Problem Solving Workshop #29 Tech Interviews and Competitive Programming Meetup

October 1, 2017 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, "Late Stragglers" (Easy)

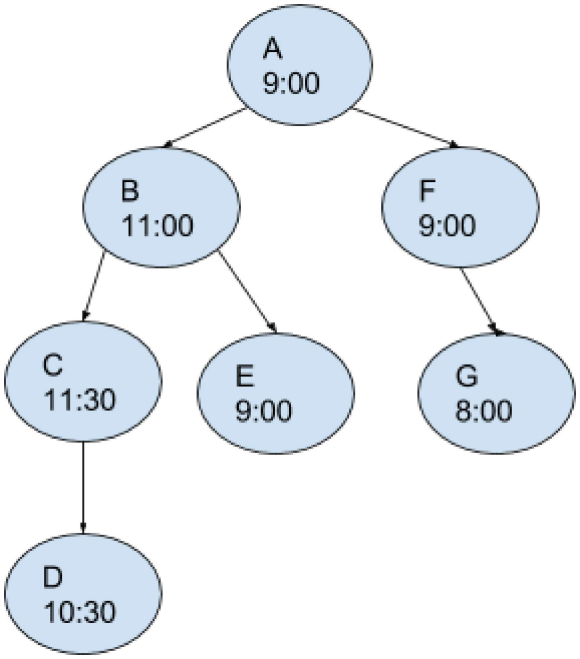
You're given a tree that represents a hierarchy of employees at a company (reporting hierarchy). Each node represents a person, and each person has a time at which they come into work, which is stored in that node.

We say that a person A (directly or indirectly) reports to their boss (parent node in the hierarchy), plus all the people their boss reports to in turn. Given the reporting hierarchy and each person's arrival time to work, find the largest discrepancy across the entire org between the arrival times of an employee and someone they report to.

Note that the tree is not necessarily binary. Each node's structure looks like:

```
class Node {
  List<Node> children;  // Possibly empty if this is a leaf node.
  String employeeName;
  Time arrivalTime;
}
```

Example Input:



Output: C and A, with 2:30 difference.

Explanation: C arrives at 11:30 and A, which C reports to indirectly, arrives at 9:00. These times differ by 2 hours and 30 minutes. (It would have qualified as a difference regardless of whose time is earlier.) This is the biggest difference between an employee and someone they report to in this tree. Note that though the difference between G and C is even larger, that is not the output because G does not report to C or vice versa.

Problem #2, "Trading Places" (Medium)

You're given a string S, and a list L of pairs where each pair (i, j) is a rule that indicates that you may at any time swap the character at position i in S with the character at position j in S. Generate the lexicographically largest (last in dictionary order) string that you can by swapping characters. The same rule (same (i, j) pair) may be reused any number of times to perform as many swaps as needed.

Example Input:
S = "abdc"
L = [(0,3), (2,3)]

Output: dbca

Explanation: The swapping rules are that you can at any time swap S[0] and S[3], or S[2] and S[3]. It is possible to get the strings abcd, abdc, cbad, cbda, dbac, dbca through some sequence of these two swaps. You should print only dbca which is lexicographically largest (last in dictionary order) out of these. You can get it by starting with abdc, using the (2,3) rule to swap positions 2 and 3 to produce abcd, and

then using the (0,3) rule to produce dbca. d**c**ba is even larger lexicographically, but it's not possible to obtain it by swaps (quite obviously since there are no rules that can move anything from position 1).

Complexity Guidance: Something like O(n log n) is a good solution here; better solutions still are possible.

Problem #3, "Dual Paths" (Hard)

(Source: Codeforces, online judge available for a very similar but not identical problem, see below.)

You are given an undirected, unweighted graph with V vertices and E edges (assume any reasonable representation of the graph, such as an adjacency list format). You are then given four distinct vertices in the graph, S1, D1, S2, and D2. What is the smallest set of edges in the graph such that if you deleted all edges except for those in the set, there would exist a path between S1 and D1, and also a path between S2 and D2?

Note that if this problem just required there to be a path between **two** given vertices S1 and D1, this would be a shortest path problem (since the shortest path contains the fewest edges). However, finding the edges satisfying the condition above is not the same as finding the two shortest paths S1 -> D1 and S2 -> D2 in the original graph and then combining them, because it is possible, for example, that you may choose to take some non-shortest path between S1 and D1, if it is a path that has a lot of edges that can be reused by the path from S2 to D2.

Complexity Guidance: $O(n^2)$ is a good solution here.

Online judge for problem. Don't follow this link until you've solved the problem and want to code it, as there may be spoilers: http://codeforces.com/contest/544/problem/D . Note that in the Codeforces problem statement, there are additional constraints that path_length(S1 -> D1) <= L1 and path_length(S2 -> D2) <= L2, for some additional parameters L1 and L2. This doesn't change the problem much conceptually.