[22.12.10] 최대최소값찾기

TS최대최소값찾기

김 태 현

문 제

수가 순서대로 저장된 수열이 있다.

새로운 수열은 오른쪽 끝에 추가된다.

특정 구간의 수들이 삭제되기도 한다.

오른쪽에서 K번째까지의 수 중 가장 큰 수와 가장 작은 수를 찾아 그 차이를 반환해야 한다.

`void init(int N, int mValue[])`

N개의 수열 값이 mValue[] 에 주어진다.
N : 1 ~ 30,000
mValue : 0 ~ 100,000,000

`void erase(int mFrom, int mTo)`

제일 앞에 있는 수를 1번째 수라고 정의했을 때,
mFrom 부터 mTo 번째 수까지 삭제한다.
"mTo – mFrom" 은 0 이상 100 이하임이 보장된다.

`void add(int M, int mValue[])`

M개의 mValue[] 값이 수열의 오른쪽에 추가된다.

`int find(int K)`

오른쪽 첫번째 수부터 K번째까지의 수 중 가장 큰 수와
작은 수의 차이를 반환한다.

※ 제약사항

init(), add() 에서 전달된 수의 개수 총합 <= 200,000개
add() <= 2,000 | erase() <= 500 | find() = 30,000

1. Naive

A[] : 수열이 담긴 배열
N : A의 길이 <= 200,000

add(M, val[]) < 2,000

A의 오른쪽에 val[] 등록
total O(200,000)

erase(s, e) < 500

A에서 s-1 index부터 e-1 index까지 삭제
worst O(N)

find(K) < 30,000

A의 N-K index부터 N-1 index까지 min, max값 확인
worst O(N)

2. min, max 미리 구해놓기

아래와 같이 정의된 배열 두개를 구성하여 init(), add(), erase() 과정에서 미리 구해 놓는다.

- minA[i] = A의 i ~ N-1 범위 값 중 가장 작은 값
- maxA[i] = A의 i ~ N-1 범위 값 중 가장 큰 값

그러면, find(K)는 **maxA[N-K] – minA[N-K]** 로 O(1)에 해결 가능하다.

1. init(N, val[])

뒤에서부터 진행하며,
maxA[i] = max(maxA[i+1], A[i]) , minA[i] = min(minA[i+1], A[i])
O(N)

	0	1	2	3	4	5
A	1	5	6	3	2	4
maxA	6	6	6	4	4	4
minA	1	2	2	2	2	4

2. min, max 미리 구해놓기

2. add(M, val[])

- 오른쪽에 M개 값 추가하고, M개는 init()과 동일하게 진행

	0	1	2	3	4	5	6	7	8
A	1	5	6	3	2	4	9	8	7
maxA	6	6	6	4	4	4	9	8	7
minA	1	2	2	2	2	4	7	7	7

- 기존에 있던 값들도 뒤에서부터 min, max 업데이트
  기존의 min, max가 그대로 유지되면 더 이상 진행하지 않고 종료
  (min, max 따로 진행)

	0	1	2	3	4	5	6	7	8
A	1	5	6	3	2	4	9	8	7
maxA	9	9	9	9	9	9	9	8	7
minA	1	2	2	2	2	4	7	7	7

호출시마다 연산은 worst case O(N)번 진행된다.
2,000번 호출동안 O(200,000) 일어난다라고 하면
O(4억) 이긴 하지만 아래 이유로 문제 없을 것으로 판단

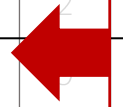
- 연산이 상당히 간단
- 호출되면서 원소개수가 늘어나는 것이기 때문에
  N이 200,000보다 작음
- **업데이트를 중간에 멈추게 되는 상황 多**

2. min, max 미리 구해놓기

3. erase(s, e)

- 지워지는 구간 오른쪽 모든 값을 그대로 왼쪽으로 갖고 온다.

	0	1	2	3	4	5	6	7	8
A	1	5	6	3	2	4	9	8	7
maxA	9	9	9	9	9	9	9	8	7
minA	1	2	2	2	2	4	7	7	7



	0	1	2	3	4	5
A	1	5	4	9	8	7
maxA	9	9	9	9	8	7
minA	1	2	4	7	7	7

- 지워진 구간 왼쪽 값들의 max,min을 뒤에서부터 각각 업데이트 한다.
  maxA[i] = max(maxA[i+1], A[i]) | minA[i] = min(minA[i+1], A[i])
  만약, 바뀌지 않는다면 역시 더 이상 진행하지 않고 종료한다.

	0	1	2	3	4	5
A	1	5	4	9	8	7
maxA	9	9	9	9	8	7
minA	1	4	4	7	7	7

호출시마다 연산은 worst case O(N)번 진행된다.
500번 호출동안 O(200,000) 일어난다라고 하면
O(1억)이고 아래 이유로 문제 없을 것으로 판단

- 연산이 상당히 간단
- 호출되면서 원소개수가 감소하는 것이기 때문에 N이 200,000보다 작음
- **업데이트를 중간에 멈추게 되는 상황 多**

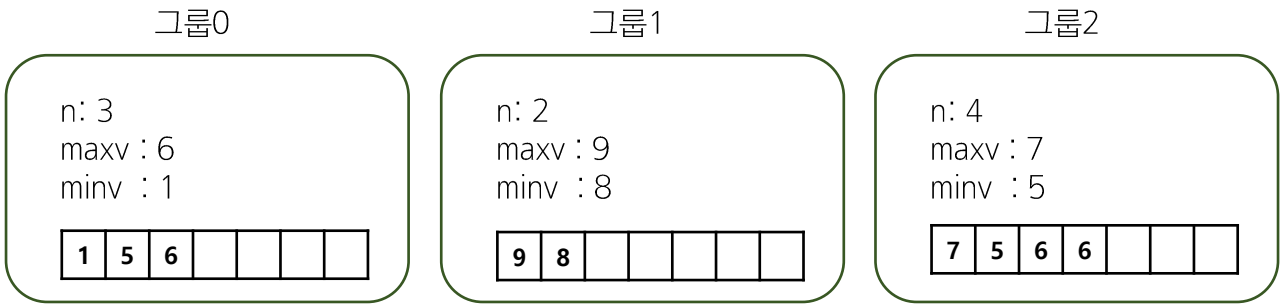
3. sqrt decomposition

- 수열을 SQ = sqrt(N)개의 원소로 이루어진 그룹으로 나누어 max, min을 구해 놓는다.
- add(), erase(), find() 모두 O(sqrt(N)) 의 비용으로 해결 가능하다.
  (참고로, segment tree는 O(log N) 으로 가능)
- find()의 비용이 늘어나 평균적으로는 시간이 느려질 수 있어도 worst case에도 안정적으로 돌아간다.
- 여기서 erase()가 있어서 까다로운 부분이 생기는데 이를 위해 크게 두가지 형태로 구현해볼 수 있다.

1. 원소를 삭제하지 않고 exist = 1/0 으로 표시하며 그룹 원소 개수를 기록한다.

	0	1	2	3	4	5	6	7	8	9	10	11
A	1	5	6	~~3~~	~~2~~	~~4~~	9	8	7	5	6	6
exist	1	1	1	0	0	0	1	1	1	1	1	1
cnt	3				2				4			
maxA	6				9				7			
minA	1				8				5			

2. 그룹 정보를 객체로 두고 그 내부에서 수열을 기록하고 추가/삭제 해 나간다.

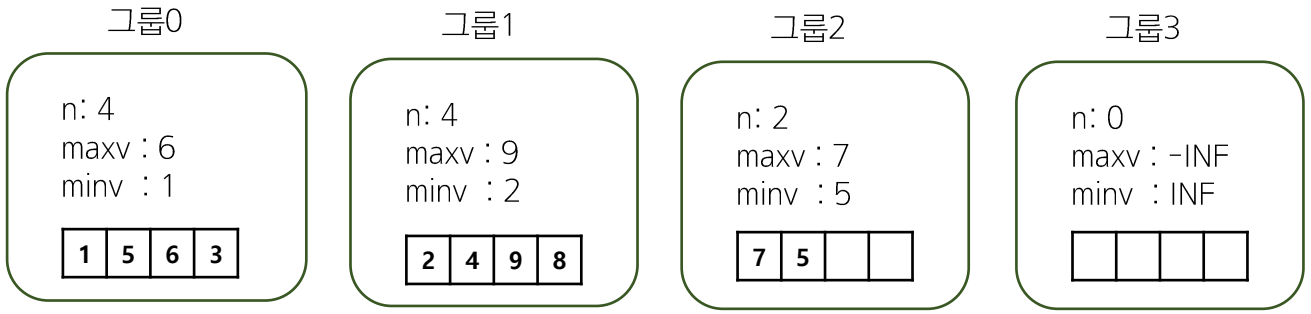


3. sqrt decomposition

1. init(N, val[])

- 아래 형태로 초기화
- 최대 원소 개수 = 16, 그룹 크기 = 4

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	1	5	6	3	2	4	9	8	7	5						
exist	1	1	1	1	1	1	1	1	1	1						
cnt	4				4				2							
maxA	6				9				7							
minA	1				2				5							

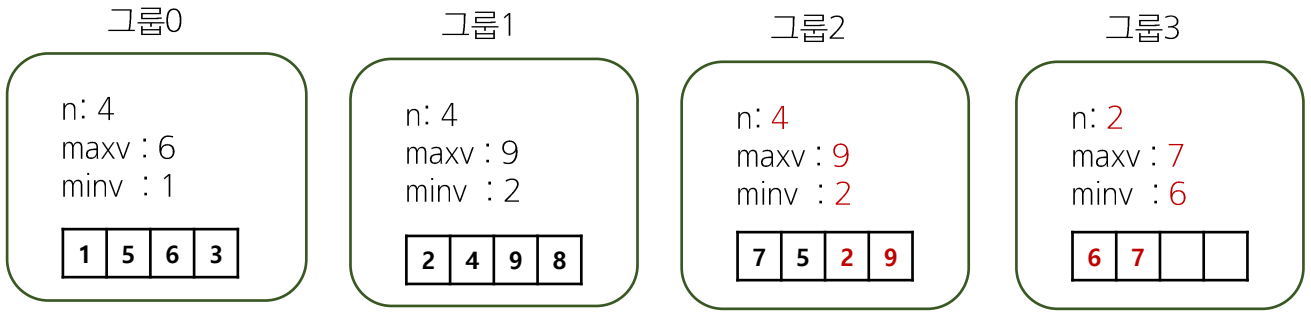


3. sqrt decomposition

2. add(M, val[])

- 마지막 그룹 오른쪽부터 M개 추가
  그룹이 다 찼으면 다음 그룹에 추가

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	1	5	6	3	2	4	9	8	7	5	2	9	6	7		
exist	1	1	1	1	1	1	1	1	1	1	1	1	1	1		
cnt	4				4				4				2			
maxA	6				9				9				7			
minA	1				2				2				6			



3. sqrt decomposition

2. erase(s, e)

- cnt, exist 활용하여 앞에서 s번째 위치를 찾고 e번째 위치까지 삭제한다.
  그룹의 원소 개수가 0개면 그룹단위로 넘어간다.
- 값이 한 개라도 삭제 되는 그룹은 max, min을 새로 구한다.

erase(3,9) 인 경우

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	1	5	6	3	2	4	9	8	7	5	2	9	6	7		
exist	1	1	0	0	0	0	0	0	0	1	1	1	1	1		
cnt	2				0					3			2			
maxA	5				-INF					9			7			
minA	1				INF					2			6			

erase(2,4) 인 경우

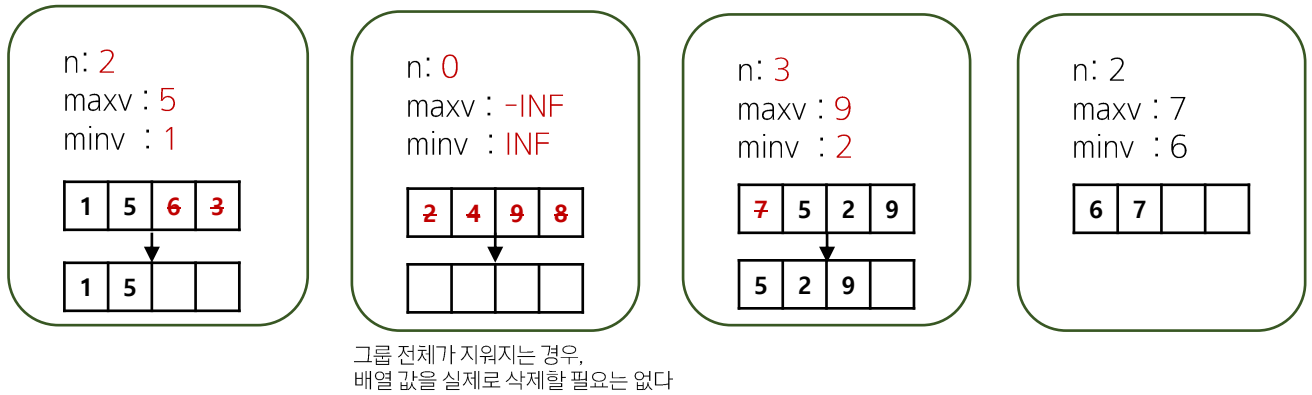
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	1	5	6	3	2	4	9	8	7	5	2	9	6	7		
exist	1	0	0	0	0	0	0	0	0	0	0	1	1	1		
cnt	1				0					1			2			
maxA	1				-INF					9			7			
minA	1				INF					9			6			

3. sqrt decomposition

2. erase(s, e)

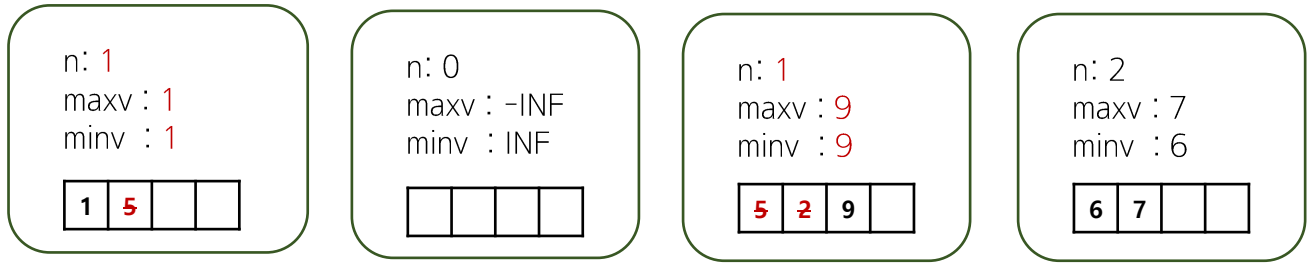
- 각 그룹의 삭제 해야하는 구간을 찾아 삭제한다.
- 일부가 지워지는 그룹은 많아야 2개 뿐이다.
  일부를 지우는 경우 그룹 원소를 전부 검색하여 min, max를 재설정한다.
  전부를 지우는 경우 n=0, maxv=-INF, minv=INF 로 설정한다.

erase(3,9) 인 경우



그룹 전체가 지워지는 경우,
배열 값을 실제로 삭제할 필요는 없다

erase(2,4) 인 경우



3. sqrt decomposition

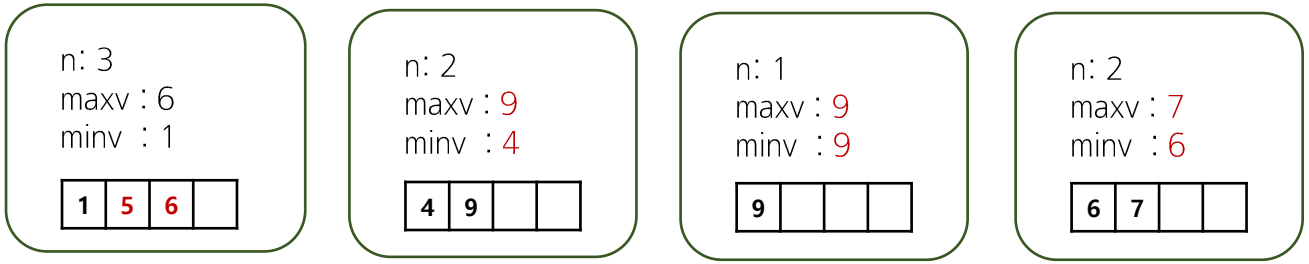
2. find(K)

- 그룹 내의 원소 개수를 활용하여 뒤에서 K번째 위치까지 검색한다.
- 그룹이 완전히 포함된다면 그룹의 max, min값으로 통째로 확인하여 넘어가고 완전히 포함되지 않는다면 낱개로 확인한다.

find(7) 인 경우

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A	1	5	6	3	2	4	9	8	7	5	2	9	6	7		
exist	1	1	1	0	0	1	1	0	0	0	0	1	1	1		
cnt	3				2				1				2			
maxA	6				9				9				7			
minA	1				4				9				6			

max= 9 min = 4 | max= 9 min = 4 | max= 9 min = 4 | max= 9 min = 6 | max= 7 min = 6 | max=-INF min = INF




감사합니다