shortest path { A → D } between A and D is also the shortest path between B and D.

The Algorithm uses Greedy Approach in the sense that we find the next best solution, hoping that the end-result is the best result.

Pseudo-Code for Dijkstra Algorithm

Here is the Pseudo-Code, in C++ format :-

```
auto dijkstra(auto G, auto S)
{
    for (auto V : G)
        distance(V) = numerics<> :: infinity ;
        previous(V) = nullptr ;
        if (V != S) Q.add(V) ;
    distance(S) = 0 ;
    while (!Q.is_empty())
        U = Q.min() ;
        for (auto V : U | unvisited)
        if (auto t = distance(V) + edge_w(U,V); t < d(V))
            distance(V) = t ;
            previous(V) = U ;
    return ;
}
```

Steps for Dijkstra's Algorithm

- Start with a weighted graph
- Choose a Starting vertex and assign '∞' path values to all other devices.
- Go to each verter and update its path length
- If the path length of the adjecent verter is lesser than new path length, don't update it.
- Avoid updating path lengths of already visited vertices.
- After each iteration, pick the unvisited vertex with least path length.
- Sometimes, vertex has its path length updated twice.
- Repeat until all the vertices have been visited.

↳ Complexities

Time Complexity : $O(E.\log(V))$
Space Complexity : $O(V)$

↳ Applications

- Finds the Shortest Path
- Social Networking Applications
- Telephone Networks
- Internet Routers
- Google Maps, and Bing
- Routing Protocols
- Cellular Networks