AI Lab

Assignment 2: *SEARCH*

Note:

1. From here onwards, any code that you write will be helpful in the examinations. Please protect your codes and do not share your codes. The TAs/portal do a random check on plagiarism. If any part of the code is copied, even if it the standard code of a search, the TA will regard it as copied. We are not following the policy of the last year where copying the base codes was allowed. For the same reasons it may be risky to take codes from online repository or your seniors' repository.
2. Any form of copying will be attract negative marks and penalties
3. You can code inside lab as well as outside lab after the normal working hours, however the code should be your own
4. Any magnitude of verbal discussion outside lab is allowed. No codes must be exchanged
5. Please protect your own codes. Copying will more severely affect the person whose codes are taken.
6. The TAs/portal itself will check for plagiarism.
7. If your performance is very good during the semester and poor in the mid-sem and end-sem, it would be assumed that you have copied and all marks would be zeroed.
8. In the exams you will be allowed to carry your own codes, written apriori, but not of anybody else.

In this question you'll be solving the n-puzzle problem. A board of size $n \times n$ exists with tiles numbered from 1 to n^2-1. One of the tiles is numbered as blank (-1). Before going forward with this question make sure that you have identified the state representation scheme, made a function for computing the actions and legal actions, identified step and path costs. Make sure the solutions you make are generalized enough so that you may extend them to make the solutions of the future lab sessions and examinations. The preference of actions is left (most preferred), up, right and down (least preferred).

1. Solve the problem using Breadth First Search.

Input: The first line of input is the number of test cases. Each test case starts with a single input, the number n. Then there are exactly n lines, with n integers each, denoting the current state of the board. Blank is represented by -1.

Output: For each test case, print a variable number of lines, each with n^2 integers representing the solution path, starting from the source state to the goal state. Each line prints the board array in a row major format.

2. Solve a simpler problem, which is to take only the blank from its current position to its final position. What state variables do you need? Print only the path of the blank in X-Y

coordinates, one point per line. (0,0) is assumed to be the origin. X axis is the vertical axis. Y axis is the horizontal axis.

3. Solve another simpler problem, wherein the blank needs to visit the 4 corners (0,0), (0,*n*-1), (*n*-1,*n*-1) and (*n*-1,0) in the same order. Print only the path of the blank in X-Y coordinates, one point per line.
4. Solve another simpler problem, wherein the blank needs to visit the 4 corners (0,0), (0,*n*-1), (*n*-1,*n*-1) and (*n*-1,0) in any order. Print only the path of the blank in X-Y coordinates, one point per line.
5. Modify the original puzzle such that the game ends with a win in either of the configurations formed by rotating the board in the winning state at angles of 90 degrees. That is the winning states for an 8-puzzle problem are:

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

6. Modify the original puzzle with a new related puzzle game which has 2 blank pieces in place of 1. Only one blank can be operated at a time. All actions corresponding to a blank at an earlier row (primary) and an earlier column (secondary) get a preference.
7. Modify the program above with a new related puzzle game which has 2 blank pieces in place of 1. One or both blanks can be operated at a time, which constitutes a unit move. All actions corresponding to a blank at an earlier row (primary) and an earlier column (secondary) get a preference.
8. Solve the original puzzle using Depth First Search. For each test case print only whether the goal is reachable from the source (1) or not (0).
9. Solve the original puzzle using Iterative Deepening Algorithm.