Project Assignment #2: Sliding Puzzle Solver

In this project your goal is to solve the sliding puzzle with 8 and 15 numbers using breadth-first, depth-first, and A* search algorithms. In this game your goal is to sort the number tiles in the counting order as given in the below figure.

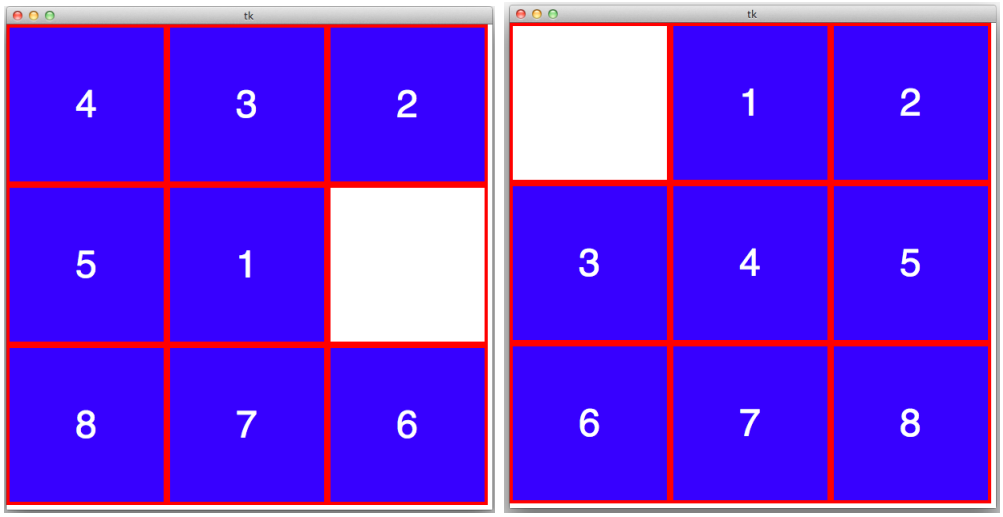

Figure 1: Sliding puzzle game with 8 numbers. On the left is the initial state of the game, on the right is the goal state of the game.

1 Preparation To-Dos

1. Make sure you have Python 2.7 installed
2. Make sure you install Tkinter package
3. Download the source code (slidingpuzzle.tar.gz) from LMS, extract it to your local disk

2 BFS and DFS performance analysis

(40 points) In this section your goal is to asses the performance of breadth-first search (BFS) and depth-first search (DFS) solvers, which we will also need to compare with the A* search solver. In the source code made available to you BFS and DFS solvers are

implemented in `BFSSolver.py` and `DFSSolver.py` respectively. You can run `test.py` to run BFS and DFS search on any game. Using the *easyCase* as the initial game state which is given in `test.py`, make sure your BFS and DFS solvers achieve the solution in a single move. Make sure you also visualize the solution by uncommenting the necessary line. The *Visualizer* class expects any array of game states, and visualizes them sequentially by pausing in between two states. Hence, you can see the moves taken to achieve the solution.
Using the *moderateCase* as the initial game state, run both DFS and BFS solvers. To asses the performance of the two solvers, produce two plots.

1. First plot is the level of total number of uniquely visited nodes versus the node depth.

2. Second plot is the total number of uniquely visited nodes versus the size of the fontier (number of nodes in the frontier).

How does the tree depth change with respect to the total number of nodes in the two search algorithms? How does the frontier size change? Compare the two search algorithms using these plots. For the *moderateCase*, BFS searches 115262 nodes and achieves the solution at depth level 23 (that is 23 moves), while DFS searches 124837 nodes and the tree depth grows to 53977 meaning that we need to make 53977 moves to find the solution. Clearly both uninformed search algorithms can do better and that is why we have A* search algorithm.

3 Implementing the A* Search algorithm

(60 points)

You are asked to implement the A* search algorithm. You can write your own code in function *astar* in file `AStarSolver.py`. You will be able to implement the algorithm with a minor change in either BFS or DFS solvers. A* algorithm sorts the nodes in the frontier according to their total cost. Hence, you need to keep a sorted list while generating new nodes and inserting them into the frontier. Using the bisect algorithm you can perform this with a lower complexity. You might want to use *bisect* library and the *insort* function.
The cost of node is the some of of its cost from the root (depth level in this case) and estimation of the cost from the node to the goal. This estimation can be done using heuristic functions such as *h1* and *h2*. Implement these heuristic functions in `PuzzleState.py`. *PuzzleState.getCost()* function should return depth level plus the heuristic function's value.
In `test.py` you are given the *hardCase* initial state using 15 numbers. Try and see BFS and DFS miserably fail in solving this harder game. You should be able to solve this game with A* search using h1 or h2. Produce the same two plots as in the above

section using the A* search algorithm for both h1 and h2. Comment on the differences. Also how many moves does it take to find the solution using *h1* or *h2*?

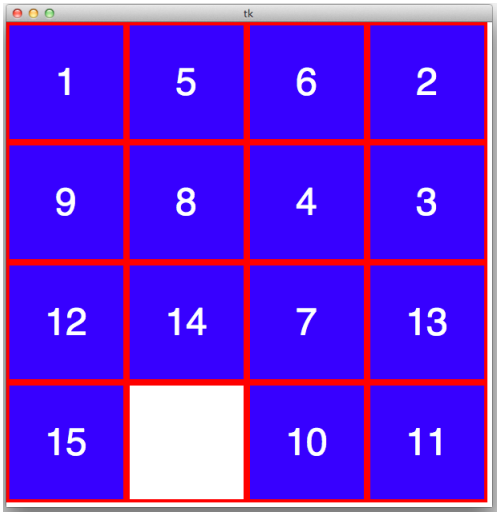

Figure 2: Sliding puzzle game with 15 numbers.

4 Bonus Question

(20 points)
Implement the iterative depth-first search algorithm, which should also find the best solution (smallest number of moves) as A*?

Good luck!