CS 461 – ARTIFICIAL INTELLIGENCE

HOMEWORK #2 (5% or 10 points)

Assigned: Wed Oct 14, 2020

Due: Wed Oct 28, 2020 **2:00 pm**

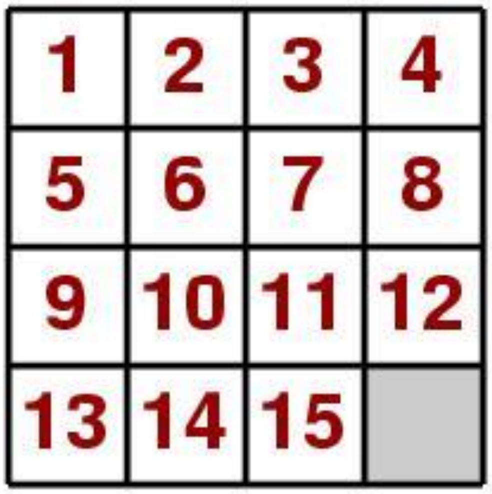
Do not forget to indicate the group members submitting the homework (at most 5 names). Submit your homework according to your TAs' guidelines.

Feel free to use any programming language as long as any of you can, if requested, give a demo using your notebook computer.

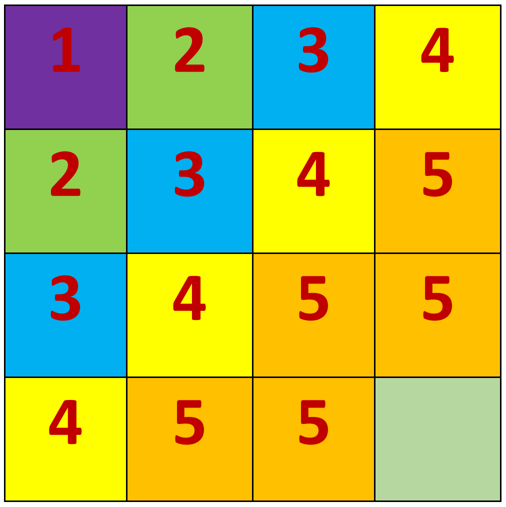
The usual late submission policy applies.

PROBLEM

In this homework, you'll solve a variant of the 15-puzzle, something I would like to call the E15-puzzle because it is easier to solve. (It is well-known that the 15-puzzle is difficult to solve; Wikipedia notes that lengths of optimal solutions can be as large as 80 moves.) The first figure below is the goal state for the (classical) 15-puzzle. The second figure below is the goal state for the E15-puzzle. It is clear that due to the repeated tiles, the E15-puzzle should take fewer moves to solve in general.



15-puzzle (goal state)



E15-puzzle (goal state)

(colors are just to emphasize the duplicate tiles)

Now implement the A* search algorithm (no other algorithm is acceptable) and use it to solve a given instance of the E15-puzzle <u>optimally</u>, that is, with a minimum number of moves.

(Winston covers A* in chapter 5. It is up to you to add the dynamic programming part; no points will be deducted for lack of it. On the other hand, you must definitely implement loop-checking.)

As you know, A* requires admissible heuristics. The following are used in solving the 15-puzzle and with some modifications (if necessary) should work for the E15-puzzle, too. The admissibility proofs given below within parentheses are for the 15-puzzle though.

- h1: number of misplaced tiles

  (h1 is an admissible heuristic for the 15-puzzle, since every tile that is out of position must be moved at least once.)

- h2: sum of Manhattan distances of the tiles from their proper positions

  (h2 is an admissible heuristic for 15-puzzle, since in every move, one tile can only move closer to its goal by one step and the Manhattan distance is never greater than the number of steps required to move a tile to its proper position.)

Here's what you should do:

- Write a 'puzzle generator' first. Starting from the goal state of the E15-puzzle (cf. preceding figure), the generator returns a reasonably garbled initial state (S) by randomly shuffling the puzzle 10 times. This means that starting with the goal state, you 'move' the blank tile (up, down, left, or right) 10 times in succession, each move being decided by a call to a pseudo-random number generator. **Notice that the puzzle generator thus guarantees that this (initial state) S will be solvable.**
- Run your generator as many times as necessary to obtain a dozen distinct initial states of the E15-puzzle. For the benefit of a reader (e.g., our TAs), clearly print these states and give them names, viz. S1, S2, …, S12.
- Solve each Si, where i is between 1 and 12, via A*. In your A* implementation, you'll use admissible heuristics similar to h1 or h2 (see presently).

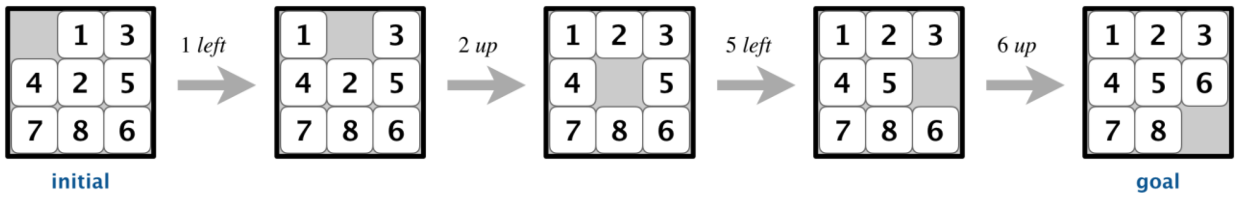
A submission consists of:

- Listing of your program code with a clear indication of what parts, if any, of it are borrowed from elsewhere. (It is best if you write your own original code because points will definitely be deducted for code heavily borrowed from another source.) To explain your program, you should include comments. It is crucial that you give, in the beginning of your program and in block comments, a characterization of the heuristic you're using and an admissibility proof for it.

(HINT: Do the duplicate tiles in the E15-puzzle pose a problem re: the admissibility of h1 or h2?)

- The list of a dozen initial states.
- For two of your initial states (you are free to pick them arbitrarily), a graphical <u>solution sequence</u> starting with the initial state and ending with the goal, fully tracing the moves of your program. **See the part typeset in dark red below for details.**
- A graph showing the queue size for the <u>remaining</u> 10 states. The x-axis of your graph should have the names of the states. The y-axis should indicate the maximum number of paths in the queue at any time for a particular state.

It is of utmost importance that a solution sequence is shown graphically. The drawings need not be sophisticated or in color, etc. Just to give you an example of what I've in mind, consider the following solution sequence for the 8-puzzle:



Thus, I expect two drawings like this from you but, needless to say, for the E15-puzzle!