RBE 550 Homework 1 Documentation

Alex Tacescu — Spring 2021

I. Introduction

This document accompanies my homework 1 submission. It is designed to explain my thought process when developing my code. It will also explain the 4 algorithms implemented: Breath First Search (BFS), Depth First Search (DFS), Dijkstra's Algorithm, and A*.

II. General Code Setup

The code is split up into two main sections, with 2 classes developed to accompany the search algorithms while also simplifying the code. The `map2d` class (`map2d.py`) is designed to store the map and add features such as returning local neighbors, getting distances between points, and getting occupancy data of nodes. It allows the main code in search.py to be significantly more brief and easier to read, while avoiding duplicate code. Similarly, the `PriorityQueue` class (`priorityqueue.py`) was created to keep a priority queue style data structure. It is important to note that lists were used instead of `numpy` arrays to minimize dependencies, especially when running on a professor/TA computer.

Testing the algorithms can be separated in to two sections. First of all, the search algorithms were developed using a basic map found in `test_map.csv`. This allowed me to hone in the software and ensure that everything worked as planned. Afterwards, 3 additional maps were created (other than `map.csv`) in order to test certain expected qualities of each algorithm. For example, a map was created to show the difference between BFS and DFS. The graphs shown in this paper will use the `map.csv` default for familiarity. The graphing tool from the next subsection also was used to ensure all algorithms were performing as expected.

A. Additional Tool: Graphing Visited Nodes and Their Weights

In order to better debug and visualize the weights and priorities while implementing Dijkstra's algorithm (Section V) and A* (Section VI), a graph was developed that displays the visited nodes and their weights. This was added to the map2d class to be easily used between algorithms. You can see an example in Figure VI-B

III. Breath First Search (BFS)

The main concept behind BFS is that the algorithm explores a graph or map equally in all directions. It is extremely simple and useful, and usually leads to a complete search and an optimal path. However, it takes a long time to compute since there is no heuristic pushing the algorithm to look toward the goal. It is also important to note that BFS algorithms assume there is no difference moving between points on a graph/map - a major limitation especially when path planning.

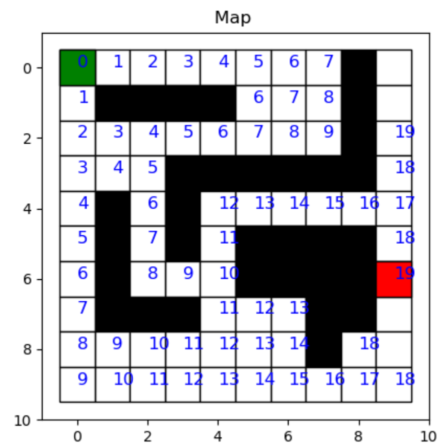

Fig. 1. Example of a map populated with the heuristic weights from Dijkstra's Algorithm

A. Implementation

BFS is one of the most simple algorithms, usually taking only a few lines of code to fully implement. Below is some basic Python pseudocode describing my implementation of BFS: [1]

```
frontier = list()
frontier.append(start_node)
visited = dictionary()
visited[start_node] = None

while frontier not empty:
    current = frontier.get_first()
    if goal found in visited:
        exit loop and return path
    for neighbor in neighbors of current:
        if neighbor exists and \
            not in visited and \
            not obstacle:
            frontier.add_to_end(neighbor)
            visited[neighbor] = current
```

The output of the code above only gives a list of visited nodes and where they were visited from. Therefore, an additional section of code was needed to generate a path:

```
path = list()
curr_point = goal_node
while curr_point is not start:
        path.append(curr_point)
        curr_point = visited.get(curr_point)
path.append(start)
path.reverse()
```

Finally, to get the number of steps taken, I simply took the length of the `visited` dictionary.

B. Results

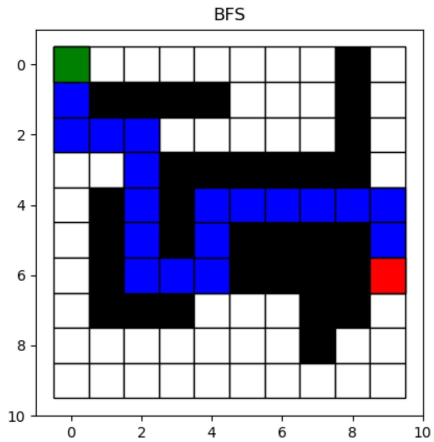

Fig. 2. Breath First Search Results for a 10x10 map. The algorithm took 64 steps to find the solution.

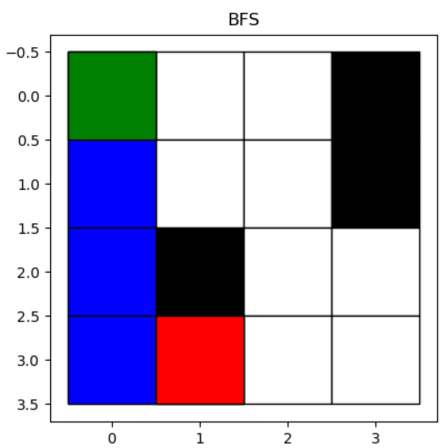

Fig. 3. Breath First Search result for a 4x4 map. The algorithm took 10 steps to find the solution

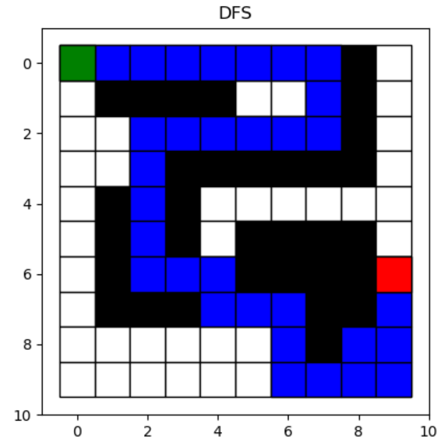

Fig. 4. Depth First Search result for a 10x10 map. The algorithm took 33 steps to find the solution

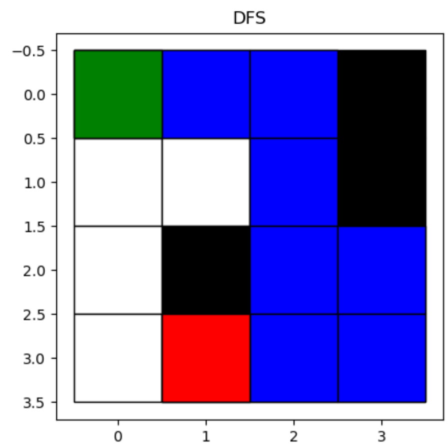

Fig. 5. Depth First Search result for a 4x4 map. The algorithm took 9 steps to find the solution

IV. Depth First Search (DFS)

A. Implementation

B. Results

V. Dijkstra's Algorithm

Dijkstra's algorithm extends the usefulness Breath First Search by considering path heuristics. This is extremely useful to get a complete and optimal path from a start node to a goal node considering the movement effort going from one node to another. However, like BFS, it takes a very long time since there is no heuristic to push the algorithm to the goal node faster than other methods.

A. Implementation

B. Results

VI. A*

A. Implementation

B. Results

References

[1] RedBlob Games, May 2014. [Online]. Available: https://www.redblobgames.com/pathfinding/a-star/introduction.html.

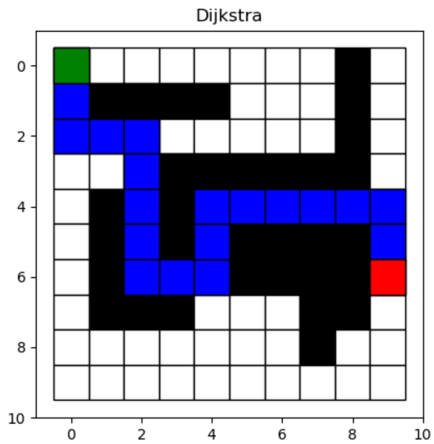

Fig. 6. Dijkstra's Algorithm search result for a 10x10 map. The algorithm took 64 steps to find the solution

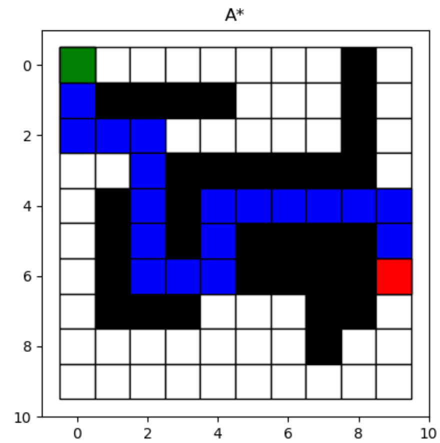

Fig. 8. A* search result for a 10x10 map. The algorithm took 48 steps to find the solution

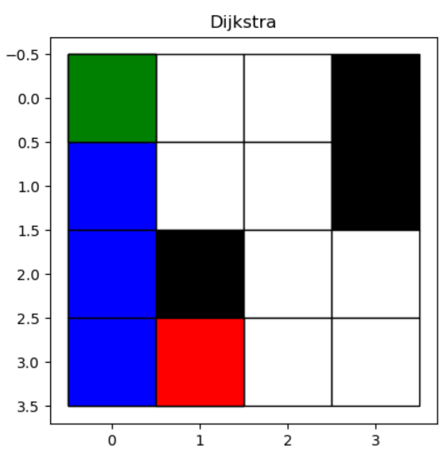

Fig. 7. Dijkstra's Algorithm search result for a 4x4 map. The algorithm took 10 steps to find the solution

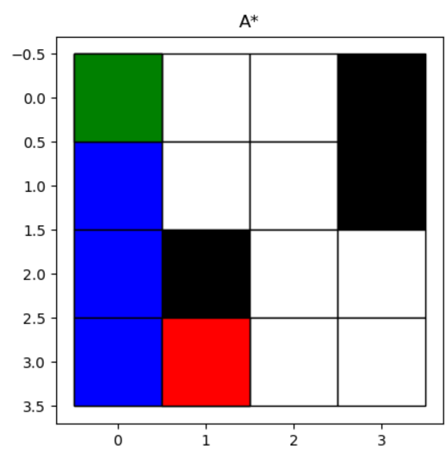

Fig. 9. A* search result for a 4x4 map. The algorithm took 8 steps to find the solution