Problem Solving Workshop #36 Tech Interviews and Competitive Programming Meetup

July 21, 2018 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, "Land Purchase"

You have a 2D array representing an MxN grid of land. For each square of the grid, there is an associated cost of purchasing the land at that location. You want to purchase a rectangular (axis-aligned rectangle) plot of land having maximum area, subject to a given maximum budget B.

Example:
Input:
Grid (4x4):
6 6 2 5
3 5 9 7
2 2 2 3
1 4 3 3

B = 21

Output: 4x2 rectangle, left upper corner at row 2, col 0.
Explanation:
6 6 2 5
3 5 9 7
2 2 2 3
1 4 3 3

The highlighted segment has total cost 2+2+2+3+1+4+3+3 = 20, which fits within the budget (you don't have to get the exact budget total, you just can't exceed it). Out of all rectangles you could have picked, this 4x2 rectangle has the largest area (8) out of all the rectangles that have total cost <= the budget. You can assume costs and the budget total are positive integers. If there are several solutions having the same maximum area, return any one of them.

(i) **[Easy-Medium]** Consider the one dimensional case of this problem. That is, given a 1D line of land specified as a 1D array and a budget (e.g. [6, 3, 3, 4, 5], B = 10), find the longest contiguous stretch of land fitting the budget. Here it would be [3, 3, 4]. A good solution is linear time here, O(N).

(ii) **[Medium]** Solve the 2D version of the problem as originally stated earlier. Your solution is not expected to be linear time, but try to improve as much as you can beyond naive time complexity.

(iii) **[Medium-Hard]** Suppose the input is very large and you'd like to speed up the algorithm substantially, even if it means that you might not always find the optimal solution. Come up with an efficient approximation for the solution to this problem. This should be some efficient algorithm that (for example) finds a solution that is always within some factor M of the optimal solution. (If M=2, the algorithm finds a solution that's at least half as good as optimal. Or if M= 1.01, the solution would be 1/1.01 of the optimal solution.)

Problem #2, "Knight Dialer"

Consider a phone dial pad:

1	2	3
4	5	6
7	8	9
*	0	#

Suppose that a chess knight starts on the 0 square of the grid. The knight can now move the way it does in chess (move in L-shapes where you move two squares in one direction and one square in the perpendicular direction). For example, you can move to 4 or 6 from 0; from 4 you can move to 3 (2 right, 1 up), 9 (2 right, 1 down), or 0 (1 right, 2 down). You have to stay in-bounds as you move, and you have to dial a number, which means you can't land on * or #.

(i) If the knight starts on 0, how many different N-digit numbers can be dialed? Example: for N=3 the answer is 6. You can dial 040, 043, 049, 060, 061, 067. **[Easy]** exponential-time solution; **[Medium]** O(N), assuming arithmetic operations take O(1). (Arithmetic doesn't necessarily take O(1) here because the numbers could be super large, but let's ignore that.) **[Medium-Hard]** O(log N) solution.

(ii) **[Easy]** If you removed the bottom row (only 3x3 grid having 1-9 remains), and the knight starts on 1, how many N-digit numbers can be dialed? (***Hint:*** there's a reason this is being asked. Don't just say "same concept as first problem.")

Problem #3, "Scalable 3-Sum" (Medium-Hard)

You're probably familiar with the classic 3-sum problem: find 3 numbers in an array (at distinct array locations) that sum to a target number T. (Solve that problem first if you are not familiar with it. You can also look up "3-sum problem". Make sure you understand some of the common solutions to it.) Let's say in this case that the numbers are 64-bit long integers.

(i) Let's say that, on a single embedded processor, you only have enough memory to load 10^4 numbers at once, but the input is 10^6 numbers, on disk. (You have extra free disk space available, too.) How would you solve the problem?
(ii) Let's say we're talking 10^{10} numbers and machines whose memory can hold 10^9 numbers. How would you solve this with a distributed system (e.g. thousands of machines)?

Problem #4, "Probability is Hard"

(***Note:*** direct math questions like this on probability are not usually encountered in technical interviews. However, this was a fun problem, and is good training for improving thinking about probability, which can come into play in problems that do show up.)

You take a fair 6-sided die and keep rolling it until you get a 6. What is the expected value for the number of rolls this will take, **given that** you know all digits rolled prior to the 6 were even (that is, 2 or 4, since you stop on 6)?

Hint: if you think you know the right answer, or you don't, you could write a program to simulate sone sequences. Might show you why you're wrong, or how to get the right solution.