CS 461 – ARTIFICIAL INTELLIGENCE

HOMEWORK #4 (6.5% or 13 points)

Assigned: Wed Mar 17, 2021

Due: Wed Mar 31, 2021 **2:00 pm**

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

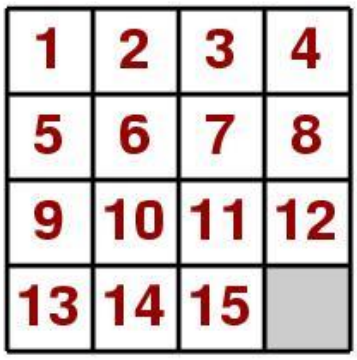
Feel free to use any programming language as long as any of you can, if requested, give a demo using their notebook computer.

The usual late submission policy applies.

PROBLEM

In this homework --- which you'll notice is similar to Homework #3 in obvious ways --- you'll solve some instances of the 15-puzzle. (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 figure below depicts the goal state for the 15-puzzle.



15-puzzle (goal state)

First, implement the A* algorithm, viz. branch-and-bound search, with a lower bound estimate of remaining distance (a.k.a. admissible heuristics), combined with the dynamic programming principle. (No other approach is acceptable.)

Then use it to solve a given instance of the 15-puzzle optimally, that is, with a minimum number of moves. (You are required to add Dynamic Programming; otherwise, you'll receive no credit whatsoever. Needless to say, you must implement loop-checking too.)

Here's what you should do:

- Write a 'puzzle generator' first. Starting from the goal state of the 15-puzzle (cf. preceding figure), the generator returns a garbled initial state (S) by randomly shuffling the puzzle. This means that starting with the goal state, you 'move' the blank tile (up, down, left, or right) numerous 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.** (N.B. There's a delicate balance that needs to be observed here: shuffling too much may lead to initial states which are demanding! See presently.)
- Run your generator as many times as necessary to obtain 10 distinct initial states of the 15-puzzle. For the benefit of a reader (e.g., our TAs), print these states and give them names, viz. S1, S2, …, S10.
- Solve each Si, where i = 1, …, 10, by running your program (A*, as described above). I suggest that you avoid those Si's which cause trouble (i.e., deplete your hardware resources). Therefore, if a particular Si seems to have a very long solution, just delete it from your set of 10 states, and create and add a new Si which is hopefully less problematic. You should repeat this process until each and every Si can be handled by your program comfortably. (It is okay for a program to spend minutes, even hours, to solve a particular instance. We are not interested how long your program took to compute the solution. So, you'll be the judge, based on the strength of your computing resources.)

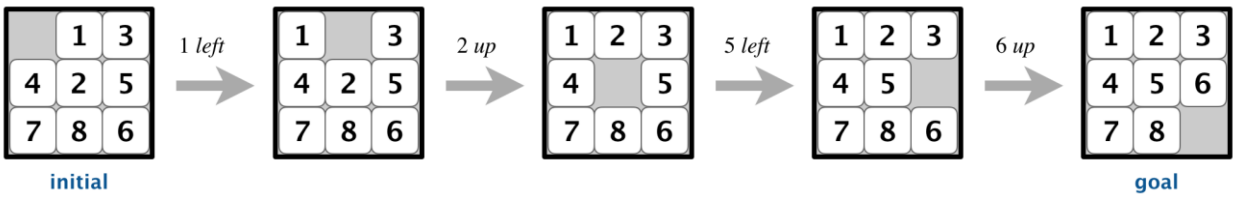
A submission consists of:

- 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.) Do not forget to explain in detail, using block comments, (i) which admissible heuristics you've used and (ii) how you've implemented DP. **You are free to use any heuristics as long as**

you give a proof of admissibility for it, again in block comments. This proof need not be original with you. You can just rewrite someone else's proof in your own words and cite the original source.

- **(1 pt)** A printout of 10 initial states.
- **(3 pts)** A simple graph. The x-axis of your graph should have the names of the states. The y-axis should indicate the length of the shortest solution for each state. **Do not forget to show, on your graph, the average (over 10 instances) solution length as a number correct to 3 decimal places. The group whose average solution length is the largest will receive a BONUS of 3 points.**
- **(6 pts)** For 2 of your initial states (you are free to pick them arbitrarily), a graphical solution sequence starting with the initial state and ending with the goal, fully tracing the moves of your program. **See the part typeset in red below for details.**
- **(3 pts)** Quality of your software.

It is of utmost importance that your 2 solution sequences are 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 2 drawings like this from you but, needless to say, for the 15-puzzle!