

Wednesday 3/9/22

Coding interview on Number theory:

https://www.youtube.com/watch?v=LyCcBOTb5Fo&ab_channel=CodingInterviewPrep

https://www.youtube.com/watch?v=6YdVek9XSdI&ab_channel=CodingInterviewPrep

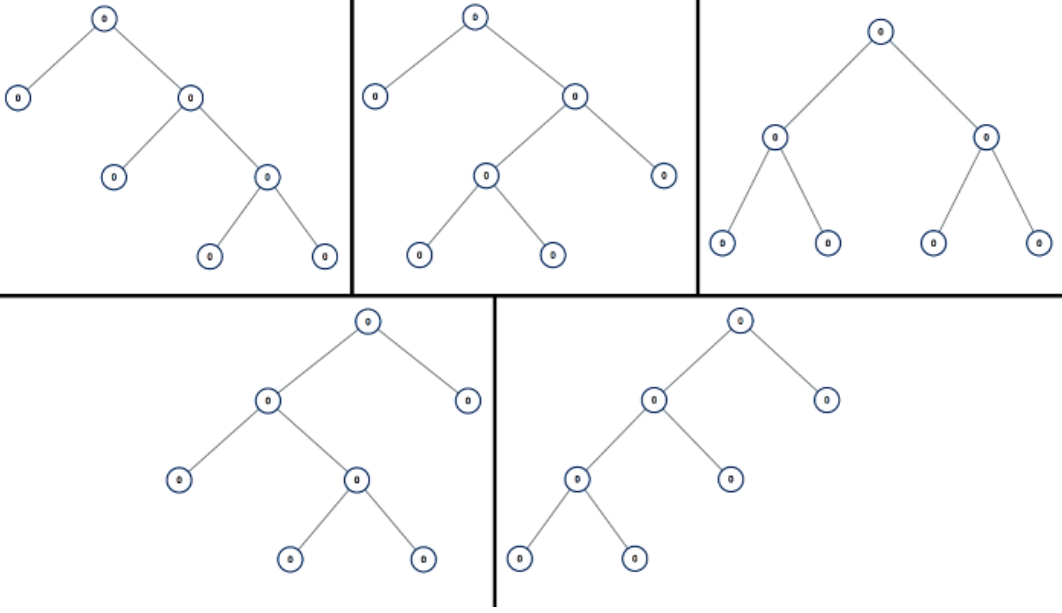
https://www.youtube.com/watch?v=7I92aIYuF2M&ab_channel=MiranFattah

Virtual table: https://www.youtube.com/watch?v=Qn719zRFyao&ab_channel=EmbeddedOS

Pi estimates with rand():

https://www.youtube.com/watch?v=pvimAM_SLic&ab_channel=JomaTech

HW6:

1	<p>Given an integer n, return a list of all possible full binary trees with n nodes. Each node of each tree in the answer must have <code>Node.val == 0</code>.</p> <p>Each element of the answer is the root node of one possible tree. You may return the final list of trees in any order.</p> <p>A full binary tree is a binary tree where each node has exactly 0 or 2 children.</p> <p>Example 1:</p>  <p>Input: $n = 7$ Output: [[0,0,0,null,null,0,0,null,null,0,0],[0,0,0,null,null,0,0,0,0],[0,0,0,0,0,0,0,0],[0,0,0,0,0,null,null,0,0],[0,0,0,0,0,0,null,null,0,0]]</p> <p>Example 2: Input: $n = 3$</p>
---	---

	<p>Output: <code>[[0,0,0]]</code></p> <p>Constraints:</p> <ul style="list-style-type: none"> • $1 \leq n \leq 20$
2	<p>Return the result of evaluating a given boolean expression, represented as a string.</p> <p>An expression can either be:</p> <ul style="list-style-type: none"> • <code>"t"</code>, evaluating to True; • <code>"f"</code>, evaluating to False; • <code>"!(expr)"</code>, evaluating to the logical NOT of the inner expression <code>expr</code>; • <code>"&(expr1,expr2,...)"</code>, evaluating to the logical AND of 2 or more inner expressions <code>expr1</code>, <code>expr2</code>, ...; • <code>" (expr1,expr2,...)"</code>, evaluating to the logical OR of 2 or more inner expressions <code>expr1</code>, <code>expr2</code>, ... <p>Example 1: Input: <code>expression = "!(f)"</code> Output: <code>true</code></p> <p>Example 2: Input: <code>expression = " (f,t)"</code> Output: <code>true</code></p> <p>Example 3: Input: <code>expression = "&(t,f)"</code> Output: <code>false</code></p> <p>Constraints:</p> <ul style="list-style-type: none"> • $1 \leq \text{expression.length} \leq 2 * 10^4$ • <code>expression[i]</code> consists of characters in <code>{'(', ')', '&', ' ', '!', 't', 'f', ' '}</code>. • <code>expression</code> is a valid expression representing a boolean, as given in the description.
3	<p>Special binary strings are binary strings with the following two properties:</p> <ul style="list-style-type: none"> • The number of 0's is equal to the number of 1's. • Every prefix of the binary string has at least as many 1's as 0's. <p>You are given a special binary string <code>s</code>.</p> <p>A move consists of choosing two consecutive, non-empty, special substrings of <code>s</code>, and swapping them. Two strings are consecutive if the last character of the first string is exactly one index before the first character of the second string.</p> <p>Return the lexicographically largest resulting string possible after applying the mentioned operations on the string.</p> <p>Example 1: Input: <code>s = "11011000"</code> Output: <code>"11100100"</code> Explanation: The strings <code>"10"</code> [occurring at <code>s[1]</code>] and <code>"1100"</code> [at <code>s[3]</code>] are swapped. This is the lexicographically largest string possible after some number of swaps.</p>

	<p>Example 2: Input: s = "10" Output: "10"</p> <p>Constraints:</p> <ul style="list-style-type: none"> • $1 \leq s.length \leq 50$ • s[i] is either '0' or '1'. • s is a special binary string.
4	<p>There are n friends that are playing a game. The friends are sitting in a circle and are numbered from 1 to n in clockwise order. More formally, moving clockwise from the ith friend brings you to the (i+1)th friend for $1 \leq i < n$, and moving clockwise from the nth friend brings you to the 1st friend.</p> <p>The rules of the game are as follows:</p> <ol style="list-style-type: none"> 1. Start at the 1st friend. 2. Count the next k friends in the clockwise direction including the friend you started at. The counting wraps around the circle and may count some friends more than once. 3. The last friend you counted leaves the circle and loses the game. 4. If there is still more than one friend in the circle, go back to step 2 starting from the friend immediately clockwise of the friend who just lost and repeat. 5. Else, the last friend in the circle wins the game. <p>Given the number of friends, n, and an integer k, return the winner of the game.</p> <p>Example 1:</p> <p>Input: n = 5, k = 2 Output: 3 Explanation: Here are the steps of the game:</p>

	<p>1) Start at friend 1. 2) Count 2 friends clockwise, which are friends 1 and 2. 3) Friend 2 leaves the circle. Next start is friend 3. 4) Count 2 friends clockwise, which are friends 3 and 4. 5) Friend 4 leaves the circle. Next start is friend 5. 6) Count 2 friends clockwise, which are friends 5 and 1. 7) Friend 1 leaves the circle. Next start is friend 3. 8) Count 2 friends clockwise, which are friends 3 and 5. 9) Friend 5 leaves the circle. Only friend 3 is left, so they are the winner.</p> <p>Example 2: Input: $n = 6, k = 5$ Output: 1 Explanation: The friends leave in this order: 5, 4, 6, 2, 3. The winner is friend 1.</p>
5	<p>You are given a positive integer p. Consider an array <code>nums</code> (1-indexed) that consists of the integers in the inclusive range $[1, 2p - 1]$ in their binary representations. You are allowed to do the following operation any number of times:</p> <ul style="list-style-type: none"> Choose two elements x and y from <code>nums</code>. Choose a bit in x and swap it with its corresponding bit in y. Corresponding bit refers to the bit that is in the same position in the other integer. <p>For example, if $x = 1101$ and $y = 0011$, after swapping the 2nd bit from the right, we have $x = 1111$ and $y = 0001$.</p> <p>Find the minimum non-zero product of <code>nums</code> after performing the above operation any number of times. Return this product modulo $10^9 + 7$.</p> <p>Note: The answer should be the minimum product before the modulo operation is done.</p> <p>Example 1: Input: $p = 1$ Output: 1 Explanation: <code>nums</code> = [1]. There is only one element, so the product equals that element.</p> <p>Example 2: Input: $p = 2$ Output: 6 Explanation: <code>nums</code> = [01, 10, 11]. Any swap would either make the product 0 or stay the same. Thus, the array product of $1 * 2 * 3 = 6$ is already minimized.</p> <p>Example 3: Input: $p = 3$ Output: 1512 Explanation: <code>nums</code> = [001, 010, 011, 100, 101, 110, 111] - In the first operation we can swap the leftmost bit of the second and fifth elements. - The resulting array is [001, 110, 011, 100, 001, 110, 111]. - In the second operation we can swap the middle bit of the third and fourth elements.</p>

	<p>- The resulting array is [001, 110, 001, 110, 001, 110, 111]. The array product is $1 * 6 * 1 * 6 * 1 * 6 * 7 = 1512$, which is the minimum possible product.</p> <p>Constraints:</p> <ul style="list-style-type: none"> • $1 \leq p \leq 60$
6	<p>The set [1, 2, 3, ..., n] contains a total of n! unique permutations. By listing and labeling all of the permutations in order, we get the following sequence for n = 3:</p> <ol style="list-style-type: none"> 1. "123" 2. "132" 3. "213" 4. "231" 5. "312" 6. "321" <p>Given n and k, return the kth permutation sequence.</p> <p>Example 1: Input: n = 3, k = 3 Output: "213"</p> <p>Example 2: Input: n = 4, k = 9 Output: "2314"</p> <p>Example 3: Input: n = 3, k = 1 Output: "123"</p> <p>Constraints:</p> <ul style="list-style-type: none"> • $1 \leq n \leq 9$ • $1 \leq k \leq n!$