극솟값 제거 – 풀이

작성자: 장근영

풀이

각 쿼리 (l, r, t)에서는 부분수열 $A_l, A_{l+1}, \ldots, A_r$만을 따로 떼어 내고, 극솟값을 동시에 제거하는 과정을 t번 적용한다.

양쪽 이웃이 모두 존재해야 제거될 수 있으므로, 현재 수열의 양 끝 원소는 절대로 제거되지 않는다.

부분문제 1

이 부분문제에서는 모든 쿼리가 전체 수열을 대상으로 하며, $N \le 5000$이다.

현재 수열을 직접 저장하면서 각 변화를 그대로 시뮬레이션한다.

한 번의 변화를 처리할 때 현재 수열을 순회하여 극솟값을 모두 표시한다. 그 뒤 표시되지 않은 원소들만 순서대로 모아 새로운 수열을 만든다.

모든 극솟값은 동시에 제거되어야 하므로, 수열을 순회하는 도중에 원소를 바로 제거하면 안 된다는 점에 유의해야 한다.

s번 변화시킨 뒤 남은 원소의 개수를 C_s라고 저장하면, 쿼리 $(1, N, t)$의 답은 C_t이다.

한 번의 변화에 $\mathcal{O}(N)$의 시간이 들고, 변화는 최대 N번만 수행하면 된다.

전체 시간 복잡도는 $\mathcal{O}(N^2 + Q)$이다.

부분문제 2

이 부분문제에서도 모든 쿼리가 전체 수열을 대상으로 한다.

각 원소 A_i가 전체 수열에서 몇 번째 변화에 제거되는지를 D_i라고 하자. 끝까지 제거되지 않는 원소에 대해서는 $D_i = \infty$로 정의한다.

현재 살아 있는 원소들 사이의 인접 관계를 이중 연결 리스트로 관리한다.

처음에는 모든 극솟값을 후보 목록에 넣는다. 한 번의 변화에서는 후보 목록의 원소들을 모두 동시에 제거하고, 제거된 원소들의 기존 양쪽 이웃만 다시 극솟값인지 검사한다.

원소 하나가 제거되었을 때 극솟값 여부가 달라질 수 있는 원소는 기존의 양쪽 이웃뿐이다. 또한 각 원소는 최대 한 번만 제거된다.

따라서 모든 D_i를 구하는 데 필요한 시간은 $\mathcal{O}(N)$이다.

각 시각까지 제거된 원소의 누적 개수를 저장하면 쿼리 $(1, N, t)$의 답은 $N - \#\{i \mid D_i \le t\}$이다.

전체 시간 복잡도는 $\mathcal{O}(N + Q)$이다.

부분문제 3

이 부분문제에서는 항상 $t = 1$이다.

첫 번째 변화에서 위치 i의 원소가 제거될 조건은

$$A_{i-1} > A_i < A_{i+1}$$

이다.

각 $2 \le i < N$에 대해, i가 극솟값이면 $C_i = 1$, 그렇지 않으면 $C_i = 0$으로 정의하고 C의 누적 합을 구한다.

쿼리 $(l, r, 1)$에서 제거될 수 있는 위치는 양 끝을 제외한 $l + 1, l + 2, \ldots, r - 1$뿐이다.

따라서 제거되는 원소 수는 $\sum_{i=l+1}^{r-1} C_i$이고, 답은

$$r - l + 1 - \sum_{i=l+1}^{r-1} C_i$$

이다.

전체 시간 복잡도는 $\mathcal{O}(N + Q)$이다.

부분문제 4

이 부분문제에서는 항상 $t = N$이다.

원소가 제거되는 변화에서는 적어도 하나의 원소가 없어지므로, N번 변화시킨 뒤에는 반드시 더 이상 제거할 원소가 없다.

구간 $[l, r]$을 왼쪽에서 오른쪽으로 보았을 때, 지금까지 등장한 모든 원소보다 큰 원소를 **접두사 최댓값**이라고 하자.

마찬가지로 오른쪽에서 왼쪽으로 보았을 때, 지금까지 등장한 모든 원소보다 큰 원소를 **접미사 최댓값**이라고 하자.

접두사 최댓값은 자신의 왼쪽에 자신보다 큰 원소가 없다. 따라서 어떤 순간에도 왼쪽 이웃이 자신보다 커질 수 없으므로 극솟값이 될 수 없다.

접미사 최댓값도 같은 이유로 제거되지 않는다.

반대로 어떤 원소의 왼쪽과 오른쪽에 모두 자신보다 큰 원소가 존재한다고 하자.

값이 더 작은 원소부터 생각하면, 해당 원소와 양쪽의 더 큰 원소 사이에 있는 작은 원소들은 결국 모두 제거된다. 그러면 해당 원소의 양쪽 이웃이 모두 자신보다 커지므로 해당 원소도 제거된다.

따라서 최종적으로 남는 원소들은 정확히 접두사 최댓값이거나 접미사 최댓값인 원소들이다.

각 위치 i에 대해 다음을 정의하자.

- L_i: i의 왼쪽에서 가장 가까운, A_i보다 큰 원소의 위치

- R_i: i의 오른쪽에서 가장 가까운, A_i보다 큰 원소의 위치

해당하는 원소가 없다면 각각 $L_i = 0$, $R_i = N + 1$로 둔다.

모든 L_i와 R_i는 단조 스택을 이용하여 $\mathcal{O}(N)$에 구할 수 있다.

구간 $[l, r]$의 접두사 최댓값들은 정확히

$$l,\ R_l,\ R_{R_l}, \ldots$$

의 형태로 나타난다. 위치가 r을 넘어가면 중단한다.

마찬가지로 접미사 최댓값들은

$$r,\ L_r,\ L_{L_r}, \ldots$$

의 형태로 나타난다. 위치가 l보다 작아지면 중단한다.

두 수열은 구간 $[l, r]$의 최댓값 위치에서 정확히 한 번 겹친다.

L/R을 여러 번 적용하는 것을 빠르게 하기 위해서 아래와 같이 희소 배열을 만들어준다.

$$\mathrm{RJ}_0(i) = R_i, \qquad \mathrm{RJ}_{k+1}(i) = \mathrm{RJ}_k(\mathrm{RJ}_k(i))$$

$\mathrm{RJ}_k(i)$는 i에서 R을 2^k번 적용한 결과이다.

마찬가지로

$$\mathrm{LJ}_0(i) = L_i, \qquad \mathrm{LJ}_{k+1}(i) = \mathrm{LJ}_k(\mathrm{LJ}_k(i))$$

를 정의한다.

각 쿼리에 대해 희소 배열에서의 이분탐색을 사용하면 접두사 최댓값과 접미사 최댓값의 개수를 각각 $\mathcal{O}(\log N)$에 구할 수 있다.

두 개수를 더한 뒤, 구간의 최댓값이 두 번 세어졌으므로 1을 빼면 답을 얻는다.

전체 시간 복잡도는 $\mathcal{O}((N + Q) \log N)$이다.

부분문제 5

전체 수열은 어떤 위치 p를 기준으로

$$A_1 > A_2 > \cdots > A_p < A_{p+1} < \cdots < A_N$$

의 형태이다.

따라서 어떤 구간을 선택하더라도 그 구간은 단조 수열이거나 V자 형태이다.

구간 $[l, r]$을 생각하자.

현재 수열에 극솟값이 존재한다면 극솟값은 정확히 하나이다. 이 원소를 제거한 뒤에도 남은 수열은 다시 V자 형태가 된다.

따라서 매 변화마다 원소가 정확히 하나씩 제거된다.

$M = \min(A_l, A_r)$라고 하자.

구간 내부에서 값이 M보다 작은 원소가 남아 있다면, 그중 가장 작은 원소가 현재 수열의 극솟값이 되어 제거된다.

반면 값이 M보다 작은 원소가 모두 제거되면, 값 M을 가지는 양 끝 원소 중 하나가 남은 수열의 최솟값이다. 양 끝 원소는 제거되지 않으므로 이후에는 더 이상 극솟값이 존재하지 않는다.

따라서 전체 과정에서 제거되는 원소의 수는

$$K = \#\{i \mid l \le i \le r,\ A_i < M\}$$

이다.

매 변화마다 정확히 하나가 제거되므로, t번 변화시킨 뒤 남는 원소의 개수는

$$r - l + 1 - \min(t, K)$$

이다.

수열은 p의 왼쪽에서 단조 감소하고 오른쪽에서 단조 증가한다. 따라서 $A_i < M$을 만족하는 원소의 개수는 양쪽에서 각각 한 번씩 이분 탐색하여 구할 수 있다.

전체 시간 복잡도는 $\mathcal{O}(N + Q \log N)$이다.

핵심 관찰

이제 일반적인 구간에 대해 여러 번의 변화를 처리하는 방법을 알아보자.

전체 수열에서의 제거 시각

각 위치 i에 대해 D_i를 전체 수열에서 A_i가 제거되는 시각이라고 정의한다. 끝까지 제거되지 않는다면 $D_i = \infty$이다.

앞서 정의한 L_i와 R_i를 이용하면 모든 D_i를 값이 작은 원소부터 차례대로 계산할 수 있다.

각 위치 i에 대해 다음 집합을 정의하자.

$$C_i = \{x \mid L_x = i \text{ 또는 } R_x = i\}$$

$x \in C_i$이면 $A_x < A_i$이다.

또한 x가 살아 있는 동안에는 i의 한쪽에 i보다 작은 원소가 남아 있으므로, i는 x보다 먼저 제거될 수 없다.

$L_i = 0$이거나 $R_i = N + 1$이면 i의 한쪽에는 자신보다 큰 원소가 없다. 따라서 i는 절대로 제거되지 않는다.

그 외의 경우에는 C_i의 원소들이 모두 제거된 다음 변화에서 i가 제거된다.

따라서 다음 관계가 성립한다.

$$D_i = \begin{cases} \infty, & L_i = 0 \text{ 또는 } R_i = N + 1, \\ 1 + \max\limits_{x \in C_i} D_x, & \text{그 외.} \end{cases}$$

C_i가 비어 있다면 최댓값은 0으로 정의한다.

모든 $x \in C_i$에 대해 $A_x < A_i$이므로, 원소들을 값이 작은 순서대로 처리하면 D_i를 계산할 때 필요한 모든 값이 이미 계산되어 있다.

A는 $1, 2, \ldots, N$의 순열이므로 값이 v인 원소의 위치를 미리 저장하면 모든 D_i를 $\mathcal{O}(N)$에 계산할 수 있다.

구간에서도 제거되는 조건

위치 i가 전체 수열에서 D_i번째 변화에 제거된다고 해서, 임의의 구간에서도 같은 시각에 제거되는 것은 아니다.

예를 들어 $L_i < l$이라면, 구간 $[l, r]$ 안에는 i의 왼쪽에 A_i보다 큰 원소가 없다.

따라서 i는 구간을 왼쪽에서 보았을 때의 접두사 최댓값이고, 구간 안에서는 절대로 제거되지 않는다.

마찬가지로 $R_i > r$이면 i는 구간의 접미사 최댓값이므로 제거되지 않는다.

이제 $L_i \geq l$이고 $R_i \leq r$이라고 하자.

L_i와 R_i는 모두 구간 안에 있으며 값이 A_i보다 크다. 또한 i가 제거되기 전까지는 두 원소 모두 제거되지 않는다.

따라서 L_i와 R_i 사이에서 일어나는 제거 과정은 그 바깥 원소들의 영향을 받지 않는다.

그러므로 구간 $[l, r]$에서도 i는 전체 수열에서와 같은 시각 D_i에 제거된다.

결론적으로 위치 i가 쿼리 (l, r, t)에서 제거될 필요충분조건은 다음과 같다.

$$D_i \leq t, \qquad L_i \geq l, \qquad R_i \leq r$$

구간의 경계에서 살아남는 원소

먼저 구간 $[l, r]$ 안에서 $D_i \leq t$인 원소를 모두 세었다고 하자.

그중 실제 구간에서는 제거되지 않는 원소는 $L_i < l$이거나 $R_i > r$인 원소이다.

$L_i < l$인 원소들은 구간의 접두사 최댓값들이며, 정확히

$$l,\ R_l,\ R_{R_l}, \ldots$$

의 형태로 나타난다.

$R_i > r$인 원소들은 구간의 접미사 최댓값들이며, 정확히

$$r,\ L_r,\ L_{L_r}, \ldots$$

의 형태로 나타난다.

이 두 체인은 구간의 최댓값 위치에서만 겹친다.

따라서 실제로 제거되는 원소의 수는 다음과 같이 구할 수 있다.

1. 구간 안에서 $D_i \leq t$인 원소를 모두 센다.
2. 왼쪽 끝에서 시작하는 R 체인 중 $D_i \leq t$인 원소를 뺀다.
3. 오른쪽 끝에서 시작하는 L 체인 중 $D_i \leq t$인 원소를 뺀다.
4. 구간의 최댓값을 두 번 뺐다면 한 번 다시 더한다.

체인을 따라가면 제거 시각이 증가한다

R_i가 존재한다고 하자.

R_i의 정의에 의해 R_i는 i의 오른쪽에서 가장 가까운, A_i보다 큰 원소이다. 따라서 R_i의 제거 시각을 계산할 때 i는 C_{R_i}에 포함된다.

그러므로 D_i가 유한하다면

$$D_{R_i} \geq D_i + 1$$

이다.

같은 이유로 L_i가 존재하고 D_i가 유한하다면

$$D_{L_i} \geq D_i + 1$$

이다.

따라서 R을 반복해서 적용한 체인에서는 제거 시각이 엄격히 증가한다.

$$D_i < D_{R_i} < D_{R_{R_i}} < \cdots$$

마찬가지로 L을 반복해서 적용한 체인에서도 제거 시각이 엄격히 증가한다.

$$D_i < D_{L_i} < D_{L_{L_i}} < \cdots$$

따라서 각 체인에서 $D_i \leq t$인 원소들은 항상 체인의 앞쪽 연속 구간을 이룬다.

이 성질 덕분에 체인의 모든 원소를 하나씩 확인하지 않고, 이분 탐색으로 조건을 만족하는 마지막 원소를 찾을 수 있다.

부분문제 6

전체 수열에 과정을 20번 적용한 뒤 더 이상 제거되는 원소가 없다.

따라서 모든 유한한 제거 시각은 $D_i \leq 20$을 만족한다.

각 $1 \leq s \leq 20$에 대해 다음 누적 합을 만든다.

$$P_s(i) = \#\{j \mid 1 \leq j \leq i,\ D_j \leq s\}$$

쿼리 (l, r, t)에서 전체 수열을 기준으로 t번 이내에 제거되는 원소의 수는

$$P_{\min(t,20)}(r) - P_{\min(t,20)}(l-1)$$

이다.

이제 왼쪽 끝 l에서 R을 반복해서 적용하면서, 위치가 r 이하이고 제거 시각이 t 이하인 원소들을 뺀다.

체인에서 제거 시각은 엄격히 증가하며 유한한 제거 시각은 최대 20이므로, 최대 20개의 원소만 확인하면 된다.

오른쪽 끝 r에서도 L을 반복해서 적용하여 같은 작업을 한다.

두 체인이 같은 원소까지 도달했다면 그 원소를 두 번 뺀 것이므로 한 번 다시 더한다.

쿼리 하나를 $\mathcal{O}(20)$에 처리할 수 있다.

전체 시간 복잡도는 $\mathcal{O}(20N + 20Q)$이고, 메모리 복잡도는 $\mathcal{O}(20N)$이다.

부분문제 7

일반적인 경우에는 체인의 길이가 클 수 있으므로 이분 탐색을 사용한다.

각 위치 i에 대해 시각 D_i에 위치 i를 활성화한다고 생각하자.

쿼리들도 t가 작은 순서대로 처리한다.

현재 처리하는 쿼리의 시간이 t라면, $D_i \le t$인 모든 위치 i를 Fenwick Tree에 추가한다.

그러면 구간 $[l, r]$에서 활성화된 원소의 수

$$C = \#\{i \mid l \le i \le r,\ D_i \le t\}$$

를 Fenwick Tree의 구간 합으로 구할 수 있다.

위치의 활성화와 쿼리 처리를 모두 시간순으로 정렬하면 전체 시간 복잡도는 $\mathcal{O}((N+Q)\log N)$이다.

왼쪽 체인과 오른쪽 체인 위의 위치 i 중 $D_i \le t$를 만족하는 i의 개수를 세는 것은 희소 배열에서의 이분탐색을 사용하면 된다.

전처리에는 $\mathcal{O}(N \log N)$이 필요하고, 각 쿼리는 $\mathcal{O}(\log N)$에 처리된다.

전체 시간 복잡도는 $\mathcal{O}((N+Q)\log N)$이고, 메모리 복잡도는 $\mathcal{O}(N \log N)$이다.

다른 만점 풀이 1: CDQ 분할 정복

위치 i가 쿼리 (l, r, t)에서 제거될 필요충분조건은

$$L_i \ge l, \qquad R_i \le r, \qquad D_i \le t$$

이다.

첫 번째 부등식의 방향을 바꾸기 위해 $X_i = N - L_i + 1$로 정의하자.

그러면 $L_i \ge l$은 $X_i \le N - l + 1$과 동치이다.

각 원소 i를 다음 3차원 점으로 생각한다.

$$(N - L_i + 1,\ R_i,\ D_i)$$

쿼리 (l, r, t)는 세 좌표가 모두

$$(N - l + 1,\ r,\ t)$$

이하인 점의 개수를 묻는다.

따라서 문제는 3차원 dominance counting 문제가 된다.

점과 쿼리를 첫 번째 좌표 순서대로 정렬하고 CDQ 분할 정복을 수행한다.

각 분할 정복 단계에서는 왼쪽 절반에 있는 점이 오른쪽 절반에 있는 쿼리에 주는 기여를 계산한다.

왼쪽의 점과 오른쪽의 쿼리를 두 번째 좌표 순서로 처리하면서, 두 번째 좌표가 현재 쿼리 이하인 점들을 Fenwick Tree에 추가한다.

Fenwick Tree에서는 세 번째 좌표를 관리한다. 그러면 세 좌표가 모두 쿼리 이하인 점의 수를 셀 수 있다.

센 점의 개수는 제거되는 원소의 수이므로, 구간의 길이에서 이 값을 빼면 답을 얻는다.

각 분할 정복 단계에서 정렬을 수행하면 전체 시간 복잡도는 $\mathcal{O}((N+Q)\log^2(N+Q))$이다.

다른 만점 풀이 2: 2차원 Fenwick Tree

각 원소 i를 다음 2차원 점으로 생각하자.

$$x_i = N - L_i + 1, \qquad y_i = R_i$$

점들을 D_i 순서로 정렬하고, 쿼리들을 t 순서로 정렬한다.

쿼리를 시간순으로 처리하면서 $D_i \le t$가 된 점 (x_i, y_i)를 2차원 Fenwick Tree에 추가한다.

쿼리 (l, r, t)에서는

$$x_i \le N - l + 1, \qquad y_i \le r$$

인 점의 개수를 세면 된다.

즉, 원점부터 $(N-l+1, r)$까지의 직사각형 안에 있는 점의 개수를 구하면 된다.

일반적인 $N \times N$ 크기의 배열을 사용하면 메모리가 너무 크다.

대신 바깥쪽 Fenwick Tree의 각 정점에 실제로 추가될 수 있는 y좌표들만 미리 모아 각각 좌표 압축한다.

점 하나는 바깥쪽 Fenwick Tree의 $\mathcal{O}(\log N)$개 정점에 포함되므로, 저장되는 좌표의 총개수는 $\mathcal{O}(N \log N)$이다.

점 추가와 직사각형 합 쿼리는 모두 $\mathcal{O}(\log^2 N)$에 처리할 수 있다.

전체 시간 복잡도는 $\mathcal{O}((N+Q)\log^2 N)$이고, 메모리 복잡도는 $\mathcal{O}(N \log N)$이다.