CS 361: Artificial Intelligence

Course Project

Project Details

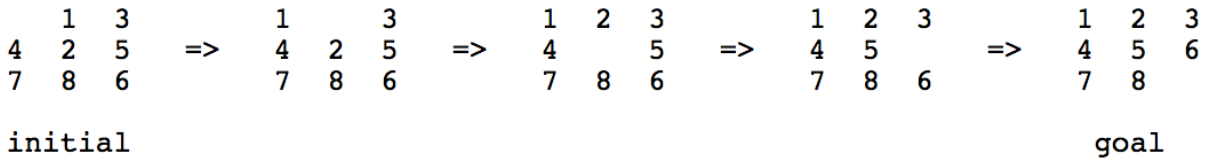
Team Size: 4
Project Deadline: Wednesday April 18th, 2018.
Total Points: 10 points
Number of Problems: 2
Programming Language: Python (other programming languages are not allowed)
Deliverables (for each problem):

1. Source Code on GitHub.
2. README file that includes instructions on how to run the code.
3. Presentation on the day of delivery.

Submit Your Team:
Form a team and send your names to abdelrahman@aun.edu.eg with the subject "**CS361: Artificial Intelligence Course Project Team**" by Wednesday April 4th. Write the email subject as in above (any different subjects will be moved to trash automatically). If you fail to submit your team names by that time, your project will not be graded.

Problem #1 (5 pts.)

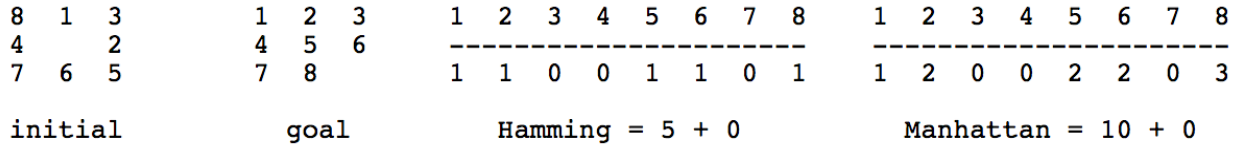
The problem. The 8-puzzle problem is a puzzle invented and popularized by Noyes Palmer Chapman in the 1870s. It is played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square. Your goal is to rearrange the blocks so that they are in order. You are permitted to slide blocks horizontally or vertically into the blank square. The following shows a sequence of legal moves from an initial board position (left) to the goal position (right).



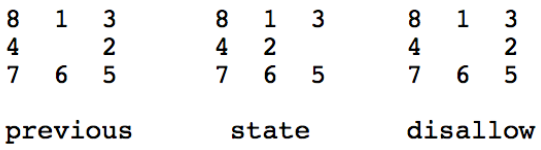
Best-first search. We now describe an algorithmic solution to the problem that illustrates a general artificial intelligence methodology known as the A* search algorithm. We define a *state* of the game to be the board position, the number of moves made to reach the board position, and the previous state. We consider two priority functions:

- *Hamming priority function.* The number of blocks in the wrong position, plus the number of moves made so far to get to the state. Intuitively, a state with a small number of blocks in the wrong position is close to the goal state, and we prefer a state that have been reached using a small number of moves.
- *Manhattan priority function.* The sum of the distances (sum of the vertical and horizontal distance) from the blocks to their goal positions, plus the number of moves made so far to get to the state.

For example, the Hamming and Manhattan priorities of the initial state below are 5 and 10, respectively.



A critical optimization. After implementing best-first search, you will notice one annoying feature: states corresponding to the same board position are explored many times. To prevent unnecessary exploration of useless states, when considering the neighbors of a state, don't move the neighbor if its board position is the same as a previous state.



Your Task:
Write a program **solver.py** that reads the initial board from a file and prints to standard output a sequence of board positions that solves the puzzle in the fewest number of moves. Also print out the total number of moves.
0 represents the blank tile.

Example:

puzzle.txt

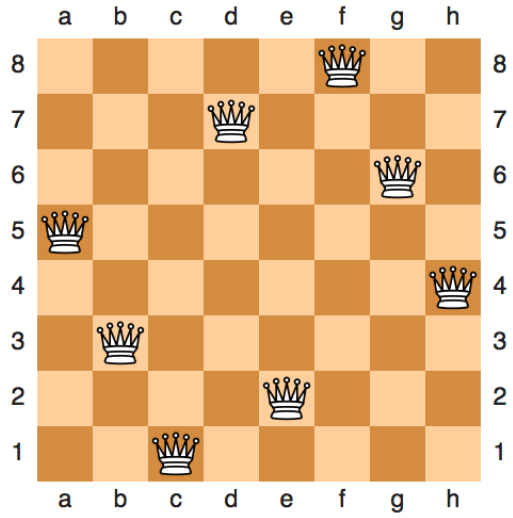
```
0  1  3
4  2  5
7  8  6
```

python solver.py puzzle.txt

```
    1  3
 4  2  5
 7  8  6

 1     3
 4  2  5
 7  8  6

 1  2  3
 4     5
 7  8  6

 1  2  3
 4  5
 7  8  6

 1  2  3
 4  5  6
 7  8

 Minimum number of moves = 4
```

Problem #2 (5 pts.)

The **eight queens puzzle** is the problem of placing eight chess queens on an 8×8 chessboard so that no two queens threaten each other. Thus, a solution requires that no two queens share the same row, column, or diagonal.



This problem can be solved by searching for a solution. The **initial state** is given by the empty chess board. Placing a queen on the board represents an action in the search problem. A goal state is a configuration where none of the queens attacks any of the others. Note that every goal state is reached after exactly 8 actions.

This formulation as a search problem can be improved when we realize that, in any solution, there must be exactly one queen in each of the columns. Thus, the possible actions can be restricted to placing a queen in the next column that does not yet contain a queen. This reduces the branching factor from (initially) 64 to 8.

Furthermore, we need only consider those rows in the next column that are not already attacked by a queen that was previously on the board. This is because the placing of further queens on the board can never remove the mutual attack and turn the configuration into a solution.

Your Task:
Write a program **solver.py** that prints to standard output a sequence of queen assignments that solves the queens' problem in the fewest number of assignments. Also print out the total number of moves.
0 represents the blank tile. 1 represents a tile with a queen.

Example:

```
python queen-solver.py

0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0

:
:
:
:

0   0   0   0   0   1   0   0
0   0   0   1   0   0   0   0
0   0   0   0   0   0   1   0
1   0   0   0   0   0   0   0
0   0   0   0   0   0   0   1
0   1   0   0   0   0   0   0
0   0   0   0   1   0   0   0
0   0   1   0   0   0   0   0
```