Assignment 1 – Tree Based Search

- **Due** *11:59pm Friday 19th April 2024 (Week 7)*
- **Contributes** 30% to your final subject result, subject to moderation if required.
- **Individual**.

This assignment requires you to work individually to get working tree based search algorithms.

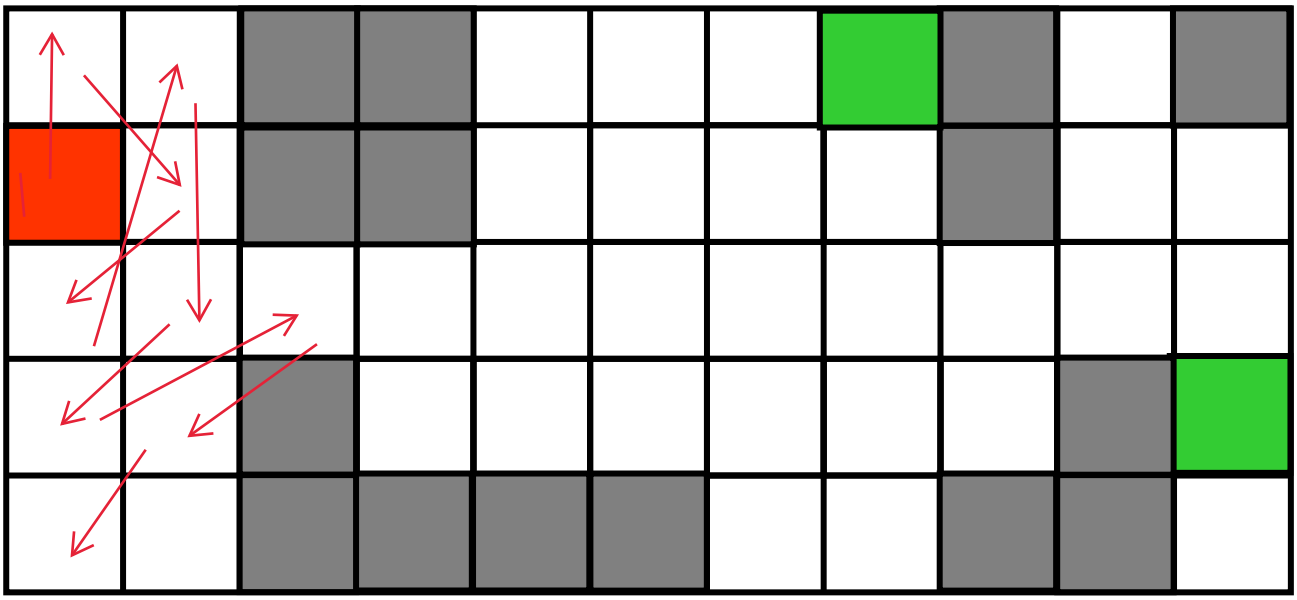
You need to implement tree-based search algorithms in software (from scratch) to search for solutions to the ***Robot Navigation*** problem. Both ***informed*** and ***uninformed*** methods will be required. You will also need to do some self-learning to learn several search methods (not covered in the lectures).

Implementation

You are encouraged to implement your software using Python. If you prefer to implement your assignment in one of the alternative languages Java, C++ or C#, you need to discuss with your tutor to seek their permission. **You must gain permission from the convenor before using anything else.** Assignment work will be tested on a standard **Microsoft Windows 10** system.

The Robot Navigation Problem

In the lectures you have seen the Robot Navigation problem: The environment is an NxM grid (where N > 1 and M > 1) with a number of walls occupying some cells (marked as grey cells). The robot is initially located in one of the empty cells (marked as a red cell) and required to find a path to **visit one of the designated cells of the grid** (marked as green cells). For instance, the following is one possible environment:



Assume that the cells of the grid are located by their coordinates with the leftmost top cell being considered to be at the coordinate (0, 0). A wall is a rectangle whose leftmost top corner occupies a cell (x,y) and whose width (w) and height (h) capture its size. For instance, the above environment can be expressed by the following specification:

```
[5,11]          // The grid has 5 rows and 11 columns
(0,1)           // initial state of the agent – coordinates of the red cell
(7,0) | (10,3)  // goal states for the agent – coordinates of the green cells
(2,0,2,2)       // the square wall has the leftmost top corner occupies cell (2,0) and is 2 cells wide and 2 cell high
(8,0,1,2)
(10,0,1,1)
(2,3,1,2)
(3,4,3,1)
(9,3,1,1)
(8,4,2,1)
```

File Format: The problems are stored in simple text files with the following format:

- First line contains a pair of numbers [N,M] – the number of rows and the number of columns of the grid, enclosed in square brackets.
- Second line contains a pair of numbers (x1,y1)– the coordinates of the current location of the agent, the initial state.
- Third line contains a sequence of pairs of numbers separated by |; these are the coordinates of the goal states: (x_{G1},y_{G1}) | (x_{G2},y_{G2}) | … | (x_{Gn},y_{Gn}), where n ≥ 1. That is, the goal states may have just one goal state, or two goal states, or many goal states.
- The subsequent lines represent the locations of the walls: The tuple (*x,y,w,h*) indicates that the leftmost top corner of the wall occupies cell (*x,y*) with a width of *w* cells and a height of *h* cells.
- We will only be interested in search algorithms. Therefore, you can assume that the problem files will contain valid configurations. For instance, if N=5 and M = 11 then you don't have to worry that the agent is initially located at coordinates (15, 3).

Search Algorithms

The following describe a number of tree based search algorithms. DFS, BFS, GBFS and AS have been covered in the lectures and the tutorials. CUS1 and CUS2 are two algorithms you may learn by yourself (from the textbook, from the Internet or any other sources).

NOTE 1: The objective is to reach one of the green cells.

NOTE 2: When all else is equal, nodes should be expanded according to the following order: the agent should try to move UP before attempting LEFT, before attempting DOWN, before attempting RIGHT, in that order. Furthermore, **when all else is equal**, the two nodes N_1 and N_2 on two different branches of the search tree should be expanded according to the chronological order: if node N_1 is added BEFORE node N_2 then N_1 is expanded BEFORE node N_2.

Search Strategy	Description	Method
Uninformed		
depth-first search	Select one option, try it, go back when there are no more options	DFS
breadth-first search	Expand all options one level at a time	BFS
Informed		
greedy best-first	Use only the cost to reach the goal from the current node to evaluate the node	GBFS
A* ("A Star")	Use both the cost to reach the goal from the current node and the cost to reach this node to evaluate the node	AS
Custom		
Your search strategy 1	An **uninformed** method to find a path to reach the goal. For this problem, the uniform-cost search (UCS) is not an acceptable custom search strategy.	CUS1
Your search strategy 2	An **informed** method to find a shortest path (with least moves) to reach the goal.	CUS2

Command Line Operation

Your program needs to operate from a DOS command-line interface to support batch testing. A DOS command-line interface can be brought up in Windows 7/8/10 by typing **cmd** into the search box at the ***Start*** button. However, please ensure that your program works on Windows 10 because we will be testing your program on Windows 10. This can be accomplished with a simple DOS .bat (batch) file if needed. Below are the three different arguments formats you need to support. Note the unique argument count for each.

```
C:\Assignments> search <filename> <method>
```

where **`search`** is your .exe file or a .bat (batch) file that calls your program with the parameters.

(if you program in Python, we can accept Python scripts and execute your scripts using the following command from the CLI:

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

When a goal can be reached, standard output needs to be in the following format:

```
filename method
goal number_of_nodes
path
```

where **goal** is the goal node your search method reached and **number_of_nodes** is the number of nodes your program has created when perform this search strategy, and **path** is a sequence of moves in the solution that brings you from the start-configuration to the end-configuration. Line breaks are ignored (so use them if you want to).

For instance, the following screenshot shows an example of running the program:

```
PS C:\Users\bvo\OneDrive - Swinburne University\Teaching\2024\AI\Assignments\A1\Sol2> python .\search.py .\RobotNav-test.txt BFS
3.10.11 (tags/v3.10.11:7d4cc5a, Apr  5 2023, 00:38:17) [MSC v.1929 64 bit (AMD64)]
.\RobotNav-test.txt BFS
<Node (7, 0)> 34
['down', 'right', 'right', 'right', 'up', 'up', 'right', 'right', 'right']
PS C:\Users\bvo\OneDrive - Swinburne University\Teaching\2024\AI\Assignments\A1\Sol2>
```

(**NOTE:** If you use the implementation from **aima-python**, you'll see a different **number_of_nodes** from the above example. My implementation is to ensure consistency across ALL search methods.)

When a goal cannot be reached, standard output needs to be in the following format:

```
filename method
No goal is reachable; number_of_nodes
```

For instance, the following screenshot shows an example of running the program on a test case in which the goal is not reachable:

```
PS C:\Users\bvo\OneDrive - Swinburne University\Teaching\2024\AI\Assignments\A1\Sol2> python .\search.py .\n1.txt BFS
3.10.11 (tags/v3.10.11:7d4cc5a, Apr  5 2023, 00:38:17) [MSC v.1929 64 bit (AMD64)]
.\n1.txt BFS
No goal is reachable; 13
PS C:\Users\bvo\OneDrive - Swinburne University\Teaching\2024\AI\Assignments\A1\Sol2>
```

Report file

You must also include a report which has to be either in Microsoft Word or in PDF whose name is your student ID (for example, 1234567.PDF) containing your report. The aim of this report is for you to summarise your understanding of the problem, to introduce the search algorithms used in your assignment (including the standard ones), to discuss how you implemented them. You'll also need to compare and discuss your strategies, in particular the custom strategies developed, **using data obtained by running your software**.

Report Details: The report must be between 10 and 14 pages (excluding cover page and TOC).

- **Cover page:** including your student details (i.e., your full name and student ID).
- **Table of contents (TOC)**.
- **Instructions:** Basic instructions of how to use your program. You can also include a **note** containing anything else you want to tell the marker, such as how to use the GUI version of your program, and something particular about your implementation.
- **Introduction:** Introduce the *Robot navigation Problem*, basic graph and tree concepts and other related terminology that may be needed later in your report. (*Hint:* using a glossary)
- **Search Algorithms:** Present and discuss the qualities of the search algorithms used in your assignment. Which algorithms are better and why? **Use data collected to support your points.**
- **Implementation:** Briefly present how each search program was implemented. Class diagram and flow charts (pseudo code) are all suitable. Point out and briefly discuss differences in implementation or approach. Note important references.
- **Testing:** Provide an overview of the test cases you have created to test your program (either manually or automatically). Report the results of testing your program.
- **Features/Bugs/Missing:** Include a list of the features you have implemented. Clearly state if a required feature has not been implemented. Failure to do this will result in penalties. Include a list of any known bugs. Also, anything else you want to tell the marker, such as how to use the GUI version of your program, and something particular about your implementation.
- **Research (if applicable):** If you managed to do some additional research to improve the program in some ways (see a number of ideas for research initiatives), please report it here.

- **Conclusion:** Conclude with a discussion about the best type of search algorithm you would use for this type of problem. Include thoughts about how you could improve performance.
- **Acknowledgements/Resources:** Include in your report a list of the resources you have used to create your work. A simple list of URL's is not enough. Include with each entry a basic description of how the person or website assisted you in your work.
- **References:** Cite the sources you referred to in your Assignment (implementation, report, etc.)

Tips:

- All figures and tables need to be properly captioned with sensible descriptions.
- Report presentation should include header/footer information (pages numbers, etc.)

Marking Scheme & Submission

You must submit your work via the online assignment collection system ESP https://esp.swin.edu.au/

Create a single zip file with your code and a working version of your program. Do not use deep folder hierarchies. Do not include the data files (we have them☺). The upload limit to ESP will be 10 MB. Please consult early if you have a large binary/package for some.

Standard late penalties apply - 10% for each day late, more than 5 days late is 0%.

Marking Scheme

Requirements (or equivalent sections)	***Mark***
If you get Depth-First Search (DFS) work well	**12**
If you get A* (AS) work well	**10**
For each of the 4 methods Breadth-First Search (BFS), greedy best-first search (GBFS), CUS1 and CUS2, if you can get it work well	**8(x4)**
Testing: At least 10 test cases have been created to cover different problem scenarios. Test results have been checked and documented.	**10**
Report: Clear and provide sufficient information about your programs and your algorithm/solution.	**16**
Research: If you show some initiatives in researching about the problem and solutions, or carrying out extensive tests to provide interesting data about the algorithms, or getting some clever optimization, etc. with a well-written **Research** section in the report to discuss and demonstrate these initiatives	**20**
Total	**100**
You need to follow good programming practice (e.g., well-designed, well-structure codes with clear and helpful comments). Failure to do so get penalty.	**Up to -10**
Failure to provide in class updates of the assignment progress to your tutor	**Up to -40**

Ideas for research initiatives

- Can you modify your program so that the robot will visit ALL green cells with the SHORTEST path? Please include in your report the challenges you had to overcome to address this requirement and how you solved them.
- Can you extend your program to allow the robot to have four additional actions **jump_up**(*n*), **jump_down**(*n*), **jump_left**(*n*), and **jump_right**(*n*); where *n* is the number of squares the robot jumps to (i.e., when *n* = 1 then **jump_x**(1) is the same as moving to the direction **x**). The action **jump_x**(*n*) allows the robot to jump over the obstacles with the exponential cost: The cost of **jump_x**(*n*) should be: $2^{(n-1)}$. For example, the cost of **jump_x**(4) is 8. Can you compare the optimal solutions when the robot is allowed to use the actions **jump_x**() and when it is not allowed to?
- In addition to the mandatory command-line-based user interface, can you build a GUI to display the environment and how the algorithm is trying to find the solution? This option should also include a visualizer to show the changes happening to the search tree.

(**NOTE:** If you choose one of the above three options, please make sure that this extension comes with a command-line argument so that the default option of your program is still for the original problem files we will use to test your program.)

- Can you produce comprehensive test suites to ensure that as many bugs as possible can be caught? Can you automate the test case generation? Your test automation technique should also include data collection and report generation.

Note: You don't have to realise too many ideas ☺ Choose one idea and execute it very well and write a clear **Research** section for the report, then you can get the 20 marks awarded for doing a research initiative.