**Assignment 1**

Tree Based Search

**Due:** 4:30pm Friday 21st April 2023 (Week 7)

**Contributes:** 30% of unit mark

**Task Type:** Individual.

**Student**: Nathan Hoorbakht (103865794)

|  |  |
| --- | --- |
| Instructions | 1 |
| Introduction | 2 |
| Search Algorithms: | 2 |
| Implementation | 3 |
| Depth-First Search Algorithm (DFS) | 3 |
| Breadth-First Search Algorithm (BFS) | 3 |
| Greedy Best-First Search Algorithm (GBFS) | 4 |
| A Star Search Algorithm (AS) | 4 |
| Random Search (RND) | 5 |
| Features/Bugs/Missing: | 6 |
| Notable Features | 6 |
| Bugs & Missing Features | 6 |
| Research | 7 |
| Test Suites | 7 |
| UI visualizer | 8 |
| Conclusion: | 9 |
| Acknowledgements/Resources | 9 |
| References | 9 |

**Instructions**:

To run the program, navigate to the directory where search.exe file is location – Robot Navigation/Obj/Debug. Then open command prompt from that directory.

Input

Type the command: “Robot Navagation.exe” <path\_to\_file> <search\_strategy>

Search strategies:

* BFS
* DFS
* AS
* GBFS
* RS

Output:

Example Output

*TestInput.txt*

*BFS: 23*

*down; right; right; right; right; up; up; right; right; right;*

Additionally, once the program is run a command-line UI will be generated for the map. The symbols will indicate the value of each cell: initial state (i), goals (g) and walls (w).

Text

Description automatically generated

**Introduction**:

The goal of the Introduction to AI Assignment 1 is to assess your understanding and ability to apply tree-based search algorithms in solving two different problems: the NxM-Puzzle problem and the Robot Navigation problem. By completing this assignment, you will demonstrate your knowledge of informed and uninformed search methods, as well as your ability to write software programs in my chosen language of C#.

For this assessment I have selected option B of the assignment which requires a solution to the Robot Navigation problem. The program will use 2 of each informed and uninformed tree-based search algorithms: DFS BFS GBFS and A\*. Additionally, 2 other search algorithms are to be researched and added to the program.

**Search Algorithms**:

1. **Depth-First Search** (DFS) is an uninformed search algorithm that explores as far as possible along each branch before backtracking. DFS is easy to implement and memory-efficient, making it useful for large graphs. However, DFS does not guarantee the shortest path to the goal node, and it can get stuck in infinite loops if the graph contains cycles.
2. **Breadth-First Search** (BFS) is another uninformed search algorithm that explores all the nodes at the current depth before moving on to the next depth level. BFS guarantees the shortest path to the goal node and is more suitable for finding the optimal solution in unweighted graphs. However, BFS can be memory-intensive and may not work well for large graphs.
3. **Greedy Best-First Search** (GBFS) is an informed search algorithm that uses a heuristic to select the best node to explore next. GBFS is faster than BFS and more memory-efficient than A\*, but it may not find the optimal solution to the problem.
4. **A\* Search**, also known as A Star, is an algorithm that considers two factors when determining the optimal path. These factors are the G-Score, which is the path cost between nodes, and H-Score, which is an estimate of the distance to from the current location to the goal. Utilizing these factors, A\* can search for the most direct path while minimising the number of nodes that need to be explored. This results in a faster and more efficient system that uses less memory.
5. **Random Search** is a brute-force approach to an uninformed search algorithm that randomly selects a node from the available paths and explores it until it finds the goal node. Random search does not use any heuristics or search strategies, making it simple to implement but not very efficient or effective.

Overall, the best algorithm depends on the problem domain and the specific requirements of the search. However, for Robot Navigation Task A\* is the most effective search algorithm as it guarantees the optimal path to the goal.

**Implementation**:

**Depth-First Search (DFS)**

The Depth-First Search algorithm can also be implemented iteratively using a stack data structure instead of recursion. The iterative DFS algorithm works by pushing the starting node onto the stack and then repeatedly popping the top node off the stack, marking it as visited, and pushing its unvisited neighbours onto the stack. The iterative DFS algorithm works on the following pseudo-code:

****

**Breadth-First Search (BFS)**

The Breadth-First Search algorithm is a graph traversal algorithm that explores all the vertices of a graph in a level-wise manner. It starts at the root node and explores all the nodes at the current level before moving to the next level. BFS is implemented using a queue data structure to store nodes.

To implement BFS, we first initialize a queue with the root node. Then, we dequeue the node from the front of the queue and examine its neighbours. Any unvisited neighbours are added to the end of the queue. The process continues until the queue is empty or the goal node is found. This works on the following pseudo-code.

****

**Greedy Best-first Search (GBFS)**

The Greedy Best-First Search algorithm is a search algorithm that explores a graph by expanding the node that appears to be closest to the goal node, as determined by a heuristic function. GBFS maintains a priority queue of nodes to visit, sorted by their estimated distance to the goal node.

To implement GBFS, we first initialize a priority queue with the root node. Then, we dequeue the node from the front of the queue and examine its neighbours. We calculate the heuristic value for each neighbour and add them to the priority queue based on their heuristic value. The node with the lowest heuristic value is always placed at the font of the queue as highest priority. This works on the following pseudo-code.

****

**A\* ("AStar") (AS)**

The A\* algorithm evaluates each node on the open list, selecting the node with the lowest combined cost (the cost to reach the node plus the estimated cost to get from that node to the goal) for expansion. The estimated cost is calculated using a heuristic function, which is admissible (never overestimates the actual cost) and consistent (monotonically decreasing along any path).

For this assessment the Manhattan Heuristic function was used as follows:



Here is an example pseudocode for the A\* algorithm:

****

**Random Search (RS)**

The Random search algorithm selects a random path to expand and adds the destination node to the list of visited nodes. If the selected path has no valid destination nodes, the algorithm terminates and returns "No solution".

****

**Features/Bugs/Missing:**

**Features**

The program contains all the algorithms in a single executable. Algorithm selection is accomplished by specifying program arguments during launch. To use one of the available algorithms - DFS, BFS, GBFS, AS, or RS - simply enter the corresponding code. The program will then load the appropriate environment from file and execute the selected algorithm.

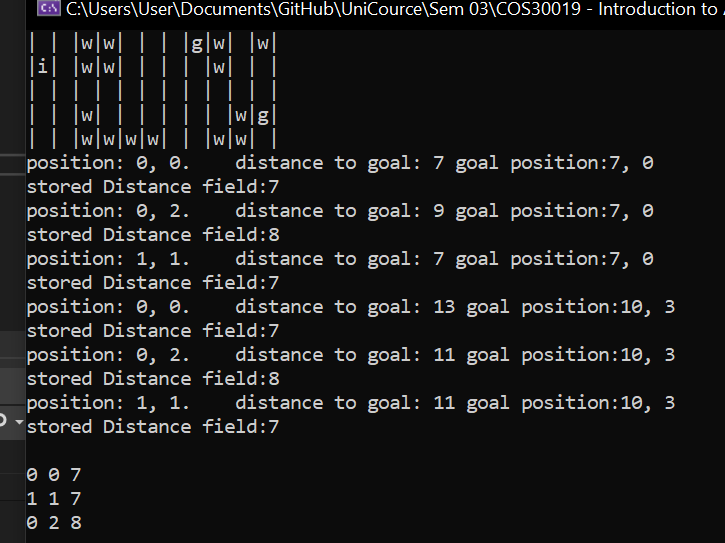
UI Visualizer: a user interface that draws the map as described in the file. Where ‘g’ is goal, ‘i’ is the initial point, ‘w’ are the walls, and ‘x’ are the movements of the algorithm.

Test Suite: in code methods that enable bug testing.

**Bugs & Missing Features**

BUG – When the algorithm finds the goal the UI will print both x and g which will momentarily create a cell larger than the map. However, once it confirms the goal, it corrects itself replacing the goal with an x marker.

BUG – There is a bug in the GBFS (Greedy Best First Search) algorithm, where the distance to the goal stored in the nodes does not match the input. Specifically, the distance to the goal for position 0,2 is calculated as 9, but the field (.DistanceToGoal) in the nodes only stores up to 8. Even after trying to artificially increase this value, it remains limited to 8. Further investigation showed that for y-coordinates greater than 2, the distance is reduced to 8. Although this produces an almost correct solution, the first step is downwards instead of to the right, and the sequence is slightly longer as it checks other positions.



MISSING – The random search algorithm lacks the ability to backtrack when encountering a dead end, resulting in premature termination of the program. This can lead to incomplete or incorrect results and limit the algorithm's effectiveness. Enabling backtracking would allow the algorithm to revisit previously explored nodes and uncover new solutions, improving the accuracy and potential for the success of the algorithm.

MISSING – Program does not include CUS2 (Informed search).

**Research**:

**Test Suite**

The printInfo method prints information about the map, such as its size, initial state, goal states, and walls. It's used to check the accuracy of the input data and ensuring the map is generated correctly.

The printMap method prints details about each grid on the map, including its position, whether it's a wall or not, and available paths. It's used to test the code's pathfinding function and verify the agent's ability to navigate the map.

**UI Visualizer**

The Draw method is utilized to visually represent the grid map, using characters to denote the value of each cell. For instance, the letter I indicates the initial position of the robot, G represents the goal position, W represents walls, and X represents the current position of the agent.

Similarly, the DrawPath method functions in a similar manner to the draw method, with the additional feature of also mapping out the path between the robot's initial position and the goal position. During this process, the method prints "x" for each cell that falls within the path.

Pseudo code for Draw method:



Pseudo code for DrawPath method:Top of Form



**Conclusion**:

For similar cases, I prefer using the A\* algorithm due to its efficiency and ability to quickly provide the shortest path to the destination. To improve its performance, I can enhance the overestimation function (heuristic) for the cost of the remaining path, which would reduce the number of explored nodes. Other search methods may be considered based on any changes made to the initial task.

**Acknowledgements/Resources**:

The Data Structures and Patterns unit covers both breadth-first search and depth-first search algorithms. These algorithms were developed using the knowledge gained in the unit and further research on A\*, GBFS, and Random search from the links provided below helped to deepen my understanding.

**References:**

GeeksforGeeks. (n.d.). Breadth First Search or BFS for a Graph. Retrieved April 1, 2023, from <https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/>

GeeksforGeeks. (n.d.). Greedy Best-First Search (GBFS). Retrieved April 3, 2023, from <https://www.geeksforgeeks.org/greedy-best-first-search-gbfs/>

GeeksforGeeks. (n.d.). A\* Search Algorithm. Retrieved April 1, 2023, from <https://www.geeksforgeeks.org/a-search-algorithm/>

GeeksforGeeks. (n.d.). Depth First Search or DFS for a Graph. Retrieved April 1, 2023, from <https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>

GeeksforGeeks. (n.d.). Randomized Algorithms. Retrieved April 3, 2023, from <https://www.geeksforgeeks.org/randomized-algorithms/>

Wikipedia contributors. (2023, Feburary 27). Breadth-first search. In Wikipedia, The Free Encyclopedia. Retrieved April 1, 2023, from <https://en.wikipedia.org/wiki/Breadth-first_search>

Wikipedia contributors. (2023, March 12). Depth-first search. In Wikipedia, The Free Encyclopedia. Retrieved April 1, 2023, from <https://en.wikipedia.org/wiki/Depth-first_search>

Wikipedia contributors. (2023, February 24). Best-first search. In Wikipedia, The Free Encyclopedia. Retrieved April 3, 2023, from <https://en.wikipedia.org/wiki/Best-first_search>

Wikipedia contributors. (2023, March 16). A\* search algorithm. In Wikipedia, The Free Encyclopedia. Retrieved April 3, 2023, from [https://en.wikipedia.org/wiki/A\*\_search\_algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm)

Wikipedia contributors. (2023, March 5). Heuristic (computer science). In Wikipedia, The Free Encyclopedia. Retrieved April 12, 2023, from <https://en.wikipedia.org/wiki/Heuristic_(computer_science)>

GeeksforGeeks. (n.d.). Search Algorithms in AI. Retrieved April 10, 2023, from <https://www.geeksforgeeks.org/search-algorithms-in-ai/>

GeeksforGeeks. (n.d.). Best First Search (Informed Search). Retrieved April 13, 2023, from <https://www.geeksforgeeks.org/best-first-search-informed-search/>

GeeksforGeeks. (n.d.). A\* Search Algorithm. Retrieved April 10, 2023, from <https://www.geeksforgeeks.org/a-search-algorithm/>

Javatpoint. (n.d.). Breadth-First Search Algorithm. Retrieved April 10, 2023, from <https://www.javatpoint.com/breadth-first-search-algorithm>

JavaTpoint. (n.d.). Depth First Search Algorithm in Data Structure. Retrieved April 10, 2023, from <https://www.javatpoint.com/depth-first-search-algorithm>

Wikipedia contributors. (n.d.). Random search. In Wikipedia. Retrieved April 3, 2023, from <https://en.wikipedia.org/wiki/Random_search>

Visualgo.net. (n.d.). BFS/DFS. Retrieved from <https://visualgo.net/en/dfsbfs?slide=1>