COMP 472/6721 **Mini-Project 1** **Fall 2018**

Due date:	October 14, 2018
Late submission:	20% per day
Teams:	Students registered in COMP 472 can do the project individually or in teams of 2. Students registered in COMP 6721 must do the project individually. Teams must submit only 1 copy of the project.
Purpose:	The purpose of this project is to make you practice and experiment with heuristic search.

Heuristic Search

Assume a variation of the 8-puzzle called 11d-puzzle. The 11d-puzzle is identical to the 8-puzzle, except for 2 differences:

1. the board is a 3x4
2. diagonal moves into the empty tile are legal (assume it can be done on a physical board). So we have at most 8 possible moves: UP, UP –RIGHT, RIGHT, DOWN–RIGHT, DOWN, DOWN–LEFT, LEFT, UP–LEFT.

The goal configuration of the 11d-puzzle is:

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

To represent the positions on the board, let us use the letters **a** to **l** as indicated by the superscripts below:

1^a	2^b	6^c	4^d
5^e	9^f	7^g	3^h
0^i	10^j	11^k	8^l

For example, in the figure on the left:
tile #1 is at position **a**, ...
tile #6 is at position **c**, ...
the empty tile (denoted by the 0) is at position **i**...

Since a tile can only move into the empty position, all that is required to represent a move is to record the tile that moved. For example, given the configuration above, the sequence of moves: **`[f g k]`** should be interpreted as:

1. move tile **f** DOWN−LEFT to position **i** (where the empty tile is)
2. (now the empty tile is at position **f**) move tile **g** LEFT to position **f**
3. (now the empty tile is at position **g**) move tile **k** UP to position **g**

Your task:
Implement a solution for the 11d-puzzle using the following 3 search algorithms:

A. Depth-first search (DFS)
B. Best-first search (BFS)
C. Algorithm A* (A*)

Ties between equivalent nodes should be broken in a clockwise direction starting with UP. This means that equivalent moves are ordered this way:

UP > UP –RIGHT > RIGHT > DOWN- RIGHT > DOWN > DOWN –LEFT > LEFT > UP–LEFT
(most preferred moves) *(least preferred moves)*

For BFS and A*, you must experiment with 2 different heuristics *h1* and *h2* of your choice. *h1* and *h2* should both be used with BFS and with A*.

Input:
Your program will read from the console an initial puzzle, where each tile will be represented by a unique integer (0 to 11), where the zero will denote the empty tile. The order of the tiles will follow the alphabetic order of the tile names (**`a b c d e f g h i j k l`**). For example, the puzzle:

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

will be represented as: **`1 0 3 7 5 2 6 4 9 10 11 8`**

Output:
Given an initial puzzle (from the console), your program must output its results in the 5 text files below:

1. **`puzzleDFS.txt`:**
   A trace of the puzzle configuration after each move on the final solution found by depth-first search. First, you display the initial configuration preceded by a 0 (zero), then for each move in the solution path, you should display the move followed by the new configuration in list format, one puzzle per line. For example, if your final solution path is:
   `[c d h l k g c d b f k l]`
   Then **`puzzleDFS.txt`** should contain:

```
0 [1, 0, 3, 7, 5, 2, 6, 4, 9, 10, 11, 8] // 0 then the initial configuration
c [1, 3, 0, 7, 5, 2, 6, 4, 9, 10, 11, 8] // move c, then configuration after move c
d [1, 3, 7, 0, 5, 2, 6, 4, 9, 10, 11, 8] // move d, then configuration after move d
h [1, 3, 7, 4, 5, 2, 6, 0, 9, 10, 11, 8] // ...
l [1, 3, 7, 4, 5, 2, 6, 8, 9, 10, 11, 0]
k [1, 3, 7, 4, 5, 2, 6, 8, 9, 10, 0, 11]
g [1, 3, 7, 4, 5, 2, 0, 8, 9, 10, 6, 11]
c [1, 3, 0, 4, 5, 2, 7, 8, 9, 10, 6, 11]
d [1, 3, 4, 0, 5, 2, 7, 8, 9, 10, 6, 11]
b [1, 0, 3, 4, 5, 2, 7, 8, 9, 10, 6, 11]
f [1, 2, 3, 4, 5, 0, 7, 8, 9, 10, 6, 11]
k [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0, 11]
l [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 0]
```

   Please follow the format indicated above.

2. **`puzzleBFS-h1.txt`:** same as above but for BFS with heuristic *h1*.
3. **`puzzleBFS-h2.txt`:** same as above but for BFS with heuristic *h2*.
4. **`puzzleAs-h1.txt`:** same as above but for A* with heuristic *h1*.
5. **`puzzleAs-h2.txt`:** same as above but for A* with heuristic *h2*.

Programming Environment:
You can use Java, C, C++, Python or Prolog. If you wish to use another language, please check with me first.

Experimentations:
Note that this is a mini-project, not an assignment. The difference is that it is open-ended and in addition to the required work stated above, you are expected to be creative, perform additional experimentations and analyse the results of your experimentations. Examples of experimentations include trying different heuristics, modifying the size or rules of the puzzle,...

Deliverables:
The submission of the project will consist of 3 deliverables:

1. The Code
2. A Report
3. A Demo

The Code:
Submit all files necessary to run your code in addition to a `README.txt` which will contain specific and complete instructions how to run your program on the desktops in the computer labs. If the instructions in your readme file do not work or are incomplete, you will not be given the benefit of the doubt.

The Report:
Your deliverable must be accompanied by a written report (~3-5pages, about 12pts single space) that:

- Describes and justifies your heuristics *h1* and *h2*;
- Describes any difficulties that you have encountered and how you addressed them;
- Analyses the results of your experiments. In particular, you must compare and contrast the use of *h1* and *h2* and the search algorithms.

Please note that your report must be analytical. This means that in addition to stating the facts (ex. how many moves does a search algorithm yield), you should also analyse them (e.g. explain why something behaves this or that way, in which case something would be better than another...).

Your report should have a reference section (not included in the page count) that properly cites all relevant resources that you have consulted (books, Web sites...), even if it was just to inspire you. Failure to properly cite your references constitutes plagiarism and will be reported.

The report must be in PDF format and must be called:

- `472_Report1_StudentID1_StudentID2.pdf` (for team work) or
- `472_Report1_StudentID.pdf` (for individual work in COMP 472) or
- `6721_Report1_StudentID.pdf` (in COMP 6721)

Students in COMP 6721 (especially thesis students) are strongly encouraged to write their report in Latex (see https://www.overleaf.com).

The Demo:
All submissions will be demoed during the lab time on the lab machines. You will not be able to demo on your laptop. Regardless of the demo time, you will demo the program that was uploaded as the official submission on or before the due date. The schedule of the demos will be posted on Moodle. No special preparation is necessary for the demo (no slides or prepared speech). Your TA will ask you questions on your code, and you will have to answer him/her.

Evaluation Scheme:

	COMP 472	COMP 6721
Implementation (functionality, design, programming style, ...)	35%	30%
Demo (clear answers to questions, knowledge of the program, ...)	15%	15%
Report (clarity and conciseness, depth of the analysis, presentation, grammar,...)	35%	30%
Experimentations (thoroughness, originality,...)	15%	25%
Total	100%	100%

Submission:
The code and the report must be handed-in electronically by midnight on the due date.

1. Create one zip file, containing all necessary files to run your program, the README.txt file and your report. Remember that your report must be in PDF and must be named as indicated in the section "The Report" above
2. Name your zip file:
    - **`472_Project1_StudentID1_StudentID2.zip`** (for team work) or
    - **`472_Project1_StudentID.zip`** (for individual work in COMP 472) or
    - **`6721_Project1_StudentID.zip`** (in COMP 6721)
3. Upload your zip file at: https://fis.encs.concordia.ca/eas/ as **`project1`**.

In addition, please bring a paper copy of your report to class on Monday October 15.

Have fun!