EECS 649: PROBLEM SET #3

Reading:

- R&N 3.1-6

Total Points: 100

Format:

- Use standard sheets of paper (8.5 by 11 inches) for scanning.
- Perform all work neatly. When asked to write a paragraph, it should be typed (e.g., using a computer and LaTeX or Word/Docs).

Problem 3.1 [15 points] *Search problem formulation*

Give a complete problem formulation for each of the following scenarios. In each case, choose a formulation that is precise enough to be implemented.

a. Using only four colors, you have to color a planar map in such a way that no two adjacent regions have the same color.

d. You have three jugs, measuring 12 gallons, 8 gallons, and 3 gallons, and a water faucet. You can fill the jugs up or empty them out from one to another or onto the ground. You need to measure out exactly one gallon.

For each part, also identify the entire state space, *X*.

Problem formulation means States, Initial State, Actions, Transition model, Goal test, Path cost (plus I asked you to add the entire State Space as well.) Examples of these descriptions appear in R&N in Sections 3.2.1 and 3.2.2. For the 8-puzzle, the entire state space is the set of 9! configurations of the tiles.

Problem 3.2 [15 points] *Search spaces*

Do the following problem:

3.15 Consider a state space where the start state is number 1 and each state k has two successors: numbers $2k$ and $2k + 1$.

a. Draw the portion of the state space for states 1 to 15.

b. Suppose the goal state is 11. List the order in which nodes will be visited for breadth-first search, depth-limited search with limit 3, and iterative deepening search.

c. How well would bidirectional search work on this problem? What is the branching factor in each direction of the bidirectional search?

d. Does the answer to (c) suggest a reformulation of the problem that would allow you to solve the problem of getting from state 1 to a given goal state with almost no search?

e. Call the action going from k to $2k$ Left, and the action going to $2k + 1$ Right. Can you find an algorithm that outputs the solution to this problem without any search at all?

Problem 3.3 [20 points] *Understanding A**

(a) Trace the operation of A∗ search applied to the problem of getting to Bucharest from Lugoj **(see Figure 3.1)** using the straight-line distance heuristic. That is, show the sequence of nodes that the algorithm will consider and the f, g, and h score for each node.

Draw a figure like the one in **Figure 3.18(f)** of R&N but make sure each node is labeled with f = g+h and a circled number relating to its order of expansion. The map is **Figure 3.1** and the straight-line distances are in **Figure 3.16**.

(b) Redo (a) using a different heuristic, h', based on h and **one-step look-ahead**. That is, let

h'(n) = min over all i that are successors n of { c(n, i) + h(i) }.

So, let's say a node n has two neighbors: a and b

Then h' = min(cost(n, a) + h(a), cost(n, b) + h(b))

The following problem requires significant programming:

Problem 3.4 [50 points] *Planning in a grid world*

This problem is about planning obstacle-free paths on grids, which arises in applications involving autonomous vehicles and video games.

Specifically, you are to implement routines that solve planning problems that take place in an N by M grid of cells, with initial and goal cells specified by integer pairs: (iinit, jinit) and (igoal, jgoal). The actions available are moving left (decrementing i), right (incrementing i). Down (decrementing j), and up (incrementing j). Moves are possible only if the destination cell is within the grid **and** not an obstacle cell. Each move has a cost of 1.

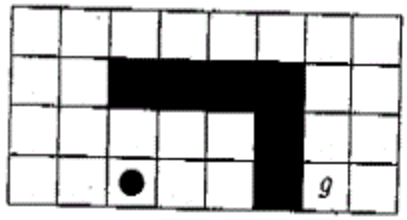
In all cases, the cost-to-reach is the number of moves and the heuristic estimate of the cost-to-go you should use is the "straight-line" Manhattan distance, that is, the sum of the absolute differences in both dimensions (which is admissible because it is the cost associated with the relaxed problem involving no obstacles). For example,

h(xinit) = | iinit - igoal | + | jinit - jgoal |

(a) Implement a general program for such worlds with routines for
   (i) Breadth-First Search (BFS)
   (ii) Greedy Best-First Search
   (iii) A*

In each case, implement your routine *according to the template we used in lecture for FORWARD-SEARCH (also covered in this handout*). You may also see the this linked note. **Turn in your code.**

(b) As a test, try your routines on the example grid we used in class, this linked handout:



Briefly comment on, **but do not turn in**, your results.

(c) Run your routines on the 100 by 100 discrete "bug trap" problem [linked page is from an early edition of Steve LaValle's *Planning Algorithms*], with initial, goal, and obstacle locations given by the following linked code code in both Matlab and Python. Matlab also appears belowl **[Note: Matlab indexes from 1, while Python indexes from 0]**:

```
iinit = 50;
jinit = 55;
igoal = 75;
jgoal = 70;

obs=zeros(100,100); % initialize to "no obstacles"
for i=1:100,
  for j=1:100,
    if i<50,     % rear of bugtrap
      d=abs(i-51)+abs(j-50);
      if d==50, obs(i,j)=1;  end;
    else         % front of bugtrap
      if j>50,  % upper lobe
        d=abs(i-50)+abs(j-75);
        if d==24, obs(i,j)=1; end;
      end;
      if j<50,  % lower lobe
        d=abs(i-50)+abs(j-25);
        if d==24, obs(i,j)=1; end;
      end;
    end;
  end;
end;
```

For each of the three routines, report the number of cells in *closed*, the number of cells on *fringe*, and the length of the path found. There is no need to draw/report the path. Above, *closed* refers to those cells that have been reached and expanded; *fringe* are those that have been reached, but not yet expanded.

(d) **Repeat** the experiment in (c) above with the initial and goal points reversed.

(e) Comment on your results and on the potential of bidirectional search for this class of problems.