Patrick Griffin (pgriffi1@uno.edu)

CSCI 4525

Question 1 *(3.14 from AI: A Modern Approach)* – ***10 points total, 2 points each***

Which of the following are true and which are false? Explain your answers.

a.) Depth-first search always expands at least as many nodes as A* search with an admissible heuristic.
   a. False. If A* uses an admissible heuristic that would make it a consistent heuristic, which is cost-optimal. Whereas depth-first search is not cost-optimal. Meaning it will always return the first goal state found, independent of the cost. This could very well lead to an overestimate of the cost. Something that A* can never do.

b.) h(n) = 0 is an admissible heuristic for the 8-puzzle.
   a. True. An admissible heuristic is one that never overestimates the cost to reach a goal. Seeing how h(n) = 0 would mean that you're at the goal state this would make it impossible to overestimate.

c.) A* is of no use in robotics because percepts, states, and actions are continuous.
   a. True. Continuous environments have infinite branching factors. A* has a memory usage issue. It can often time expand nodes exponentially in length. There could never be enough memory to explore all possible states found by A* in a continuous environment.

d.) Breadth-first search is complete even if zero step costs are allowed.
   a. True. Breadth-first search is systematic therefore it is complete. Completeness guarantees that a solution will be found if one exists and reports a failure otherwise. Completeness has nothing to do with whether a search's resulting cost is optimal.

e.) Assume that a rook can move on a chessboard any number of squares in a straight line, vertically or horizontally, but cannot jump over other pieces. Manhattan distance is an admissible heuristic for the problem of moving the rook from square A to square B in the smallest number of moves.

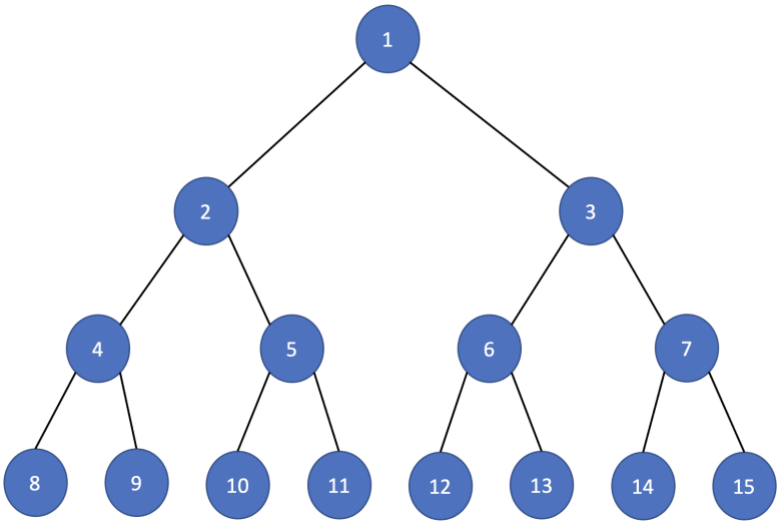
a. False. Manhattan distance gets its value by counting single tiles from square A to square B. Yet the rook is able to move over multiple times in a single move. This would mean that Manhattan distance is inadmissible in this situation.

Question 2 (*3.15 from AI: A Modern Approach*) – **10 points total, 2 points each**

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

a. State space: 1
    i. K = 1
    ii. 2(1), 2(1)+1
b. State space: 1, 2, 3
    i. K=2
    ii. 2(2), 2(2)+1
c. State space: 1, 2, 3, 4, 5
    i. K=3
    ii. 2(3), 2(3)+1
d. State space: 1, 2, 3, 4, 5, 6, 7
    i. K=4
    ii. 2(4), 2(4)+1
e. State space: 1, 2, 3, 4, 5, 6, 7, 8, 9
    i. K=5
    ii. 2(5), 2(5)+1
f. State space: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
    i. K=6
    ii. 2(6), 2(6)+1
g. State space: 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13
    i. K=7
    ii. 2(7), 2(7)+1
h. State space: 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 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.
   a. Breadth-first: 1, 2, 3, 4, 5, 6, 7, 9, 10, 11
   b. Depth-limited w/ limit of 3: 1, 2, 4, 8, 9, 5, 10, 11
   c. Iterative deepening: 1, 1, 2, 3, 1, 2, 4, 5, 3, 6, 7, 1, 2, 4, 8, 9, 5, 10, 11

c.) How well would bidirectional search work on this problem? What is the branching factor in each direction of bidirectional search?
   a. Assuming that we know where the goal state is and are able to reason backwards, bidirectional search would perform much better than both depth-first and breadth-first. The branching factor of bidirectional search is b^d/2 + b^d/2, whereas BFS and DFS have a branching factor of b^d.

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?
   a. If you were able to perform bidirectional search then this means you know where the goal state is located. Since the state would be a successor it should know how to reach its parent node. If you continue to reach the parent nodes until you reach the root you would know the path to the goal state.

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

a. By using these actions, we could infer that moving left for all even number and right for all odd numbers. If we take each node value % 2, starting from the goal state value, and look for remainders of 1 (move right) or 0 (move left) we could get the path to the root in reverse.