고급 10회차
Persistent Segment Tree

서강대학교 전해성(seastar105)

Persistent Data Structure

변경이력들을 가지고 있는 자료구조를 Persistent Data Structure라고 한다.

초기 상태부터 변경 쿼리가 있을 때마다 그 변경 쿼리를 실행했을 때의 상태를 전부 가지고 있는 것을 말합니다.

Persistent Segment Tree

Persistent Segment Tree는 세그먼트 트리에서 업데이트 쿼리가 있을 때마다 그 상태를 저장해두는 자료구조 입니다.

정말 간단하게 생각하자면 각 쿼리가 있을 때마다 세그트리를 복사해서 갖고 있으면 되겠죠.

그러나 메모리가 매우 많이 듭니다.

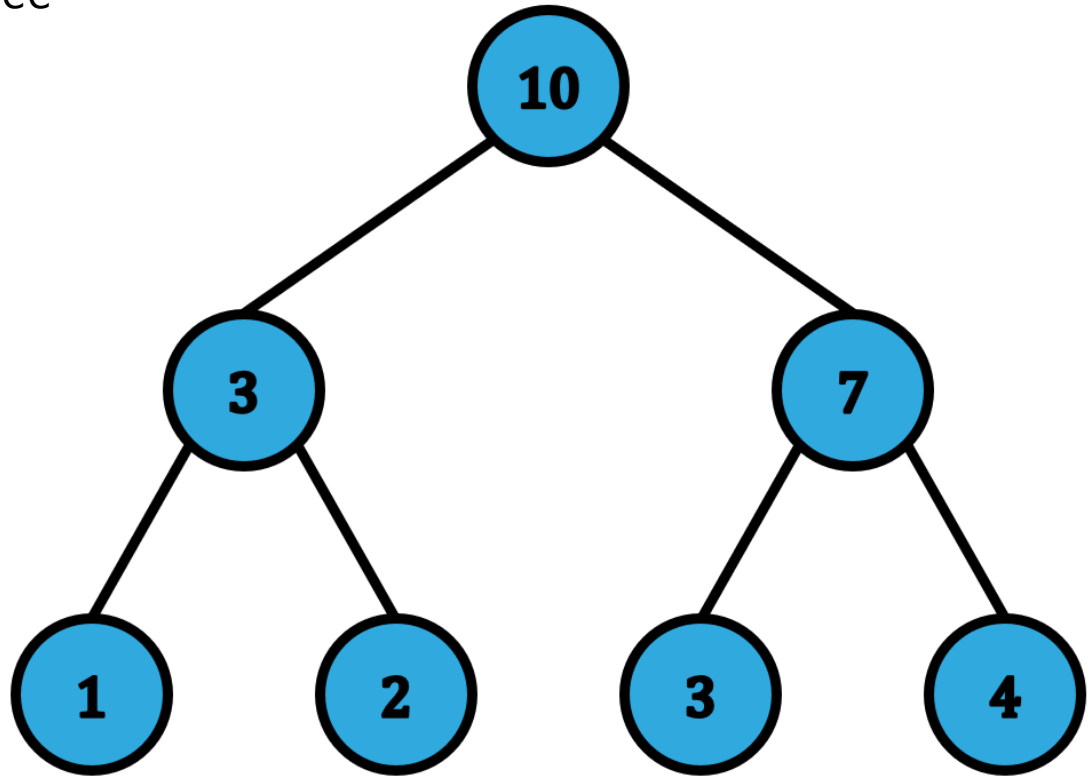
Update only O(logN) nodes

세그먼트 트리에서 점 하나를 업데이트하는 경우를 생각해봅시다.

총 세그먼트 트리에서 변경되는 노드들은 높이당 하나로 O(logN)이 됩니다.

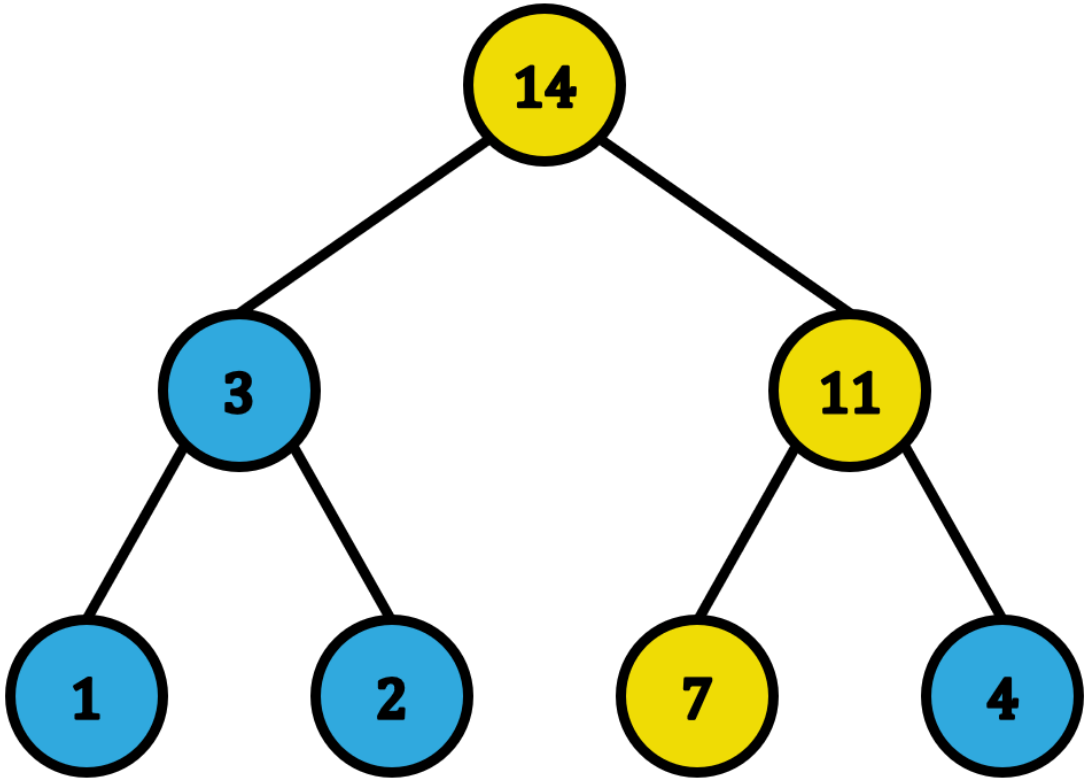
Update only O(logN) nodes

Initial Segment tree



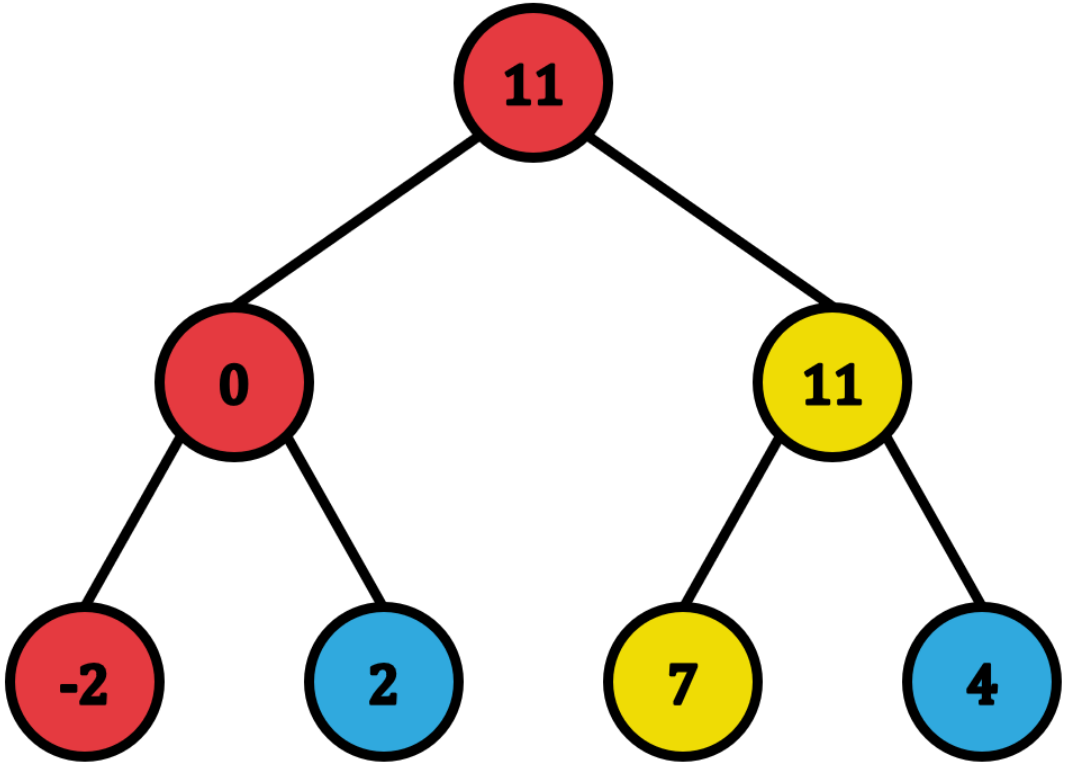
Update only O(logN) nodes

Change a[3] to 7



Update only O(logN) nodes

Change a[1] to -2



Update only O(logN) nodes

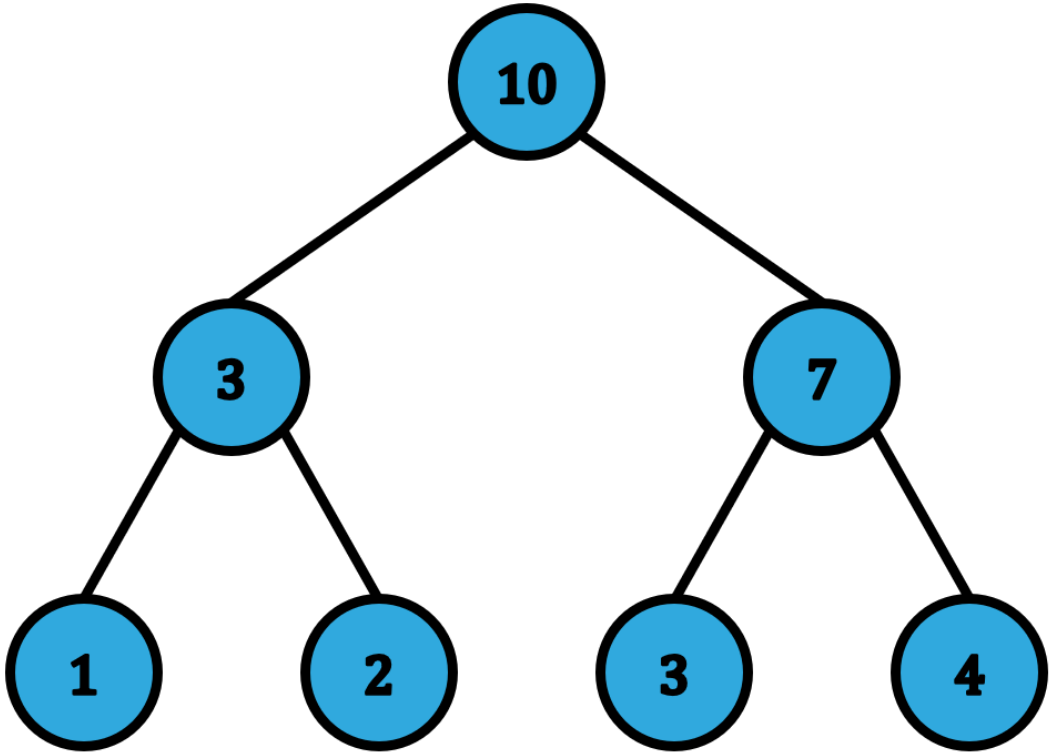
이 점을 이용하면 쿼리당 O(logN)만큼의 메모리만 사용해서 모든 상태를 저장할 수 있습니다.

변경 쿼리가 일어나면, 일반 세그트리처럼 노드를 타고 내려가되, 변경이 일어나는 곳들의 노드만 새로 만듭니다.

그리고, 변경이 일어난 노드와 변경이 없는 노드를 구분해서 새로이 연결해주면 됩니다.

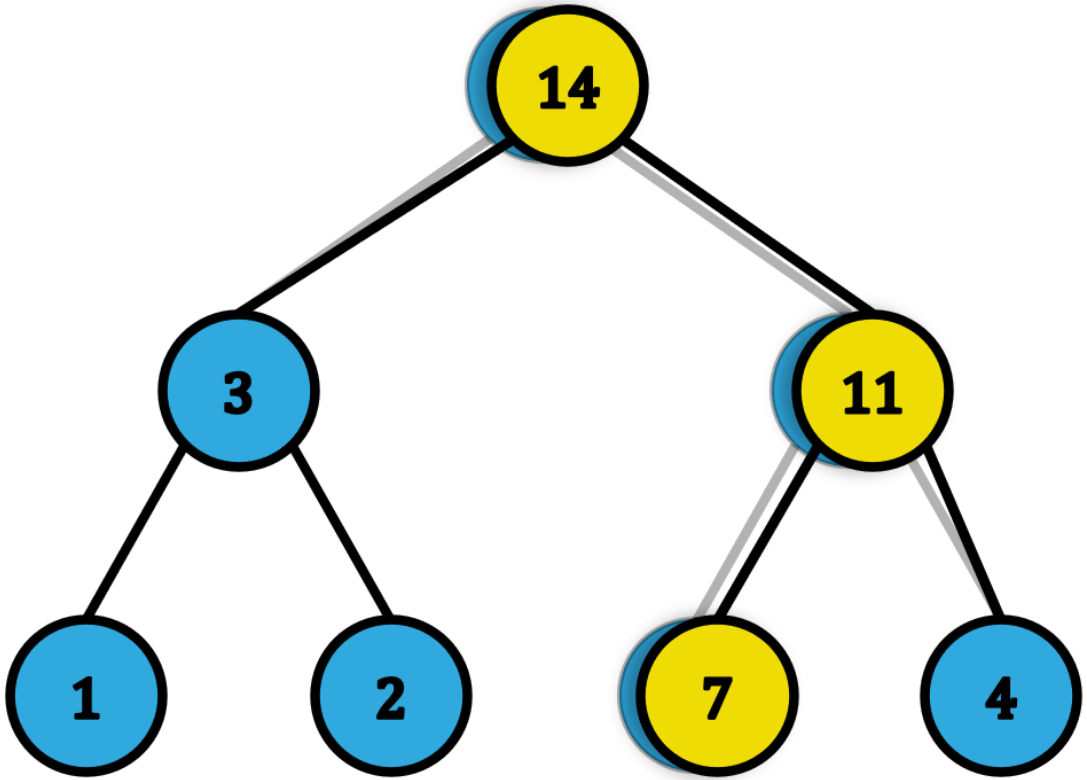
Persistent Segment Tree

Initial Segment Tree



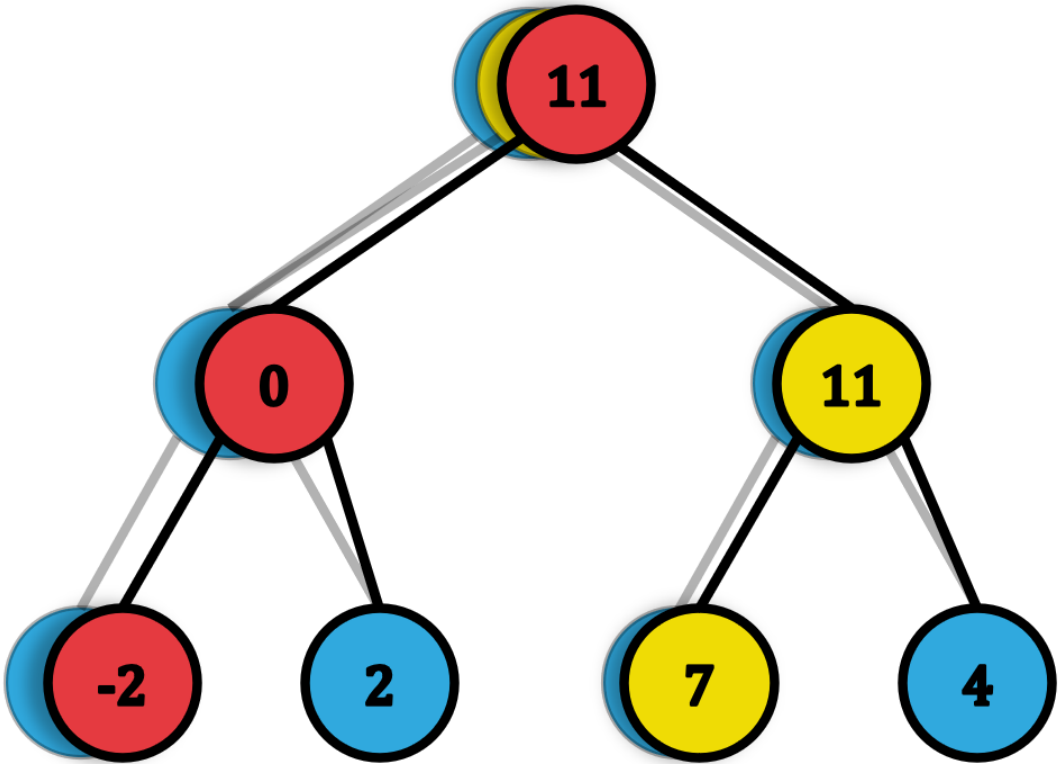
Persistent Segment Tree

Change a[3] to 7



Persistent Segment Tree

Change a[1] to -2



Implementation Issue

사용하게 되는 메모리는 O(N+QlogN)이 됩니다. 이에 유념하셔서 노드들을 동적할당 하는게 아니라 배열로 만들어 두셔서 가지고 오시는 걸 추천 드립니다.

쿼리 별로 관리하는 것은 어떤 쿼리에 맞춰 변경이 일어났을 때 새로 생긴 루트 노드들을 관리함으로 쿼리 별로 그 때의 상태에 접근이 가능합니다.

이제 현재 노드의 인덱스를 두 배로 취해도 자손이란 보장이 깨지므로 각 노드 별로 자손의 노드 위치를 기억하고 있어야 됩니다.

Update Implementation

```
void update(int node, int s, int e, int val, int idx) {
    if(s == e) {    // leaf node
        tree[node].val = val;
    }
    else {
        int mid = s + e >> 1;
        int lc = tree[node].lc, rc = tree[node].rc;
        if(idx <= mid) {    // update left child
            tree[node].lc = node_cnt;
            tree[node_cnt++] = tree[lc];
            update(tree[node].lc, s, mid, val, idx);
        }
        else {              // update right child
            tree[node].rc = node_cnt;
            tree[node_cnt++] = tree[rc];
            update(tree[node].rc, mid+1, e, val, idx);
        }
        tree[node].val = tree[tree[node].lc].val + \
                         tree[tree[node].rc].val;
    }
}
```

구간 합 세그먼트 트리에서 점 업데이트를 하는 코드입니다.

update 함수에 호출하기 전에 업데이트 대상 노드를 미리 만들고 호출합니다.

Application

그래서 이 PST를 어떤 문제에 쓰이는지를 대표적으로 두 개 알아보겠습니다.

어떤 수열의 구간이 주어졌을 때, 해당 구간에서 K번째 수가 무엇인지를 알 수 있습니다. (boj.kr/7469)

2차원 좌표 공간 상에 점들이 놓여있고 직사각형 쿼리가 주어졌을 때, 그 직사각형 안에 위치하는 점의 개수를 셀 수 있습니다. (boj.kr/11012)

백준 7469번 K번째 수

방금 전에 얘기한 문제 중에 하나입니다.

이 문제를 PST로 어떻게 풀 지 알아봅시다.

주어진 값들에 대해서 먼저 좌표 압축을 진행합니다.

백준 7469번 K번째 수

좌표 압축한 배열의 크기를 N이라고 합시다.

배열 a[1…N]에 대해서 세그먼트 구간 합을 지원하는 세그먼트 트리를 만듭시다.
세그트리의 초기 상태는 모든 원소가 0입니다.

백준 7469번 K번째 수

a[1]부터 a[N]까지 돌면서 a[i]의 값에 해당하는 위치의 원소를 1씩 키웁니다.

이제 roots[i]를 정의합시다.

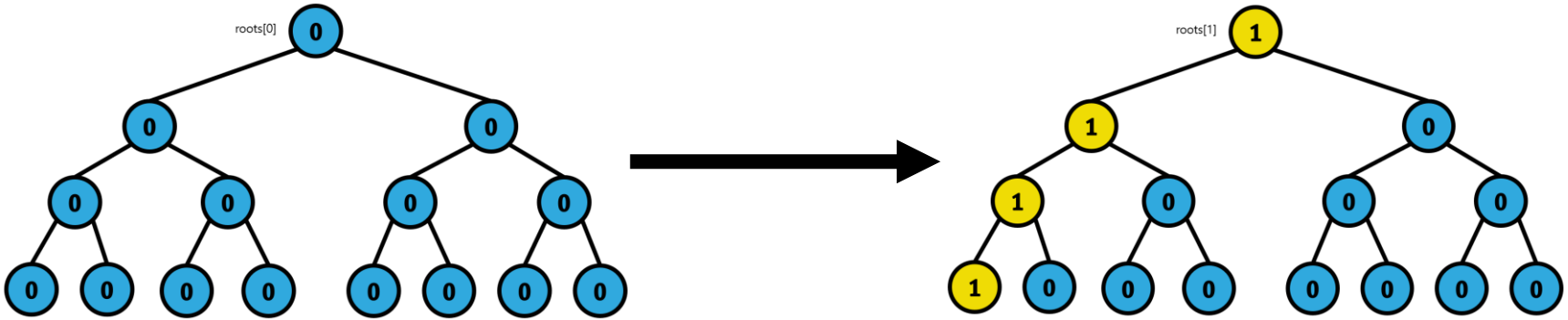
roots[i] = a[1…i]까지 해당 업데이트를 진행한 세그트리의 루트노드

PST를 이용하면 업데이트가 이뤄진 노드들만 추가함으로 roots[i]를 관리할 수 있습니다.

백준 7469번 K번째 수

예시

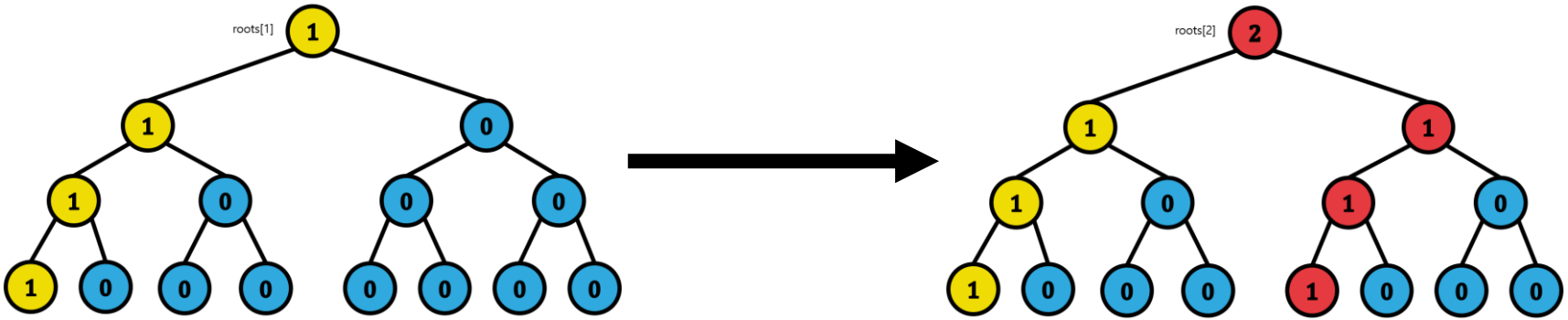
[1,5,2,6,3,7,4] → [0,4,1,5,2,6,3]



백준 7469번 K번째 수

예시

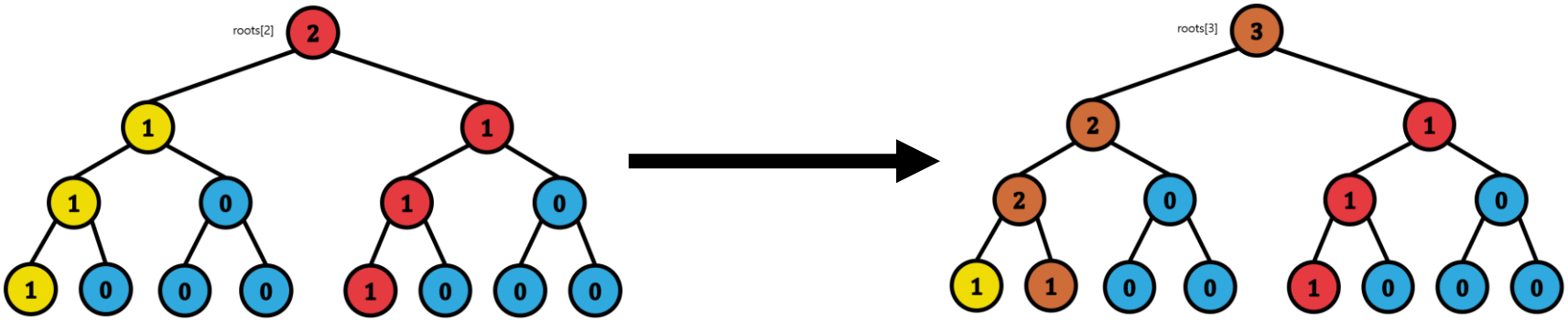
[1,5,2,6,3,7,4] → [0,4,1,5,2,6,3]



백준 7469번 K번째 수

예시

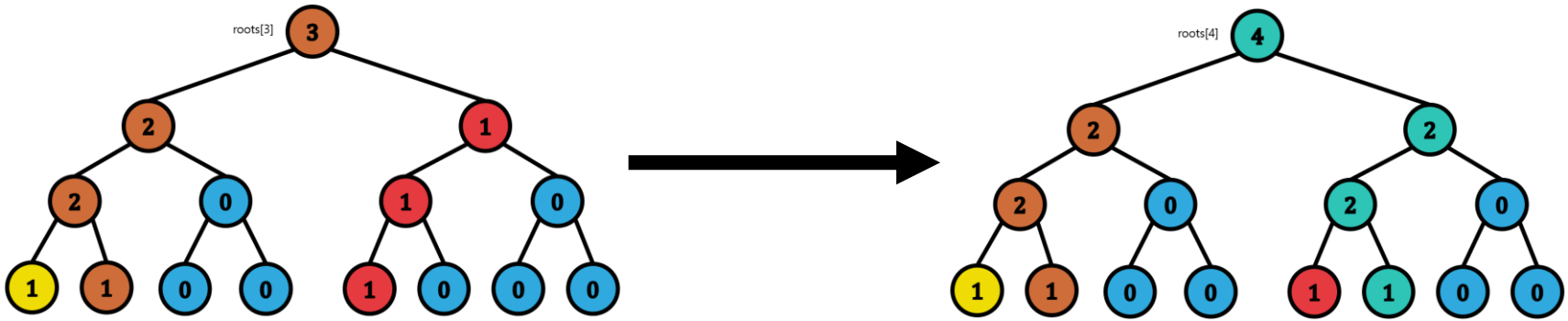
[1,5,2,6,3,7,4] → [0,4,1,5,2,6,3]



백준 7469번 K번째 수

예시

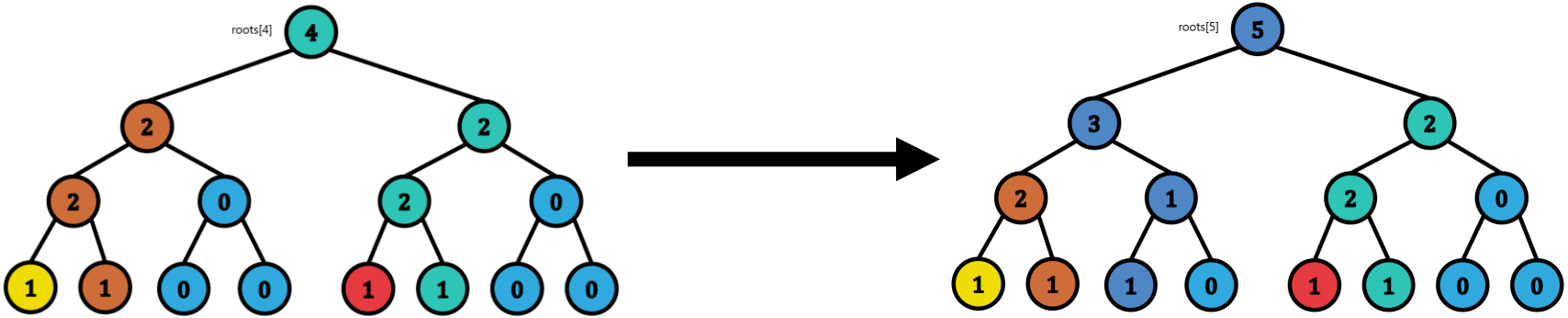
[1,5,2,6,3,7,4] → [0,4,1,5,2,6,3]



백준 7469번 K번째 수

예시

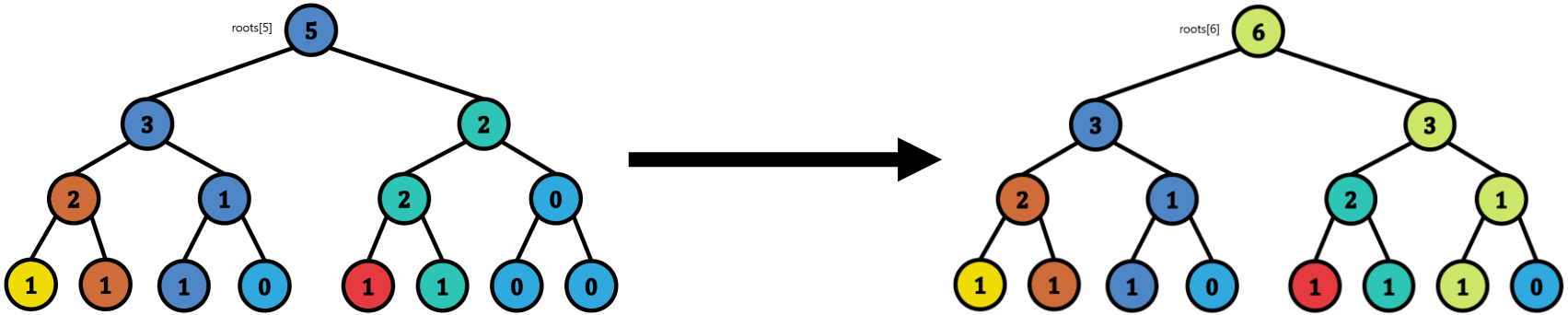
[1,5,2,6,3,7,4] → [0,4,1,5,2,6,3]



백준 7469번 K번째 수

예시

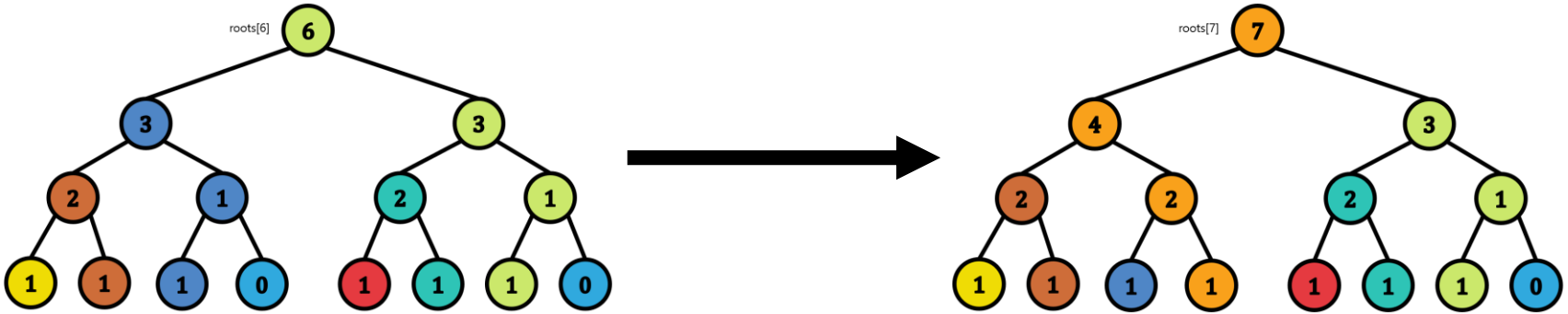
[1,5,2,6,3,7,4] → [0,4,1,5,2,6,3]



백준 7469번 K번째 수

예시

[1,5,2,6,3,7,4] → [0,4,1,5,2,6,3]



백준 7469번 K번째 수

이런 PST를 잘 구성했다고 칩시다.

쿼리로 [l, r, k]가 들어옵니다. 이를 어떻게 처리할까?

일단 세그먼트 트리를 위와 같이 구성했을 때, [1…i]구간에서 K번째 수를 어떻게 찾으면 될까요?

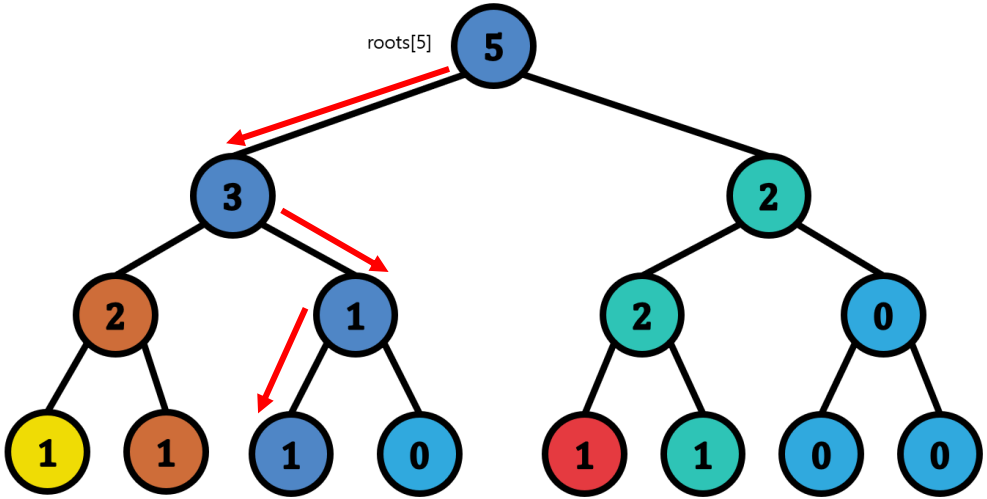
roots[i]의 세그먼트 트리는 1…i 구간의 원소들이 켜져 있는 상황입니다. 따라서, 해당 세그먼트 트리에서 1…j의 합이 K가 되는 j의 위치에 해당하는 수가 K번째 수가 됩니다.

백준 7469번 K번째 수

예시

[1,5,2,6,3,7,4] → [0,4,1,5,2,6,3]

1…5에서 세번째 수는? 좌표압축 했을 때 세번째 수(3)



백준 7469번 K번째 수

이를 이용해서 이제 쿼리를 처리합시다.

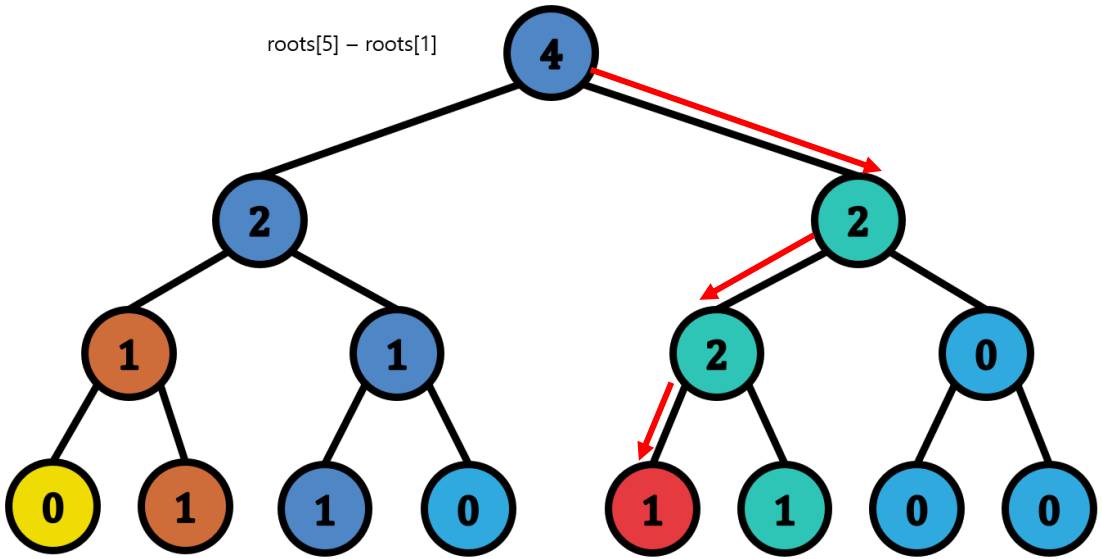
[l, r, k]면 roots[r]과 roots[l-1]을 고려합시다. roots[r]의 세그먼트 트리에서 roots[l-1]의 세그먼트 트리의 값들을 빼면 이제 [l, r]의 원소들만 켜지게 됩니다.

이 상황에서 K번째 수를 구하면 쿼리에서 원하는 답이 됩니다.

백준 7469번 K번째 수

예시

[1,5,2,6,3,7,4] → [0,4,1,5,2,6,3], [2,5,3] 쿼리



백준 7469번 K번째 수 - Code

```
int get_kth(int nxt_node, int prv_node, int s, int e, int k) {
    if (s == e) return s;
    int lsz = tree[tree[nxt_node].l].val - tree[tree[prv_node].l].val;
    int mid = s + e >> 1;
    if (lsz >= k) return get_kth(tree[nxt_node].l, tree[prv_node].l, s, mid, k);
    else return get_kth(tree[nxt_node].r, tree[prv_node].r, mid + 1, e, k - lsz);
}
```

nxt_node가 root[r]이고, prv_node가 roots[l-1]이라고 보시면 됩니다.

왼쪽의 합이 k보다 크거나 같으면 왼쪽으로 가고, k보다 작다면 그만큼 빼주고 오른쪽으로 내려가면 k번째 원소를 찾을 수 있습니다.

백준 11012번 Egg

2차원 좌표 상에서 점이 N개 놓여 있습니다. 쿼리로는 직사각형이 주어집니다.

직사각형 안에 놓여있는 점들의 개수를 계산해야 합니다.

입력되는 좌표의 범위는 최대 10만(=MAX)입니다.

2D 세그트리를 사용하면 쿼리당 $O(\log^2 MAX)$로 해결이 가능합니다.

그러나, 메모리가 $O(MAX^2)$으로 만들 수가 없습니다.

이를 PST로 해결해봅시다.

백준 11012번 Egg

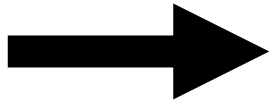
직사각형 쿼리가 l, r, b, t로 들어오는데 이 부분에 대한 구간합을 묻는 것과 같습니다.

2d세그처럼 생각을 해서 tree[x][y]를 x좌표가 [0…x]인 점들 중에서 y인 점들의 개수라고 생각합시다.

그럼 해당 쿼리는 tree[r][b…t] - tree[l-1][b…t]를 통해서 구할 수 있습니다.

백준 11012번 Egg

			1		
		1			
	1				



0	0	0	1	1	1
0	0	0	0	0	0
0	0	1	1	1	1
0	0	0	0	0	0
0	1	1	1	1	1
0	0	0	0	0	0

Tree[0] Tree[1] Tree[2] Tree[3] Tree[4] Tree[5]

Query [1,2,1,3]
2 - 0

백준 11012번 Egg

tree[x]를 일일이 다 만들면 또 다시 메모리 초과가 납니다.

tree[x]를 tree[x-1]와 비교 했을 때, 변하는 부분은 x좌표가 x면서 점이 있는 y부분만 변경됩니다.

따라서, 제일 처음에 모든 원소가 0인 세그트리를 만들고 tree[x]를 0부터 시작해서 만들어 나가는 방법을 생각해볼 수 있다.

이 경우 추가로 사용되는 메모리 양은 점의 개수 N과 추가되는 노드 수 O(logMAX)로 O(NlogMAX)가 된다. 게다가 구간 합 쿼리도 O(logMAX)로 처리 가능하다.

“Any Question?”