Problem Solving Workshop #31 Tech Interviews and Competitive Programming Meetup

December 16, 2017 https://www.meetup.com/tech-interviews-and-competitive-programming/

Instructor: Eugene Yarovoi (can be contacted through the group Meetup page above under Organizers)

More practice questions: leetcode.com, glassdoor.com, geeksforgeeks.org
Books: Elements of Programming Interviews, Cracking the Coding Interview
Have questions you want answered? Contact the instructor, or ask on Quora. You can post questions and follow the instructor and other people who write about algorithms.

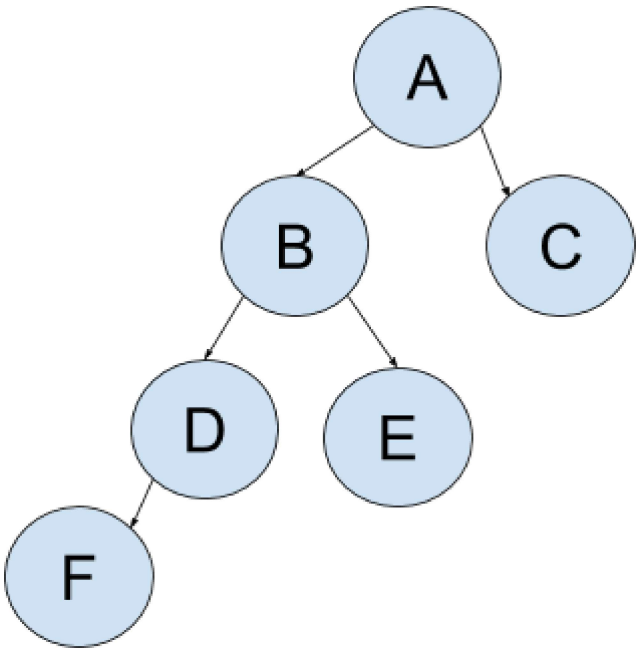
Try to find optimized solutions, and provide a time and space complexity analysis with every solution.

Problem #1, "Tree Burning"

You're given a binary tree, and you can choose a node and light it on fire. When you light a node on fire, the node burns up immediately, and the fire spreads to all the nodes connected to the burning node (the parent plus the child nodes) in 1 second, to all the nodes connected to those nodes in turn after another second, and so on, until the whole tree burns up.

You want to burn your tree up as quickly as possible, to make room for a virtual Christmas tree soon. Which node should you light on fire to minimize the tree's burn time (if several nodes are equally good, return any one of them)?

Example Input:



Output: B

Explanation: B is the best node to light on fire because it is the only node that can burn the tree in just two seconds. (B -> A -> C and B-> D -> F are the longest paths, and they only have two edges). At t=0, B will be on fire; at t = 1 D, E, and A will catch fire; at t=2, F and C will catch fire.

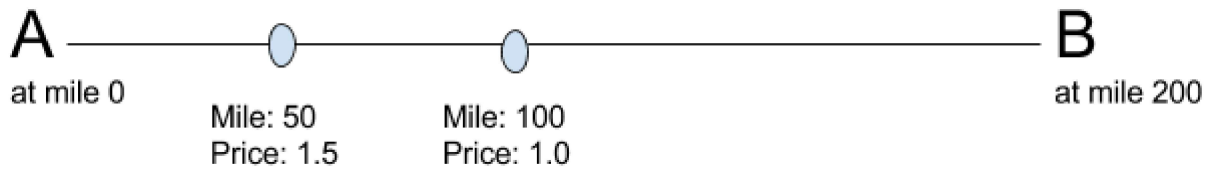
(a) **[Easy]** Solve in $O(n^2)$ where n is the number of nodes.
(b) **[Easy-Medium]** O(nh) where h is the height of the tree.
(c) **[Medium]** O(nh) but the tree can be an arbitrary (non-binary) tree now.
(d) **[Medium-Hard]** O(n) for arbitrary trees.

Problem #2, "Cheapo Drive"

You are driving on a line from point A to point B. You will always move in the direction of B, never doubling back. Along the drive, there is some number of gas stations. Each gas station has a different price for fuel (denoted in $ / mile: here we assume we can interpret gas quantites in miles of road that they can cover). As you reach each gas station, you can purchase some (zero or greater) amount of fuel, for which you pay fuel_cost * amount_purchased at that gas station. You want to reach point B at minimum cost, subject to the constraint that you must never run out of fuel along the way.

Without loss of generality, we will say that A is located at mile 0. You will be given the (positive) location in miles of B, and the location of, and fuel price at, every gas station along the way (you can assume these are given sorted by distance from A). You also have some amount of starting fuel in the tank, which will be given as well. Output the minimum cost of getting to B.

Example Input: B_mile = 200, stations = [(mile = 50, price = 1.5), (mile = 100, price = 1.0)], starting_fuel = 80



Output: 130
Explanation: At the 50 mile mark, you should buy 20 units of fuel to tide you over until you reach the 100-mile mark (you start with 80 miles of gas, so you need those 20 units). This will cost 20 * 1.5 = 30. Then, at the 100-mile mark, get 100 miles worth of gas, which costs 100*1 = 100. The total cost is 30 + 100 = 130. You could have (among other possibilities) bought 120 units of gas at the first station, but this would have cost 120 * 1.5 = 180, which is more expensive.

(a) **[Medium]** Optimal solution is O(N), where N is the number of gas-stations.
(b) **[Medium-Hard]** Follow-up: what if in addition, there is a limit to how much gas you may have in the tank at any given time? You are given an additional parameter "capacity". You can never have

more than that much gas in the tank at a time.

Problem #3, "Array Runs" (Medium)

You're given an array of 0s and 1s. Design a data structure that, after some preprocessing of the array, can support all of the following operations efficiently:

1. Given an index i and a 0/1 value V, set A[i] = V
2. Given an index i, get A[i].
3. Get the length and position of the longest consecutive streak of 1s in the array.

Example:
Starting state: A = [0, 0, 1, 1, 1, 0, 1, 1]
Operation 3 invoked -> (max_streak_length = 3, max_streak_start_index = 2)
Operation 1 invoked with i = 5, V = 1. State of array is now A = [0, 0, 1, 1, 1, 1, 1, 1]
Operation 3 invoked again -> (max_streak_length = 6, max_streak_start_index = 2)

Problem #4, "Wildcard Parens"

You're given a string consisting of "(", ")", and "?" characters only. If each "?" can be replaced with either nothing, "(", or ")", output a boolean indicating whether the string can be made to have well-formed parentheses. Parentheses are said to be well-formed when each opening parenthesis can be matched with a closing parenthesis and vice versa (e.g. ()() or (()) are well-formed, ())) is not).

Example Input 1: "((?)))"
Output: true
Explanation: The ? would have to be replaced with an open parenthesis, and the string is then "((()))", which is well-formed.

Example Input 2: "(?))))??"
Output: false
Explanation: No matter what you do with the ?, there's no way to get well-formed parentheses here.

(a) **[Medium]** Solve this in O(N) where N is the length of the string.
(b) **[Hard]** What if now, we allow 3 different kinds of parentheses, (), {}, []. We allow a "?" to mean any one of these 6 symbols or nothing (7 possibilities total). The parentheses are only considered well-formed if they're matched up by type correctly, e.g. "({}) is well-formed, but not "({)}".