Assignment 1

Tree Based Search

Authors

Vu Duc Tran 104175614

Table of Contents

Instruction

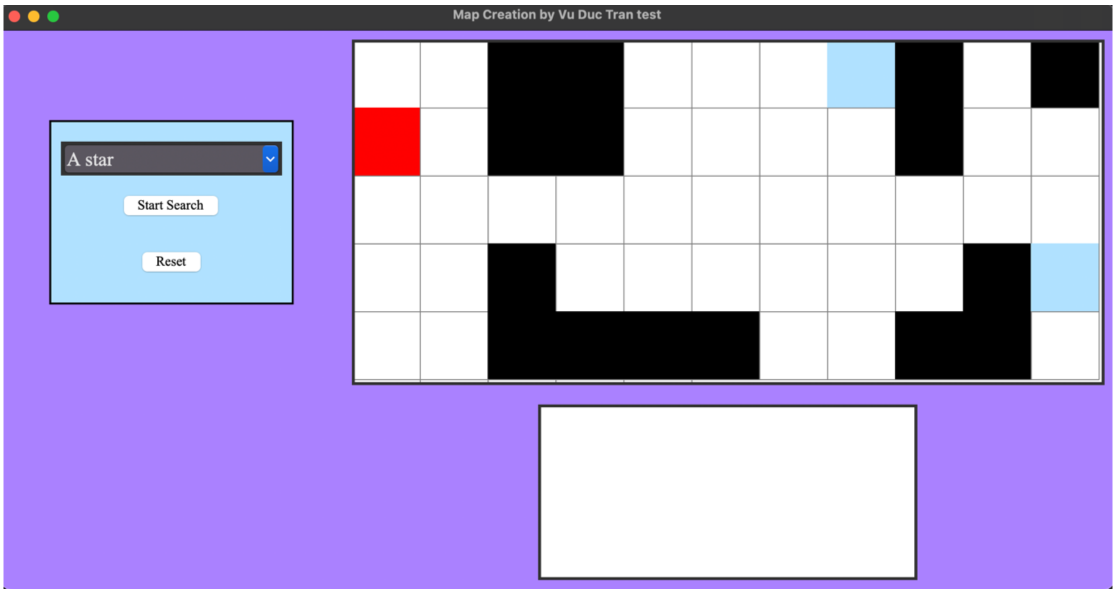
Executing the scripts using the following command from the CLI:

```
C:\Assignments> python search.py <filename> <method>
```

It will illustrate the goal node, number of nodes has created, a sequence of moves in the solution that brings you from the start to the end.

Running Visualisation Maze steps:

- Run "display.py" file



- Choose a search algorithm
- Start searching by "Start Search"

The explored nodes and the path to the goal will be drawn

The white canvas below the maze illustrates number of nodes has created, a sequence of moves in the solution that brings you from the start to the end.

- Before search by an another algorithm, click "Reset" to reset the maze

Introduction

Search Algorithms

Algorithm	**Optimality**	**Time Complexity**

Breadth-First Search	Find the shortest path, guarantee optimality	$O(b^d)$ in the worst case.
Depth-First Search	May find a non-optimal solution	$O(b^d)$ in the worst case.
A* Search	Guarantee optimality if the heuristic function is admissible	Depend on heuristic $O(b^d)$ in the worst case.
Greedy Best-First Search	May stuck in local minima	Depend on heuristic $O(b^d)$ in the worst case.
Iterative Deepening Depth-First Search	May not find the optimal solution (graph search)	$O(b^d)$ in the worst case.
Iterative Deepening A∗	Guarantee optimality if the heuristic function is admissible	$O(b^d)$ in the worst case.

While BFS guarantees optimality and completeness and has linear space complexity, it may suffer from exponential time complexity. DFS, on the other hand, has linear space complexity but does not guarantee optimality and may get stuck in infinite branches. A* and GBFS are informed search algorithms that often find good solutions quickly but may not be optimal.

IDS combines the completeness of DFS but it may still suffer from exponential time complexity because it could miss a shorter path due to exploring only a subset of the graph.

IDA offers optimality guarantees similar to A*, but with reduced memory overhead by performing depth-limited searches with increasing limits until a solution is found.

Implementation

Algorithm	**Pseudocode**	**References**
Breadth-First Search	Start Initialize queue, explored set, and visited_nodes counter Add initial node to the queue While queue is not empty Dequeue node from the front of the queue Check if node is the goal state Expand node and add children to the queue Mark node as explored Goal state found?	Tutorial Week 3 COS30019 Introduction to Artificial Intelligent

	End	
Depth-First Search	Start Initialize stack, explored set, and visited_nodes counter Push initial node onto the stack While stack is not empty Pop node from the top of the stack Check if node is the goal state Expand node and push children onto the stack Mark node as explored Goal state found? End	
A* Search	Start Initialize priority queue, g_cost dictionary, explored set, and visited_nodes counter Add initial node to priority queue with f_cost 0 While priority queue is not empty Pop node with lowest f_cost from priority queue Check if node is the goal state Expand node and calculate f_cost for each child For each child Calculate tentative g_cost Calculate f_cost as sum of g_cost and heuristic value If child not in explored or frontier Add child to priority queue with f_cost Update g_cost for child Else if child in frontier with higher f_cost Update child's f_cost and g_cost in priority queue Mark node as explored Goal state found? End	

Greedy Best-First Search	Start Initialize priority queue, explored set, and visited_nodes counter Add initial node to priority queue with heuristic value While priority queue is not empty Pop node with lowest heuristic value from priority queue Check if node is the goal state Expand node and calculate heuristic value for each child For each child If child not in explored or frontier Add child to priority queue with heuristic value Mark node as explored Goal state found? End	
Iterative Deepening Depth-First Search	Start Initialize depth_limit to 0 Repeat until goal state found: Perform depth-limited search with current depth_limit Increase depth_limit by 1 for next iteration Goal state found? End	https://people.engr.tamu.edu/guni/csce421/files/AI_Russell_Norvig.pdf
Iterative Deepening A*		

Differences in approach:

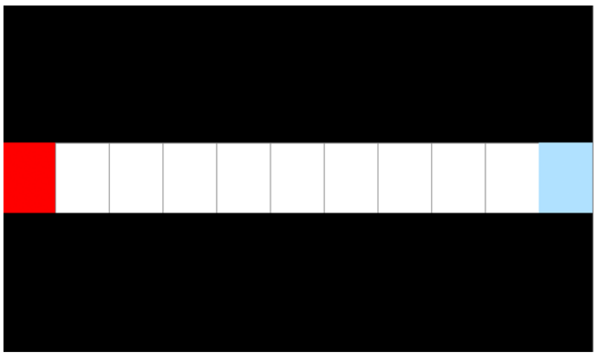
- When exploring a node, DFS pushes its children onto the stack. Thus, A stack follows the Last-In-First-Out (LIFO) principle, making it suitable for implementing DFS.
- A deque supports efficient insertion and removal from both ends, making it suitable for BFS. It allows nodes at the current level to be explored before moving to the next level.
- A priority queue organizes elements based on their priority and in n A* and GBFS, nodes are prioritized based on a combination of path cost and heuristic

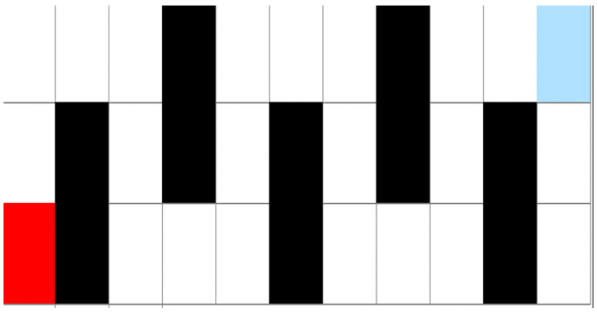
value. A priority queue ensures that nodes with lower estimated total cost are explored first.
- When implementing Iterative Deepening Depth-First Search, I combine DFS with iterative deepening. This approach addresses the main limitation of pure DFS.

Testing

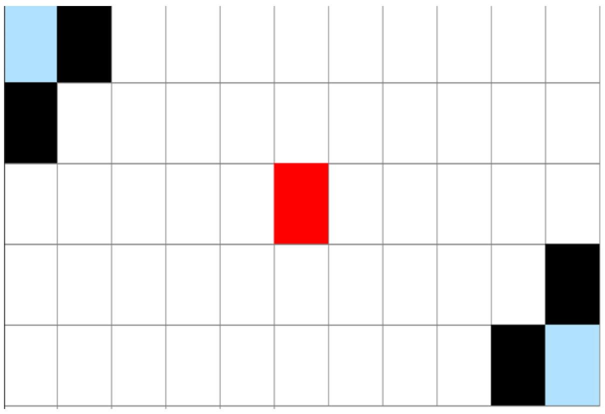
I manually create 10 different mazes. These include different situations: **a Single Path, two Paths with equal nodes, Blocked Goal, Blocked Start,…**

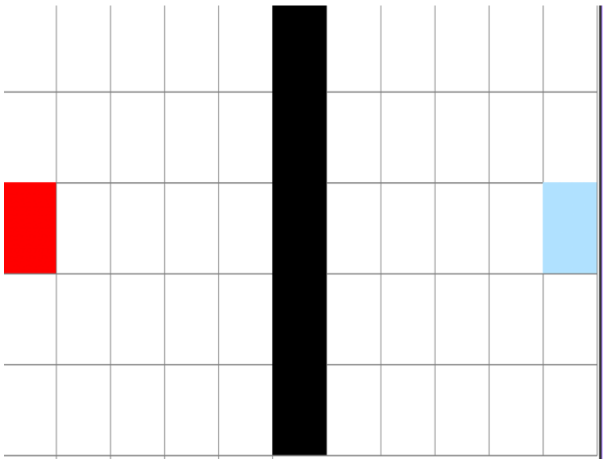
Only One Solution Maze

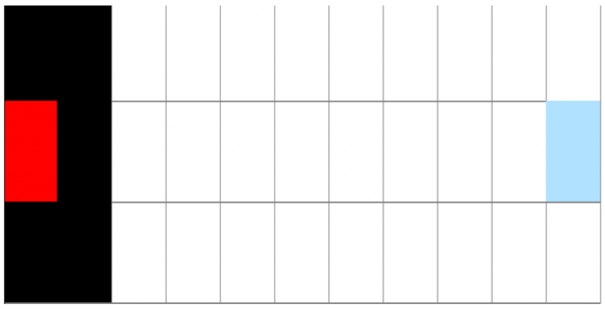

test 1


test 6

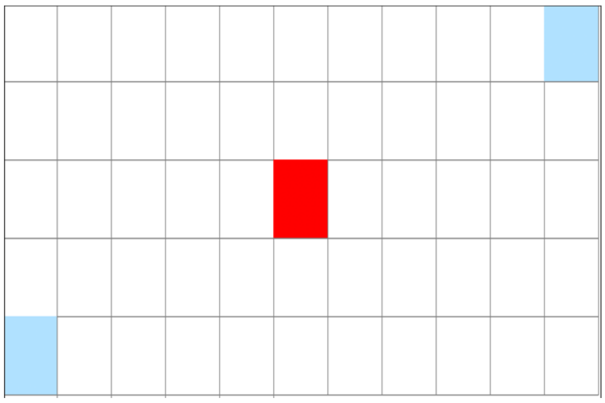
No Solution Maze

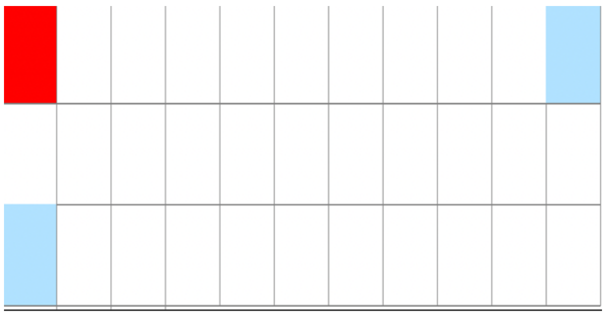

test 8

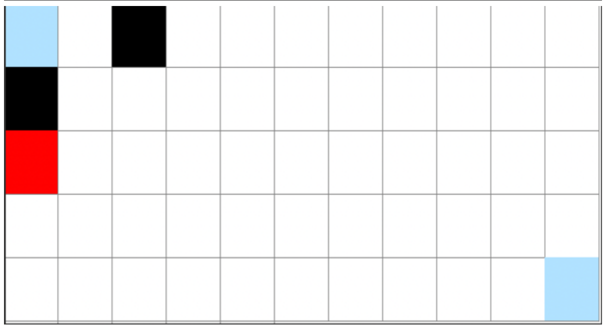

test 3


test 4

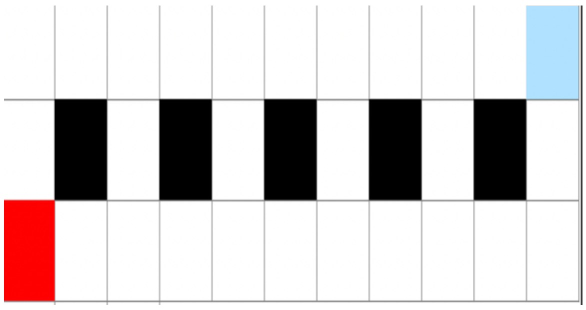
Multiple Goals

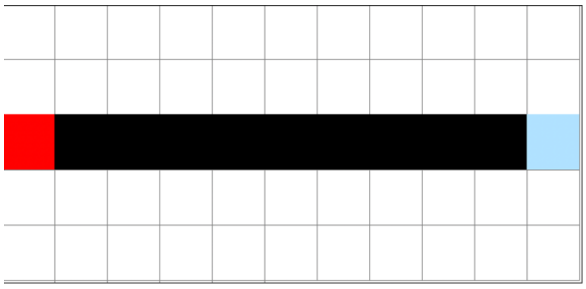

test 10


test 7


test 5

Multiple Paths Maze


test 9


test 2

Maze (Test number)	**Algorithms**	**Number of Nodes explored to get to the goal (or after no goal is unreachable)**
Number 1	Breadth-First Search	**11**
	Depth-First Search	**11**
	A* Search	**11**
	Greedy Best-First Search	**11**
	Iterative Deepening Depth-First	**11**

	Search	
	Iterative Deepening A∗	
Number 2	Breadth-First Search	**46**
	Depth-First Search	**25**
	A* Search	**45**
	Greedy Best-First Search	**25**
	Iterative Deepening Depth-First Search	**14**
	Iterative Deepening A∗	
Number 3	Breadth-First Search	**25**
	Depth-First Search	**25**
	A* Search	**25**
	Greedy Best-First Search	**25**
	Iterative Deepening Depth-First Search	**25**
	Iterative Deepening A∗	
Number 4	Breadth-First Search	**1**
	Depth-First Search	**1**
	A* Search	**1**
	Greedy Best-First Search	**1**
	Iterative Deepening Depth-First Search	**1**
	Iterative Deepening A∗	
Number 5	Breadth-First Search	**6**
	Depth-First Search	**21**
	A* Search	**5**
	Greedy Best-First Search	**5**
	Iterative Deepening Depth-First Search	**7**
	Iterative Deepening A∗	
Number 6	Breadth-First Search	**21**
	Depth-First Search	**21**
	A* Search	**21**
	Greedy Best-First Search	**21**
	Iterative Deepening Depth-First Search	**21**

	Iterative Deepening A∗	
Number 7	Breadth-First Search	**19**
	Depth-First Search	**33**
	A* Search	**10**
	Greedy Best-First Search	**10**
	Iterative Deepening Depth-First Search	**19**
	Iterative Deepening A∗	
Number 8	Breadth-First Search	**49**
	Depth-First Search	**49**
	A* Search	**49**
	Greedy Best-First Search	**49**
	Iterative Deepening Depth-First Search	**49**
	Iterative Deepening A∗	
Number 9	Breadth-First Search	**28**
	Depth-First Search	**18**
	A* Search	**27**
	Greedy Best-First Search	**18**
	Iterative Deepening Depth-First Search	**13**
	Iterative Deepening A∗	
Number 10	Breadth-First Search	**55**
	Depth-First Search	**34**
	A* Search	**20**
	Greedy Best-First Search	**20**
	Iterative Deepening Depth-First Search	**21**
	Iterative Deepening A∗	

Features/Bugs/Fixing

Key features that I have implemented in this assignment:

- Breadth-First Search
- Depth-First Search
- A* Search
- Greedy Best-First Search

- CUS 1 (Iterative Deepening Depth-First Search)
- CUS 2 (Iterative Deepening A∗)
- 10 test cases
- GUI versions

Bugs:

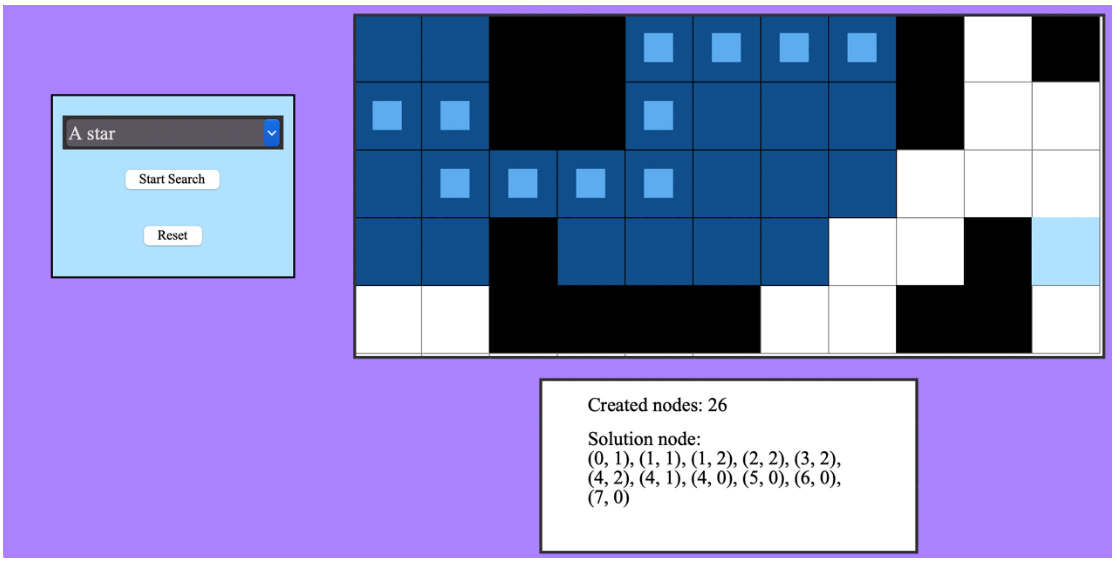
- I cannot print the explored nodes during expanding in Iterative Deepening Depth-First Search Algorithm.
- My program is not very well-structured, it's not OOP.
- I tried to implement Iterative Deepening A* but it's not very consistent.

Extensions

Building a GUI using Tkinter.

Running Visualisation Maze steps:

- Run "display.py" file



Created nodes: 26

Solution node:
(0, 1), (1, 1), (1, 2), (2, 2), (3, 2), (4, 2), (4, 1), (4, 0), (5, 0), (6, 0), (7, 0)

- Choose a search algorithm
- Start searching by "Start Search"

The explored nodes and the path to the goal will be drawn

The dark blue square is the explored/created nodes.

The small light blue square indicates the way from the starting point to the goal point.

The white canvas below the maze illustrates number of nodes has created, a sequence of moves in the solution that brings you from the start to the end.

- Before search by an another algorithm, click "Reset" to reset the maze

Conclusion

In conclusion, the report has explored and compared six different search algorithms: Breadth First Search (BFS), Depth First Search (DFS), A* Search, Greedy Best First Search (GBFS), Iterative Deepening Depth First Search (IDDFS), and Iterative Deepening A* (IDA*). Through testing and analysis, I have shown insights into their performance across various problem domains.

Each algorithm has its strengths and weaknesses, making them suitable for different scenarios. BFS guarantees optimality and completeness but may suffer from exponential time complexity. DFS has linear space complexity but does not guarantee optimality. A* and GBFS are informed search algorithms that often find solutions quickly but may not always be optimal. IDDFS combines the completeness of DFS with iterative deepening, mitigating the risk of infinite branches. IDA* offers optimality guarantees similar to A* with reduced memory overhead.

Overall, each algorithm has different solution optimality, time complexity.

Acknowledgements/Resource

- **(W3) Tutorial COS30019**

https://swinburne.instructure.com/courses/56950/pages/w3-tutorial?module_item_id=3841164

This tutorial explains and implements BFS, DFS search algorithms, give a based knowledge and logic code of reading and creating maze from a file, applying search algorithms in solving maze.

- **Artificial Intelligence: A Modern Approach**

https://people.engr.tamu.edu/guni/csce421/files/AI_Russell_Norvig.pdf

This book gives me an overall look of 6 search algorithms I have implemented.

It mainly contributes to my understand of Iterative Deepening Depth-First Search and Iterative Deepening A∗.

- **Iterative Deepening Search | IDS Search | DFS Algorithm in Artificial Intelligence by Mahesh Huddar**

https://www.youtube.com/watch?v=BK8cEWKHCkY&ab_channel=MaheshHuddar

This video demonstrates how IDDFS works, help me visualize the differences between IDDFS and DFS

References

Russell, Stuart J. (Stuart Jonathan), 1962-. (2010). Artificial intelligence : a modern approach. Upper Saddle River, N.J. :Prentice Hall, retrived from https://people.engr.tamu.edu/guni/csce421/files/AI_Russell_Norvig.pdf

https://swinburne.instructure.com/courses/56950/pages/w3-tutorial?module_item_id=3841164