Problem Solving Workshop #24 Tech Interviews and Competitive Programming Meetup

May 13, 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 (eugene.yarovoi AT gmail.com), 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, "Integer Distance" (Easy-Medium)

You're given an array of N integers.

(i) Determine whether there exist two values in the array that have a difference of at most K (K is a given integer). Good time complexity: O(N log N).
(ii) Determine whether there exist two values in the array that are identical, and their index differs by at most D (they are distance D apart or closer in the array). D is a given integer. Good time complexity: O(N) or O(N log N), depending on your approach.
(iii) Determine whether there exist two values in the array that satisfy both conditions at once: they have a difference of at most K **and** are at most D distance apart (given both K and D). Target time complexity: O(N * some log factors. It's also possible to solve in O(N), independent of K and D, but this is hard, so try it once you've completed everything else.

Problem #2, "Road Repairs"

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 of road (in the direction of your choice) 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) **[Medium]** 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.

Guidance: first find the $O(N^2)$ solution, then try to improve to O(N log N), the target time complexity. It's also possible to solve this problem in O(N), but this is hard, so try it once you've completed everything else.

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.

System Design Problem (HARD)

The interviewer states the problem like this: Consider an existing large website where there are lots of images submitted by users. For example, Wikipedia, Pinterest, DeviantArt. How would you design a web crawler to crawl all pages belonging to that domain, and download all the images?

At this point, you would need to ask clarifying questions before you can attempt the problem. Imagine the conversation goes as follows:

You: How large is the website and how many images do you estimate there are?
Interviewer: The site may have on the order of 1 billion images.
You: Each image may be 100KB-1MB, so we're talking hundreds of terabytes of data. I assume we want to download this data into some kind *distributed* database, since 1 machine can't hold this.
Interviewer: Yes. And no requirement to organize the images in any way, just download them all.
You: Can I assume I have the storage set up and an API call that adds 1 image to the storage?
Interviewer: Yes. Just design the crawler.
You: How do we know what pages are available in the domain? Is the website organized so that we will find all the pages by starting from the homepage for the domain and following links?
Interviewer: Yes. Also assume for the time being you don't have to deal with robots.txt or the like.

You: Can I assume a static page structure too, the link structure of the site isn't changing much?
Interviewer: Start with that assumption, you can think how to weaken this assumption later.
You: Can I assume images are referenced by img tags, and they're not embedded in some other medium like Flash?
Interviewer: Yes. The images will be easily accessible like that.
You: Because there's so much data, this process may take a long time.
Interviewer: Yes. Think about your design in the context of this huge scale.
You: And probably the websites will detect you're crawling them and block you.
Interviewer: True. Assume that doesn't happen, or that you can circumvent it.

At this point, there's still plenty of ambiguity, but assume reasonable answers of your choice for your remaining questions.

Design the system.