Problem Solving Workshop #8 Tech Interviews and Competitive Programming Meetup

June 11, 2016 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 for the algorithms questions.

Coding / Algorithm Problem #1

A road of length N has good parts, and bad parts that need to be repaired. The road is discretized into chunks of size 1. We receive as input a boolean array A of size N, where A[i] is true if the road is good at coordinate i, and A[i] is false if the road is bad there. For example, [true, true, true, false, true, true, false] would be a road of size 7 where the bad parts occur at positions 3 and 6, and the rest of the road is good.

We want to repair all bad parts of the road. A bad chunk gets repaired if at least one repair truck visits it. There are T repair trucks and each one starts with F units of fuel. Each truck starts on some chunk of your choosing, and can move to *adjacent* chunks from there until it runs out of fuel, spending 1 unit of fuel per chunk it visits (including the starting chunk).

(a) **(Easy)** In addition to the road array, you're given F, the amount of fuel per truck, as input. Give an efficient algorithm to compute T, the minimum number of repair trucks you need in order to fix all the bad chunks.

Example Input: A = [false, true, true, false, false, false], F = 2
Output: 3
Explanation: You need one truck just to get the bad chunk at the start of the road, since the truck that fixes it can't get to any other bad chunk before running out of fuel. The 3 bad chunks at the end require 2 more trucks to fix, since each truck can visit at most 2 chunks. So, 3 trucks total.

(b) Now, in contrast to the previous problem, you're given the road array and T, the number of repair trucks you have, and you're asked to find the minimum F you need in order to be able to fix all the bad chunks.

(i) **(Easy)** Find any algorithm for this.
(ii) **(Medium)** Give an O(N log N) solution.

(iii) **(Hard)** Give an O(N) solution.

Example Input: A = [false, true, true, true, true, true, true, false, true, false, false, false], T = 2

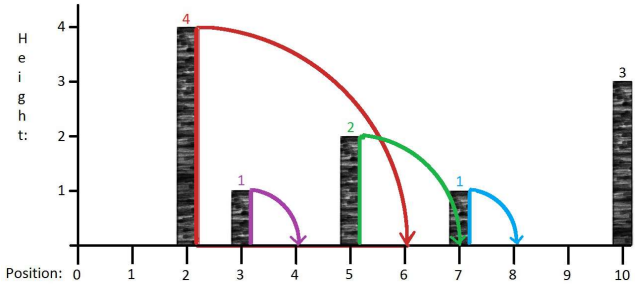
Output: 5

Explanation: We need at least 5 units of fuel for each truck. If we have 2 trucks with 5 units, we can start one truck at position 0, and the other at position 7 (second F in the array), and drive both forward as far as they go. There's no way to cover all Fs with less than 5 units of fuel per truck. The first truck needs only one unit, but all trucks start with the same amount of fuel, and the second truck needs all 5 units.

(c) **(Medium-Hard)** We now allow up to B bad chunks (where B is given as an argument) to remain after the repairs are done. Modify your solution from part (b) to take this into account. The time complexity that's expected is O(BN) or O(BNlog N), depending on the efficiency level you got for (b).

Algorithm Problem #2

You have dominoes of different heights arranged in a line. When a domino is knocked over, it may knock over other dominoes, which may in turn knock over even more dominoes, and so on in a cascade. More precisely, the effect of knocking over to the right a domino at position x is that any and all dominoes with position greater than x and less than or equal to x + *dominoHeights[x]* are knocked over to the right as well. These dominoes may then knock over even more dominoes and so on until nothing else is in range to be knocked over.



In the above diagram, we have five dominoes of different heights. If we knock over to the right the red domino at position 2 (on the x-axis), which has height 4, it falls and knocks over all the dominoes up to and including position 2 + 4 = 6. The purple and the green dominoes are knocked over in the same direction in this process. The green domino then knocks over the blue domino. The rightmost domino is not affected by this cascade. We say that the cascade ends at position 8, since that is the farthest right that any domino participating in the cascade travels.

The goal of this problem is to compute the end position of the cascade for each possible starting domino. This is easy to accomplish in $O(n^2)$ by running an O(n) algorithm for each domino, but the goal is to come up with a more efficient algorithm.

The input for this problem will be given as an integer array, where the value at the i-th index will represent the height of the domino at position i. A value of 0 will denote the absence of a domino at that position. For the image above, the input is [0, 0, 4, 1, 0, 2, 0, 1, 0, 0, 3]. The output should be an array of the same size indicating the cascade end position for each domino, in this case [0, 1, 8, 4, 4, 8, 6, 8, 8, 9, 10, 13] (where there is no domino, the cascade ends at the position it begins).

System Design Problem

How would you design a maps application such as Google Maps, Apple Maps, Bing Maps, Mapquest, etc.? There could be a lot of features within such an app, but focus on the following:

- How would you adjust the map shown to the user as they zoom in and out, or scroll the view?
- How would you design the feature that overlays the map with icons for businesses and points of interest?
- How would you implement the feature that provides directions from point A to point B?

Generally speaking, in design questions, there's always lots of ambiguity, so you would start by asking clarifying questions to the interviewer. Since you don't have an interviewer here, assume what you think are reasonable answers to the questions you would ask.