Problem Solving Workshop #44 Tech Interviews and Competitive Programming Meetup

March 10, 2019 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, geeksforgeeks.org, hackerrank.com
Books: Cracking the Coding Interview, Elements of Programming Interviews
Have questions you want answered? Contact the instructor on Meetup, 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, "Country Club" (Easy)

N players come together for a sports game. They need to be divided into teams consisting of K players each. Each player has a country they represent, and to promote diversity, each team should have all of its players be from different countries; that is, no two players who are on the same team should be from the same country. Given a list showing which country each player is representing, and the parameter K, what is the maximum number of teams you can form?

Example Input: [England, Egypt, England, England, USA, USA, England, England], K = 2
Expected Output: 3
Explanation: You can form the teams [England, USA], [England, Egypt], [England, USA]. At this point you'd only have 2 players left over and they're both from England, so a 4th team can't be formed, because then both team members would be from the same country. There's no other possible arrangement that results in 4 teams that have all their team members from distinct countries, either, so the maximum number of teams you can form is 3.

Problem #2, "Max Sum Subarray" (Easy-Medium)

This is a classic problem. Given an array A, find the contiguous subarray that adds up to the maximum sum.

Example Input: [5, -7, 3, 2, -1, 4, -10]
Expected Output: start = 2, end = 5, sum = 8.
Explanation: The maximum sum subarray starts at index 2 (0-indexed) and ends at index 5. This is the subarray [3, 2, -1, 4]. This subarray has a sum of 8. No other contiguous subarray has a higher sum.

Problem #3, "Sequential"

You're given an MxN grid of letters. Each row of the grid is interpreted as a word, though it can be gibberish and doesn't have to be any recognizable word in any particular language. For example, if the grid is

X C A C
Y D D B
X D D A

The words are "XCAC", "YDDB", "XDDA".

You want to make these words sorted lexicographically from top to bottom (dictionary order). You're not allowed to remove any of the rows to do so, but you can delete some of the columns. When you delete a column, you remove the entire column (so you're removing one letter, in the same position, from each word). For example, if you decide to remove the leftmost column, the grid will now look like:

C A C
D D B
D D A

...and the words now read "CAC", "DDB", and "DDA". In this case, they're still not in correct lexicographic order, because "DDB" comes after "DDA" in dictionary order, but "DDB" appears above "DDA" in the grid.

Question [Medium]: Given the MxN grid, what is the minimum number of columns you need to remove in order to make the words lexicographically sorted from top to bottom?

In the example shown here, if you remove the leftmost and rightmost columns from the initial (4-column) input, keeping the middle 2 columns, the words will be ["CA", "DD", "DD"] from top to bottom, which are in lexicographic order. This involved removing 2 columns. There's no way to do it by removing only 1 column, so the desired output is: **2**

Follow-up [Medium-Hard]: What if now, you're not allowed to delete columns, but you can delete rows? Obviously, if you delete all rows except one, the collection will be in lexicographic order (because one row is always trivially in order, considering it has nothing else to be in order relative to). Sometimes, it would be possible to delete fewer rows, however. Find the minimum number of row deletions.

Problem #4, “Longest Subsequence From Dictionary”

You’re given a string S and a dictionary D of words. Find the longest word in D that is a subsequence of S.

The definition of “W is a subsequence of S” is that if we delete zero or more characters from S, we can get W with no further rearrangement of the characters. For example, “aba” is a subsequence of “adba” (the “d” can be deleted”), but not of “aabcd”.

Example Input:
S = “bappbable”
D = [“able”, “apple”, “kangaroo”]
Expected Output: “apple”
Explanation: “able” and “apple” are both subsequences of S. Of these two, “apple” is the longer word. “kangaroo” would have been even longer, but is not a subsequence of S.

When analyzing time complexity, you can think about expressing it in terms of (some of) these parameters:

- W, the number of words in the dictionary
- N, the length of S
- L, the average word size in the dictionary

This problem can be any range of difficulty depending on the time complexity attained. It is quite easy to get to a simple solution and quite tricky to get to the optimal solution.

(a) **[Easy-Medium]** O(NW) running time.
(b) **[Medium-Hard]** Achieve a significant improvement upon the previously-mentioned time complexity. Generally, we might expect that N will be much greater than L, since the words in the dictionary have to be found inside the string S. Try to see why O(NW) is not very good in that case.

Complexity Goals

1. O(N log N) is a good solution. It's actually possible to solve the problem in O(N) too.
2. O(N) is possible. O(N log N) is not too bad.
3. The original question can be solved in O(MN) time, which is clearly the best possible complexity. The follow-up can be solved in $O(M^2N)$ where M is the number of rows and N is the number of columns.
4. Like the problem already mentions, a good complexity for a simple solution (naive but not too naive) is O(NW). When N (length of S) is similar in magnitude to L (avg. dictionary word size), this is actually optimal, since then the dictionary has LW = ~NW chars, and you have to read all of them. But, when N >> L, you could hope for a solution that is O(N + LW): linear in the total size of the input. It is possible to get this optimal O(N + LW) complexity, but it is tricky. You could get O((N+LW) log N) and that would still be fairly good. Also, some solutions (depending on the approach) may get a good time complexity but only in the case where the letters are in a small range such as A-Z. In that case, an extra challenge is to avoid depending on the letters being in a small range.