Combinatoric Search Practice Exercises (Chpt. 3)

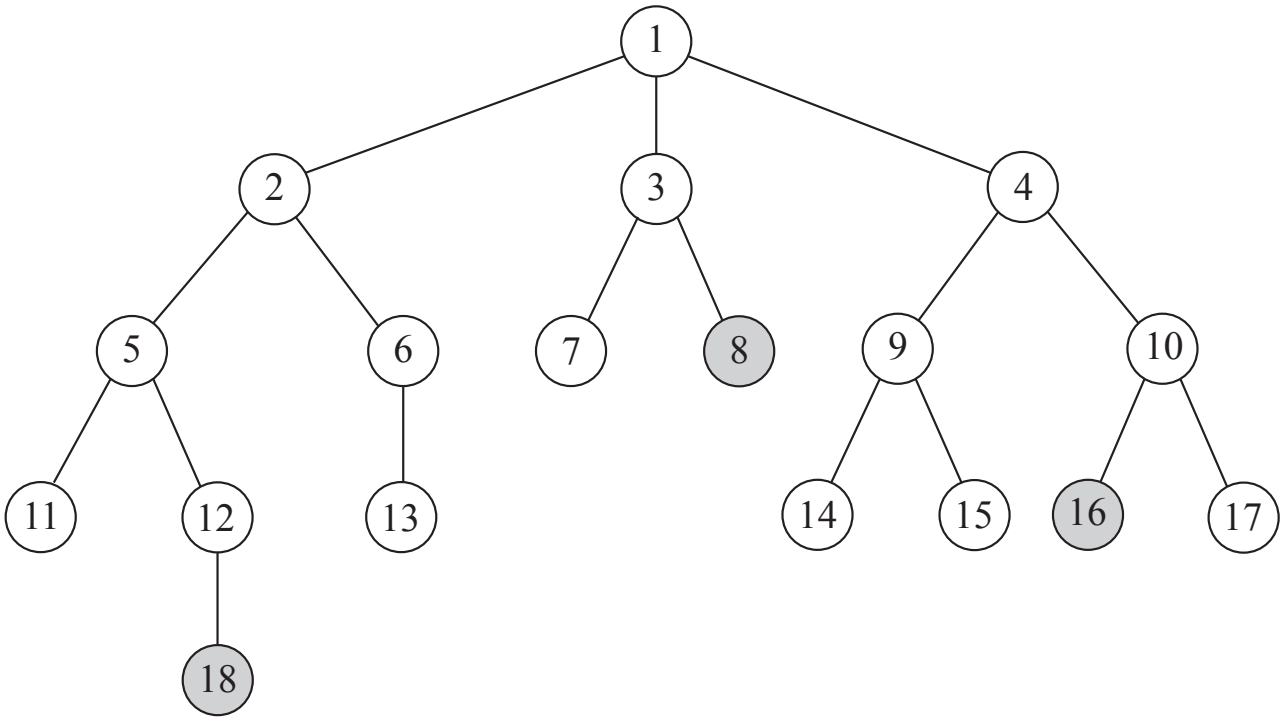
Prof. Jim Rehg
CS 3600 Introduction to Artificial Intelligence
School of Interactive Computing
Georgia Institute of Technology

February 9, 2018

These practice exercises are for your benefit in preparing for the exams. They will not be collected or graded. Solutions will be provided. Note that not all questions are representative of questions you will find on the exam, but the material covered by these questions will also be covered by the exams.

Question 1. Consider the following search tree, where the shaded nodes correspond to goal nodes. List the order in which nodes will be visited for the following four search strategies.[1] You should also show the evolving frontier/queue in order to get partial credit.

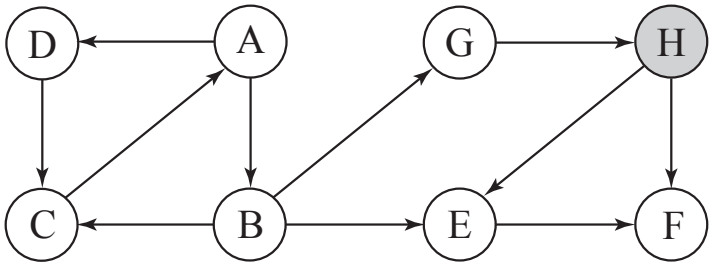
1. Breadth-First Search
2. Depth-First Search
3. Iterative Deepening Depth-First Search
4. Select a search algorithm with the property that node 16 will be selected as the solution (you may make any appropriate modification of the search tree as needed)



[1]The question is asking for the explored set, sometimes called closed list, of visited nodes at the time the search terminates.

Question 2. Consider the following graph. Assume we start the search at state "A" with goal state "H." At any point during search, if ties need to be broken, the node which is closest to the start of the alphabet (closest to A) will be selected. List the order in which nodes will be visited for the following two search methods:

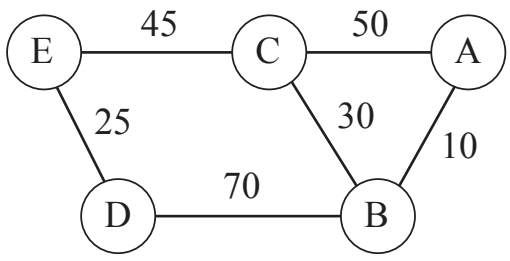
1. Breadth-First Search
2. Depth-First Search



Question 3. Use **A* Search** to solve the map problem below, where the numbers give the distances between states.

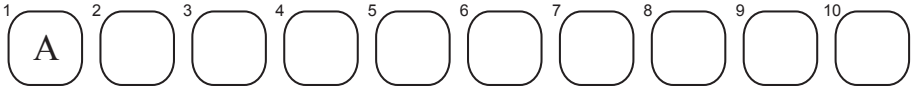
- **Initial State** = A
- **Goal State** = E
- The value of the heuristic function for each node is given in the table on the left.

n	$h(n)$
A	60
B	55
C	30
D	20
E	0

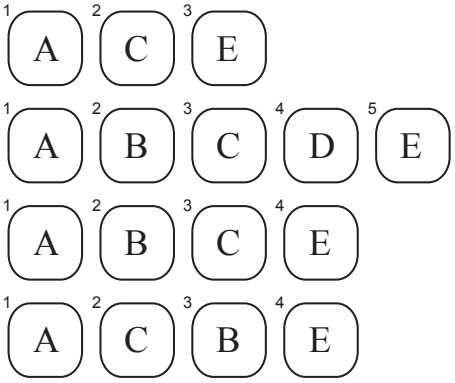


(a) Draw the search tree for A*, showing the evolving frontier.

(b) Indicate below the order in which nodes will be expanded (i.e. put in the explored set)



(c) For each of the possible node expansion orders shown below, indicate which *uninformed search algorithms* could have produced the given node order when applied to the graph above. Note: Be explicit about how you are breaking ties for nodes at the same depth.



_____

_____

_____

_____

(d) Show that the heuristic function for this problem satisfies the *consistency* criteria.

Question 4. Work all parts of problem 3.26 in Russell and Norvig.

Question 5. The graph separation property of Graph-Search (see Fig. 3.9 in your text) states that the frontier divides the explored and unexplored regions of state space.

(a) Prove the graph separation property by induction.

Let U, F, E be the *unexplored*, *frontier*, and *explored* sets of nodes, respectively. Separation is often written as $U \operatorname{sep} E|F$, which means that sets U and E are separated by F. This is true if and only if, for every $x \in U$ and $y \in E$ connected by a path P, there exists at least one node $n \in P$ such that $n \in F$. In words, every path between nodes in U and E passes through F.

Note: To prove by induction, show that the statement holds (trivially) at the start node. Assume that it is true after t steps of expanding a node, and then prove that it must also hold after $t + 1$ steps.

(b) Suppose your friend invents a more efficient variant of Uniform Cost Search in an undirected graph, where instead of generating all of the successor states for a given node n, only a subset of the possible successors, randomly selected, are generated. Is the separation property preserved under this modification? (why or why not) How is the optimality of Uniform Cost search affected?

(c) When performing graph search in a directed graph, expanding a parent node generates all of the child successors, where edges are directed from parent to child (e.g. in Problem 2 above, node A has two children, B and D, and is itself a child of C). Does graph separation hold in this case? (why or why not) Show that Uniform Cost Search is optimal in a directed graph.