Purpose: A* Search and the Eight Puzzle

Summary: You’ll solve the “Eight Puzzle” using A* search with the Manhattan heuristic.

Details: In file “lab3.py” I have provided a large amount of code related to setting up and solving the eight puzzle. Here are a few details about the data representations used:

- A node in the search tree is an object of class `node` with the following instance variables:
  - `state`: a list of size 9 containing the digits from 0 through 8. The “0” represents the blank space in the puzzle. This list is provided as a parameter to the constructor. Example:
    
    ```python
    state = [3,1,0,2,4,8,7,6,5]
    ```

  - `parent`: the parent of this node (unless it is the root, in which case `parent=None`). This is provided as a parameter in the constructor (you must use `None` when creating the root)
  
  - `g`: the cost of reaching this node from the root (automatically set to 1 + `parentNode.g`)
  
  - `h`: the estimated distance from this state to the goal (provided as a parameter in the constructor)
  
  - `value`: the sum of `g` and `h` (just provided for convenience)

- Class `eightpuzzle` holds most of the code for the Eight Puzzle. To construct a puzzle you must provide one parameter, `n`, specifying the maximum number of moves required to solve the puzzle. If you use a negative value for `n` a random puzzle instance will be generated, otherwise a puzzle will be generated that is guaranteed to have a solution requiring no more than `n` moves.

  The main instance variables of class `eightpuzzle` are:
  
  - `n`: a copy of the `n` parameter, unless it was -1, in which case `n` is a random integer between 1 and 64
  
  - `state`: a list of nine digits; the current state (see above for example)
  
  - `initial`: a list containing the initial state (randomly generated to guarantee a solution of at most `n` moves)
- **goal**: the goal state (currently [1,2,3,4,5,6,7,8,0])
- **currentNode**: the current node being visited in the search tree; note that the variable `state` is redundant, since `currentNode.state = state`.
- **open**: a list of nodes that have been generated but have not yet been expanded. Initially this contains only the root of the tree.
- **closed**: a list of nodes that have been expanded (in other words, all of their successors have been generated).

After a few node expansions, a typical search tree might look like:

```
state=[3,1,0,2,4,8,7,6,5];
g=0; h=12; value=12

/    /
/    /
/    / 
/    /
/    /
/    /
/    /
/    /
/    /
/    /
/state=[3,0,1,2,4,8,7,6,5];
g=1; h=13; value=14

/state=[3,1,2,4,8,7,6,5];
g=2; h=12; value=14

/state=[3,0,1,2,4,8,7,6,5];
g=1; h=13; value=14

/state=[3,1,2,4,8,7,6,5];
g=2; h=14; value=16
```

Inside the `eightpuzzle` class you will need to provide the code for two methods, `h` and `removeMin`. The comments in the program explain what these must do.

Method “solve” carries out the $A^*$ search. It is nearly complete, but you need to add a few lines about updating the `open` and `closed` lists.

Run the program several times and provide me with sample output in addition to your fully commented source code.

If you have time before the homework is due, do the following experiment: for each of the values $n = 10, 15, 20, 25, 30$, generate a large number (maybe 50 or 100?) of random puzzles (you can easily do this in a loop). For each value of $n$ collect statistics on the number of nodes expanded during the search. I want to know, for each of the given values of $n$, the minimum number of nodes expanded, the maximum number expanded, and the average number expanded. If you can, compute the standard deviations also. Ask if you’re not sure what you’re expected to do here.

Try to put this information together in a table for me, and we’ll compare results.

If you are really ambitious, find an admissible heuristic for the 8 puzzle that is more informed than the Manhattan heuristic (do a Web search or figure one out on your own?) and re-run the experiment using the improved heuristic.