



Flooding Wall

The amount of water collected on each wall segment is the difference between the level of water at that segment and that segment's height. Now, instead of thinking about the segments individually, let's consider the level of water at a certain height across all segments. The number of segments where the level of water is at least p is defined by the leftmost and the rightmost wall segments whose height is at least p . Therefore for a given sequence of wall segment heights h_1, h_2, \dots, h_N the amount of water collected on the wall is

$$\sum_{p=1}^{\max h_i} (\text{right}(p) - \text{left}(p)) - \sum_{i=1}^n h_i$$

where $\text{right}(p)$ denotes the rightmost position in h such that $h_i \geq p$, and $\text{left}(p)$ denotes the leftmost position in h such that $h_i \geq p$, minus 1. The first sum represents the heights of the water level from the ground, from which the heights of the wall segments is subtracted.

We want to compute the sum of these sums over all 2^N possible walls. The second sum is easy, so we won't discuss it further.

1 Idea

The main idea is to sweep over the heights from the largest to the smallest. Let the current height be p . We increase the total answer by the number of positions, in all possible walls, with level of the water being at least p . For efficiency reasons, we will not consider each p explicitly, and instead work with intervals of heights that behave in the same manner.

Thus, we have a dynamic sequence of integers t_1, t_2, \dots, t_N , where t_i denotes how many of the two possible wall segment heights at position i are larger or equal than the current p . Initially, the sequence is $0, 0, \dots, 0$.

We split the expression into 2 parts: $\text{right}(p)$ and $\text{left}(p)$.

1.1 $\text{right}(p)$

For each possible value of $\text{right}(p)$ we will calculate how many times it appears in the sum. Consider position i . There can be anything on the left of i , and everything on the right of i must be smaller. Thus the contribution is:

$$i \cdot t_i \cdot 2^{i-1} \cdot \prod_{j=i+1}^n (2 - t_j)$$

1.2 $\text{left}(p)$

By a symmetric argument the contribution is:

$$(i-1) \cdot t_i \cdot 2^{n-i} \cdot \prod_{j=i}^{i-1} (2 - t_j)$$



2 Solution

The sequence t_1, t_2, \dots, t_N changes $2N$ times during the sweep. All the formulas can be maintained with some segment trees in $O(\log N)$ time, resulting in the overall complexity of $O(N \log N)$.

2.1 Segment tree

More precisely, we need a segment tree that aggregates each formula over the positions i . The vertices in the segment tree can contain 3 integers:

- A : initially 0.
- B : initially 1.
- Sum : maintains the product of $A \cdot B$.

We then implement the following operations in this segment tree:

- $\text{SetA}(i, c)$: sets the value of A at position i to c .
- $\text{MulB}(i, j, c)$: multiplies the value of B at each of the positions between i and j by c .

2.2 Operations performed on the segment trees

2.2.1 Before starting the sweep

For each i we initialize the $\prod(2 - t_j)$ part of the formulas. For instance, in the segment tree for the formula of $\text{right}(p)$ we would perform $\text{MulB}(i, i, 2^{n-i})$.

2.2.2 Each time t_i turns into 1

- We use SetA to initialize the value at position i . For instance, in the segment tree for the formula of $\text{right}(p)$ we would perform $\text{SetA}(i, i \cdot 2^i)$.
- We need to divide the product $\prod(2 - t_j)$ by 2 for a range of positions where t_i is part of that product. Since we are working modulo $10^9 + 7$, we can perform this division with multiplication by the multiplicative inverse of the number 2. For instance, in the segment tree for the formula of $\text{right}(p)$ we would perform $\text{MulB}(1, i - 1, 5 \cdot 10^8 + 4)$.

2.2.3 Each time t_i turns into 2

1. Use MulB to multiply the value at position i by 2, because the t_i multiplier in the formula changes from 1 to 2.
2. For a range of positions where t_i is part of the product $\prod(2 - t_j)$, we need to multiply the value by 0.

3 Alternative solutions

1. Instead of iterating over segment heights in order of decreasing height, we can iterate in order of increasing height. This is what is implemented in our published solution code.
2. We can iterate over segments from left to right, and have a segment tree that counts the number of possibilities that the water will be of each possible height at this segment. Further,



it needs to provide the sum of the water levels over all possibilities. The implementation could be rather slow and get time limit exceeded so some minor optimizations could be needed.

3. We can iterate over segments in order of either decreasing or increasing height, and have a segment tree that stores how many possibilities there are for water to be of at least the current height at each segment, and also computes the sum.

Credits

- Task: Bartłomiej Czarkowski (Poland)
- Solutions and tests: Justas Klimas, Martynas Budriūnas (Lithuania)