Problem Solving Workshop #47 Tech Interviews and Competitive Programming Meetup
September 29, 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.

"Stairway To Heaven" (Easy)

You have a staircase of N steps. You can climb 1, 2, or 3 steps at a time. How many distinct ways are there to climb the staircase?

For example, one possible way to climb a staircase of 4 steps: climb 1 step, then climb 2 steps, then climb the last step.

Example Input: N=4
Expected Output: 7
Explanation: the possible ways to climb it are [1, 1, 1, 1], [1, 1, 2], [1, 2, 1], [1, 3], [2, 1, 1], [2, 2], [3, 1]. So, the correct output is 7 because there are 7 ways to climb.

Follow-Up: what if you get "tired" after climbing 3 steps and cannot climb 3 steps twice consecutively? That is, after climbing 3 steps, you must next climb 1 or 2 steps. You can climb 3 steps again after that.

"Grid Paths" (Easy)

You have an MxN grid. You start at the upper left corner, and you have to reach the lower right corner, moving either one cell to the right or one cell downward in each step. Additionally, some cells are blocked off and not traversable. You're given a boolean matrix of size MxN that represents which cells cannot be moved to.

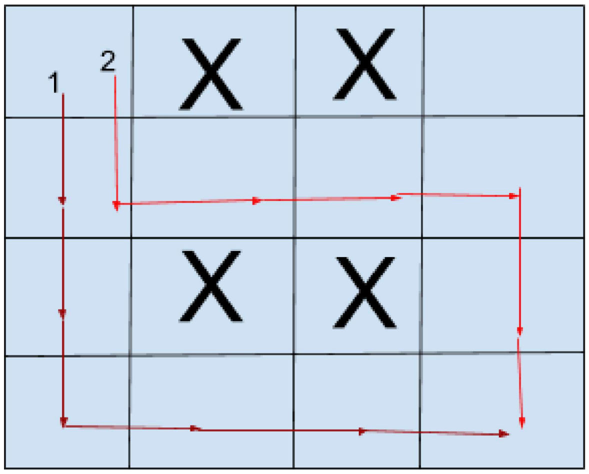
How many distinct paths can you take to get from the upper left corner to the lower right corner?

Example Input (2D array, T = true, F = false, about whether a cell is blocked):

```
FTTF
FFFF
FTTF
FFFF
```

Expected Output: 2

Explanation: This 4x4 grid has 2 distinct paths that involve moving only down and right (X's are blocked-off cells):



"Image Compression" (Hard)

As you might know, grayscale images in uncompressed format are often represented as a 2-D array of pixels, where each pixel is specified as a 0-255 value: a value of 0 means the pixel is completely black, a value of 255 means the pixel is completely white, and values in between are various shades of gray. Here we'll allow the 255 to generalize and call the parameter R.

We can perform compression of the image by choosing a "color palette" of K distinct values, with each value between 0 and R. Then, we change each pixel's value to the numerically closest value in the palette.

For example, if R = 255, normally 8 bits would be required to represent each pixel, as there are 256 possible values, 0 to 255, for the pixel. If K=8, each pixel could be represented with only 3 bits: the 3 bits identify which of the 8 values in the palette the pixel will take. Additionally, the image will store information about the palette colors, but this uses negligible space because this information is stored once for the entire image and is not per-pixel. You can think of smaller values of K as resulting in stronger compression at the cost of greater quality loss.

It may seem that the best way to choose the K values would be to pick 0, R/(K-1), R * 2/ (K-1),...., R (uniformly spaced), but this may not be so if the distribution of pixels in the image is not uniform. To reduce the amount by which this compression distorts the image, we instead want to minimize a least-squares cost function:

Cost = sum over all pixels [(original pixel value - new pixel value after compression)^2]

Given an image, and the parameter K, return the set of K choices that results in minimizing the cost (if there are several, choose any one). For example, if the image is standard grayscale with R = 255, if K = 4 maybe we'll pick a set like {25, 50, 75, 255} (depends on the image).

"Robot Mines" (Hard)

You have N mines in a minefield that need to be cleared by futuristic robots. When a robot finds a mine, it steps on it, destroying the mine and the robot.

The trouble is, you only have one robot, and potentially more than one mine. Luckily, the robot can also build other robots. It takes a robot T units of time to build another robot, during which it can't do anything else. The new robot is identical in every respect to the original, including the ability to build more robots in turn. You have unlimited raw materials with which more robots can be constructed.

For each mine i, you have some value M_i that denotes how long it will take for a robot to find and destroy that mine. When you send a robot to destroy a mine, the command has to be completed all the way (robot cannot be recalled). The robot can't do anything else while it's finding the mine, and is destroyed together with the mine when the command is complete. Multiple robots can search for different mines at the same time, but two robots can't search for the same mine because they'd get in each other's way.

Given the integer T that denotes how long it takes to build a robot, and a list M of mine search times (integers), find the minimum amount of time needed to clear the minefield.

Example Input: M = [2, 9, 2, 2], T = 3
Output: 12
Explanation: At time 0, we start with 1 robot. We command the robot to buid another robot, so at time 3 (build time T = 3) we have 2 robots. We command one of them to find the "9" mine, which will finish at time 12 (current time is 3 + mine time is 9). We command the other robot to build a robot. That means at time 6 (current time is 3 + build time T is 3) we have two available robots. At time 6, we command both of them to build more robots. At time 9 we have 4 available robots, and we send 3 of them to get each of the "2" mines, which get destroyed at time 11. The 9 mine is clear at time 12 , and so the whole process takes 12 units of time.

"Traveling Salesman" (Hard)

This is a classic problem. Given a directed, weighted graph with positive-weight edges, find the lowest-cost path that visits every vertex **exactly once**.

This problem was called "traveling salesman" because of a hypothetical scenario where a salesman wants to visit every city in a region at the lowest possible cost. The expected solution here is exponential time, but using dynamic programming, you can get a better exponential-time complexity than the naive

O(n!) method of trying every sequence of vertices. Think about what history you need to maintain as you visit the cities.