Problem Solving Workshop #33 Tech Interviews and Competitive Programming Meetup

February 25, 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, "Division Without Division" (Easy-Medium)

Given two integers a and b (both can be positive, negative, or zero), your task is to design an algorithm for computing (as efficiently as possible) the quotient b/a without using the division operator (suppose you're on a machine where a division operation is not present, or very slow). Your division code should behave the same way as the / operator typically does in languages such as Java or C++ (truncating division).

(a) You can use any operation except division or close equivalents.
(b) Use only the + and - artihmetic operations, and still keep your algorithm efficient.

Example Input: a = -15, b = 4
Output: -3

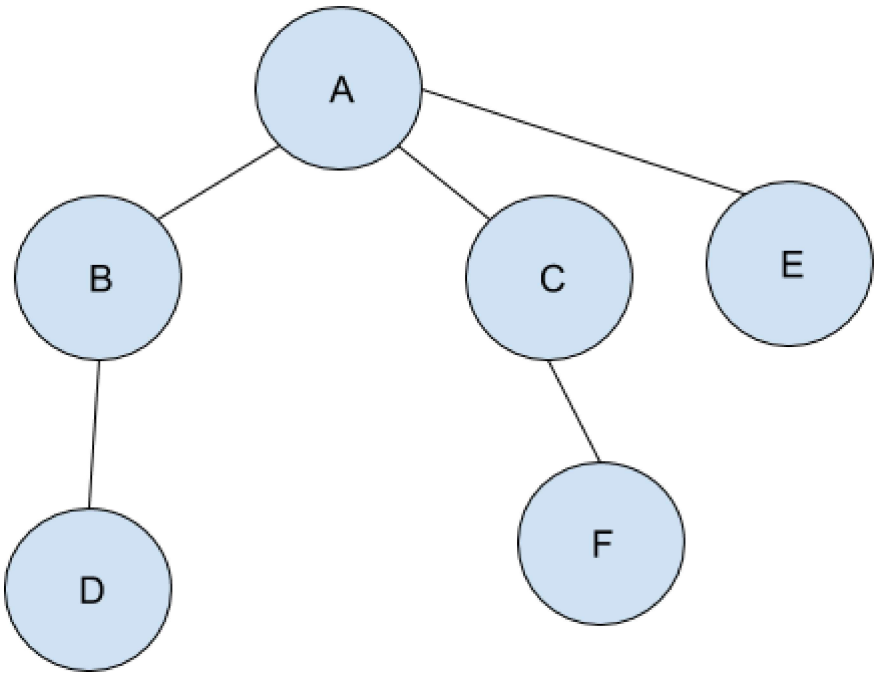
Problem #2, "Minimum Distribution Time"

You have a network of machines and links between them such that the network is a tree. That is, there is a unique path of links that connects any two machines (no cycles in the links).

We want to find out which node in the network can transmit a message to all the other nodes in the minimum time. A message would be transmitted like this: a node starts with the message at T=0. In one unit of time, it can choose any of the nodes it has a direct link to, and transmit a message to that node (the node would receive the message at T=1). Then, it could further transmit messages to other nodes it has a link to (at T=2, a second node could receive the message). Any nodes that receive the message can themselves start transmitting them.

Given a tree, return the label of the node that can get the message out to other nodes the fastest (if there are multiple that are tied, return any one), and also return the time it will take.

Example Input:



Output: node = A, time = 3
Explanation: Node A can send a message to B, with B getting it at T=1. Then A sends a message to C, C getting it at T=2. Finally, E gets it at T=3. At the same time, B sends the message to D. Since B can only start transmitting it at T=1 (when B has received it), D will get it at T = 2. By similar logic, C will start transmitting to F at T=2 and F will receive the message by T = 3. So, by T=3, all nodes will get the message.

Target time complexity: [Easy] $O(n^2)$ time, [Medium] O(n) time.

Problem #3, "Image Compression" (Medium-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 some K (K < R) distinct values (each value between 0 and R), and then changing each pixel's value to the nearest of the chosen values. For example, if K=8, each pixel could be represented with only 3 bits: the 3 bits identify which of the 8 chosen values the pixel will take, and then there is a single table for the entire image that indicates what the 8 chosen

values are. 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 chosen 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).

Don't necessarily expect to get a linear time solution here, but maybe you can do much better than exponential time?