Review of Meetings

Problem

There are N mountains, numbered from 0 through $N-1$ from left to right. The height of the mountain i is H_i ($0 \le i \le N-1$). Exactly one person lives on each mountain.

You are going to hold Q meetings, numbered from 0 through $Q-1$. To the meeting j ($0 \le j \le Q-1$), you will invite all people living on the mountains between the mountain L_j and the mountain R_j, inclusive.

For each meeting, you can choose a mountain as the meeting place. If the mountain x is chosen as the meeting place, the cost of the meeting is calculated as follows:

- The cost of the meeting is the sum of the costs of all participants.
- The cost of the participant from the mountain y is the maximum height of mountains between the mountain x and the mountain y, inclusive. Particularly, the cost of the participant from the mountain x is H_x.

For each meeting, you want to find its minimum cost.

Constraints

- $1 \le N \le 750\,000$
- $1 \le Q \le 750\,000$
- $1 \le H_i \le 1\,000\,000\,000$ ($0 \le i \le N-1$)
- $0 \le L_j \le R_j \le N-1$ ($0 \le j \le Q-1$)
- $(L_j, R_j) \ne (L_k, R_k)$ ($0 \le j < k \le Q-1$)

Subtasks and Solutions

Subtask 1 (4 points)

$N \le 3\,000$, $Q \le 10$

If a meeting place is given, you can calculate the cost of a meeting in $O(N)$. Thus, by testing every possible meeting place, the cost of a meeting can be calculated in $O(N^2)$ time.

The total time complexity is $O(N^2Q)$.

Subtask 2 (15 points)

$N \le 5\,000$, $Q \le 5\,000$

By iterating through the moutains with maintaining an upper envelope, you can get costs for all meeting places in $O(N)$ time.

The total time complexity is $O(NQ)$.

Subtask 3 (17 points)

$N \leq 100\,000$, $Q \leq 100\,000$, $H_i \leq 2$ ($0 \leq i \leq N-1$)

Find the longest contiguous subsequence consisting only of 1 by SegmentTree. The total time complexity is $O(N + Q \log N)$.

Subtask 4 (24 points)

$N \leq 100\,000$, $Q \leq 100\,000$, $H_i \leq 20$ ($0 \leq i \leq N-1$)

For each meeting, divide the range at the heighest mountains and recursively solve the problem.

If you pre-calculate the answer to some ranges, such as maximal ranges in which heights of all mountains are at most some constant, and you prepare a proper data structure for Range Minimum Queries, you can get the minimum cost of a meeting in $O(\max\{H_i\} \log N)$ time.

Pre-calculation can be done in $O(\max\{H_i\} N)$ time.

The total time complexity is $O(\max\{H_i\}(N + Q \log N))$.

Subtask 5 (40 points)

No additional constraints.

For convenience, let's assume that all values of H are distinct (this does not matter much). For each meeting, we can assume that the index of the optimal meeting place is greater than or equal to $\mathrm{argmax}_{L \leq i \leq R}(H_i)$, because by reversing the array H and solving the same problem we can get a real answer.

We denote the problem of calculating the minimum cost of a meeting with the range $[L, R]$ as query $[L, R]$.

- Let $\mathrm{Cost}(L, R)$ be the answer to the query $[L, R]$.
- Let $\mathrm{RangeL}(v)$ be the smallest x such that $\mathrm{argmax}_{x \leq i \leq v}(H_i) = v$.
- Similarly, let $\mathrm{RangeR}(v)$ be the largest x such that $\mathrm{argmax}_{v \leq i \leq x}(H_i) = v$.
- Also let $S(v)$ be the array of length

  $$\mathrm{RangeR}(v) - \mathrm{RangeL}(v) + 1$$

  such that the i-th ($0 \leq i \leq \mathrm{RangeR}(v) - \mathrm{RangeL}(v)$) value of $S(v)$ is

  $$\mathrm{Cost}(\mathrm{RangeL}(v), \mathrm{RangeL}(v) + i).$$

We are going to compute $S(v)$ for all v, and then it is easy to get answers to all queries. The order of indices in which we compute $S(v)$ is very important. Here, we use depth-first-search post-order of the cartesian tree of H.

We define the cartesian tree of H as the rooted tree such that lowest-common-ancestor of nodes u and v is the node $\mathrm{argmax}_{u \leq i \leq v}(H_i)$.

The cartesian tree can be obtained in linear time by an iteration with a stack data structure. It can be easily seen that every node of the cartesian tree has at most two children, one to the left and another to the right. Let $lc(v)$ be the left child of the node v and $rc(v)$ be the right child of the node v (here we assume that the node v has two children).

Now the remaining task is to somehow merge $S(lc(v))$ and $S(rc(v))$ into $S(v)$. Clearly, first some elements of $S(v)$ is exactly $S(lc(v))$.

All we need is to compute $\mathrm{Cost}(\mathrm{RangeL}(v), p)$, for all p ($v \le p$).

Since H_v is the maximum value in the range $[\mathrm{RangeL}(v), \mathrm{RangeR}(v)]$, you can see

$$\begin{aligned}\mathrm{Cost}(\mathrm{RangeL}(v), p) &= \min\{\mathrm{Cost}(\mathrm{RangeL}(v), v) + (p - v) \times H_v, \\ &\quad (v - \mathrm{RangeL}(v) + 1) \times H_v + \mathrm{Cost}(v + 1, p)\}\end{aligned}$$

and

$$\begin{aligned} & (\mathrm{Cost}(\mathrm{RangeL}(v), v) + (p - v) \times H_v) - ((v - \mathrm{RangeL}(v) + 1) \times H_v + \mathrm{Cost}(v + 1, p)) \\ \le\ & (\mathrm{Cost}(\mathrm{RangeL}(v), v) + (p + 1 - v) \times H_v) - ((v - \mathrm{RangeL}(v) + 1) \times H_v + \mathrm{Cost}(v + 1, p + 1))\end{aligned}$$

where $p + 1 \le \mathrm{RangeR}(v)$.

The inequality follows from the observation that

$$\mathrm{Cost}(v + 1, p + 1) - \mathrm{Cost}(v + 1, p) \le \max_{v+1 \le i \le p+1} (H_i) \le H_v.$$

It indicates that there exists a certain index z such that

$$\mathrm{Cost}(\mathrm{RangeL}(v), p) = \mathrm{Cost}(\mathrm{RangeL}(v), v) + (p - v) \times H_v$$

for all $p \le z$, and

$$\mathrm{Cost}(\mathrm{RangeL}(v), p) = (v - \mathrm{RangeL}(v) + 1) \times H_v + \mathrm{Cost}(v + 1, p)$$

for all $z < p$.

Therefore, you can get $S(v)$ in the following way:

- Let T be the array obtained by adding a certain value to all elements of $S(rc(v))$.
- Update first some elements of T with a certain linear funtion.
- Concatenate $S(lc(v))$, $\mathrm{Cost}(\mathrm{RangeL}(v), v)$, and T.

To carry out these operations fast, we use a compressed representation for $S(v)$. $S(v)$ is represented by the list of ranges. Each range has a certain linear funciton such that the values of $S(v)$ in the range can be calculaed by the linear function.

Adding a certain value can be done by lazy propagation.

To update first some elements, we simply iterate through T from the beginning. The ranges before the break point is replaced by one range, so the total number of iterations is $O(N)$.

Concatenating two arrays can be done as follows:

- We maintain end points of ranges in a global set and store the information of ranges in a global array. Then, we do not have to do anything for ranges.

- Concatenating laze propagation information for adding can be done by Weighted-union heuristic:
    - Let $W(s)$ be the value which should be added to elements of the array s.
    - When we concatenate two arrays s and t, we pick the smaller one and arrange the elements of it so that $W(s) = W(t)$.
    - By Weighted-union heuristic this can be done in $O(N \log N)$ time in total.

Getting the answer to a query requires one lower bound operation of the set. Thus in $O(Q \log N)$ time we can get answers to all queries.

The total time complexity is $O((N + Q) \log N)$.