Section 2 Notes, January 16, 2004, by Kristin Branson

1 Implement the A* algorithm

The first step of Assignment 1 is to implement the A* algorithm. The A* algorithm is a special case of the general search algorithm. The general search algorithm, described in `http://www.cs.ucsd.edu/ elkan/150/jan6.html`, is:

```
input: a properly formulated search problem
        a function "insert" to place new nodes into a queue

 fringe := make-queue(initial-node)
 loop
   if empty?(fringe) then return FAIL
   else
     X := remove-front(fringe)
     if satisfies-goal(state(X)) then return X
     else
        fringe := insert(fringe,expand(X))
 end loop
```

The special cases of the general search algorithm (e.g. A* , DFS) differ in the order in which nodes are inserted into the fringe. In the A* algorithm, nodes are inserted in order of increasing f-value. The value $f(n)$ is an estimate of the total cost of the cheapest path *through node* n from the start to any goal. We compute $f(n) = g(n) + h(n)$ where $g(n)$ is the true cost of the path from the start state to state n and $h(n)$ is an estimate of the cost of the cheapest path from state n to any goal state.

How do you make your A* code general? I would have general `node` and `state` classes. I would write a general `AStar` function in terms of objects of these general classes. For each of the different problems, I would have an `eight_puzzle_state` or `robot_state` that inherits from the `state` class. What member variables and member functions should each of these classes have?

2 The 8-Puzzle Problem

You are to apply the A* algorithm to the 8-puzzle problem using the heuristics given on page 106 of AIMA and replicate Figure 4.8. The heuristics are:

- h_1 is the number of misplaced tiles (will be between 0 and 8).
- h_2 is the sum of the Manhattan distances of the tiles from their goals (will be the sum of 8 distances).

You are to replicate four of the columns of Figure 4.8. The first two columns are the *search cost* of A* when using each of the two given heuristics. It is unclear whether this is the number of nodes generated or the number of nodes expanded. Report whichever number you like, just document what you do.

The second two columns are the *effective branching factor*. If the total number of nodes generated is N and the solution depth is d, then b^* is the branching factor that a uniform tree of depth d with $N + 1$ nodes would have. You can think of this as the average number of paths tried from each state. If the effective branching factor were 1, then our algorithm's search path would go along an actual solution path.

To replicate these four columns, you need to generate 1200 random problems with solution lengths from 2 to 24 (100 for each even number). How do you generate 1200 problems with these properties? The goal state cannot be reached from every of the 9! possible states. One way to generate random states is to start from the goal configuration and "shuffle" the tiles randomly many times. Since A* finds the shortest solution, you can choose 100 of the problems which A* found to have solutions of length 2, 100 of the problems which A* found to have solutions of length 4, and so on. Aside: how many different problems are there with solutions of length 2?

Note: We will not penalize you if the numbers in your table do not match those in Figure 4.8, or for failing to solve enough large problems, provided your code is correct and reasonably efficient.

3 The Robot Navigation Problem

The 2-D robot navigation problem is to find the shortest path between two points on a plane that has convex polygonal obstacles. You must use A* to solve the robot navigation problem. This requires:

- Defining the state space. As suggested in 3.15a and 3.15b, you do not want to use all positions (x, y) in the plane as your state space.
- Defining the actions that can be taken from each state. You must use your geometry skills for this.
- Defining two different, reasonable heuristics:
  - Heuristic $h(\cdot)$ must be *admissable*. This means that the $h(n)$ must be less than or equal to the actual cost of the cheapest path from n to any goal state.
  - $f = g + h$ must be non-decreasing along every path from the start.
- Coding.

The state space, as suggested in 3.15b, should be the vertices of the obstacles, plus the start and goal locations. The robot should travel in straight lines from one state to another. From each state, you can reach any state that you can "see" from the current state. That is, you can reach any state such that the line segment connecting the current and new states does not intersect any edge of any obstacle.

Geometry review: how do you find whether two line segments intersect? Suppose our line segments are $\overline{(x_1, y_1)(x_2, y_2)}$ and $\overline{(x_3, y_3)(x_4, y_4)}$

1. Write the equations for the lines containing the two line segments:
$$y - y_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i).$$
2. Determine if the lines intersect. The lines intersect if the slopes are not the same.
3. Find the intersection of the lines: 2 equations, 2 unknowns. Let's call the intersection (x', y').
4. Check if this intersection point lies on each line segment. The intersection is on the first line segment if $x_1 < x' < x_2$ and $y_1 < y' < y_2$. The intersection is on the second line segment if $x_3 < x' < x_4$ and $y_3 < y' < y_4$.

As with the 8-puzzle problem, you should generate a table similar to Figure 4.8, containing the search cost and effective branching factor for each of the four test problems.