Problem Solving Workshop #34 Tech Interviews and Competitive Programming Meetup
March 17, 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, "Bits and Flips"

You're given an initial array of bits (booleans), and a sequence of bit-flipping operations. Each operation is defined by two indices i1 and i2 (i2 >= i1) and flips all the bits at indices from i1 to i2, inclusive, to the opposite value. Determine the final values of all the bits in the array after all the operations.

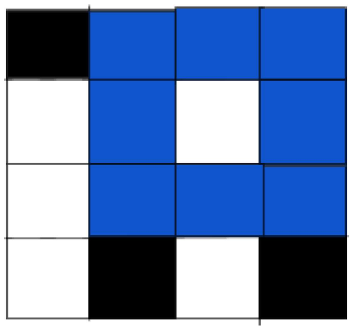
Example Input: array = [0, 1, 1, 0, 1, 0], sequence = [(i1=2, i2=4), (i1=3, i2=5)]
Output: [0, 1, 0, 0, 1, 1]
Explanation: The first operation changes the initial array [0, 1, 1, 0, 1, 0] ->[0, 1, **0, 1, 0,** 0] (the array is 0-indexed). The second operation then changes [0, 1, 0, 1, 0, 0] to [0, 1, 0, **0, 1, 1**].

Complexity Guidance: [Easy] O(NM) if array length is N, and there are M operations. **[Medium]** do much better than O(MN). The optimal solution is O(M+N).
Hint: what determines whether a value will be the same after all operations are done as it was in the beginning?

Problem #2, "Spacious Square"

You're given an NxN grid of booleans. A "true" value represents a black pixel, and a "false" value represents a white pixel. Your goal is to detect the largest square of black pixels present in the grid. A square has to have black pixels everywhere on its perimeter, but it does not have to have its interior filled in (although, the case where it does is another interesting variation of this problem). Find the position and size of the largest square in the grid (return any largest square if there are multiple that are tied).

Example Input:
```
[T T T T
 F T F T
 F T T T
 F T F T]
```
This input represents this data:



All the filled-in squares (blue and black) correspond to the boolean true values. The solution square is highlighted in blue.

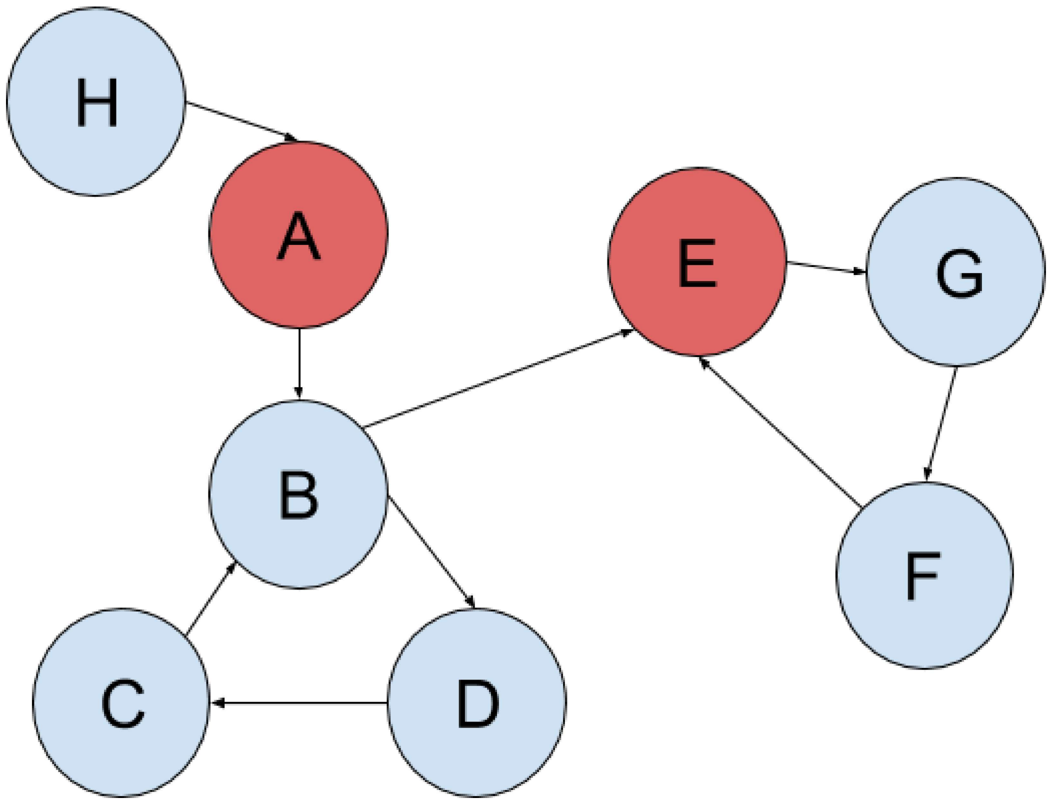
Output: Square of size 3 at position (1, 0)
Explanation: We have a square of size 3 with left-upper corner (1, 0), and there is no larger square.
Complexity Guidance: [Easy] O(N^4) **[Medium]** O(N^3) **[Hard]** O(N^2logN)

Problem #3, "Roaming Robot" (Hard)

You have a network of cities and unidirectional (directed) roads between them. A robot will start at some city, consider all the outgoing roads from that city, and pick one of them randomly, at which point it will take that road to get to a new city. It will then pick another road randomly and continue that process forever. Every city has at least one road leaving it, so the robot can never get stuck in a city. Note that it's entirely possible that the robot could make different choices with repeat visits to the same city.

Some cities, but not others, have refueling stations that refuel the robot when it passes through. The robot takes a very long time to run out of fuel, but eventually it could run out, and we want to detect in what situations this could happen. Given the city network and a list indicating which cities refuel the robot, determine for which starting cities it is ***at all possible*** that the robot could, at some point in its future, go arbitrarily long without receiving fuel if it gets unlucky. That is, for which starting cities does the robot not have a guaranteed maximum time between refuelings forever into the future? The example will clarify this further:

Example Input:
Network:



(The arrows have directions -- make sure to look closely!)
Refueling cities: [A, E] (shown in dark pink above).

Expected output: [H, A, B, C, D]
Explanation: If the robot starts in any of those 5 cities, it is possible that once the robot gets to B, it will keep picking D, then C, then B again, then D again, and so on forever. Even if it seems improbable, there's no ***guaranteed*** time by which it would escape the loop, and if it doesn't, it wouldn't be getting refueled inside that loop. However, if the robot starts at E, F, or G, it is guaranteed it will always move E-> G -> F -> E -> G -> F -> ... and so on forever (only choice), Since E refuels the robot, the robot would never go too long without fuel regardless of its luck.

Follow-up (not expecting anyone to get this far during the session): For which starting nodes does the robot have a non-vanishing probability, after many moves, of reaching a state where it's never refueled again? For example, in the above graph, there are no such starting nodes, because eventually, with probability approaching 1, the robot will get stuck in the E, G, F loop, which guarantees refueling. In this variation we are essentially not allowing the robot to have "infinitely" bad luck.