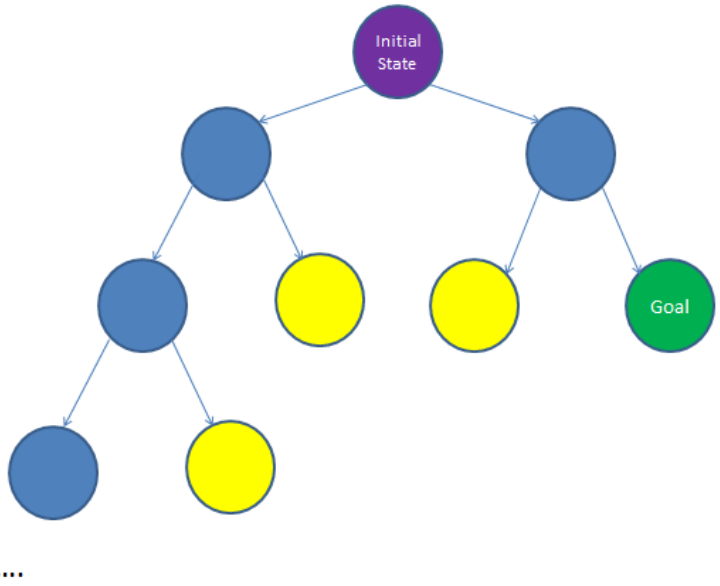
CS 4700: Foundations of Artificial Intelligence
Spring 2020
Prelim Prep Questions
Quiz 3

1. True/False Questions:
   d. If breadth-first search is able to find a solution to a search problem, then A* is guaranteed to find a solution.
   False. h need not be admissible.
   e. If h_1, h_2, and h_3 are all admissible functions, then $\frac{h_1}{6} + \frac{h_2}{3} + \frac{h_3}{2}$ is admissible.
   True. $h_1, h_2, h_3 \leq h^*$ means (1) $\frac{h_1}{6} \leq \frac{h^*}{6}$, (2) $\frac{h_2}{3} \leq \frac{h^*}{3}$, and (3) $\frac{h_3}{2} \leq \frac{h^*}{2}$, so:
   $$\frac{h_1}{6} + \frac{h_2}{3} + \frac{h_3}{2} \leq \frac{h^*}{6} + \frac{h^*}{3} + \frac{h^*}{2} = \left(\frac{1}{6} + \frac{1}{3} + \frac{1}{2}\right) h^* = h^*.$$

2. Consider a state space that is a tree with branching factor b and maximum depth m.
   c. Can depth-first search ever use more space than A* search?
      - If your answer is yes, please give an example.
      - If your answer is no, please explain why.

   Yes. Here's an example:



   ....

   Yellow designates terminal nodes. The branching factor b is 2. Imagine that the h function is perfect and directs A*directly to the goal, so that it would first go right from the initial state, then right again to the goal. Now consider the tree depicted for the left successor of the initial state, where each state has a successor that is a terminal node to the right and a similar subtree on its left. Every time DFS goes one step lower it adds the right node to Open (assuming DFS goes left-to-right), increasing the amount of memory used by 1. We can set

m to whatever value we need so that Open will contain however much memory A* would use (which in this case would only have two items on Open when it gets to the goal).

3. For each of the following search methods assume the branching factor is b. Give your answer using big-O notation.
   d. What is the worst-case size of the Open list after beam search has visited m states? (Assume the beam size is k.)

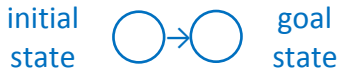
      After a state's successors are added to Open it is then pruned back to k entries. So Open will have O(k) nodes after m states are visited. Notice that in the iteration before there will have been k nodes, one is removed leaving k-1 nodes, and then the b descendants of the removed node are added back onto open, meaning there are b+k-1 nodes before it is pruned back to k. O(b+k) was therefore also taken as an acceptable answer.

4. Consider a search problem whose state space may contain cycles or be infinite. Imagine you have an evaluation function f(s) that always assigns to a state a positive integer value that is less than or equal to 100 (where lower values are better).
   Will hill climbing search always halt on such a problem?
   - If your answer is "yes", what is the maximum number of states it will visit?
   - If your answer is "no", please give an example of a search problem where it doesn't halt.

   [Corrected to have the values decreasing rather than increasing]
   Yes. Hill climbing only moves to a new state if there is a successor that has a *better* (greater) value than the current state. There are only 100 values that f(s) can take on, so in the worst case the first state will have value ~~1~~ 100, the second value ~~2~~ 99, …, all the way to a final state with value ~~100~~ 1. So the maximum is 100.

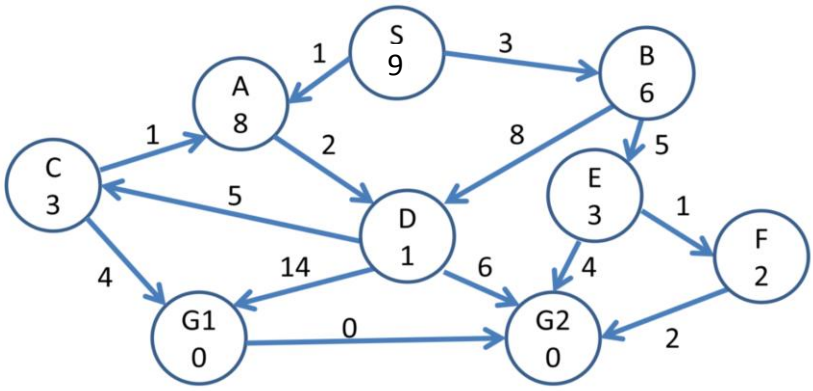
5. Give the smallest example that you can of a search problem with a corresponding h(s) where if you used f(s) = g(s) + h(s) A* would succeed in finding a solution whereas if you used f(s) as the evaluation function for hill climbing search it would fail to find a solution.

   There are many. Here's a very simple one. There are two nodes, and a single action whose cost is 1. Both states have h=0. The f value for the initial state is 0, for the second (goal) state it is 1+0=1. A* finds it, however hill climbing would stop at the first node because it has a better value than its successor.

6. Consider the following search space, where S is the initial state and G1 and G2 are goal states, where the cost of the operator that takes you from one state to the next is given

along the edge between the states, and where the value of the heuristic evaluation function h applied to the state is the number written inside the state. In cases of ties pick the state that comes earlier in the alphabet.



a. Give the sequence of states that depth-first search visits.
   S, A, D, C, G1
b. Give the sequence of states that iterative deepening visits.
   S, S, A, B, S, A, D, B, E, S, A, D, C, G1
c. Give the sequence of states that hillclimbing with f(s) = g(s) + h(s) visits (lower values are better).
   ~~S, A, D, G2~~
   S
d. Give the sequence of states that A* visits if it uses f(s) = g(s) + h(s).
   S, A, D, B, G2
e. Is h(s) admissible?
   Yes. The h value in each state is never larger than the cheapest cost to get to a goal.

7. Consider using A* search for a problem where all actions have the cost greater than or equal to 1. Imagine you're using an h(s) that does "one step look-ahead" – namely if after applying one operator to s you are in a goal state h returns 1 otherwise it returns 2. Is h(s) admissible? Please explain your answer.
   No, because if you are at a goal state h would have a value greater than 0 but should equal 0. (In all other cases it satisfies the requirements for admissibility.)