CS 4700, Foundations of Artificial Intelligence
Spring 2020
Solutions to Quiz 1

1. There were three possible questions:
    - Consider a problem whose search space is a tree, with a single goal state located at level d. How many times will a state at level d-4 have its successors generated by iterative deepening search?

      4. The states at level d have no successors generated, the states at level d-1 have successors generated just once, the states at level d-2 have successors generated twice, and so on. The states at level k have their successors generated k times.
    - If iterative deepening search runs faster than breadth-first search on a particular problem, then depth-first search will also run faster than breadth-first search on that problem. (Assume all the search methods order nodes identically, so that if they have to choose a state from some set of states they would all select the same state.)

      False. Iterative deepening uses a depth-bounded DFS, so it will never go on forever down some path. Straight DFS, on the other hand, could continue to depths well beyond the depth of the solution found by either iterative deepening or BFS, and for an infinite tree might even run forever, despite both iterative deepening and BFS returning solutions.
    - Consider a problem whose search space is a tree and whose only solution is at depth d. Depth-first search can take twice as long (expand twice as many states) as iterative deepening search to find the solution. (Assume both methods order nodes identically, so that if they have to choose a state from some set of states they will both select the same state.)

      True. Imagine a problem who solution is found by iterative deepening at depth d, and where that solution is not the first one found at level d (if we're ordering states from left to right, that means the solution is not the left-most node at level d). DFS has no depth bound so if the first state generated at level d (meaning the left-most if we order things left-to-right) has further successors, DFS would continue beyond depth d. You could put however many states down there that you needed to ensure that DFS ran for twice the length.
2. There were two possible questions:
    - Consider a search problem with branching factor b. If Breadth-First Search has thus far put n nodes on the Closed list, what is the fewest number of nodes that could be on the Open list?

      0. It could be the case that at this point all successors of all nodes in the search space have already been expanded, so nothing is left on Open.
    - Consider a search problem with branching factor b. If Depth-First Search has thus far put n nodes on the Closed list, what is the fewest number of nodes that could be on the Open list?

      0. It could be the case that at this point all successors of all nodes in the search space have already been expanded, so nothing is left on Open.
3. There were two possible questions:
    - Consider a search problem with branching factor b. If Depth-First Search has thus far put n nodes on the Closed list, what is the largest number of nodes that could be on the Open list?

      $n(b-1)+1 = nb - (n-1) = nb-n+1$. There are n nodes on closed, so if each node generated b successors we'd get nb states. One of the states on Open is the initial state. All the

others were found by generating successors of other states, so of the nb generated states n-1 of them have to have be on Closed in order to have gotten to a total of nb states, making the total number nb-(n-1) = nb-n+1.

Another way of generating this result is to imagine an infinite tree, so that all the states on Closed were generated as DFS went down the left-most branch (or in whatever ordering the DFS is using – the important thing is that each node generates b successors, we pick one of those and generate b successors, and so on so that we haven't hit a dead end and backed up to some higher-level node to start generating successors). Each time we do that we put one node on Closed – the one we just expanded – add b nodes to Open, but then take one of those off as the next state that we'll be expanding. In other words each time we do this we ultimately end up with b-1 new states on Open for every new item on Closed, and we did that n times, yielding n(b-1). However, the last time we do this we add b to Open, then take one off. Just before we take that one off we have one more item on Open, which makes its size n(b-1)+1 = nb-n+1.

- Consider a search problem where each state has exactly two successors. If Breadth-First Search has (2^k)-1 nodes on the Closed list, how many nodes are on the Open list?

  2^k. If every state has *exactly* 2 successors, then it's a complete binary tree we're talking about. A complete binary tree has 2^{k-1} nodes at level k and a total of 2^k-1 nodes in the entire tree (if we called the root level 0 – if we counted it as level it would be the same formula, but we'd be calling that same level k+1 instead of k). If BFS has 2^k-1 nodes in Closed, that means it has expanded all nodes through level k, and Open would have all the nodes at level k+1. There are 2^k nodes at that level.

4. There were four possible questions:

   - Consider a search problem with branching factor b. If during its search Depth-First Search has n nodes on the Open list, what is the smallest number of nodes that could be on the Closed list?

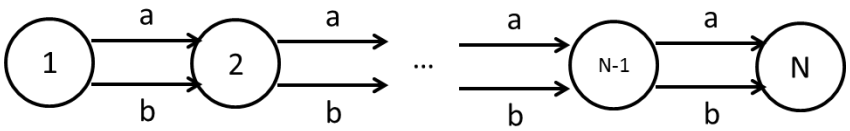
     (n-1)/(b-1). This is essentially the same as the first question under #3 above. Since both questions use the number n I'll use k instead of n within the answer to that question. If there are k nodes on Closed then the largest that Open can be is kb-k+1. So if there are kb-k+1 nodes on Open then the fewest number that can be on Closed is k. Since the question says there are n nodes on Open, setting n equal to kb-k+1 we can solve for k and get k=(n-1)/(b-1). That's the smallest number of nodes that can be on Closed.

   - Consider a search problem with branching factor b. If during its search Breadth-First Search has n nodes on the Open list, what is the fewest number of nodes that could be on the Closed list?

     ceiling(n/b). The most efficient way to generate those n nodes is for each node on Closed to have generated b of them. If n were an exact multiple of b we'd need n/b nodes on Closed to generate the n nodes on Open. If n is not an exact multiple we'd need one more than that. We can say that mathematically by saying ceiling(n/b), so that it rounds up if n is not a multiple of b.

5. There were two possible questions:

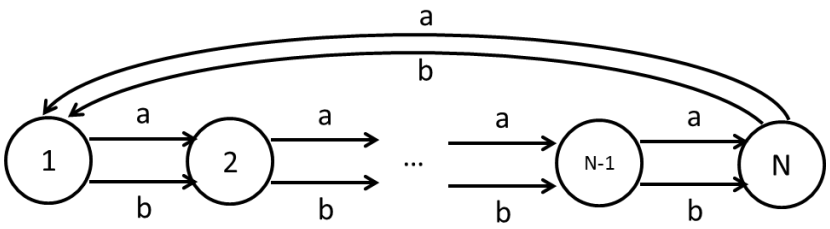
   - Consider a state space with N states, numbered 1 to N depicted in the following figure. Each state except the last has 2 actions that apply, a and b. Both actions have the same effect: If you are in state i it takes you to state i+1. The last state, N, is the only goal state and has no successors.



Depth-first search will be better than breadth-first search for this problem.

False. They behave identically. Each of them would add the successor created by a and the successor created by b to Open. Whichever one was added second would do the add, but since the state is already on Open it wouldn't be added a second time. So with either algorithm at each state only one item is added to Open. It is then removed from Open as the next state to be expanded.

- Consider a state space with N states, numbered 1 to N depicted in the following figure. There are 2 actions, a and b, that apply to each state. Both actions have the same effect: If you are in state N you return to state 1. Otherwise, if you are in state i it takes you to state i+1. The last state, N, is the only goal state.

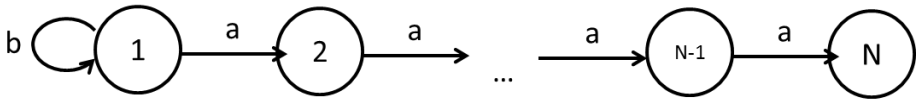


Because this state space has a cycle depth-first search can fall into an infinite loop when solving this problem.

False. Although the graph has a cycle, DFS would get to node N and then stop since it's a goal state and never get back to state 1.

6. There were two questions:

- Consider a state space with N states, numbered 1 to N depicted in the following figure. Each state except the last has 1 action, a, that takes you from state i to state i+1. In addition state 1 has a second action, b, that simply returns you to state 1. The last state, N, is the only goal state and has no successors.



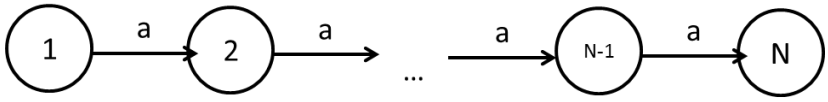
Imagine you were using breadth-first search and it did not maintain a Closed list to check for cycles (so, for example, you could return to state 1 multiple times without the algorithm realizing it). Further, when states are added to the Open list it does not redundantly add a state that is already on Open.

What is the runtime of breadth-first search on this problem? Use big-O notation.

$O(N^2)$. On the first iteration BFS would place states 1 and 2 onto Open, which takes 2 steps. It would then one-by-one take state 1 and state 2 off and generate the successor of each, which would give 1, 2, and 3 and take 3 steps. At the next step it would be 1, 2, 3, and 4 and use 4 steps. And so on. When Open contains states 1, 2, …, i, after the next iteration it would contain 1, 2, …, i, i+1 and take i+1 steps. And so on, until it eventually generated an Open list that contained 1 through N, which takes N steps. After that final iteration it would find N and stop, which at worst might take N steps depending

on the order in which states are removed from the Open list. Adding these up you get (1+…+N)+N = N(N+1)/2 + N. This is $O(N^2)$,

- Consider a state space with N states, numbered 1 to N depicted in the following figure. Each state except the last has one action that applies, a, which takes you from state i to state i+1. The last state, N, is the only goal state and has no successors.



What is the runtime of iterative deepening search on this problem? Use big-O notation.

$O(N^2)$. For the first iteration it would just get to state 1. On the next it would be states 1 and 2. For the next it would be 1, 2, and 3. And so on until it reached the iteration where it goes all the way from 1 to N. This is 1+2+3+…+N, or in other words N(N+1)/2, which is $O(N^2)$.