Problem Solving Workshop #22 Tech Interviews and Competitive Programming Meetup

April 29, 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, "Unique Random Number Generator"

You want to create a class that can be used to generate random numbers, but with the caveat that every random number that is generated should be chosen **uniformly at random** from the numbers that have not yet been generated. You want something that can be initialized like:

```
UniqueRandomNumberGenerator r = new UniqueRandomNumberGenerator (2000)
```

where the parameter of 2000 means you'll be generating numbers from 0-1999 for the life of the object. Now you should be able to call `r.nextRandom` repeatedly, and every time, it should return a number in that range uniformly at random out of those that haven't been returned yet. (You should return an error if no more numbers are available.)

You already have a regular random number generator (that doesn't guarantee unique numbers) that has the interface `getRandomInt(int max) -> int`, which takes a positive int `max` and returns an integer uniformly at random between `0` and `max - 1`. Using this, build your unique random number generator.

(a) **[Easy]** Your solution can have any time complexity.
(b) **[Medium]** Every random number should be generated efficiently (e.g. logarithmic or constant expected amortized time), assuming the underlying "regular" number generator takes O(1) time to generate a random number. **Guidance**: think carefully about whether your solution has any cases where numbers will be generated efficiently sometimes, but not always.
(c) **[Medium]** In addition to (b), your solution should satisfy the condition that the space taken is at most O(K) where K is the number of random values generated so far. In other words, if a user creates `new UniqueRandomNumberGenerator (2000000000)` but then only draws a few numbers, it should not take `2000000000` time or space to initialize this data structure, since that behavior could be surprising to the user.

(d) **[Medium-Hard]** In addition to (c), your solution should not have any long pauses between generating successive random numbers, since this might be used in a real-time application. In other words, your time complexity can't be amortized.

Problem #2, "Nearest Numbers" (Medium)

Design a data structure that maintains a set of integers and can support all of the following operations efficiently:

1) Insert a given integer.
2) Remove a given integer.
3) Answer the question "what is the smallest absolute difference between any two numbers in the set?" (in other words, what are the two numbers in the set that are closest together on a number line?)

For example, if we insert 10 and 5 into the data structure, and then we do a nearest-number query, we should get back the result of 10 - 5 = 5. Then, if we insert 7, the set of numbers is now {10, 5, 7}, so the two nearest numbers are 5 and 7, and a query now would produce the result 7 - 5 = 2. Then, if we delete 5, the numbers in the set would be {10, 7}, and a query now would produce 10 - 7 = 3.

A query on a set with fewer than 2 elements is not well-defined and should be an error. For simplicity, assume you won't deal with duplicate elements. Follow-up: How would you deal with duplicate elements?

Problem #3, "Crocodiles" (Hard)

You are in a castle having N rooms numbered from `0` to `N-1`, and M bidirectional passageways between rooms (specified like (0, 1), (1, 2) to indicate a passageway between rooms 0 and 1 and a passageway between rooms 1 and 2). One of the rooms is your "starting room", and some subset of the rooms are "exit rooms" that lead to an exit from the castle. The castle is the lair of an evil mastermind who had you trapped there, and you're trying to escape as quickly as possible. It takes you one unit of time to cross a passageway, and if you are in an exit room, you can escape immediately (0 units of time).

(a) What is the shortest time in which you can escape? (Note: at this point, this problem is a slight variation on a classic problem. This is just a warmup.)

Now, the evil mastermind happens to have K pet crocodiles. Every time you are inside a room, the mastermind will choose up to K passageways into which to place a crocodile (the mastermind may rearrange the crocodiles every time you enter a new room; that is, every time you move). Passageways containing crocodiles cannot be crossed, and as such, the mastermind hopes to delay your escape as much as possible.

Assuming the mastermind will play an **optimal** strategy to delay you as much as possible, what is the best time in which you are **guaranteed** to be able to escape?

The input consists of the value N (number of rooms), a list of all the passageways, the number of your starting room S, the set of rooms that are the exit rooms, and the parameter K (number of crocodiles).

Follow-up: what if the passageways have a travel time associated with them (e.g. one may take 3 units of time and another take 5 units of time)? How would you modify your approach?

Take-Home Problems

1. **[Medium]** You're given an array of N integers, and an integer K >= 1. How many contiguous subarrays of the array are multiples of K?
2. **[Medium-Hard]** Given a string S, find the longest repeated substring. For example, if you have S = "abcdabc", **abc** is the longest repeating substring since it's the longest substring that occurs more than once. Consider both the version of the problem where we allow the repeating substrings to overlap and the case where we don't. Follow-up: how about longest repeating anagram (two substrings that are anagrams of each other)?