Problem Tutorial: "Beautiful subsequence"

Subtask 1 — 9 points.

This subtask can be solved with the brute-force solution: by checking all possible 2^n subsequences of sequence a.

Subtask 2 — 9 points.

Check the solution for the third subtask, which is $O(min(n,m)^2 * max(n,m))$. Or you can solve it by checking all possible 2^m subsequences of sequence b.

Subtask 3 — 28 points.

Unlike previous subtasks, in this subtask you need to come up with a much more efficient solution. For now, let's forget the memory limit for this problem, I'll explain how we can reduce our memory usage after. :)

Let's solve it using *dynamic programming*.

$maxLen_{j,i,tp}$, where $1 \le j \le m, 1 \le i \le n, 0 \le tp \le 1$, is equal to the maximum possible length of a *beautiful* sequence which is a subsequence of arrays $a_1, a_2, \ldots, a_i$ and $b_1, b_2, \ldots, b_j$. The length is odd if $tp = 1$ or even if $tp = 0$. And the last element of a beautiful sequence is equal to a_i.

$f_{j,i,tp}$, with same parameters, is equal to the number of distinct *beautiful* sequences of maximum length.

Look at the pseudo code below:

```
for j = 1..m {
    for i = 1..n {
        for tp = 0..1 {
            maxLen[j][i][tp] = maxLen[j - 1][i][tp]
            f[j][i][tp] = f[j - 1][i][tp]
        }
        if b[j] = a[i] {
            good = {0}
            update your dp by checking for the sequence with length 1
            for last = i-1..1 {
                if good[a[last]] = 0 {
                    update your dp from last
                    good[a[last]] = 1
                }
        }
    }
}
```

I hope the code above is clear enough. Using an array $good[x]$ is a key idea of calculating number of only different sequences.

Subtask 4 — 54 points.

To solve for full points we need to get rid of the last for. We can get rid of it by keeping the maximum lengths of odd/even *beautiful* sequences (with keeping number of different sequences, also).

```
int mx0 = 0, cnt0 = 1
int mx1 = -inf, cnt1 = 0
for j = 1..m {
    for i = 1..n {
        for tp = 0..1 {
            maxLen[j][i][tp] = maxLen[j - 1][i][tp]
            f[j][i][tp] = f[j - 1][i][tp]
        }
        if b[j] = a[i] {
             update(j, i, 0, mx1, cnt1)
             update(j, i, 1, mx0, cnt0)
        }
        if b[j] < a[i] {
             upd(mx0, cnt0, maxLen[j - 1][i][0], f[j - 1][i][0])
        }
        if b[j] > a[i] {
             upd(mx1, cnt1, maxLen[j - 1][i][1], f[j - 1][i][1])
        }
    }
}
```

That's it! Now the only question is: how to reduce the memory usage. Fortunately, we can get rid of parameter j, by creating 2 new arrays (one for the $j-1$ and one for j).

If it's still not clear to you, then take a look at the solution code by downloading the archive of this problem.