JooHyun – Lee (comkiwer)

Sqrt Decompotion

Hancom Education Co. Ltd.

Square root Decomposition : Concept

Sqrt(Square root) Decomposition

N개의 데이터를 $\sqrt{N}$(제곱근) 크기로 나누어 처리하는 기법이다.

(평방 분할 이라고도 한다.)

N개의 데이터를 $\sqrt{N}$크기로 나누면 나누어진 구간의 개수도 $\sqrt{N}$개가 된다.

이는 N개 데이터 전체를 대상으로

탐색, 삽입, 삭제를 수행하는 것에 비해 훨씬 효과적이다.

A[]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
99	1	80	23	50	4	25	33	48	21	37	11	63	7	71	13

Sqrt(Square root) Decomposition

다음 문제를 생각해 보자.

N 개의 정수가 A[]배열에 담겨있다.

이제 다음 두가지 명령이 Q번 주어진다.

1. A[idx]값을 newValue로 업데이트한다.
2. A[s] ~ A[e] 사이의 최대값을 구한다.

이 문제를 단순히 N개 전체를 탐색하는 방법을 사용한다면
시간 복잡도는 O(N * Q)이다.

Sqrt Decomposition을 사용한다면 O($\sqrt{N}$ * Q) 가 된다.

Square root Decomposition : Build

Sqrt Decomposition : Build

N = 16이고 Block(구간)의 크기는 mod = sqrt(16) = 4인 예로 생각해 보자.

구간의 최대값을 저장할 배열 B[]를 준비하고

A[]를 순회하며 B[i / mod] = max(B[i / mod] , A[i])를 저장한다.

아래 그림은 구간별 최대값을 B[]에 구해놓은 결과이다.

A[]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
99	1	80	23	50	4	25	33	48	21	37	11	63	7	71	13

B[]

0	1	2	3
99	50	48	71

Sample code : Build

```
const int LM = 100004;
const int INF = (int)2e9;
int N, M, mod = 1;
int A[LM], B[LM];

void init(){
    scanf("%d %d", &N, &M);
    for(;(mod+1)*(mod+1) <= N; ++mod);
    for(rint i=0;i < N / mod; ++i) // zero base
        B[i] = -INF;
    for(rint i=0;i< N;++i){
        scanf("%d", &A[i]);
        B[i / mod] = max(B[i / mod] , A[i]);
    }
}
```

Square root Decomposition : update

Sqrt Decomposition : Update

A[4]를 27로 바꾸는 경우를 보자.

A[4]에 27을 채우고 B[4/4]에도 27을 채운다.

A[4]~ A[7]을 순회하며 B[1]의 값을 업데이트 한다.

이때 시간복잡도는 O(mod = 4)이다.

A[]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
99	1	80	23	27	4	25	33	48	21	37	11	63	7	71	13

50

B[]

0	1	2	3
99	50->27->33	48	71

Sample code : Update _ ver01

```
inline int max(int a, int b) { return a > b? a:b;}
inline int min(int a, int b) { return a < b? a:b;}

void update(int k, int newVal){
    rint bn = k / mod;
    rint st = bn * mod, ed = min(N, st +  mod); // 구간 인덱스 [st ~ ed) 구하기
    A[k] = B[bn] = newVal;
    while (st < ed)
        B[bn] = max(B[bn], A[st++]);
}
```

Sample code : Update _ ver02

```
const int INF = 1<<30;
const int LM = 100004;
const int BASE = 8;
const int MOD = 1 << BASE; // use powers of tow.
const int MASK = MOD - 1;
int N, M, A[LM], B[LM / MOD + 5];

inline int max(int a, int b) { return a > b? a:b;}
void update(int k, int val){
    int bn = k / MOD;
    int st = bn * MOD;
    A[k] = val, B[bn] = A[st++];

    while(st < N && (st & MASK))
        B[bn] = max(B[bn], A[st++]);
}
```

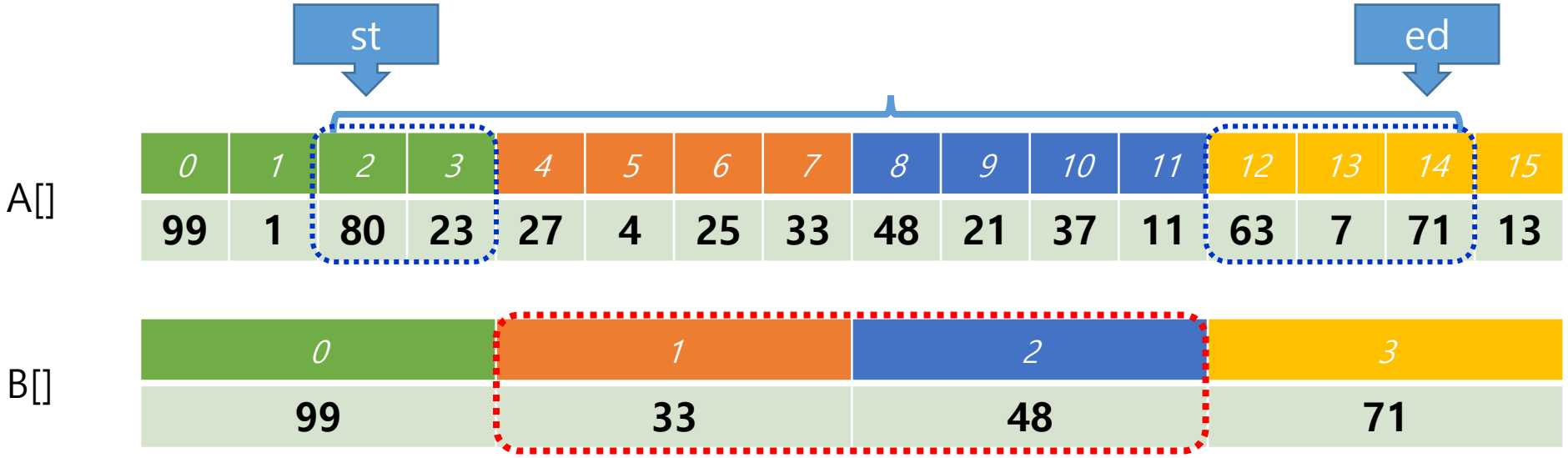
Square root Decomposition : getMax query

Sqrt Decomposition : getMax query

A[2]로부터 A[14]까지 구간에서 최대값을 구하는 경우를 보자. mod = sqrt(16) = 4

이 경우 아래 그림과 같이 3개의 탐색구간으로 나누어 탐색할 수 있다.

1. st=2부터 3까지 범위에서 ret = max(ret, A[st++])를 진행한다. : O(mod = 4)
2. ed=14부터 12까지 범위에서 ret = max(ret, A[ed--]를 진행한다. : O(mod = 4)
3. st = 4 / 4부터 ed = 11 / 4 범위에서 mod 구간단위로
   ret = max(ret, B[st++])를 진행한다. : O(mod = 4)
4. 이때 시간복잡도는 O(mod = 4)이다.



Sample code : getMax query _ ver01

```
const int LM = 50004;
const int MOD = 223;                        // (int)sqrt(50000)
int N, Q, A[LM], B[LM / MOD + 5];

int query(rint st, rint ed){                // the maximum value of the interval [s ~ e]
    int ret = -INF;
    while(st <= ed && (st % MOD))           //  processing front remnants
        ret = max(ret, A[st++]);

    while(st <= ed && ((ed + 1) % MOD)) //  processing rear remnants
        ret = max(ret, A[ed--]);

    for(; st <= ed; st += MOD)              // processing in blocks.
        ret = max(ret, B[st / MOD]);
    return ret;
}
```

Sample code : getMax query _ ver02

```
const int LM = 50004;
const int BASE = 8;
const int MOD = 1 << BASE;
const int MASK = MOD - 1;
int N, Q, A[LM], B[LM / MOD + 5];

int query(int s, int e){        // the maximum value of the interval [s ~ e)
    int ret = 0;
    while(s < e && (s & MASK))  // processing front remnants
        ret = max(ret, A[s++]);

    while(s < e && (e & MASK))  // processing rear remnants
        ret = max(ret, A[--e]);

    s = (s + MASK) / MOD, e /= MOD;
    for(;s < e; ++s)            // processing in blocks.
        ret = max(ret, B[s]);
    return ret;
}
```

관련 문제

- codepass 1726 구간의 최대값1
- codepass 3238 구간의 최대값2
- codepass 3239 구간의 최소값
- codepass 3297 동적 구간합[1D]

감사합니다.