Assignment 1 – Tree Based Search

- **Due** *4:30pm Friday 21st April 2023 (Week 7)*
- **Contributes** 30% to your final subject result, subject to moderation if required.
- **Individual**.

For this assignment, you can choose either **Option A** or **Option B**:

1. For **Option A**, we would like to solve the NxM-Puzzle problem. You will be given a Java code base allowing you to search for solutions of the NxN problem and your job is to understand how it works and possibly extend this given program. You can only achieve a maximum of 80% if you choose this option. *Note that you can also re-implement the Java code in a language of your choice for this option or you may not want to start with the provided code base and choose to implement everything from scratch yourself.*
2. For **Option B**, we would like to solve the Robot Navigation problem. You will write the whole program from scratch.

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

OPTION A - Summary

We would like to implement tree-based search algorithms in software to search for solutions of the ***NxM-Puzzle*** problem. Both ***informed*** and ***uninformed*** methods will be implemented to allow you to understand how these methods work. **You will be given a Java code base** that implements several search algorithms (BFS and GBFS) to search for solutions of the NxN-Puzzle problem. Your tasks are to:

1. **Gain a good understanding** of how search methods work in this particular problem by generating many different tests cases to see how these problems can be solved;
2. **Extend the given code base** to allow other search algorithms to be implemented, such as **DFS** and **A***;
3. Further extensions are possible:
    a. Can you **extend the given code base** to implement other search methods such as **iterative deepening depth-first search** or **IDA***?
    b. The provided code base is not very efficient (especially the BFS search method), can you find a way to improve its performance? How much does your improved version gain in terms of performance?
    c. The given code base only works for the NxN-Puzzles, can you **extend it to work with any NxM-Puzzles**? Note that if N≠M, the given code base will not work;

Note that, eventually your implementation will be tested on a standard **Microsoft Windows 10** system (i.e., please choose a computer from Swinburne computer labs and put your program there and make sure that it runs without any further modification to the system configuration).

The NxM-Puzzle Problem

In the lectures you have seen the 8-puzzle problem. The **N-Puzzle problem** (or, **sliding puzzle**, **sliding tile puzzle**) is a generalisation of the 8-puzzle problem where N=3, 8, 15, … You are referred to http://en.wikipedia.org/wiki/Sliding_puzzle and http://en.wikipedia.org/wiki/Fifteen_puzzle for informal descriptions of the problem. The 8-puzzle can also be viewed as a 3x3-puzzle as each side of the puzzle grid has 3 cells. As a generalisation of the 3x3-puzzle problem, we can have **NxM-puzzle problems** where **N** and **M** are positive integers and **N** is not necessarily equal to **M** and there is one blank cell. For instance, the following 5x3 puzzle consists of cells labelled by numbers 1..14 on a 5x3 grid and the blank cells labelled by number 0.

1	2	3
5	11	6
4	13	9
8		12
7	10	14

The above configuration will be represented by the following sequence (with the number 0 representing the empty cell):

1 2 3 5 11 6 4 13 9 8 0 12 7 10 14

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

- First line contains a string **N**x**M**: **N representing the number of rows of the puzzle** and **M represents the number of columns of the puzzle**. For example, if the first line is 5x3 then the puzzle has 5 rows and 3 columns.
- The next two lines represent the start-configuration and the end-configuration, respectively, of the puzzle your program is supposed to solve.
- We will only be interested in search algorithms. Therefore, you can assume that the problem files will contain valid configurations. For instance, if NxM=3x4 then you don't have to worry that the problem file will contain a cell labelled by number 12.

Search Algorithms

The following are standard tree-based search algorithms (covered in the lectures):

Search Type	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

Note1: **When all else is equal**, the child nodes of an expanded node *N* should be expanded according to the following order: the agent should try to move the empty cell 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.

You are given a Java code base implementing BFS and GBFS for the **NxN-Puzzle**. You can start learning about search algorithms by playing with this code base. Try to create many test cases to gain an understanding of how search algorithms work. You will find that this code base is NOT perfect. It is quite slow. It does NOT follow the instruction in **Note1** above (see the TODO in the code). Part of your job is to improve this code base.

Command Line Operation

The given program operates DOS command-line interface which can be brought up in Windows 10 by typing ***cmd*** into the search box at the **Start** button. Below is how to run the given program from DOS command line:

```
> nPuzzler <filename> <method>
```

You can try the program with the given file `N-puzzle-test.txt` and the two provided search methods `BFS` and `GBFS`. If you test it with a NxM-puzzle (e.g., the given file `NM-puzzle-test.txt`), it won't work.

Report

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, **using data obtained by running your software**.

Report Details: The report must be between 7 and 10 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 run special features of your assignment.
- **Introduction:** Introduce the *NxM Puzzle 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.
- **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 about your implementation.
- **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.)

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. The upload limit to ESP will be 100 MB. Please consult early if you have a large binary/package for some. You must also include your report in your submission. If you have generated some data files to gain your understanding of these search algorithms AND your report refers to these data files, please include these data files as well.

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 can extend the program and get DFS work well	**10**
If you can extend the program and get A* work well	**10**
If you can extend the program and get **some other informed search method** work well	**10**
If you can extend the program and get it work with NxM-Puzzles OR you can find a way to improve the performance of the program (**you only need to do one of them**).	**15**
Report: Clear and provide sufficient information (e.g., providing a clear example) about your understanding of how tree-based search works. Clear explanation of the algorithms you implement and how you implement them.	**35**
Total	**80**
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**

OPTION B - Summary

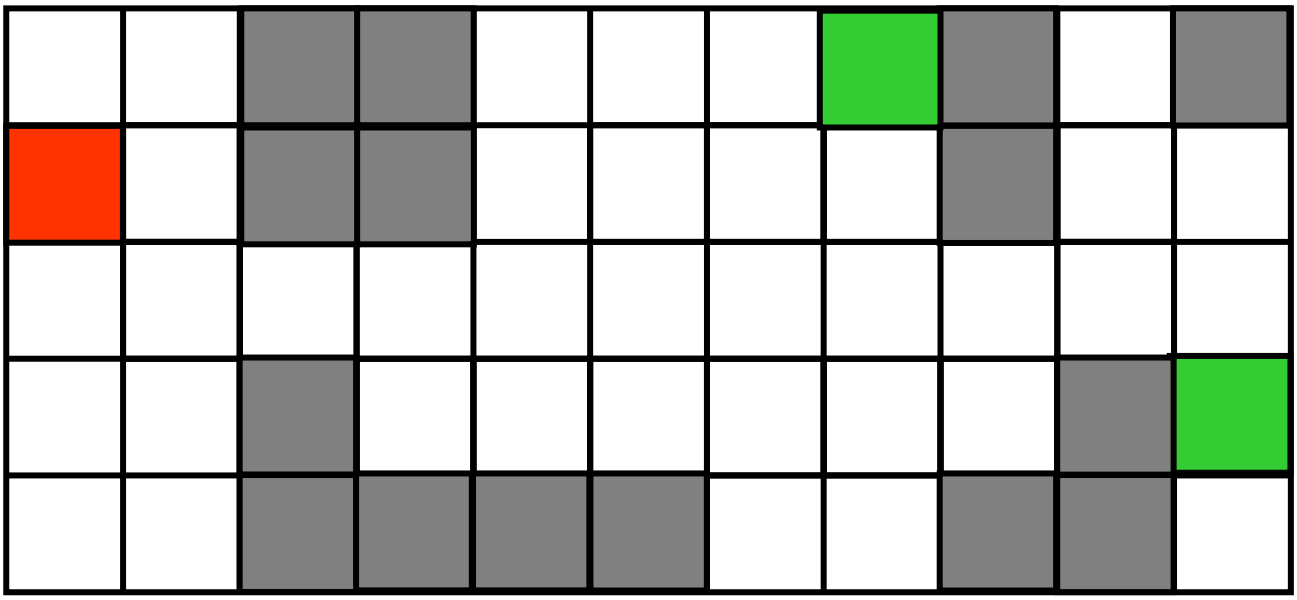
You need to implement tree-based search algorithms in software (from scratch) to search for solutions to the ***Robot Navigation*** problem. Both ***informed*** and ***uninformed*** methods will be required. You will also need to do some self-learning to learn several search methods (not covered in the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

Implementation

You are welcome to implement your software using any of the following languages; Java, Python, or C++/C#. **You must gain permission 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.

- 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).

NOTE1: The objective is to reach one of the green cells.

NOTE2: 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 Type	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.	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.

Standard output needs to be in the following format

```
filename method number_of_nodes
path
```

where **`number_of_nodes`** is the number of nodes in search tree, and **`path`** is either a message "No solution found." (when your program can not find a solution) or 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, a valid (and incomplete) path from the current configuration in the above example could be:

```
> right; down; right; right; right; right; …
```

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 8 and 12 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.
- **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 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) (if you do well, you can get up to 110% for Assignment 1)	***Mark***
If you get A* (AS) work well	**20**
If you get Breadth-First Search (BFS) work well	**15**
For each of the 4 methods depth-first search (DFS), greedy best-first search (GBFS), CUS1 and CUS2, if you can get it work well	**8(x4)**
Report: Clear and provide sufficient information about your programs and your algorithm/solution.	**18**
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 report to demonstrate these initiatives	**15**
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 -50**

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? If you choose this option, 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.
- 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.
- 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 you can get the 15 marks awarded for doing a research initiative.