Assignment 1 – Tree Based Search

- **Due** *4:30pm Thursday 24th April 2014 (Week 7)*
- **Contributes** 20% to your final subject result, subject to moderation if required.
- **Individual**.

Summary

You need to implement tree-based search algorithms in software to search for solutions to the ***NxM*-Puzzle** problem. Both ***informed*** and ***uninformed*** methods will be required. You will also develop your own techniques as a unique comparison. Your methods do not need to be better than the standard tree based measures, but should be your own novel approach.

This assignment requires you to work individually to get working tree based search algorithm for the NxM-Puzzle problem.

Implementation

You are welcome to implement your software using any of the following languages; Java or C/C++/C#. We recommend Java. You must gain permission before using anything else. Assignment work will be tested on a standard Microsoft Windows 7 system.

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 an integer number **representing the number of rows of the puzzle** - the number N of this puzzle.
- Second line also contains an integer number **representing the number of columns of the puzzle** - the number M of this puzzle.
- 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 N=3 and M = 4 then you don't have to worry that the problem file will contain a cell labelled by number 12.

Search Algorithms

You need to implement the following tree based search algorithms

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

A graphical interface is NOT encouraged; your program needs to operate in a simple user-interface form to support batch testing. This can be accomplished with a simple DOS .bat (batch) file if needed. A DOS command-line interface can be brought up in Windows 7 by typing ***cmd*** into the search box at the **Start** button. Below are the three different arguments formats you need to support. Note the unique argument count for each.

```
> 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:

```
> input.txt AS up; up; right; …
```

Readme.txt file

You must include a single readme.txt file with your work with the following details:

- **Student Details:** Your full student names, ids, and your group number (as allocated by ESP).
- **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.
- **Acknowledgements/Resources:** Include in your readme.txt file 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.
- **Notes:** 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.

Marking Scheme & Submission

You must submit your work via the online assignment collection system ESP https://esp.ict.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 one search strategy working to perfection (any of the 6 strategies)	**40**
	For the remaining 5 search strategies, if you get one strategy working to perfection, you'll get 8 marks for that strategy	**40**
	Good programming practice: Well-designed, well-structure codes with clear and helpful comments	**10**
	Readme file: Clear and provide sufficient information about your programs and your algorithm/solution.	**10**
	Total	**100**

Example user interface

An example of how such a program runs in a DOS command-line environment: