Code Jam 2013 - Round 2

Analysis: ErdÅ‘sâ€“Szekeres

Get the information

The real trick to this problem is squeezing as much information as possible from the sequences A[i] and B[i]. We will get information out in the form of inequalities between various elements of the sequence X.

First notice that if we have two indices i < j, and A[i] ≥ A[j], then X[i] > X[j]. Indeed, if it were otherwise, then the increasing sequence of length A[i] with X[i] as its largest element could be extended by adding X[j] at the end to obtain an increasing sequence with X[j] at the end, and we would have A[j] ≥ A[i] + 1. This allows us to add some inequalities X has to satisfy. We can add symmetric ones for B — if i < j and B[i] ≤ B[j], then X[i] < X[j].

These inequalities are not enough, however, to reconstruct X. Indeed, if we take A[i] = i + 1 and B[i] = **N** - i, we get no inequalities to consider, but at the same time not all permutations X are going to work. The problem is that while we know what we need to do with X so that no longer subsequences exist, we still need to guarantee that long enough sequences exist.

This is relatively simple to accomplish, though. If A[i] > 1, then the increasing subsequence ending at X[i] is formed by the extension of some sequence of length A[i] - 1. This means that X[i] has to be larger than X[j] for some j < i with A[j] = A[i] - 1. The previous set of inequalities guarantees that of the set of all such j (that is, j which are smaller than i and have A[j] = A[i] - 1) the one with the smallest X[j] is the largest j. Thus, it is enough to find the largest j with j < i and A[j] = A[i]-1 and add X[j] < X[i] to our set of inequalities. We again do the symmetric thing for B.

Use the information

Notice that the inequalities we have are indeed enough to guarantee that any sequence X satisfying them will lead to the A and B we want. It's relatively easy to check the first set of inequalities guarantees the numbers A and B will not be larger than we want, while the second set of inequalities guarantee they will be large enough.

We now reduced the problem to finding the lexicographically smaller permutation satisfying a given set of inequalities. To find the lexicographically smallest result we are foremost interested in minimizing the first value in X. To this end, we will simply traverse all the inequalities that the first value in X has to satisfy (that is, iterate over all the elements we know to be smaller, then all elements we know to be smaller than these elements, etc.). We can do this, e.g., through a DFS. After learning how many elements of X have to be smaller than X[1], we can assign the smallest possible value (this number + 1) to X[1]. Now we need to assign numbers smaller than X[1] to all these elements, in the lexicographically smallest way.

Note that how we do this assignment will not affect the other elements (since they are all going to be larger than X[1], and so also larger than everything we assign right now). Thus, we can assign this set of elements so that it is lexicographically smallest. This means taking the earliest of these elements, and repeating the same procedure recursively (find all elements smaller than it, assign the appropriate value, recurse). Note that once some values have been assigned, we need to take that into account when assigning new values (so, if we already assigned 1, 3 and 10; and now we know that an element we're looking at is larger than 4 others, the smallest value we can assign to it is 7).

Such a recursive procedure will allow us to solve the problem. For each element, we are going to traverse the graph of inequalities to find all the elements smaller than it is (O(**M** time if we do a DFS, where **M** is the number of inequalities we have), then see what's the smallest value we can assign (O(**N**) with a linear search, we can also go down to O(log**N**), but don't need to), and recurse. This gives us, pessimistically, O($\mathbf{N}^3$) time — easily good enough for the small testcase, but risky for the large.

Compress the information

Following the exact procedure above we can end up with O($\mathbf{N}^2$) inequalities. This will be too much for us (at least for the large dataset), so we will compress a bit.

The problem is with inequalities of the first type — for indices smaller than a given i there can be many with A larger or equal to A[i]. A trick we can use, though, is to take only one inequality — find the largest j < i with A[j] = A[i], and insert only the inequality X[j] &gt X[i].

Any other k < j with A[k] = A[i] will follow from transitivity — there will be a sequence of indices with A equal to A[i] connecting k to i. Any k with A[k] > A[i] will also follow, since X[k] will have to be greater than some X[l] for A[l] = A[i] and l < k (and thus < i). This means we can reduce our set of inequalities down to a set of O(**N**), which means that each DFS traversal step will take only O(**N**) time, and the solution to the whole problem will run in O($\mathbf{N}^2$).

Interesting fact: It is also possible to solve this problem in O(**N** log**N**). Can you figure it out?