2D Segment Tree

based on bottom-up approach

2021.03.06 23:30 KST

Segment Tree

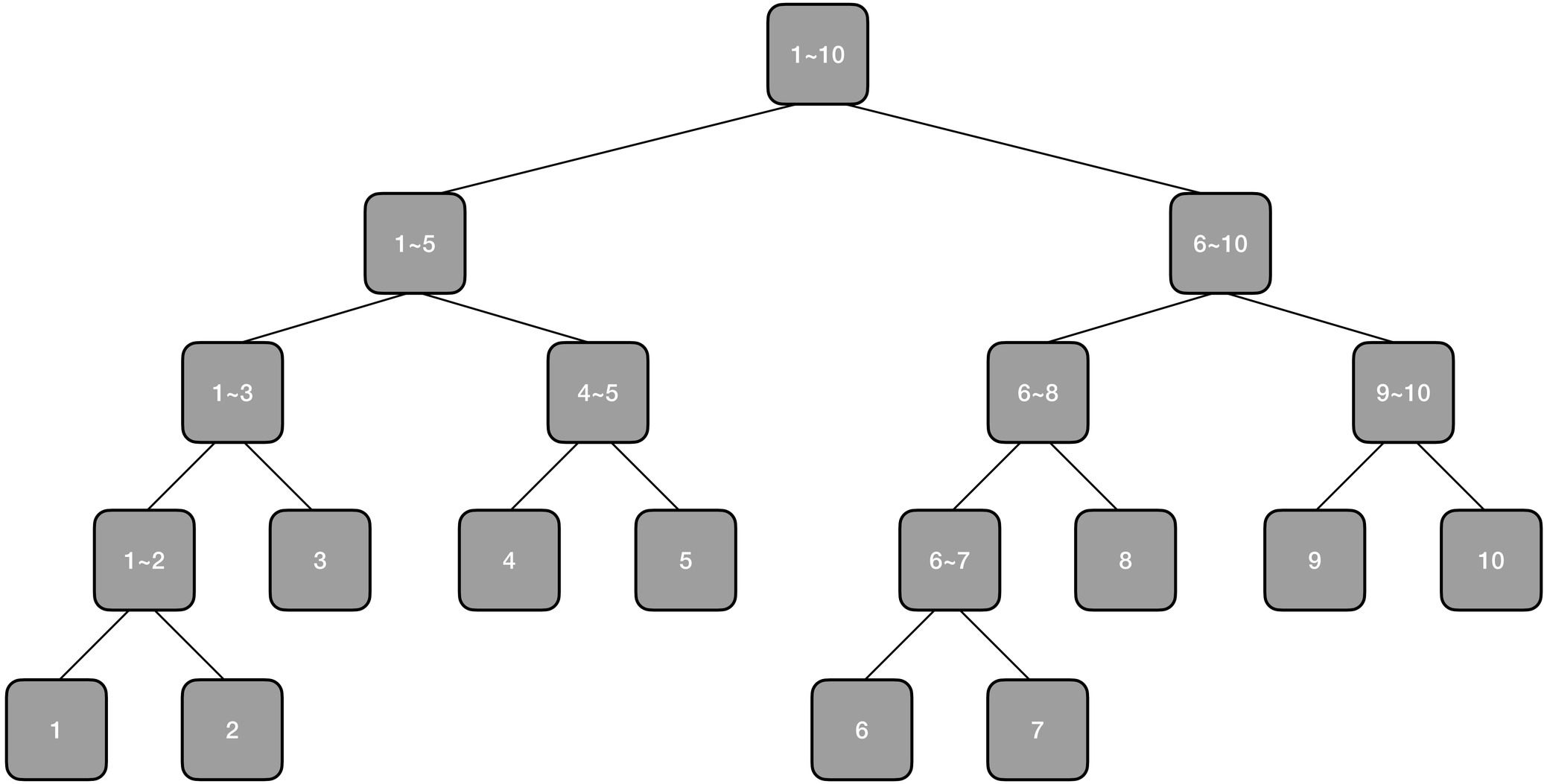
- 주어진 배열의 각 구간 정보를 트리로 저장 (구간 내의 최댓/최솟값, 구간 내 모든 수의 합 등등)

→ **O(n)**

- 1) 구간에 대한 쿼리 (주어진 배열에서 구간 [2, 5]에서의 최댓값 찾기)
  2) 배열 내 임의의 원소를 수정한 후 트리에 반영

→ **O(logn)**

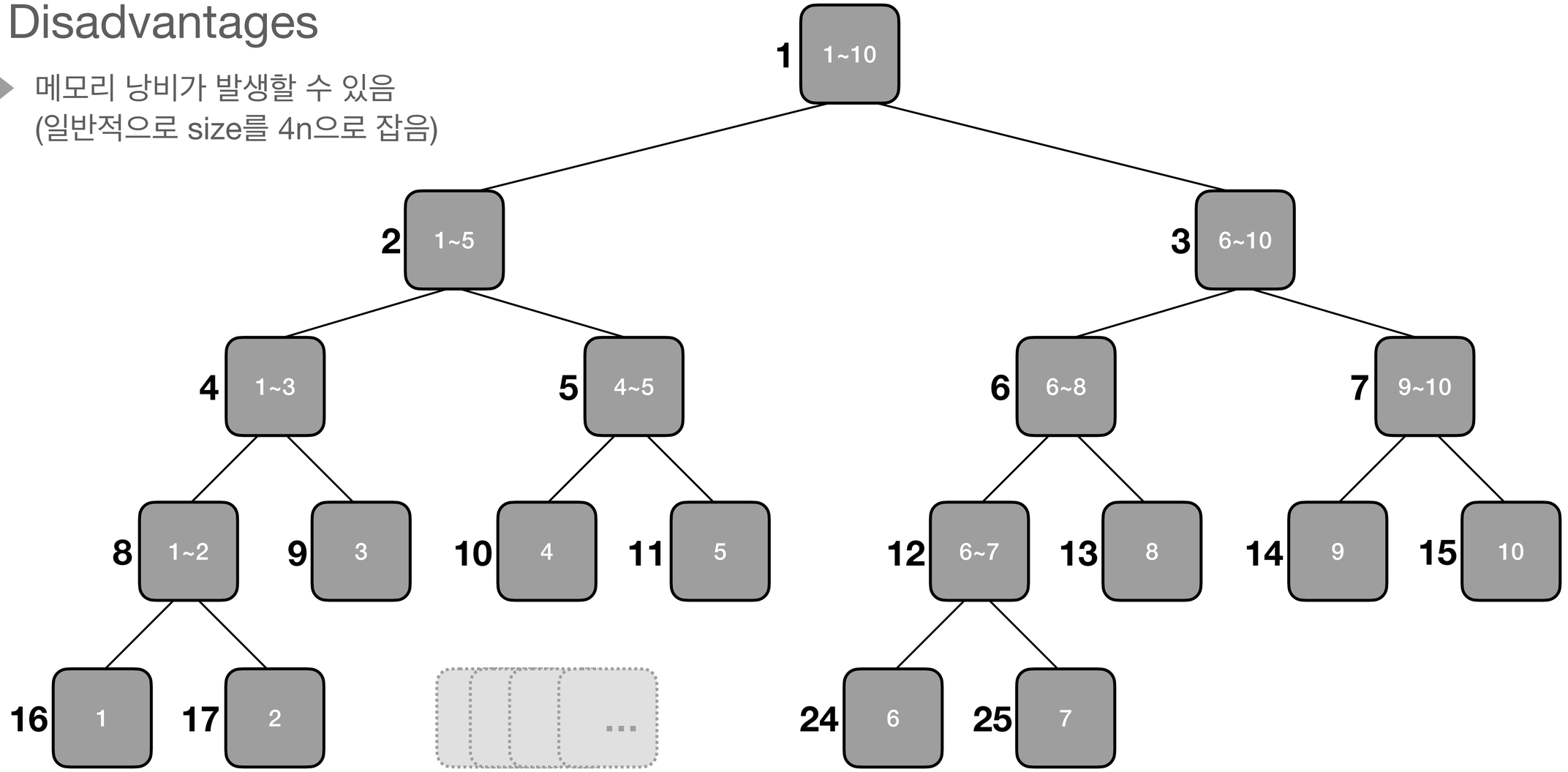
Segment Tree: Top-Down



Segment Tree: Top-Down

Disadvantages

➡ 메모리 낭비가 발생할 수 있음
(일반적으로 size를 4n으로 잡음)



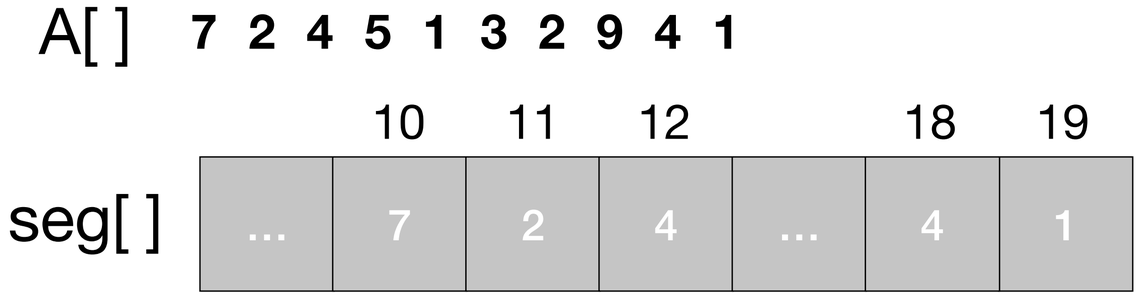
Segment Tree: Bottom-up

Advantages

- (개인적으로는 Top-Down이 더 직관적이어서 그 쪽을 선호하기는 하는데)
- 구현이 Top-Down보다 간결함
- 메모리를 **항상** 2n만 사용

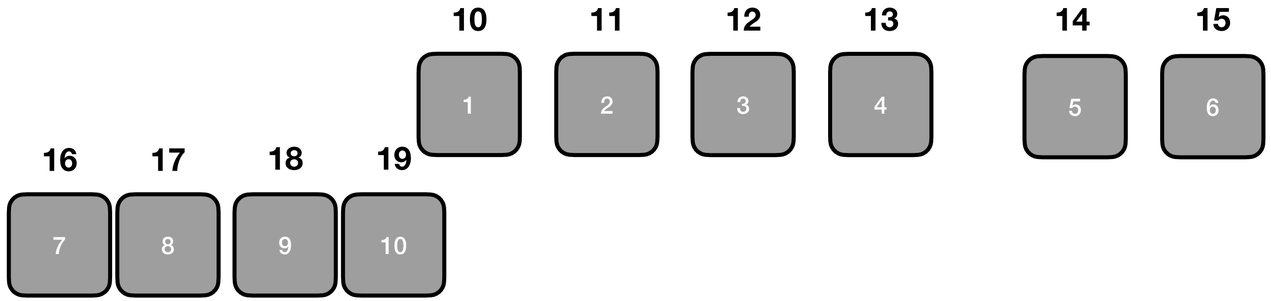
Segment Tree: Bottom-Up

Construction



construct()

1) 기존 배열 A의 길이가 N일 때, $A[i]$를 $segtree[N+i]$에 저장



Segment Tree: Bottom-Up

Construction

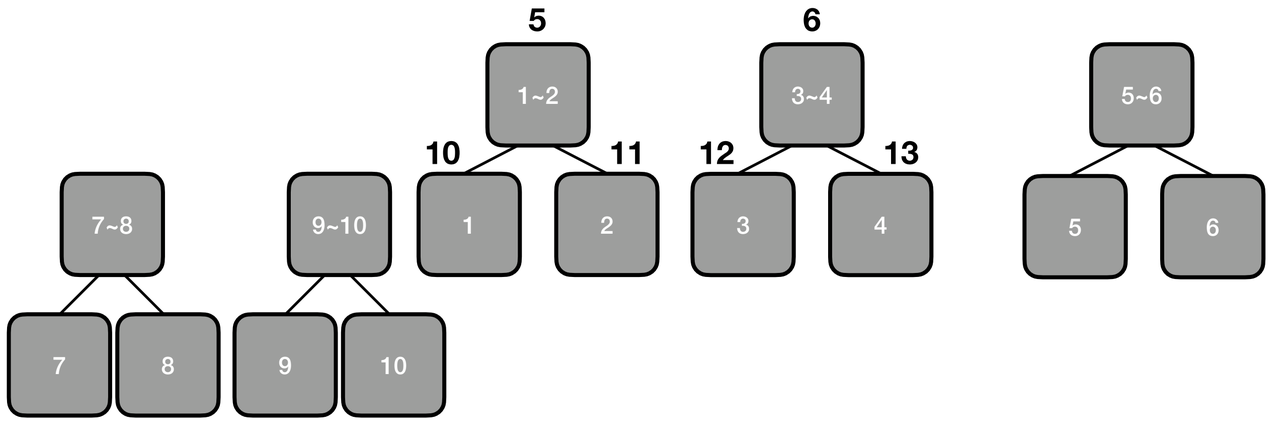
A[ ] **7 2 4 5 1 3 2 9 4 1**

	5	6		10	11	12	13
seg[ ]	9	9	…	7	2	4	5

construct()

1) 기존 배열 A의 길이가 N일 때, $A[i]$를 $segtree[N+i]$에 저장

2) $segtree[j] = segtree[2*j] + segtree[2*j+1]$ $(1 \le j \le N-1)$

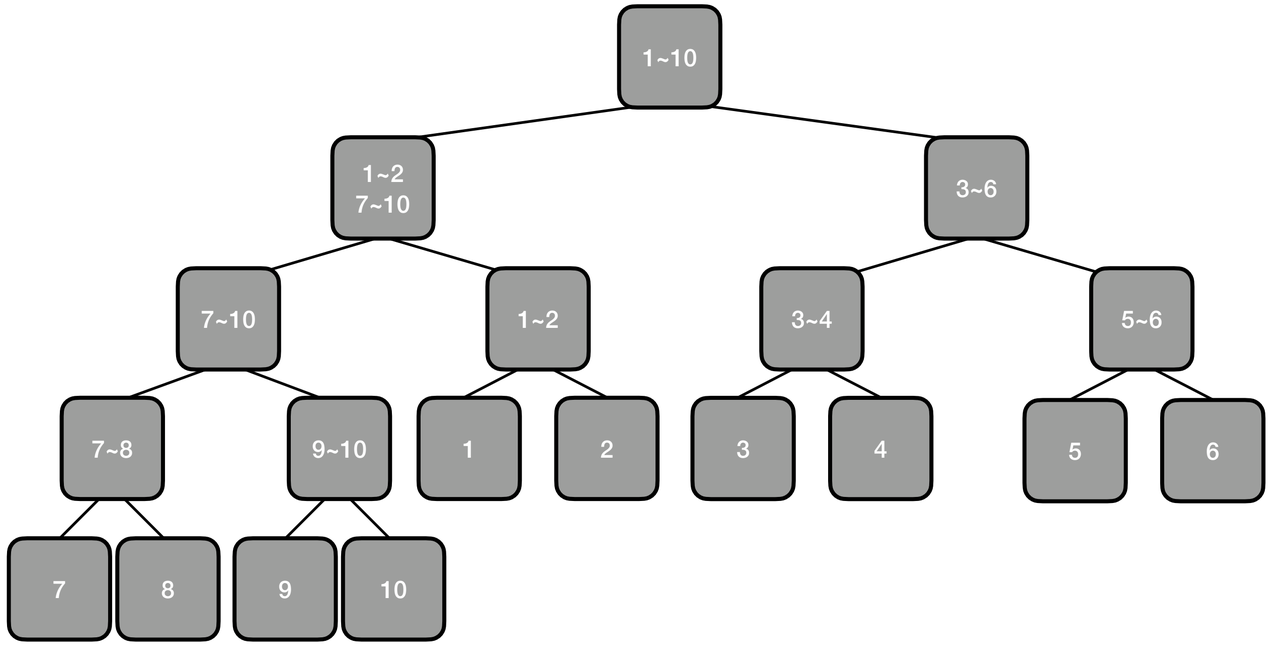


Segment Tree: Bottom-Up

Construction

A[ ] **7 2 4 5 1 3 2 9 4 1**

	5	6		10	11	12	13
seg[ ]	9	9	…	7	2	4	5



construct()

1) 기존 배열 A의 길이가 N일 때, $A[i]$를 $segtree[N+i]$에 저장

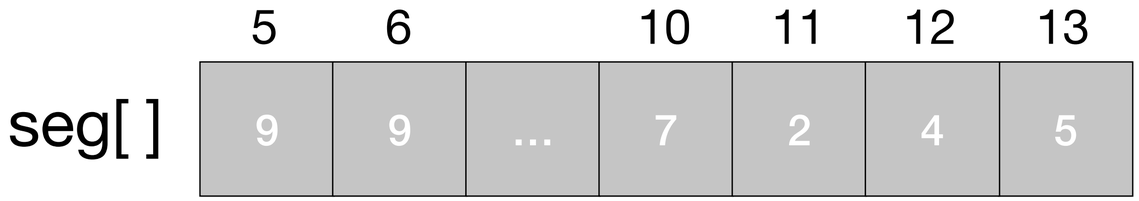
2) $segtree[j] = segtree[2*j] + segtree[2*j+1]$ $(1 \le j \le N-1)$

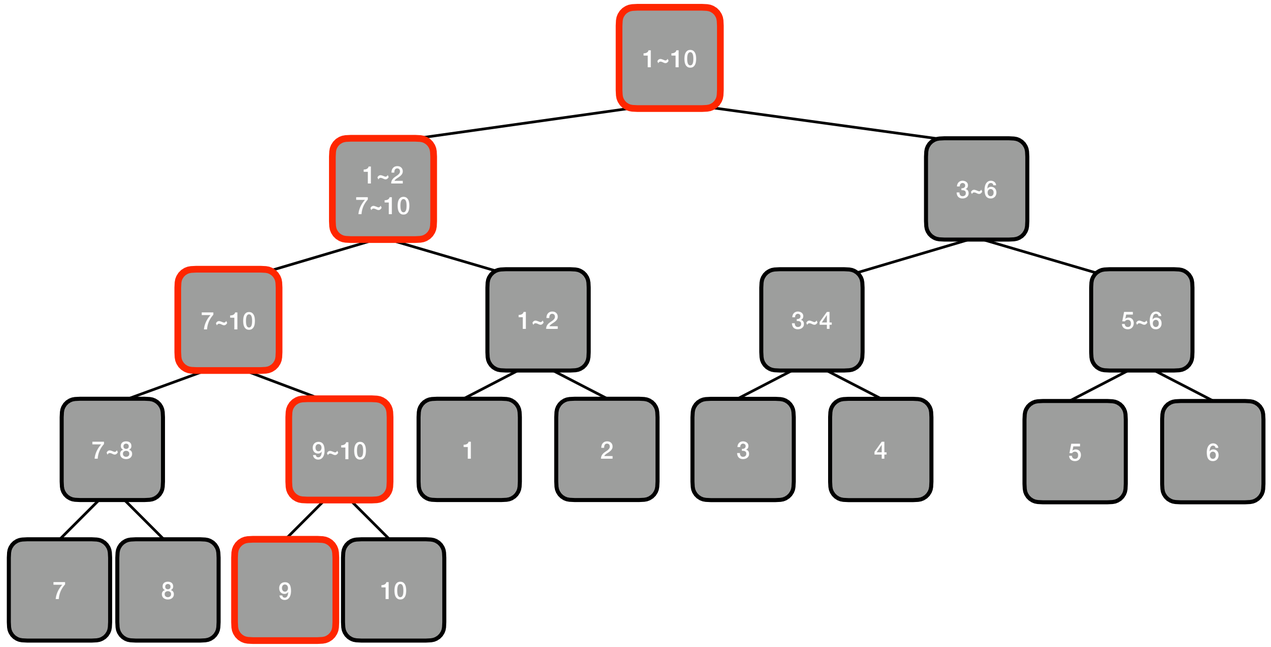
➡ 메모리를 **항상** 2n만 사용

Segment Tree: Bottom-Up

Update

A[ ] 7 2 4 5 1 3 2 9 4 1





구현 시 인덱스가 0부터 시작하는지 1부터 시작하는지를 꼭 고려하기!

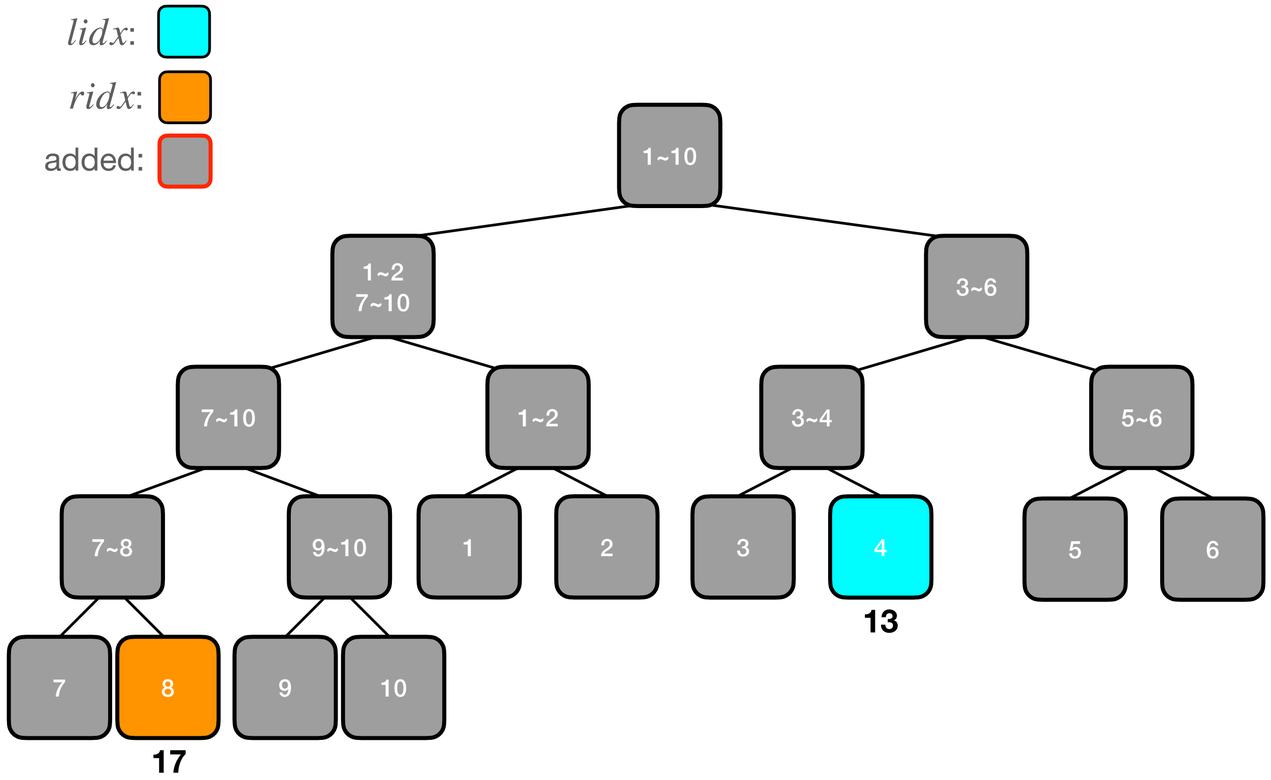
update(int target, int value)

1) $segtree[N + target]$의 값을 $value$로 변경

2) $N + target$을 2로 계속 나누면서 얻을 수 있는 값 k $(k \geq 1)$에 대해 $segtree[k]$의 값을 새로운 $segtree[2 * k] + segtree[2 * k + 1]$ 값으로 변경

Segment Tree: Bottom-Up

Query



query(4, 7) sum: 0

구현 시 인덱스가 0부터 시작하는지 1부터 시작하는지를 꼭 고려하기!

query(int left, int right)

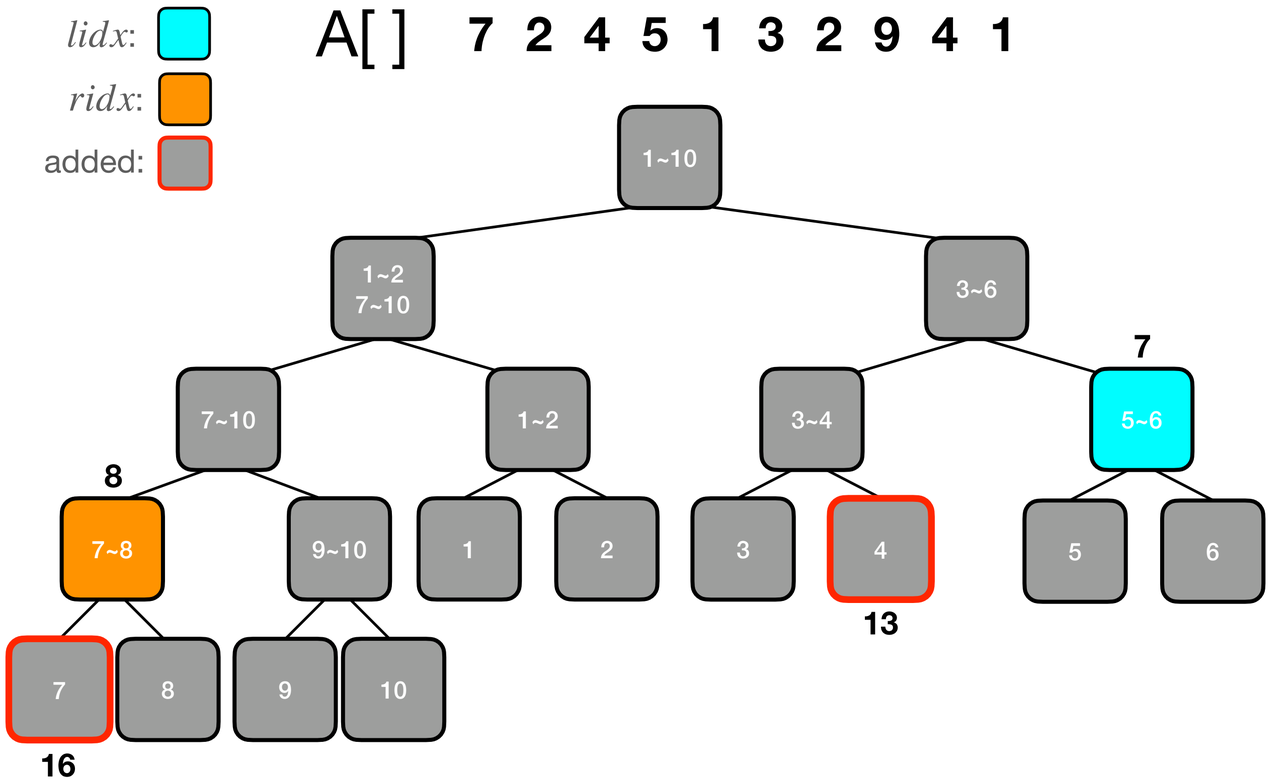
1) 변수 $lidx$와 $ridx$를 갱신하며 쿼리의 결과를 구할 것

범위 $[lidx, ridx)$ 의 합을 구할 거라고 생각하면 됨

초기값은 $lidx = N + left,\ ridx = N + right + 1$로 설정

Segment Tree: Bottom-Up

Query



query(4, 7) sum: 0 + 7 = 7

구현 시 인덱스가 0부터 시작하는지 1부터 시작하는지를 꼭 고려하기!

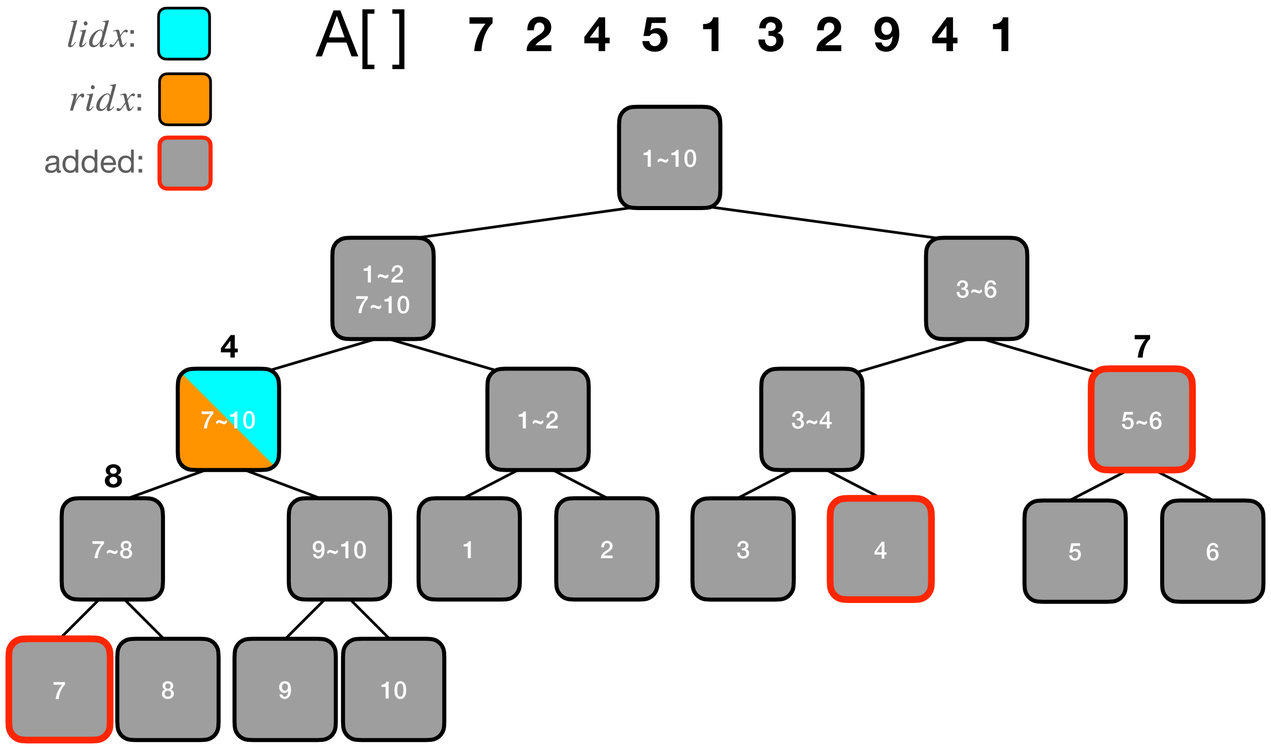
query(int left, int right)

1) 변수 $lidx$와 $ridx$를 갱신하며 쿼리의 결과를 구할 것
범위 $[lidx, ridx)$ 의 합을 구할 거라고 생각하면 됨
초기값은 $lidx = N + left,\ ridx = N + right + 1$로 설정

2) $lidx < ridx$ 인 동안 다음을 반복
- $lidx$가 트리에서 오른쪽 노드일 경우 (인덱스가 1부터 시작하면 $lidx \% 2 = 1$일 때),
$segtree[lidx]$를 결과값에 더해주고 $lidx = lidx + 1$
- $ridx$가 트리에서 오른쪽 노드일 경우,
$ridx = ridx - 1$를 한 후에 $segtree[ridx]$를 결과값에 합산
- 이후 $lidx$와 $ridx$ 모두 2로 나누어 부모 노드로 올리기

Segment Tree: Bottom-Up

Query



query(4, 7) sum: 7 + 4 = 11

구현 시 인덱스가 0부터 시작하는지 1부터 시작하는지를 꼭 고려하기!

query(int left, int right)

1) 변수 $lidx$와 $ridx$를 갱신하며 쿼리의 결과를 구할 것
범위 $[lidx, ridx)$ 의 합을 구할 거라고 생각하면 됨
초기값은 $lidx = N + left,\ ridx = N + right + 1$로 설정

2) $lidx < ridx$ 인 동안 다음을 반복
- $lidx$가 트리에서 오른쪽 노드일 경우 (인덱스가 1부터 시작하면 $lidx \% 2 = 1$일 때),
$segtree[lidx]$를 결과값에 더해주고 $lidx = lidx + 1$
- $ridx$가 트리에서 오른쪽 노드일 경우,
$ridx = ridx - 1$를 한 후에 $segtree[ridx]$를 결과값에 합산
- 이후 $lidx$와 $ridx$ 모두 2로 나누어 부모 노드로 올리기

Does it work?

저렇게 하면 쿼리 결과를 잘 구할 수 있어요??

➡ **그러게요..?(죄송…)**

➡ $lidx$에서는 $segtree[lidx]$가 나타내는 범위 및 그 이후의 것을 합산하도록 하고
$ridx$에서는 $segtree[ridx]$이전의 범위를 합산하도록
각 변수를 어케어케 잘 컨트롤하는 것으로 이해했음…

2D Segment Tree

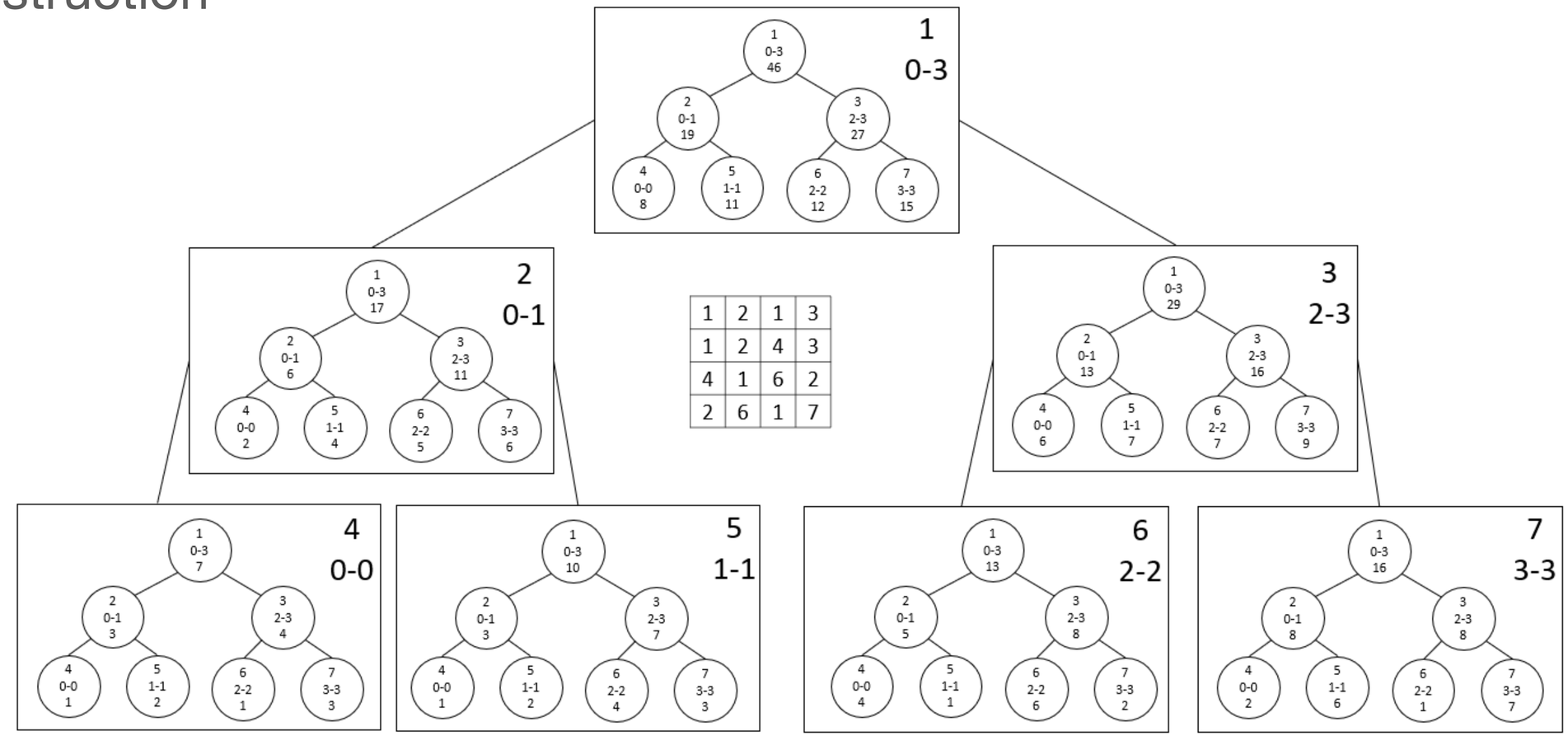
- 이차원 배열에 대한 구간 정보를 트리로 저장

➡ 많은 메모리를 필요로 하기 때문에 bottom-up 방식으로 구현하면 유리
top-down으로 하면 dynamic segment tree라는게 있다는데…😇

- 1) 구간에 대한 쿼리
  2) 배열 내 임의의 원소를 수정한 후 트리에 반영

2D Segment Tree

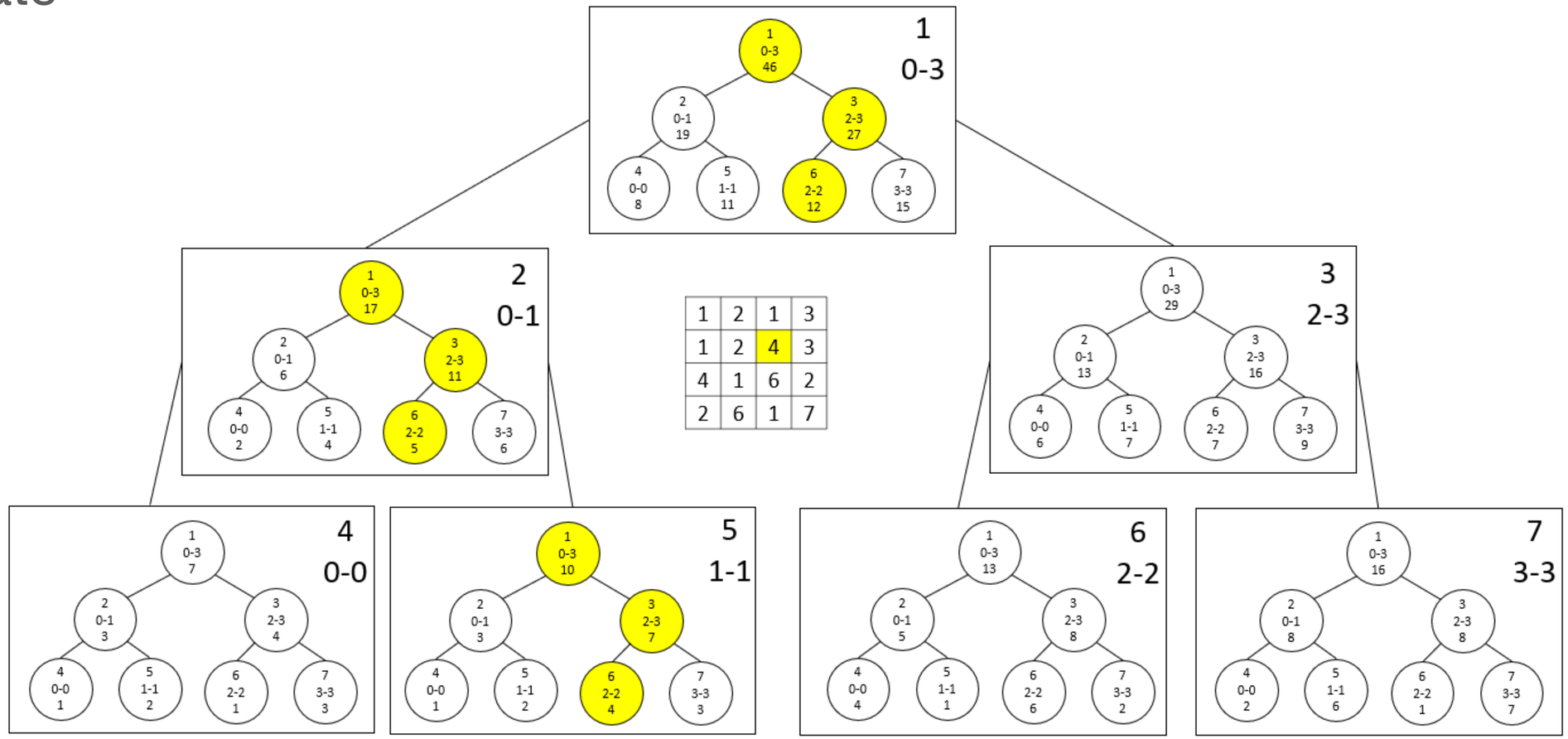
Construction



http://www.secmem.org/blog/2019/11/15/2D-segment-tree/

2D Segment Tree

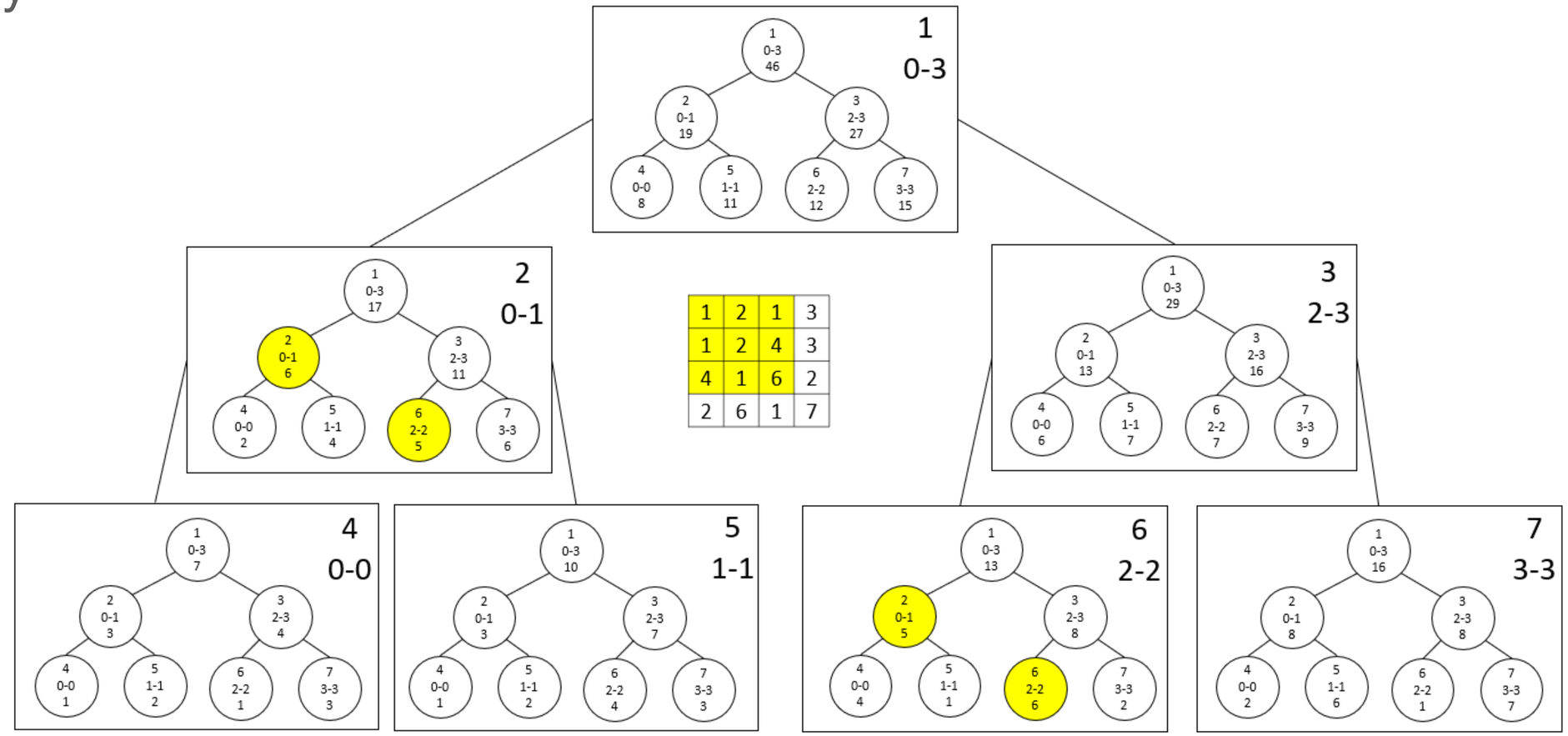
Update



http://www.secmem.org/blog/2019/11/15/2D-segment-tree/

2D Segment Tree

Query



http://www.secmem.org/blog/2019/11/15/2D-segment-tree/