CSC447/547 Artificial Intelligence - Spring 2016
Programming Assignment #2: 8-Puzzle

Introduction
The 8-puzzle is a well-known "toy" problem in AI, often used to illustrate search concepts. To solve the puzzle, you must slide 8 tiles about a 3x3 grid to reach a goal state. The 8-puzzle provides an excellent introduction to state space search.

Problem
Write a Lisp program to solve the 8-puzzle using BFS (breadth first search), DFID (depth first iterated deepening), and A*. The start position may be specified in a puzzle file, in the Lisp function call, or interactively. After solving the puzzle with each search algorithm, print out a nicely formatted list of positions, leading from the start state to the goal state. Also print out the number of moves required to reach the goal state, and the number of nodes generated and expanded (i.e., the number placed on the *OPEN* and *CLOSED* lists, respectively). This will provide a rough metric for search algorithm efficiency.

Implementation
Your Lisp program (*8puzzle.lsp*) may be run in three different ways:

Command-line usage: `clisp 8puzzle.lsp puzzlefile`
The *puzzlefile* contains an 8-puzzle start position, consisting of 9 digits separated by white space, in row-major order. The digits 1-8 represent the 8 tiles, and 0 represents the blank.

Inside CLISP:
```
(load '8puzzle)
(8puzzle [puzzlelist])
```
The optional *puzzlelist* argument contains an 8-puzzle start position, stored in row-major order as a list of 9 elements. The digits 1-8 represent the 8 tiles, and 0 represents the blank. If the *puzzlelist* argument is not supplied, prompt the used to enter the start position as 9 digits separated by white space (*not* as a list).

Search algorithms
BFS and DFID are fairly self-explanatory. Implement at least two admissible and one inadmissible A* heuristics that are designed for the 8-puzzle (e.g., number of tiles out of place). More informed heuristics that search more efficiently will be awarded greater credit.

Format your solution output as shown in the sample session below. You should display several problem states across the page (not just vertically).

Extra credit
Generalize the 8-puzzle to the N-puzzle, where N may be 3^2-1=8, 4^2-1=15, 5^2-1=24, etc. You may need to add the puzzle size to the command-line arguments and puzzle file format.

Notes

- Certain 8-puzzle positions are unsolvable. A Lisp function to determine the solvability of a given position is available on the course website (*solvable.lsp*). *(solvable L)* returns *T* if *L* represents a solvable position, *NIL* otherwise. *L* is a 9-element list such as *(1 2 3 8 0 4 7 6 5)*.

- When you are finished writing, testing, and debugging your program, submit your source code in a *zip* or *tar* archive using the *Submit It!* link on the MCS Department Website. The submit program is accessed via the MCS Web page (http://www.mcs.sdsmt.edu), by selecting the list item on the left entitled "Submit it!". Usage is self-explanatory: enter your name, choose the instructor and click "Select Instructor", choose the appropriate course, browse to the filename you wish to submit, and click "Upload".

- Submit your program by the due date (Sunday March 27) in order to receive credit for this assignment. Late programs will not be accepted for partial credit unless prior arrangements have been made with the instructor. If you have any problems with the submit program, report them to your instructor and submit your program by email instead.

- To receive full credit, your code must be readable, modular, nicely formatted, and adequately documented, as well as complete and correct. It must build and run successfully under the current CLISP interpreter (v.2.49). If your program does not run correctly, indicate why. This will make it easier to give you partial credit.

- You must work in teams of three students on this assignment. Teams should make one joint submission, not individual submissions for each team member. Each team member should complete and submit a teamwork evaluation form as well, assessing distribution of workload and team interactions. Teams for this assignment are:

    1. Johnathan Ackerman + Samuel Carroll + Benjamin Kaiser
    2. Hannah Aker + Derek Lane + Savoy Schuler
    3. Jason Anderson + John Mangold + Jacob St.Amand
    4. Stephanie Athow + Luke Meyer + Alex Nienhueser
    5. Marcus Berger + Matthew Dyke + Cassidy Vollmer
    6. Allison Bodvig + Steven Huerta + Christian Sieh
    7. Julian Brackins + Scott Carda + Leif Torgersen
    8. Dylan Geyer + Daniel Nix + Mackenzie Smith
    9. Marcus Haberling + Alex Herman + Matthew Rames

Sample Puzzle Configurations

```
Goal         Easy         Medium         Hard         Worst
1 2 3        1 3 4        2 8 1          2 8 1        5 6 7
8 0 4        8 6 2        0 4 3          4 6 3        4 0 8
7 6 5        7 0 5        7 6 5          0 7 5        3 2 1
```

Sample LISP session

```
BFS graph search
----------------
Solution found in 7 moves
350 nodes generated (219 distinct nodes), 125 nodes expanded

    2 8 3          2 8 3          2 8 3          2 8 3
    1 6      ->    1 6 4    ->    1 6 4    ->    1   4    ->
    7 5 4          7 5            7   5          7 6 5

    2   3            2 3          1 2 3          1 2 3
    1 8 4    ->    1 8 4    ->      8 4    ->    8   4
    7 6 5          7 6 5          7 6 5          7 6 5

DFID graph search
-----------------
Solution found in 7 moves
997 nodes generated (633 distinct nodes), 366 nodes expanded

    2 8 3          2 8 3          2 8 3          2 8 3
    1 6      ->    1 6 4    ->    1 6 4    ->    1   4    ->
    7 5 4          7 5            7   5          7 6 5

    2   3            2 3          1 2 3          1 2 3
    1 8 4    ->    1 8 4    ->      8 4    ->    8   4
    7 6 5          7 6 5          7 6 5          7 6 5

A* graph search (heuristic: number out of place)
---------------
Solution found in 7 moves
31 nodes generated (22 distinct nodes), 10 nodes expanded

    2 8 3          2 8 3          2 8 3          2 8 3
    1 6      ->    1 6 4    ->    1 6 4    ->    1   4    ->
    7 5 4          7 5            7   5          7 6 5

    2   3            2 3          1 2 3          1 2 3
    1 8 4    ->    1 8 4    ->      8 4    ->    8   4
    7 6 5          7 6 5          7 6 5          7 6 5
```