Homework 1
Computer Science
Spring 2017
B351

Steven Myers

February 3, 2017

All the work herein is mine.

Answers

1. Here are the definitions for the terms listed in Problem 3.10:

   (a) **State** - A particular assignment of value to all variables for a given situation.

   (b) **State space** - All of the possible assignments of values to variable in a scenario.

   (c) **Search tree** - An arrangement of states such that no state is repeated twice when traversing all states.

   (d) **Search node** - A particular node in a search tree that has a cost and ingoing or outgoing paths. Each node contains the cost of the current node, its data (state values), and the cost of visiting other accessible states.

   (e) **Goal** - A final search node with an ideal or target state that we want to obtain to resolve a problem.

   (f) **Action** - An operation that results in a change of state.

   (g) **Transition model** - The process of moving from one state to another when an action is applied.

   (h) **Branching factor** - The maximum number of "branches" or pathways that some search node can have. This is used to limit the search space and prevent a machine from running out of memory before resolving a problem when the search space is too large.

2. A state space where iterative deepning search would perform worse than DFS is one in which the following are true:

   (a) The branching factor is large.

   (b) The depth of each branch is shallow, and the goal node exists in a branch which is deeper than the others.

   A state space such as Figure 1 could produce a circumstance where DFS would perform significantly better than iterative deepening search.

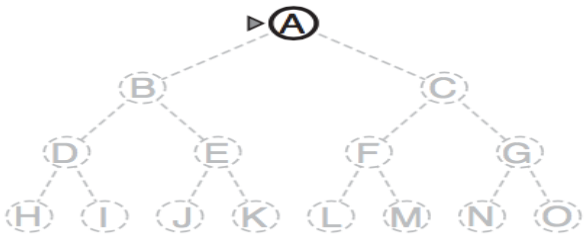

Figure 1: Attach a goal node **S** as a node **O**'s right child. (Image taken from textbook figure 3.16 on page 86, Russel & Norvig)

3. Yes, the graph is consistent. Here are the statements for each node and its sucessors:
   (a) AB: $1 \leq 6$
   (b) AC: $1 \leq 8$
   (c) BC: $4 \leq 4$
4. (a) Solution in `rv1.py`. Reads in the file `f.txt` and outputs the directions in `directions.txt` if a path exists from the robot's starting point to the goal position.
   (b) Better solution in `rv2.py`. Reads in the file `f.txt` and outputs the directions in `directions2.txt` if a path exists from the robot's starting point to the goal position. I used BFS to determine the shortest path from the current tile to the goal node. BFS is guaranteed to find the shortest path since it traverses the graph in a similar way that the robot would do so (one tile in a cardinal direction at a time). This is because my adjacent function only returns tiles NWES of the current tile and not in the diagonal directions. Once I got the shortest path from BFS, I made my robot traverse the path and rotate as necessary and recorded the steps.
   (c) In `rv1.py`, I created a very simple search that essentially models the robot randomly exploring the entire maze, always turning left when given the opportunity. So, it's as if the robot walks along the wall with its left hand on the wall and turns around at every dead-end. It explores the entire maze (perhaps the most inefficient path, depending on the maze layout). This search **fails** when the maze contains cycles, and will incur an infinite loop since the robot will wander forever. If there is no path from the robot's starting point to its goal node, then the robot's path finding function will return false. This is done by doing a DFS search from the robot's starting position and looking to see whether or not the goal position is reachable.

   In `rv2.py`, I used BFS to find the shortest path then made the robot follow that path, rotating as needed. This search also **fails** when the maze contains cycles. If the BFS search did not yield a path to the goal node, the program halts and errors out as no path exists.

   In both of my files, the $g(n)$ or weight of the edges is equivalent since all edges that are connected are exactly one tile away. The $h(n)$ or cost to reach the goal node from any given node is calculated based on its order in the shortest path. So, every node on the path to the goal node is exactly one node cheaper than the previous node. No shortest path requires more than one rotation by the robot at any given point, so we can say that the cost of intersections is also uniform throughout the entire graph.
5. I extended the rock, paper, scissors game from the last homework assignment. The file is included as `rpgs.py` as requested. This version of the game differs from the previous version since the computer and the human take bets. There is more human-interaction with the game, as you must decide when you should take bets, etc. The computer takes bets based on whether or not it's on a losing or winning streak. You lose the game if you run out of money (less than 10 dollars), and win the game if the computer runs out of money.