Problem Tutorial: "The queue"

Subtask 1. $N \leq 20$, $w \leq 1000$ — 10 points.

To solve this subtask you just need code exactly what the problem is asking for.

Subtask 2. $N \leq 10000$ — 40 points

You need to optimize the solution of the first subtask to $O(N^2)$ to solve this subtask.

Subtask 3. $N \leq 500000$ — 50 points

You need to optimize that solution even more using some data structure. In fact, this problem is very similar to the well-known problem with 2 types of query (if you imagine that you have a very big array) :

1. update the value of x_{th} element to y
2. output the sum of the prefix of the array

You can literally use almost any data structure to solve it. For instance, the author used Fenwick Tree.

Problem Tutorial: "Apples"

Subtask 1/2 — 25 points.

These subtasks can be solved using some simple formulas. If you try to simulate for both when $N = 2$ and when $N = 3$ manually, you can come up with really formula solution.

Subtask 3/4 — 75 points

The main difference between subtask 3 and 4 is the value of A_1. The subtask 3 is designed for people who found the correct solution, but couldn't multiply two long numbers.

Let's try simulate starting from the end when $N = 3$:

$\frac{1}{1} * x, \frac{1}{1} * x, \frac{1}{1} * x$

$\frac{1}{2} * x, \frac{1}{2} * x, \frac{4}{2} * x$

$\frac{1}{4} * x, \frac{7}{4} * x, \frac{4}{4} * x$

$\frac{13}{8} * x, \frac{7}{8} * x, \frac{4}{8} * x$

$\frac{13}{8} * x = A_1$ and by finding x you can find all initial values of an array. I hope you noticed a pattern, if not let's try to analyze what's going on: we are iterating from N to 1, and let i be the number of iterator. We divide every coefficient except i by 2. And we replace the coefficient of i by the new sum of coefficients. You can find out by yourself why it's true.

Because there will be at most $N = 50$ iterations, denominator will be no more than 2^{50}. You need to be careful when finding x, cause the multiplication might not fit in the limits of long long. Also don't the forget the case when there's no answer.

Problem Tutorial: "Sappers"

Let's divide our table in half at first. First $m/2$ columns belong to the 1_{st} sapper and last $m/2$ columns belong to the 2_{nd} sapper. For the sake of convenience, let the $n = 3$ and $m = 4$, and the number of mines in the first or second half is cnt_1 or cnt_2, and the overall number of mines is k. Now we'll prove that it's always possible to achieve the best answer, 0 or 1 when k is even or odd, unless $n = 1$ (corner case). Let's analyze what happens if we add cells for each component one by one:

1122	1112	1111	1111	1111	2111	2211
1122	1122	1122	1212	2211	2211	2211
1122	1222	2222	2222	2222	2221	2211

At first, the numbers of mines in each part are cnt_1 and cnt_2 respectively. But at the end, the numbers became cnt_2 and cnt_1 respectively. Because each time we either increasing or decreasing the number of mines in each component at most by 1, there will be a moment when the difference is minimum (when difference is at most 1 depending on the parity of k).

Note that, when we'll be adding the last row to the 1_{st} component, we'll start adding it starting from the end. Otherwise the component might become not connected. And don't forget the case when n is 1. For example:

1 6

* * *...

In this case, the best achievable answer would be 3.

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.