COP 4516 Spring 2020 Week 11 Team Contest #3 Solution Sketches

Mixed Set

Since $k \leq 10^6$, it's good enough just to generate the permutations in lexicographical order, using backtracking. Given a particular permutation, just try to add the next possible value. If you can, go with it, if you can't skip it (this is the backtracking) and go to the next value. Keep track of how many valid permutations you've passed until you get to the appropriate rank. At this point, set up your recursion to return true so that no further changes are made. It's key to only store ONE COPY of any permutation and change that one copy as you march through trying each permutation, as opposed to storing each permutation. The latter hogs up memory and slows down the solution.

Paint Me

First, get the total possible area of the rooms to paint. Note that you're not painting the ceiling…Then add up all the doors and windows and subtract this out. Finally, use integer division and mod to calculate the final answer. If we need to paint 1000 sq. ft. and each can paints 16 sq. feet, since 1000%16 != 0, the number of cans of paint we need is 1000/16 + 1. Alternatively, if we were painting 96 sq. feet, since 96%16 == 0, we just need 96/16 sq. ft. The formula I showed you in class previously for ceiling of y/x is (y+x-1)/x.

Perfect

Take the first number, and multiply it by itself over and over again until you either get to the target value (second value) or exceed it. Count how many times you multiplied and DO NOT USE DOUBLES!!! Note that we were really nice and made the bounds so you can use int and not overflow.

Railroad

Let dp[i][j] be true if you can form the first i+j cars of the merged car sequence with the first i cars from the first train and the first j cars from the second train. To solve the recurrence, we check to see if the (i-1) index car in train 1 matches the i+j-1 index car in the big train, AND dp[i-1][j] is true. If both things are true, then dp[i][j] is true. Basically we can form the larger sequence in this case by taking the last car from train 1. Alternatively, check if the (j-1) index car in train 2 matches the i+j-1 index car in the big train, AND dp[i][j-1] is true. If both are true, then we can set dp[i][j] to true. At the end, we just check to see if dp[n][m] is true where the first train has n cars and the second has m cars.

Stick Splitting

This follows the classic Matrix Chain DP pattern. To solve a recursive instance of splitting a sequence of all sticks from index i to index j, try splitting the sequence at each possible split point. The cost for a particular split is simply the length of the whole segment plus the recursive cost of splitting the left and the recursive cost of splitting the right. Store results to recursive calls in memo[i][j] and memoize.

Teamwork

We define f(n) = maximum sum ending at the cow at index n. (In code, we start our indexes at 0.) Consider calculating f(n) for an arbitrary value n:

We would like to try all possible groups for our "last" group of contiguous cows. These groups could be:

{n} or {n-1, n} or {n-2, n-1, n} or … {n-k+1, n-k+2, n-k+3, …, n}

Namely, given the input restrictions, the group size must be in between 1 and k, inclusive. Consider the situation where the last group of cows is {m+1, …, n}. In this situation, our maximum sum of skill levels would be:

$$f(m) + (n - m)*\max(skill[m+1], skill[m+2], \ldots, skill[n])$$

Basically, we get the best answer for any split of the first m cows and add to it the maximum cow from index m+1 to n multiplied by the number of cows in that range, which is n - m.

Thus, we want the maximum value of the expression above, for all valid m, of which there are at most k.

In terms of run time, we have an outer loop that runs n times, once for calculating the function f(i) for each different value of i. To calculate f(i) for a single value i, we have a second loop that runs upto k times. Thus, our run time is at least O(nk). Notice that the maximum bounds are $n = 10^4$and $k = 10^3$, so we already have a run-time around 10^7. This is doable, but we can't add a third nested loop. The key here is that we must be able to calculate the maximum of a set of values in O(1) time. The easiest way to do this is build the last group from smallest to largest, keeping a running maximum. (So, for example, if our list is 8, 3, 7, 6, 9, 4, 5, and we are calculating f(6), we currently store a max = 5, then when add 4 to our last group, our max stays at 5. When we add 9 to our last group, our max changes to 9, these changes occur in O(1) time and right after we finish them, we can calculate f(m) + (n - m)*max(skill[m+1], skill[m+2], …, skill[n]), in O(1) time.