COP 4516 Spring 2019 Week 7 Final Individual Contest Solution Sketches

Company Board

This exact problem was discussed (but code wasn't shown) during the binary tree lecture. The general strategy behind solving this problem for a given node is as follows:

1. Recursively solve the problem for all children of the given node.
2. Add up the results from from step #1.
3. Return the maximum of the result from step #2 and the number of connections the given node has, since you can either take the best answers from ALL other subtrees and add them, or JUST take yourself, since you can't choose yourself and anyone below you.

Lastly, the base case is a leaf node (no children). In this case, just return the number of connections of the single node.

Out of Sync

The key realization to solve this problem is noting that if gcd(a, n) = 1, then gcd(a+bn, n) = 1 for any positive integer b. Thus, the number of values relatively prime to n in the range [1, n] is the same as the number of values relatively prime to n in the range [n+1, 2n], and the range [2n+1, 3n], and so forth.

Thus, we can see that there are $\lfloor r/n \rfloor$ set of disjoint ranges of size n we must consider, each with the same number of terms to count. This number, incidentally is φ(n), which is simply defined as the number of integers in the range [1, n] relatively prime to n. In addition to these ranges, we have one potentially "last" partial range, starting at $\lfloor r/n \rfloor * n + 1$ and ending at r. Since n is no more than 10^6, we can simply run a for loop, testing the gcd of each of these values with n, to get the number of values in this sub-range that are relatively prime to n.

The bounds are small enough and the allotted time large enough that instead of calculating the Euler Phi function as shown in class, you can simply run gcd between each integer from 1 to n with n and add 1 for each term which gives a gcd of 1. Thus, this intended solution runs the gcd function at most 2,000,000 times. Once can divide this in half, roughly, but calculating φ(n) using the prime factorization of n.

Pancakes

This is a slight variant of the containers problem shown during the greedy lecture in class. In that problem, one could stack two of the same object on top of each other, but in this problem, the stacks have to be strictly decreasing in size, from the bottom to the top. Secondly, the stacks can get emptied out multiple times, which amounts to running the containers problem multiple times.

The key to the solution to the containers problem was placing each new container on the smallest possible stack that could accept the container, if one existed, or creating a new stack if the container could not be placed on any existing stack. The solution is the same here, just adjusting for the strictly smaller rule, and whenever 0 is read in, add up the number of stacks and start over.

Student Placement

This was intended to be the easiest problem in the set. The intended solution is to notice that we want to place students in the biggest room at all times. Since the total number of students that could ever fit in the rooms will never exceed 10^7, we can actually run a for loop, placing students in the rooms, one by one. But, we can speed up this process to an O(1) calculation by simply adding up the capacity of all the rooms subtracting out the number of students, and dividing by the number of rooms. The basic idea is that using the given information, we know how many total empty seats there will be. Naturally, we want to spread these out as evenly as possible, which indicates integer division. If there are 17 free seats to distribute amongst 4 rooms, we can guarantee that each room has $\left\lfloor \frac{17}{4} \right\rfloor$ free seats each, **so long as each room had that many to begin with.** The bolded observation is key to adjusting this O(1) solution. It's possible that the number produced by this calculation is larger than the capacity of one of the rooms. If this is the case, then the answer would simply be the capacity of the smallest room. So the final result is simply

$$\min(\min(cap), \left\lfloor \frac{freeseats}{numrooms} \right\rfloor)$$

where min(cap) is the minimum capacity of any of the individual rooms, freeseats is the total number of free seats available (sum of all rooms minus students in the class) and numrooms is the number of rooms the students sit in.