DAT407 Assignment 8 – Group 19

Avinash Shukla - 8 hours
Josef Rasheed - 8 hours

May 24, 2023

Question 1a

To calculate the maximum number of iterations required to reach the solution in a BFS algorithm. We need to sum up the nodes we visit while considering the maximum number of children of a node for each level. So if we say that l is the level of a node, we can use the following formula for the maximum iterations required:

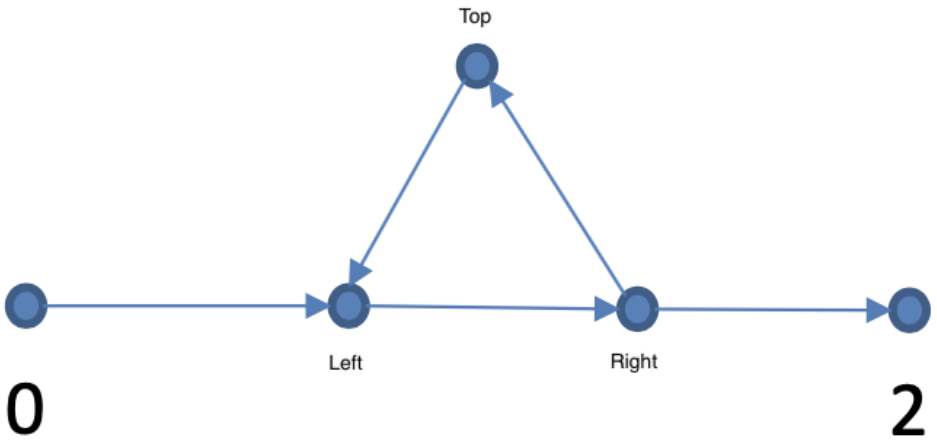
$$\sum_{l=0}^{r} d^l = \frac{d^{r+1}-1}{d-1}$$

Question 1b

If we again say that l is the level of a node, then for every level, the length of the path is $l + 1$. So for each iteration we need to store $(l + 1)d^r$ nodes. This means that we can calculate memory required with the formula:

$$\sum_{l=0}^{r} (l + 1)d^l = \frac{(r+1)d^{r+2}-(r+2)d^{r+1}+1}{(d-1)^2}$$

Question 2



Looking at the graph, we can see that the node labeled "right" has two possible directions it can go: towards the goal node and towards the top node. If the algorithm decides to explore the top node first, it will eventually come back to the "left" node, then move to the "right" node again and have to make a decision once more.

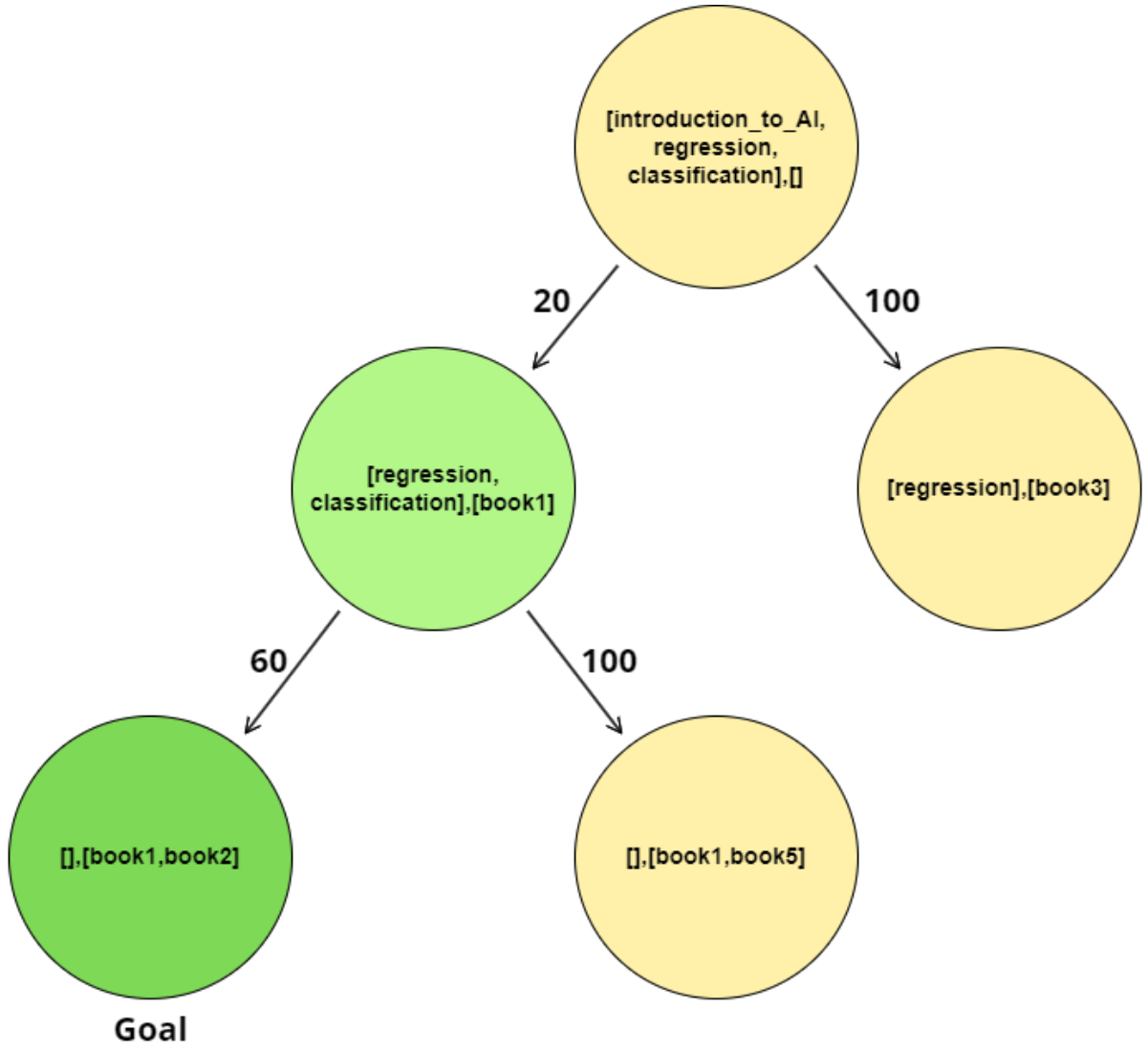
In this case, there is a tie between the two options because we assume the weights are the same. So, for Depth-First Search (DFS) to choose a node, it depends on the node with the smaller label. If we label the "top" node with a smaller label than the goal node, the algorithm will always choose the "top" node when it reaches the "right" node. This creates a loop, and the algorithm never reaches the goal node.

To address this, we can assign a label of 1 to the "top" node. Since 1 is smaller than 2 (the label of the goal node), the algorithm will always choose to go to the top node. The other two nodes only have one possible direction to move, so their labels don't affect the decision because the algorithm can only move in one direction. The labels only matter when we are at the "right" node and need to evaluate the labels of its neighbors.

We can modify DFS by using a set of visited nodes. This set keeps track of which nodes we have already visited, preventing us from choosing a path we have already traversed. During the first iteration, we check if we have visited the "top" node with a label of 1. If we haven't visited it, we go through that path. Then, when we reach the "right" node again, we check if we have visited the "top" node. If we have, we choose the other node, even if it has a bigger label.

In summary, if we have two possible nodes to choose from, before selecting the one with the smaller label, we check if we have visited that node. If we have, we choose the other node. We can also keep track of the number of possible nodes a node can visit. If, after going through the first iteration, any of the nodes we just visited have another unvisited node they can go to, we continue visiting nodes until we reach an unvisited node.

Question 3a



We start with introduction_to_AI since it is the “leftmost”. It has child nodes with book1 and book3, book1 with cost 20 is preferred, therefore the one with book3 is frontier node. The remaining topics are covered by book2 and book5.

Since book2 is cheaper with cost 60, it is chosen. We have now arrived at our goal node and the book 5 node is a frontier node.

Question 3b

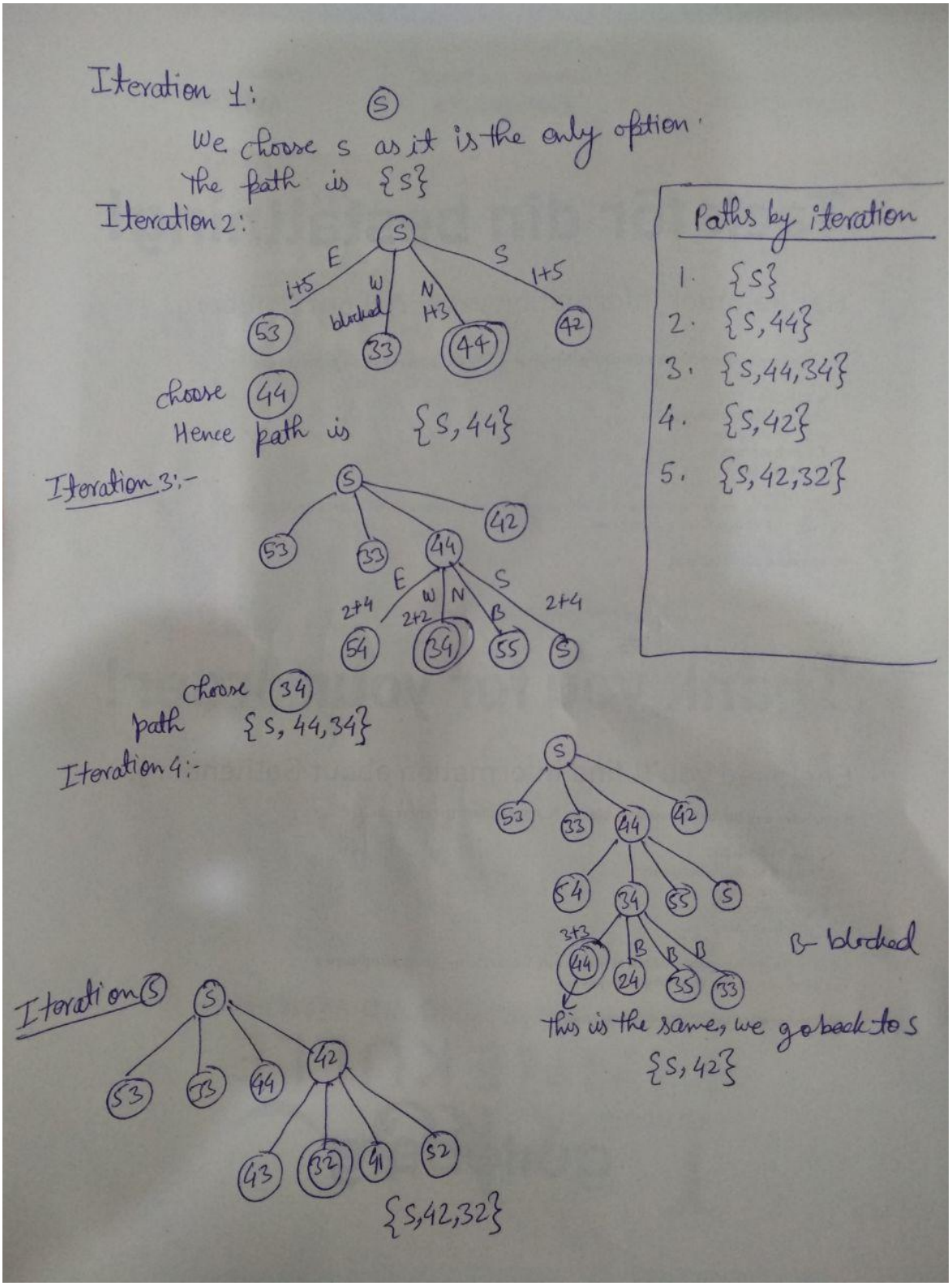
“B” is the set of books. “p” is the path. Then B(p) is the set of books and T(p) is the set topics in the last node of “p”.

The heuristic function then is:

$$h(p) = \begin{cases} \min\{B \setminus B(p)\} & \text{if } T(p) \neq \emptyset \\ 0 & \text{otherwise} \end{cases}$$

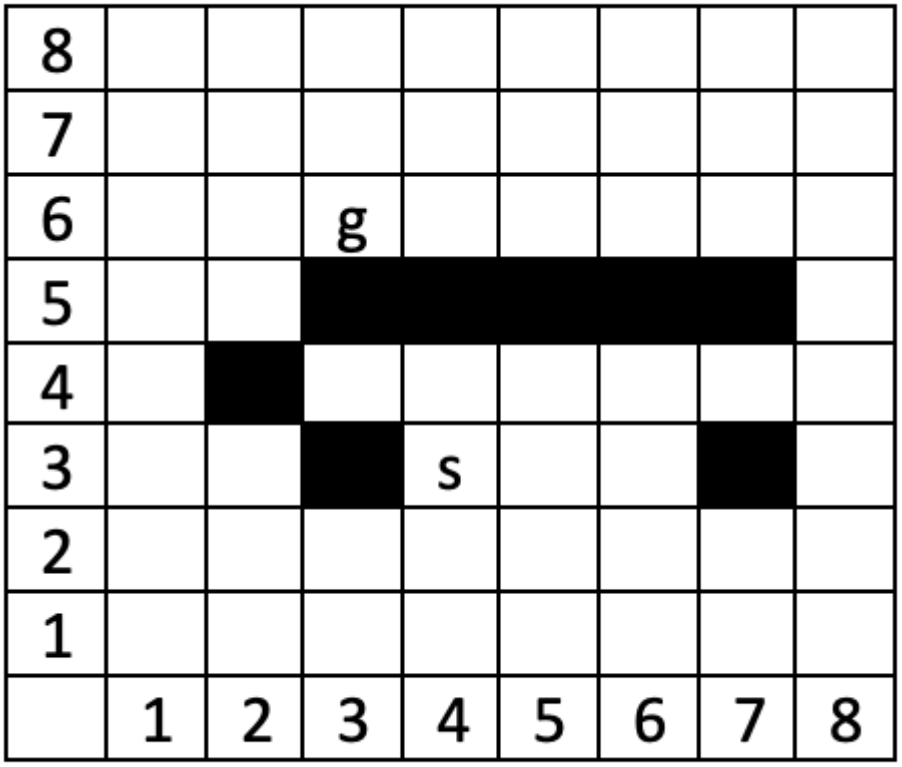
“min B” is the minimum pages from the set of books B.

Question 4a


Iteration 1:
We choose s as it is the only option.
The path is {s}
Iteration 2:
Choose 44
Hence path is {s, 44}
Iteration 3:-
Choose 34
path {s, 44, 34}
Iteration 4:-
Iteration 5
{s, 42, 32}
Paths by iteration
1. {s}
2. {s, 44}
3. {s, 44, 34}
4. {s, 42}
5. {s, 42, 32}
B- blocked
this is the same, we go back to s
{s, 42}

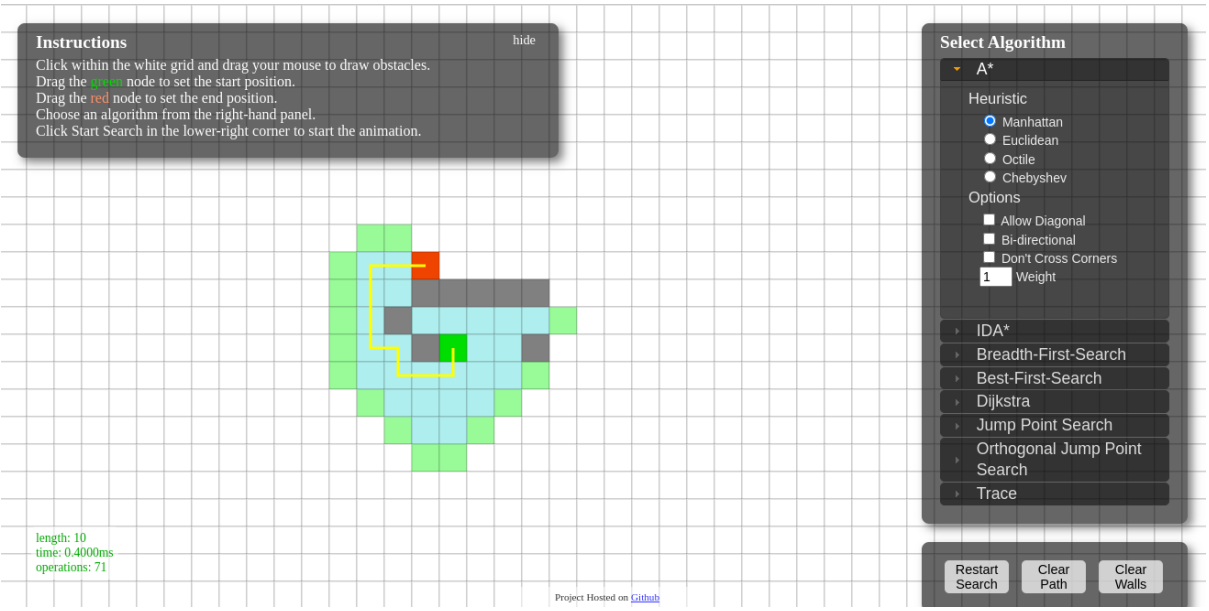
Paths by iteration:-
The coordinates of S are 43.

{S}
{S, 44}
{S, 44, 34}
{S, 42}
{S, 42, 32}



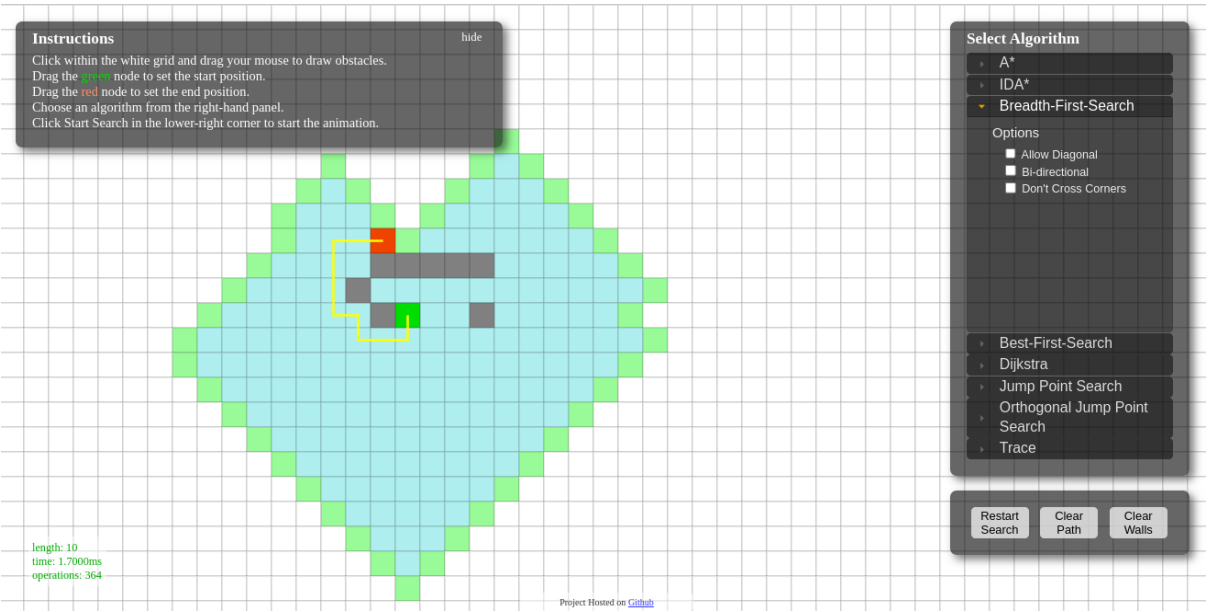
Question 4b

The search was fast, completing in just 0.4000 milliseconds. The path found had a length of 10, and the algorithm only needed 71 operations to reach the solution. Overall, it was a highly efficient method for this situation. The A* algorithm considers both the path cost and the heuristic values. In this case, we used the Manhattan distance as our heuristic, which means the movement was restricted to only four directions (up, down, left, and right), avoiding any diagonal movement.



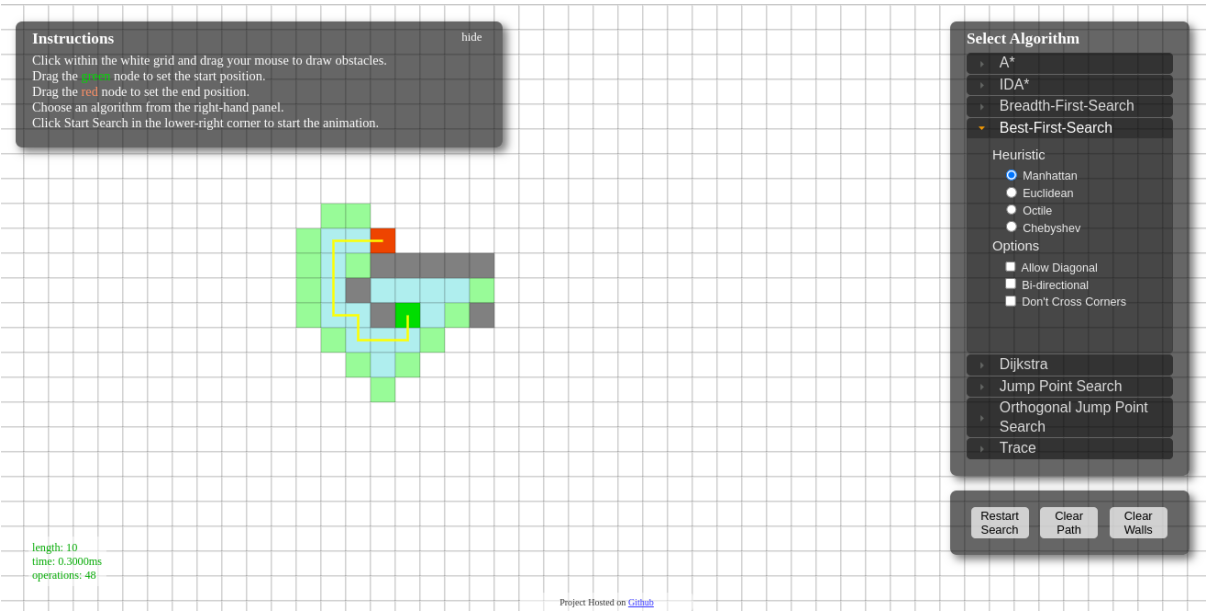
#BFS

The search took a bit longer, around 1.7000 milliseconds, but it still found a path of length 10, just like before. The downside is that it required a higher number of operations, specifically 364 operations, to reach the solution. This method, called BFS (Breadth-First Search), is different from the previous one because it doesn't use any specific information or heuristic. It explores all the nearby positions step by step, gradually moving farther away from the starting point until it finds the goal. It ensures that if there is a valid path, BFS will find the shortest one. The trade-off is that BFS needs to examine every unexplored position,making it slower compared to the previous method.



This method was fast, taking 0.3000 milliseconds to complete the search. It produced the same result as the previous two methods, finding a path of length 10. However, this method only required 48 operations, making it the

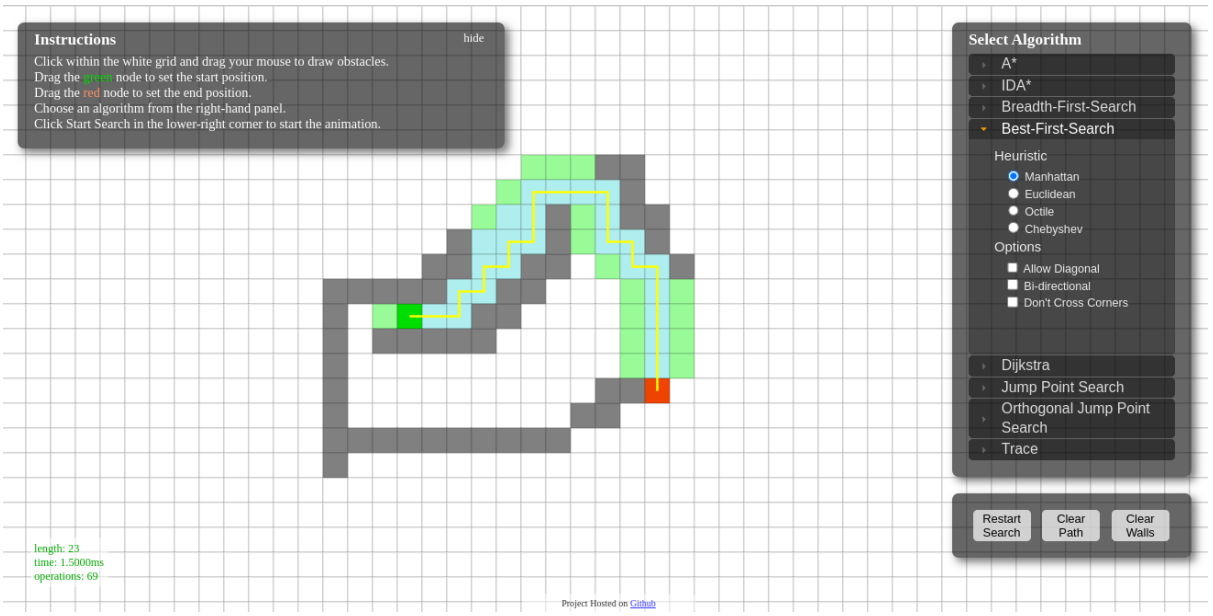
fastest and most suitable for this particular scenario. In this case, new paths are added to a list based on their heuristic score, which determines their priority. The list, known as the Frontier,is always sorted in order of the scores. This method, called Best-First Search, is faster because it doesn't consider the cost of the path to each state, unlike A*. Instead, it relies solely on the heuristic score to guide the search.



Question 4c

Breadth-First search
Length: 17

Best-First Search
Length: 23



We can see that the BFS approach is better.

BFS:
length: 17
time: 2.1000ms
398 operations

Best-First S:
length: 23
time: 1.5000ms
69 operations

The map was designed in a way that allows multiple routes to reach the goal. It wasn't restrictive and had various paths to choose from. One of the ways to get there looked like a staircase, with many steps going up and to the right. The Best-First Search algorithm decided to take this path because it appeared to have a lower estimated cost based on the heuristic. Although this algorithm managed to find a path, it wasn't the shortest one available.

In contrast, even though it required more operations, the Breadth-First Search (BFS) algorithm was able to find the shortest path. It follows a strategy of exploring nodes in the order they were discovered, which ensures that the shortest path to the solution is found first.