COP 4516 Spring 2023 Week 14 Team Contest #6 Solution Sketches

Game of Gold

Recursively, this problem is viewed as either taking the left most bag of gold or the right most, and then returning the better of the options. A quick analysis will yield that the only recursive calls possible are on consecutive subsequences of the problem, so that we can characterize f(i, j) as the best score for player 1 on the subarray starting at index i and ending at index j. Secondly, if we take the gold at index i, then our opponent's best score is f(i+1, j). This means that for us, we would score -f(i+1, j). Thus, our score when we take index i is arr[i] – f(i+1, j). A similar expression can be calculated if we take index j. Finally, if regular recursion is written, with 2 choices at each branch, the run time would be $O(2^n)$, where n could be as large as 50. Thus, memorization should be used to get the solution to run in time (or straight dynamic programming).

Extra Homework

This is the easiest problem in the set. The bounds on the input are quite small. In fact, it can be proven based on the bounds that no answer will exceed the storage of integer. Since we must output an integer, it's best to keep all calculations in integers and not use the pow function (which returns a double). Either Horner's Rule can be applied, or a nested loop structure, to accumulate the answer.

New Island and Radios

There are two well-known strategies that work here. Both involve recognizing that two towers can talk to one another if and only if they are within a distance of r. Here are the strategies:

(1) Create a complete graph connecting all pairs of towers, setting the cost of each edge to be the least integer radius that allows those two towers to be connected. Then, run Kruskal's Algorithm on these edges. Instead of keeping track of the sum of the edges of the minimum spanning tree, simply note the edge length when the MST is first completed, and this is the answer.

(2) A second idea is to binary search the value for r. When making a guess for r, then go and connect all pairs of towers that are withing a distance of r. Then, run a DFS or BFS on this graph. If the graph is connected, then the actual answer can't be any larger than this guess for r. If the graph isn't connected, then the actual answer has to be at least 1 more than this guess for r.

Sleepy Cow Sorting

The key observation here is that if the "last portion of the array" is sorted, then none of those values have to move. Consider the following example: 2 6 3 7 1 4 5 8

Ideally, since this last portion is in order, we should just be able to insert each of the previous items into that already sorted list. The only issue is figuring out **how many steps** each item should go. For example, when we are inserting 2, it's clear that 2 has to skip over 6, 3 and 7. Then, it has to skip over each item in the sorted list which it's bigger than. Since the input can be large, it's helpful to be able to answer the query: Of the already sorted items, how many are less than the item I am inserting? A binary index tree can handle this sort of query easily. Thus, the algorithm to solve the problem is as follows:

(1) Identify the smallest index x such that a[x], a[x+1],…, a[n-1] is sorted. (Alternatively, find the maximum index y such that a[y] > a[y+1], then it follows that x = y+1.)

(2) Create a BIT of size n. Add each 1 to the BIT in indexes a[x], a[x+1], …, a[n-1].

(3) Process each value, a[i] that isn't already sorted. Calculate how many positions back it should be inserted by adding the number of unsorted values it has to skip over, plus the number of sorted values that are less than it. Then add 1 in the BIT at position a[i].

Rearranging Scrabble Tiles

Due to the size of the input, this problem is fairly easy. The output is simply the number of relative inversions between the two strings. Since the max length of a string is 26, there's no need to use a binary index tree! A double for loop should suffice. There are several ways in which counting the inversions can be implemented.

Zigzag Subsequence

First, let's pretend that all the input values are in between 1 and 10^5. In one of the practice problems (LIS number), we looked at how a binary index tree could be used to could the number of increasing subsequences. To count the number of Zigzag sequences, we must actually keep track of two binary index trees: (1) one which is counting the number of "up" sequences that end at an index, and (2) another which is counting the number of "down" sequences that end at an index. For example, if we want to count how many zigzag sequences that end going "down" at the value 17, in index 10, then we want to know the number of "up" Zigzag sequences that end in a number greater than 17 because we can tack on 17 to the end of those sequences to create a Zigzag sequence that ends in this 17. Similarly, if we wanted to know the number of "up" sequences that end at this 17, we could count the number of "down" sequences that end at a number 16 or lower. In this fashion, for each index, we build the number of "up" and "down" zigzag sequences that end there. Summing this up gives us our answer…almost. The instructions say not to count zigzag sequences of length 1 or 2, thus, care has to be taken to subtract these out. Finally, notice that the input values aren't actually from 1 to 10^5. But, all that matters in determining zigzagging is relative order. Thus, we can take all of the input numbers, put them in an ordered set, so for example, [1,4] for the last two samples), and relabel the numbers by their rank: 1→ 1, 4 → 2. So, the original list 1, 4, 1, 4, 1 becomes 1, 2, 1, 2, 1 and it's guaranteed that all values in the list are in between 1 and 10^5 as desired. This idea in general is called coordinate compression.