KPSC
Algorithm 심화 스터디

01 - Segment Tree Basic

2025. 12. 19. (Fri.) 21:00 ~

Problem Definition

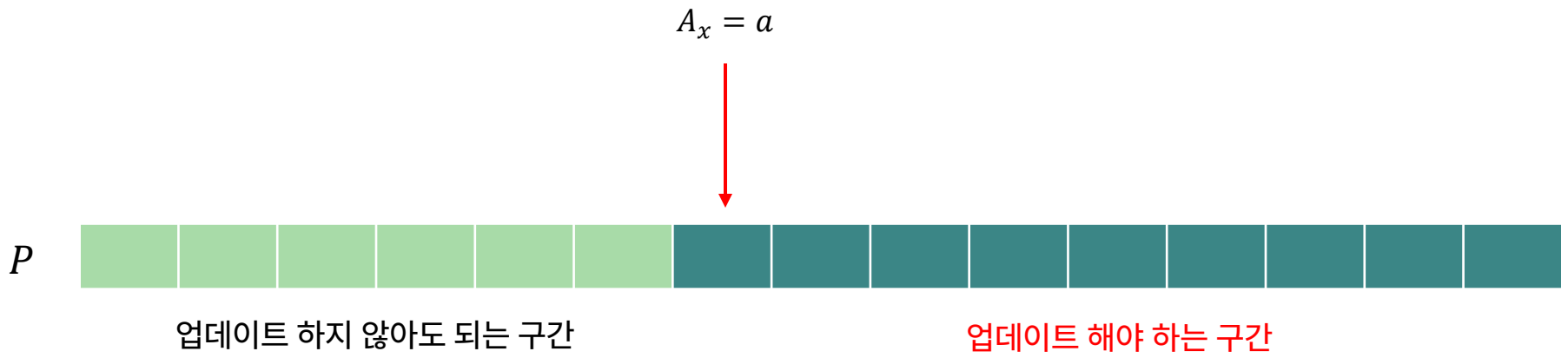
- 어떤 정수로 이루어진 배열 A가 주어지고, 다음 쿼리가 주어질 때 해결법을 고민해봅시다.

- $1\ x\ a$: $A_x = a$로 업데이트 한다.
- $2\ x\ y$: $A_x + A_{x+1} + \cdots + A_{y-1} + A_y$의 값을 출력한다.

How to Solve? – prefix sum

- 먼저, prefix sum을 사용해서 구해보고자 합니다.
- 2번 쿼리가 주어지면 $P_y - P_{x-1}$ 의 값을 출력하면 되므로, 이때 시간 복잡도는 $O(1)$이 됩니다.
- 하지만, 1번 쿼리가 주어지면 P의 값을 다시 계산해야 합니다.

How to Solve? – prefix sum



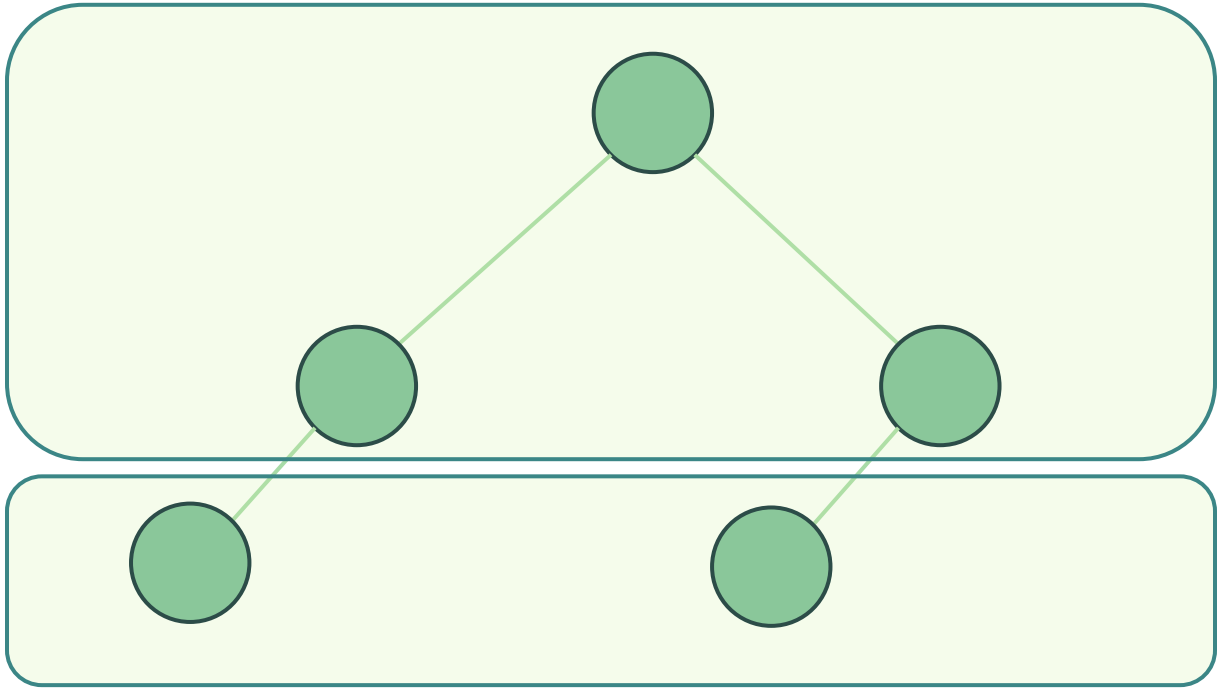
- 쿼리 1이 들어왔을 때 최악의 경우 P를 모두 업데이트 해야 하는 상황이 발생합니다.

- 따라서 각 쿼리마다 $O(N)$의 시간 복잡도를 갖게 되어 시간초과가 나게 됩니다.

Introduction to Segment Tree

- Segment Tree는 Binary Tree를 기반으로 해서 구간의 정보를 효율적으로 다룰 수 있는 자료 구조입니다.
- Segment Tree는 Complete Binary Tree입니다.

*Complete Binary Tree



상위 level은 perfect binary tree

가장 마지막 level에 있는 노드들은
왼쪽부터 우선적으로 채워짐
(level의 가장 왼쪽이 아님)

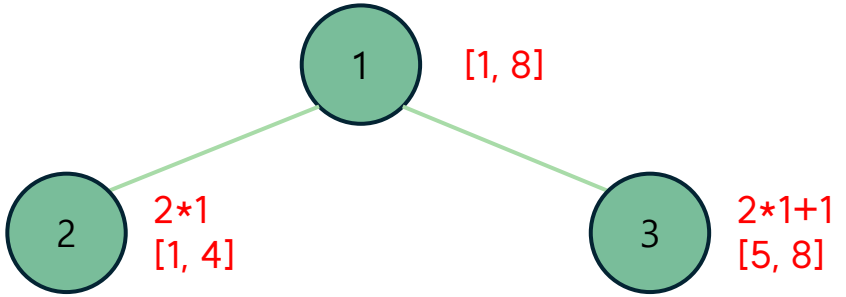
Introduction to Segment Tree

- 이제 Segment Tree를 구축하는 법을 배워보겠습니다.
- $A = [2, 3, 5, 7, 11, 13, 17, 19]$라고 하고, 우리가 원하는 것은 구간의 합이라고 가정합시다.
- 또한 각 노드에 저장된 값을 담는 배열 $Seg[]$도 만들어 둡시다.

Introduction to Segment Tree

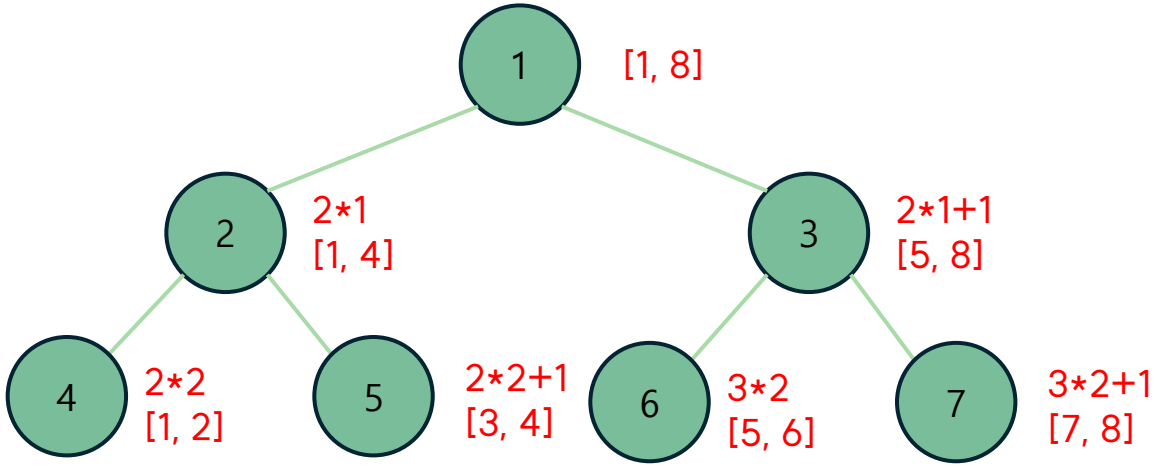
- 가장 상위 노드는 전체 구간을 관리하는 노드입니다. (노드 1) 즉, $L = 1, R = 8$을 관리합니다.
- $L \neq R$이라면, 구간을 더 쪼개서 자식 노드를 만들어야 합니다. 이때 자식 노드가 관리하는 구간들이 겹치지 않고 거의 비슷한 크기의 구간을 관리할 수 있게, $M = \frac{L+R}{2}$값을 이용해서 각각의 자식 노드가 $[L, M], [M + 1, R]$을 관리하게 합시다.

Introduction to Segment Tree



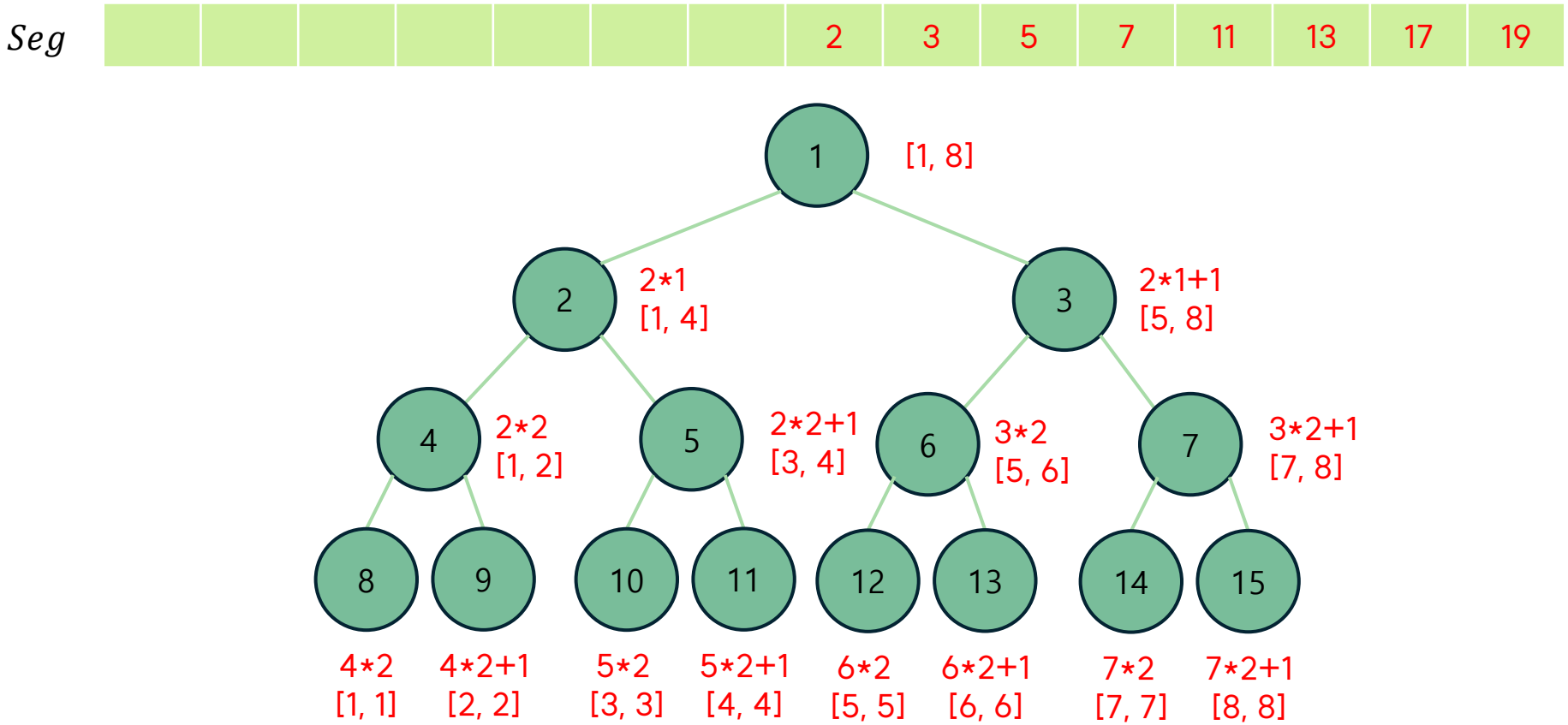
- 그 다음, 각 자식 노드들에 대해서도(노드 2, 3) 아직 $L \neq R$이므로 다시 분할하는 과정을 거칩시다.

Introduction to Segment Tree



- 그 다음, 각 자식 노드들에 대해서도(노드 4, 5, 6, 7) 아직 $L \neq R$이므로 다시 분할하는 과정을 거칩시다.

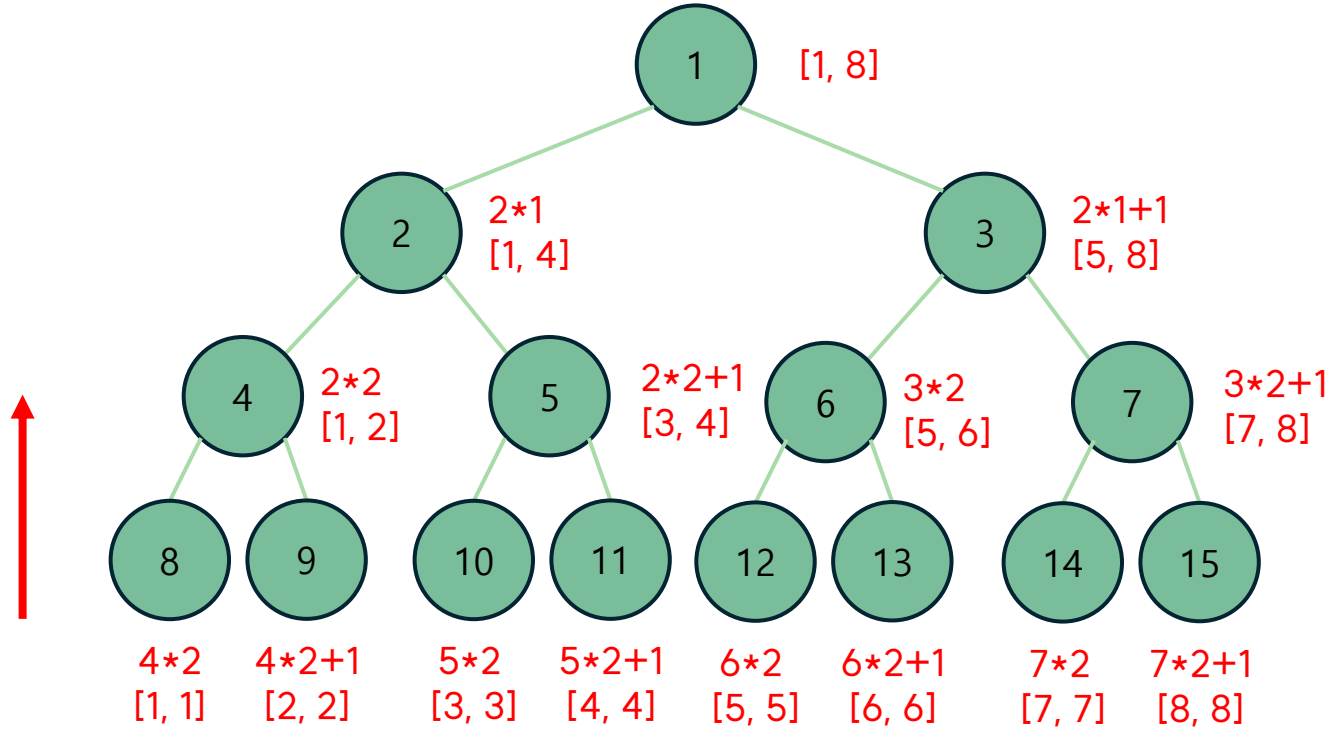
Introduction to Segment Tree



- 이제 모든 자식 노드가 $L = R$입니다.
- 각 노드에 대해서 만약 $L = R$이라면, 더 이상 분할하지 않고 종료하면 됩니다. 이때 각 노드가 원래 배열의 i번째 값을 가리키고 있으므로, $Seg[node] = A_i$를 저장합시다.

Introduction to Segment Tree

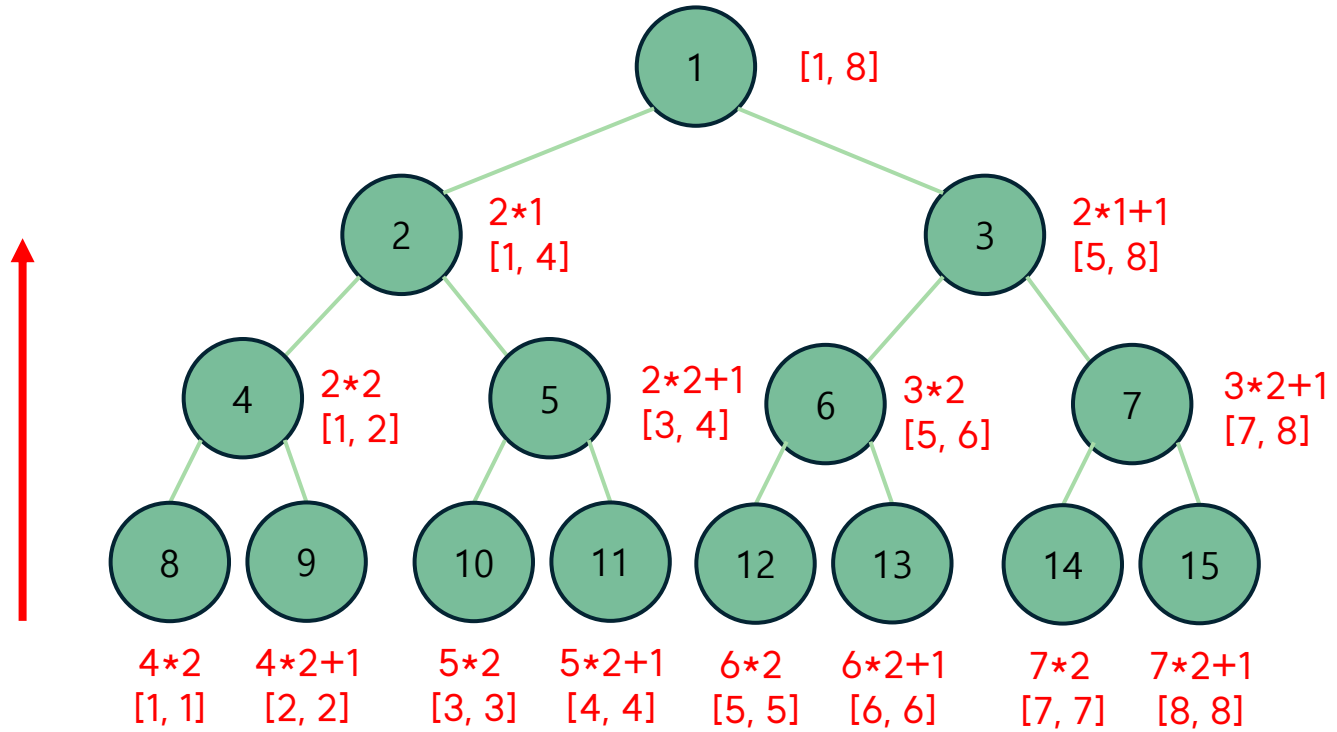
Seg				5	12	24	36	2	3	5	7	11	13	17	19



- 이제 상위 level에 있는 노드들이 자식 노드들의 정보를 담을 수 있도록, 자식 노드들의 값을 더해서 저장합시다.
  - 상위 노드가 자식 노드들이 관리하고 있는 구간의 정보를 합쳐서 관리하기 때문입니다.
- 즉, $Seg[node] = Seg[child_L] + Seg[child_R]$입니다.

Introduction to Segment Tree

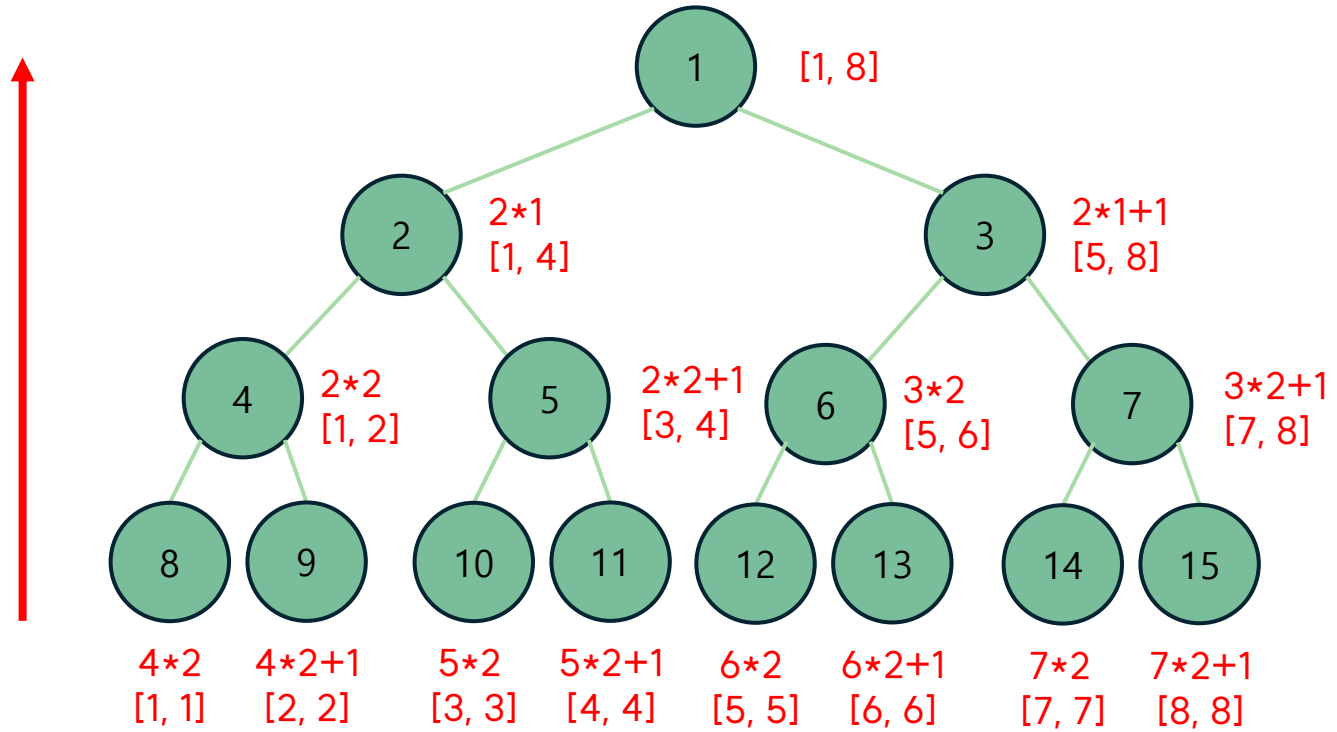
Seg		17	60	5	12	24	36	2	3	5	7	11	13	17	19



- 이제 상위 level에 있는 노드들이 자식 노드들의 정보를 담을 수 있도록, 자식 노드들의 값을 더해서 저장합시다.
  - 상위 노드가 자식 노드들이 관리하고 있는 구간의 정보를 합쳐서 관리하기 때문입니다.
- 즉, $Seg[node] = Seg[child_L] + Seg[child_R]$입니다.

Introduction to Segment Tree

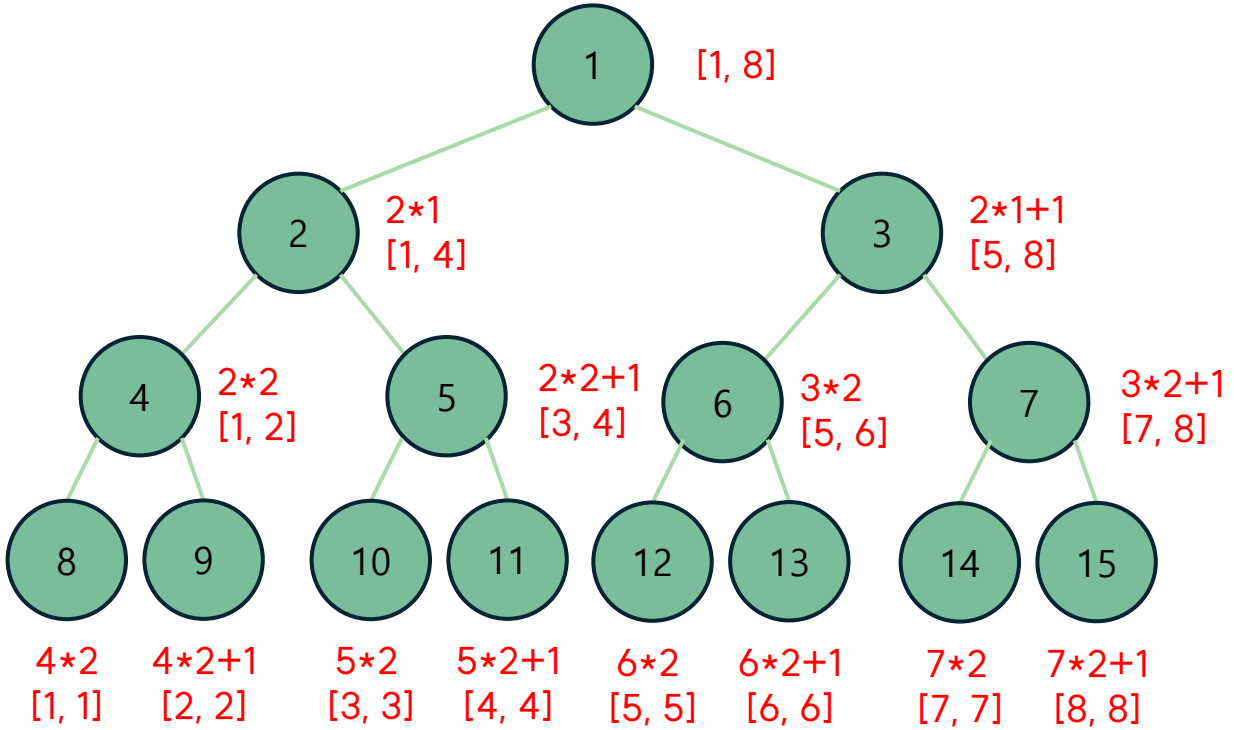
Seg	77	17	60	5	12	24	36	2	3	5	7	11	13	17	19



- 이제 상위 level에 있는 노드들이 자식 노드들의 정보를 담을 수 있도록, 자식 노드들의 값을 더해서 저장합시다.
  - 상위 노드가 자식 노드들이 관리하고 있는 구간의 정보를 합쳐서 관리하기 때문입니다.
- 즉, $Seg[node] = Seg[child_L] + Seg[child_R]$입니다.

Introduction to Segment Tree

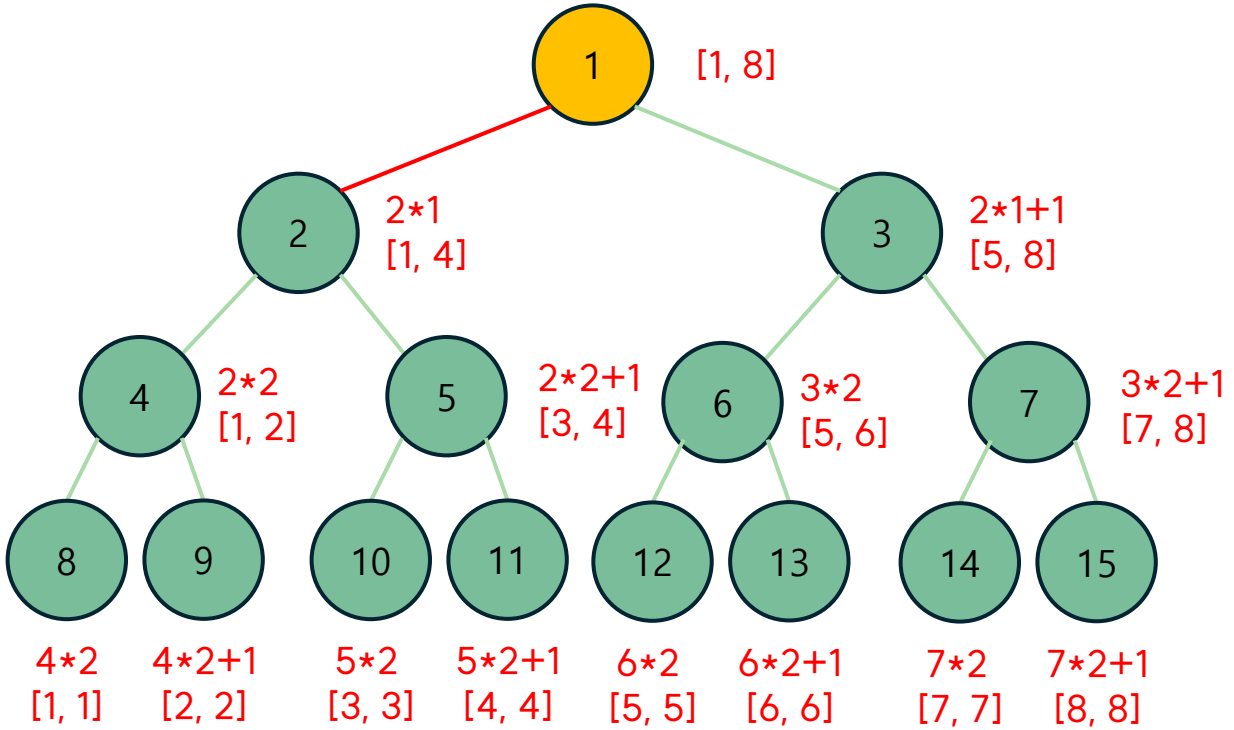
Seg	77	17	60	5	12	24	36	2	3	5	7	11	13	17	19



- 그럼 이제, 원소가 업데이트 될 때 Segment Tree에서 일어나는 일을 한번 관찰해봅시다.
- 예를 들어, $A[3] = 5 \rightarrow 31$로 업데이트 된다고 가정해봅시다.

Introduction to Segment Tree

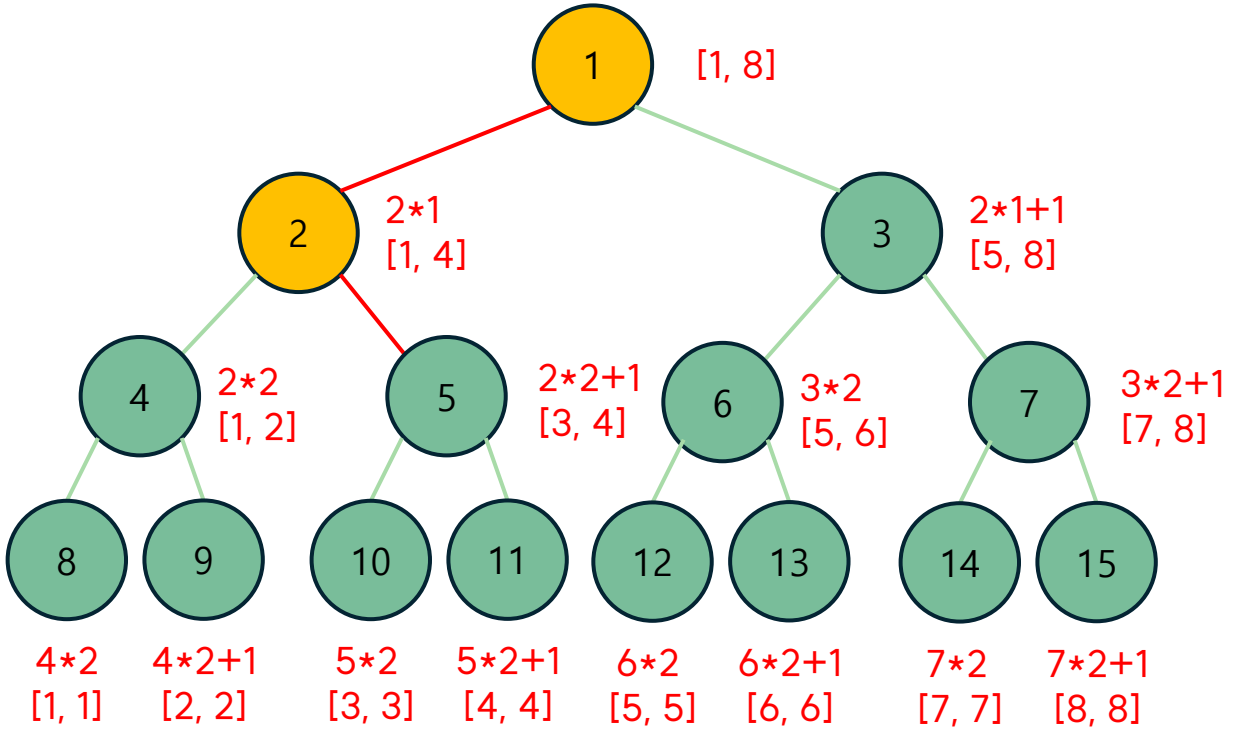
Seg	77	17	60	5	12	24	36	2	3	5	7	11	13	17	19



- 먼저 root에서 시작합니다.
- Index 3은 자식 노드가 관리하는 노드 중 왼쪽 자식 노드에 속하므로 왼쪽 자식으로 이동합니다.

Introduction to Segment Tree

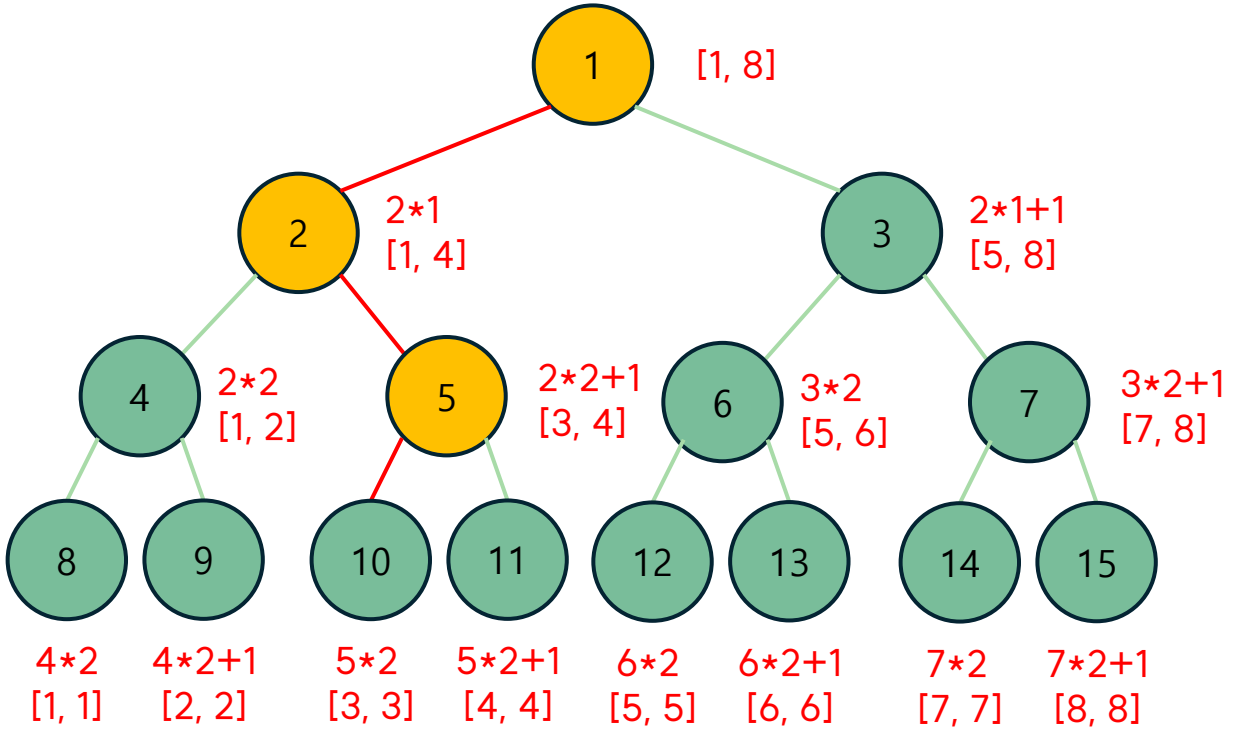
Seg	77	17	60	5	12	24	36	2	3	5	7	11	13	17	19



- 현재 노드 2에서 index 3을 관리하는 노드는 오른쪽 자식 노드입니다.
- 따라서, 오른쪽 자식 노드로 이동합니다.

Introduction to Segment Tree

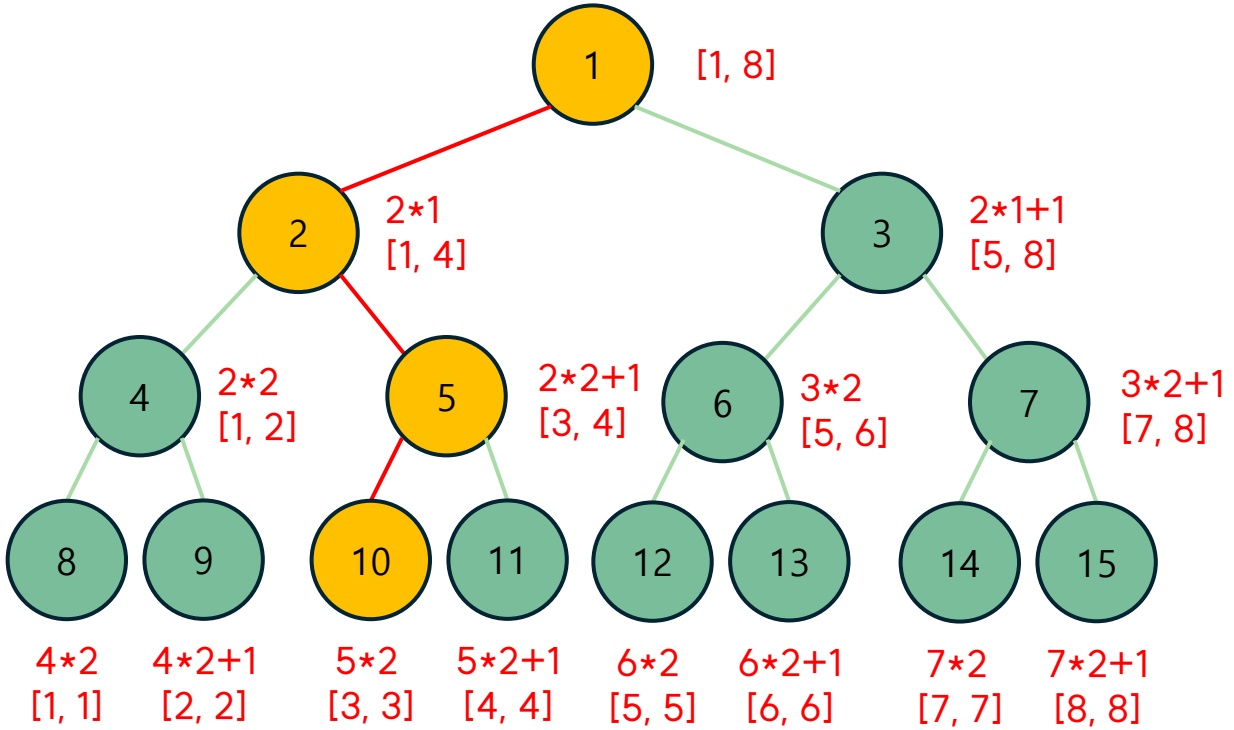
Seg	77	17	60	5	12	24	36	2	3	5	7	11	13	17	19



- 현재 노드 5에서 index 3을 관리하는 노드는 왼쪽 자식 노드입니다.
- 따라서, 왼쪽 자식 노드로 이동합니다.

Introduction to Segment Tree

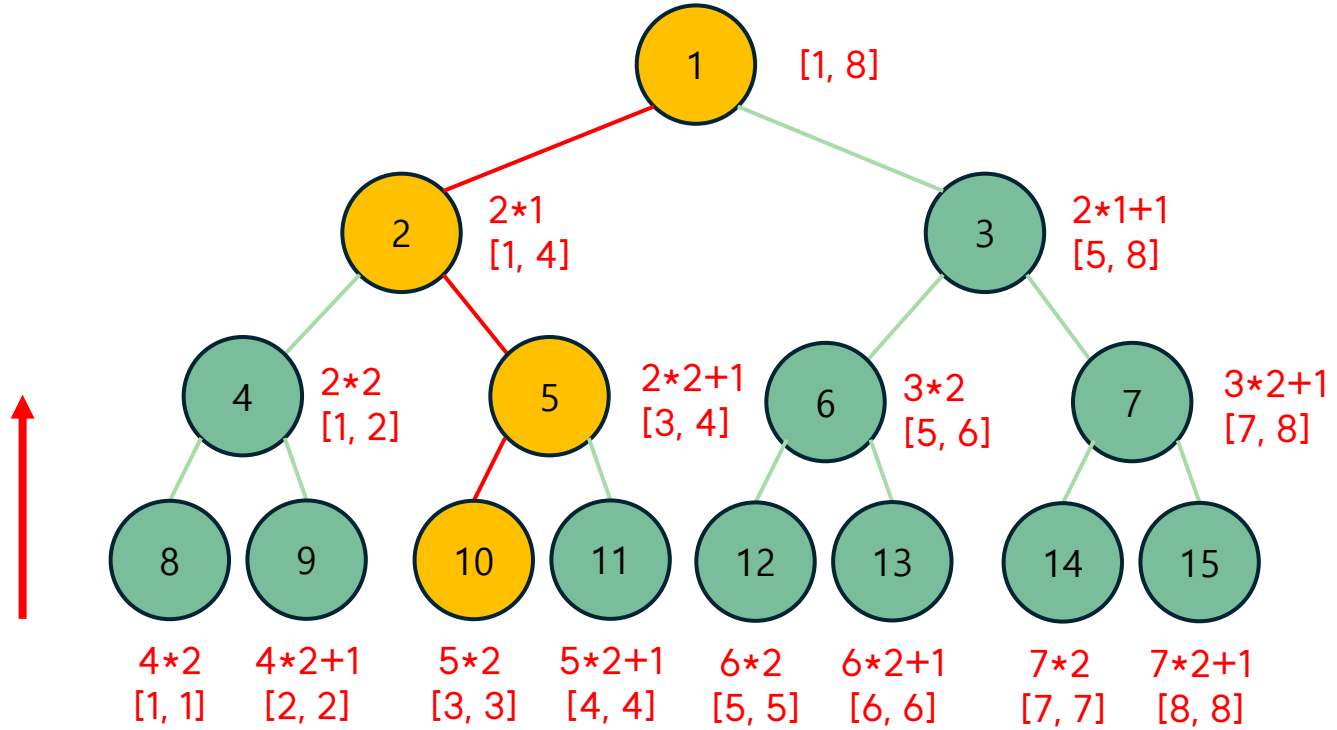
Seg	77	17	60	5	12	24	36	2	3	31	7	11	13	17	19



- 이제 더 이상 자식 노드가 없습니다. 즉, index 3을 관리하는 노드는 노드 10임을 알 수 있습니다.
- 따라서 $Seg[10] = 5 \rightarrow 31$로 바꿔줍니다.

Introduction to Segment Tree

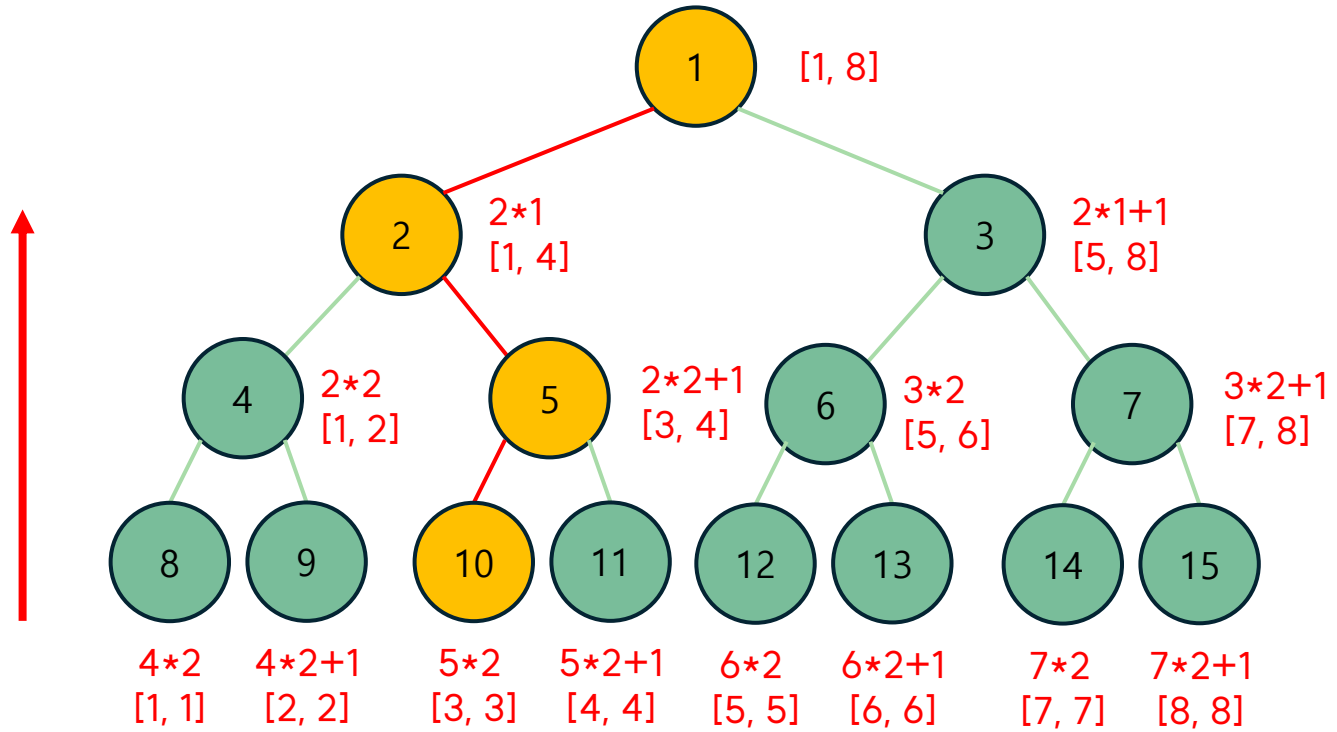
Seg	77	17	60	5	34	24	36	2	3	31	7	11	13	17	19



- 이제 자식 노드의 값이 바뀌었으니 자식들의 구간을 관리하는 부모 노드의 값들도 업데이트 해줘야 합니다.
- 거슬러 올라가서 업데이트 해줍시다.

Introduction to Segment Tree

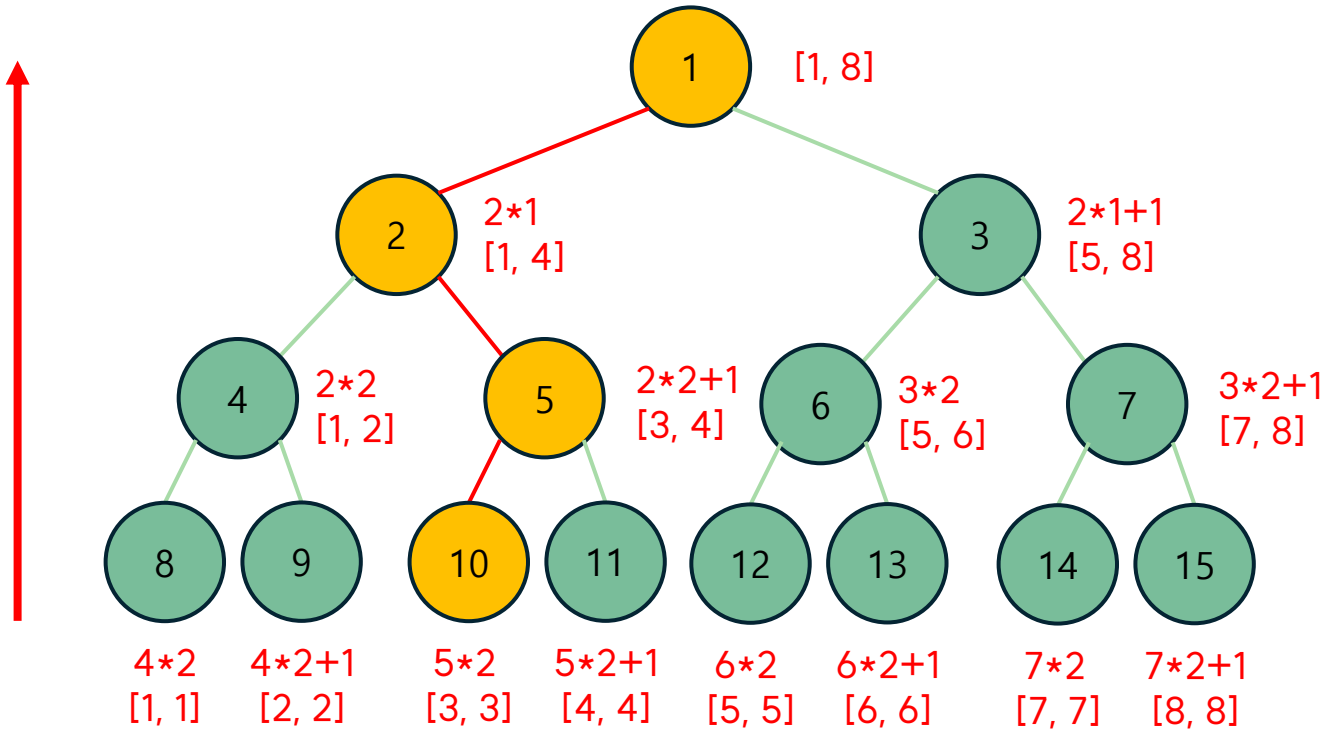
Seg	77	39	60	5	34	24	36	2	3	31	7	11	13	17	19



- 이제 자식 노드의 값이 바뀌었으니 자식들의 구간을 관리하는 부모 노드의 값들도 업데이트 해줘야 합니다.
- 거슬러 올라가서 업데이트 해줍시다.

Introduction to Segment Tree

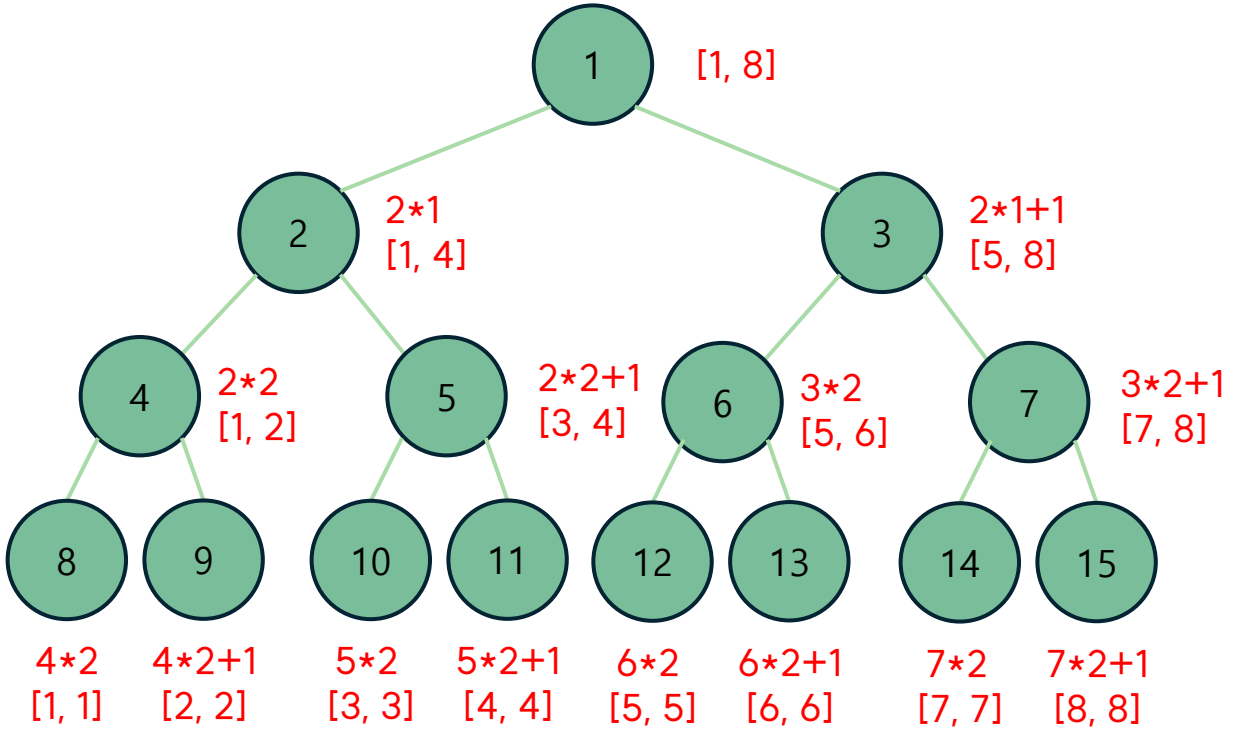
Seg	99	39	60	5	34	24	36	2	3	31	7	11	13	17	19



- 이제 자식 노드의 값이 바뀌었으니 자식들의 구간을 관리하는 부모 노드의 값들도 업데이트 해줘야 합니다.
- $Seg[node] = Seg[child_L] + Seg[child_R]$ 를 이용해서 거슬러 올라가면서 업데이트 해줍시다.

Introduction to Segment Tree

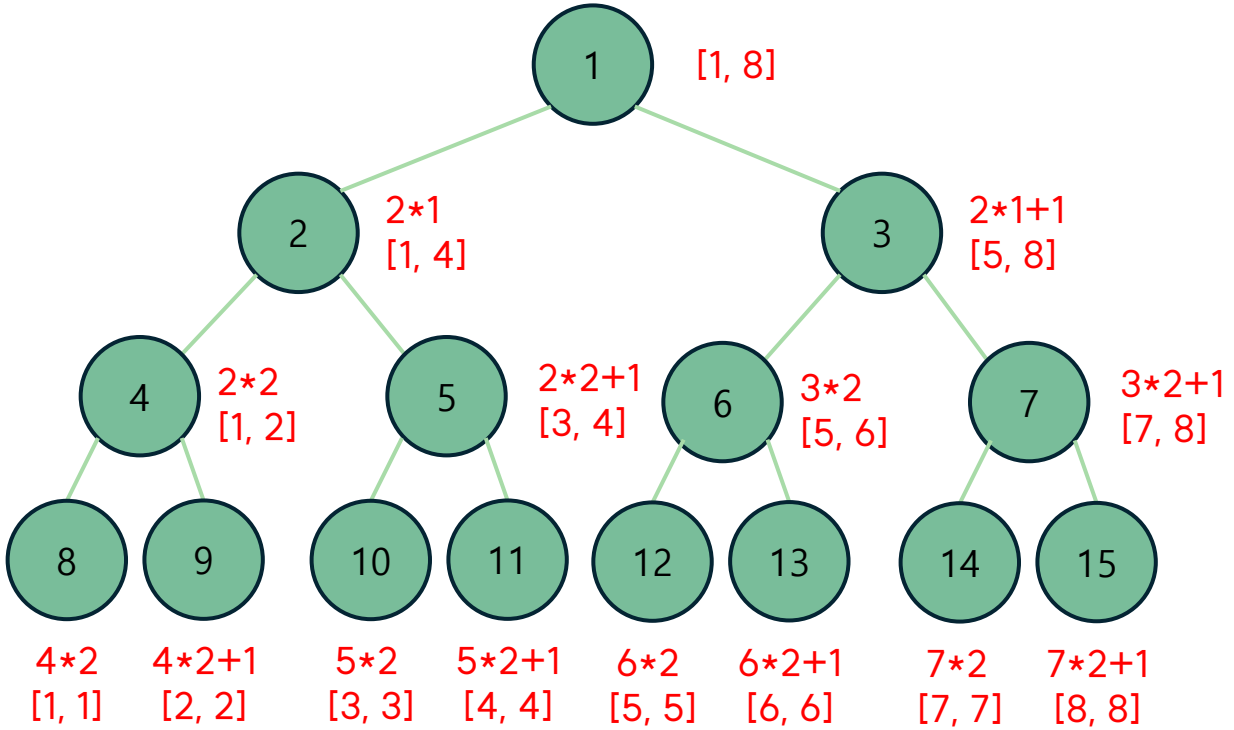
Seg	99	39	60	5	34	24	36	2	3	31	7	11	13	17	19



- 만약 세그먼트 트리에서 A의 크기가 $n = 2^k$라면, perfect binary tree가 됩니다. (위 예시)
- 이때는 필요한 노드의 개수가 $1 + 2 + \cdots + n$이므로 $(n - 1) + n = 2n - 1$이 됩니다. 따라서 시간 복잡도는 $O(n)$입니다.

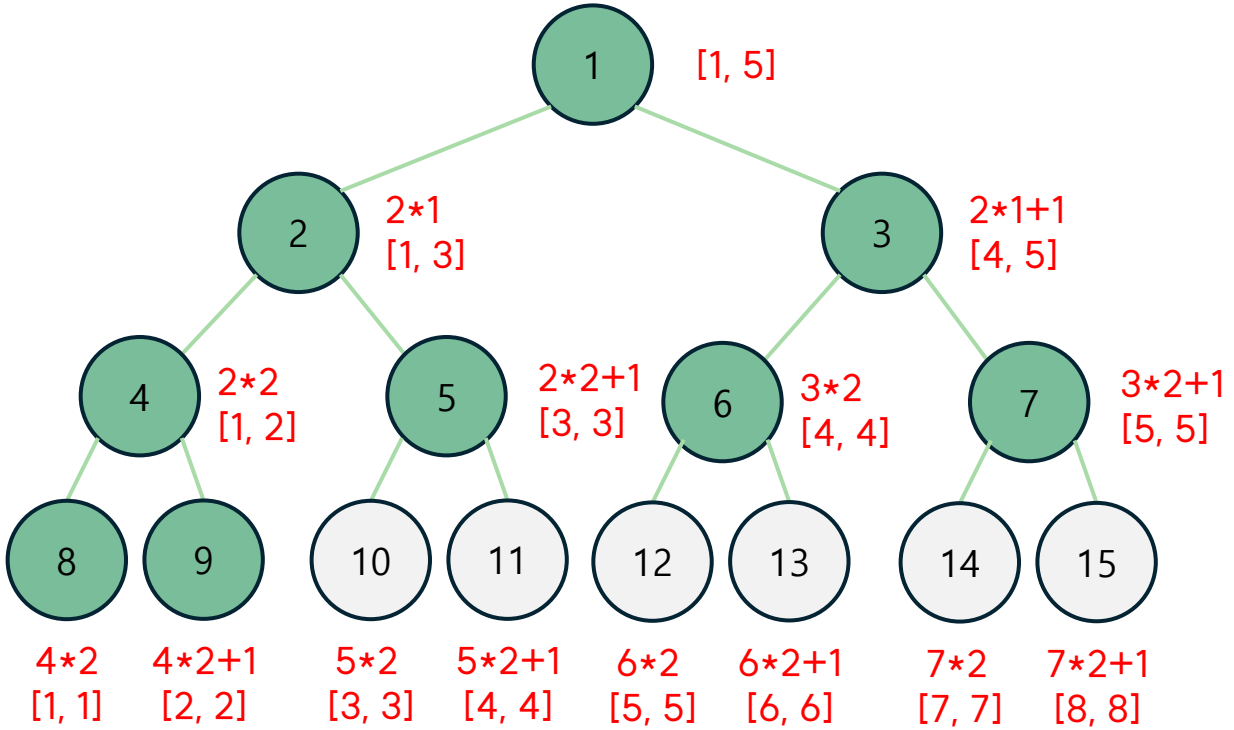
Introduction to Segment Tree

Seg	99	39	60	5	34	24	36	2	3	31	7	11	13	17	19



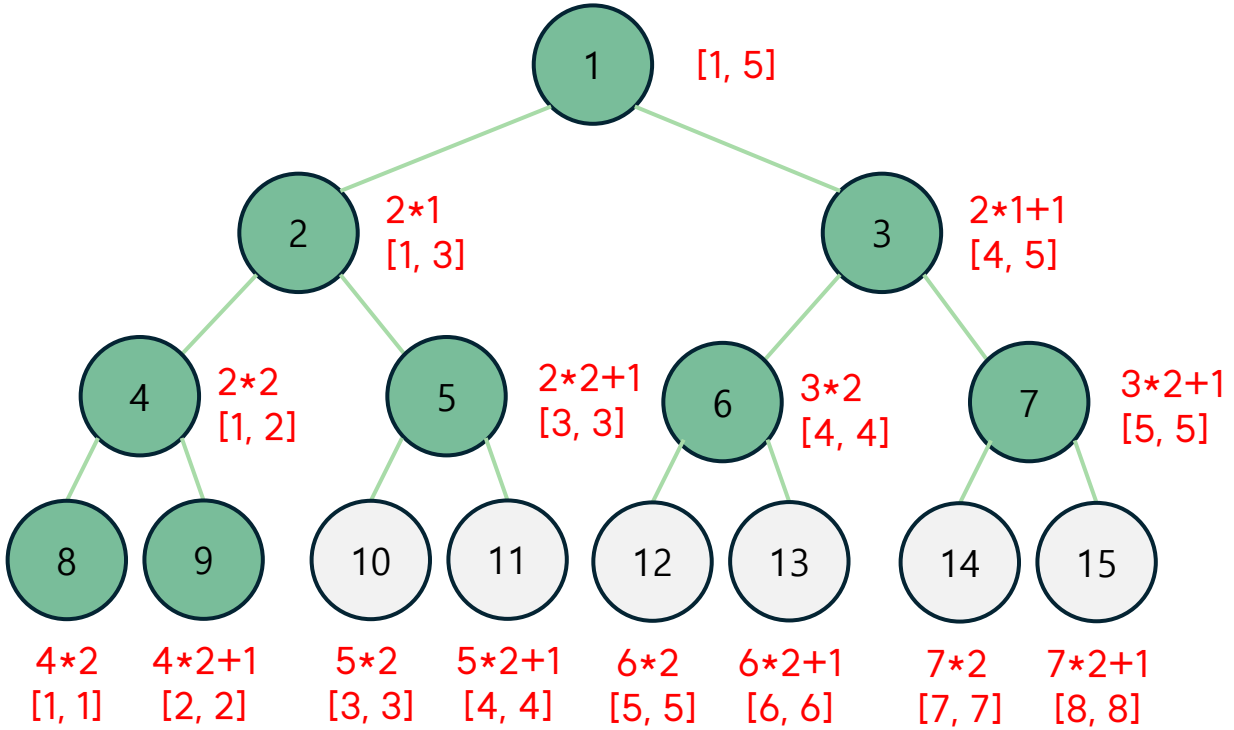
- 하지만, $n \neq 2^k$라면 조금 달라집니다.

Introduction to Segment Tree



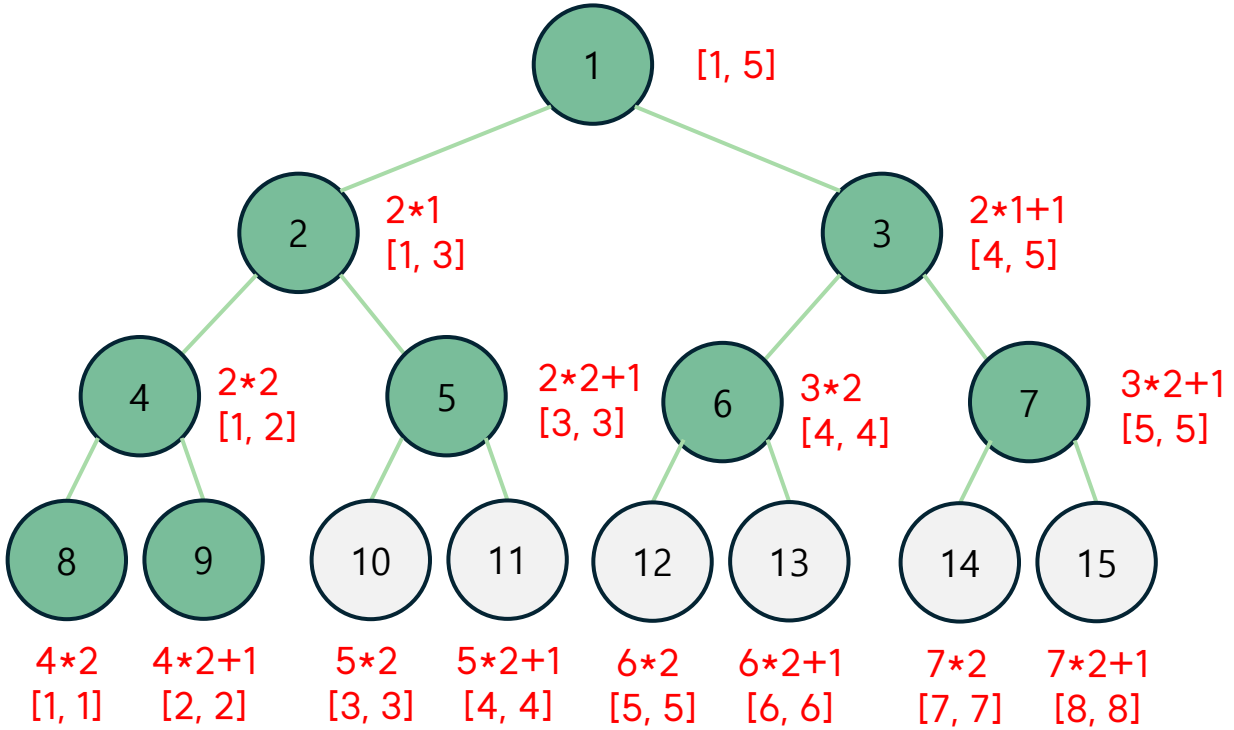
- $n \neq 2^k$라면 위의 예시와 같이 complete binary tree가 생기게 됩니다.
- 그렇다면, 마지막 level에 유령 노드들을 한번 추가해봅시다.

Introduction to Segment Tree



- 계산의 편의를 위해 유령 노드를 추가했습니다. $n \neq 2^k$인 경우 n보다 크거나 같은 $n' = 2^t$를 잡아서 계산하면 됩니다.
- Segment tree를 만들기 위해서는 $1 + 2 + 4 + \cdots + n'$개의 노드가 필요합니다.

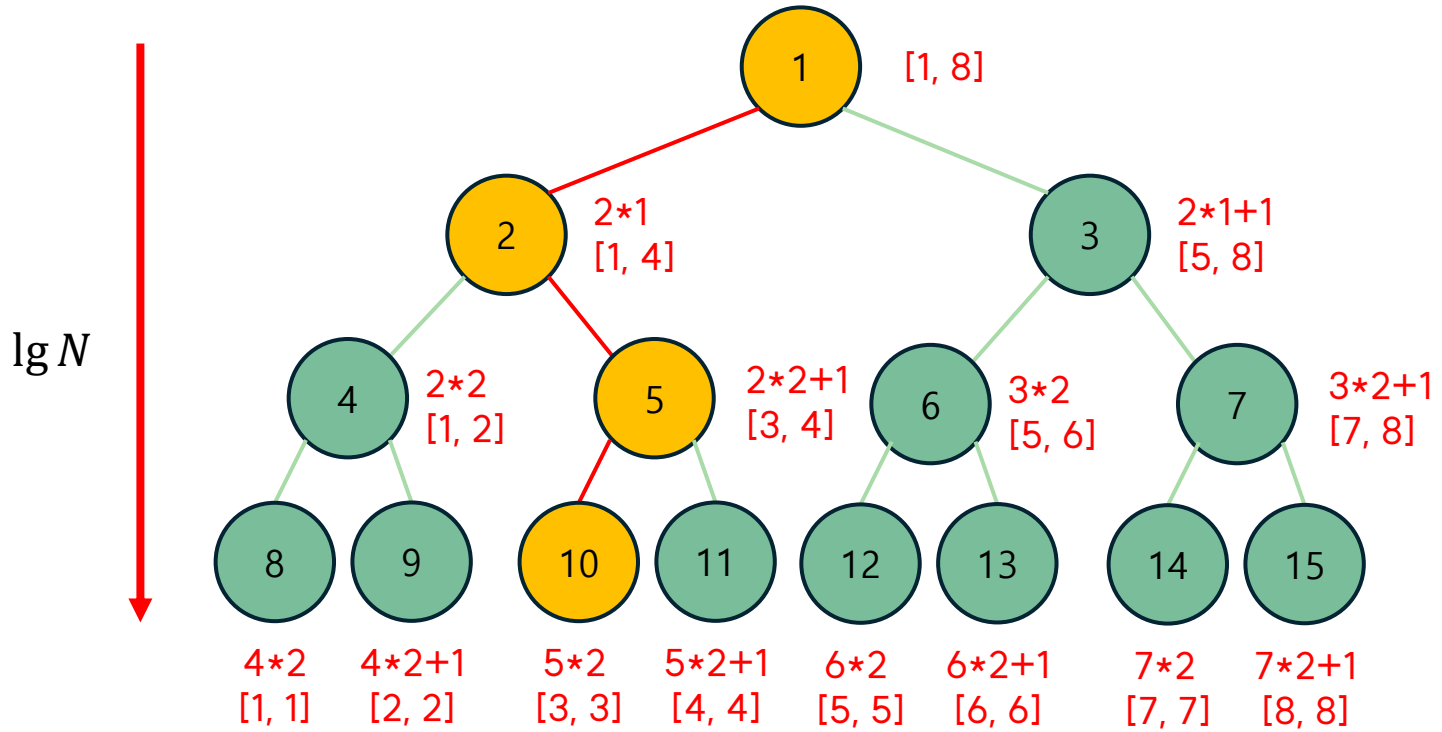
Introduction to Segment Tree



- $1 + 2 + \cdots + \frac{n'}{2}$의 값은 $n' - 1$이므로, 총 $2n' - 1$개의 노드가 필요합니다.
- 따라서 시간 복잡도는 $O(2n' - 1) = O(n')$입니다.

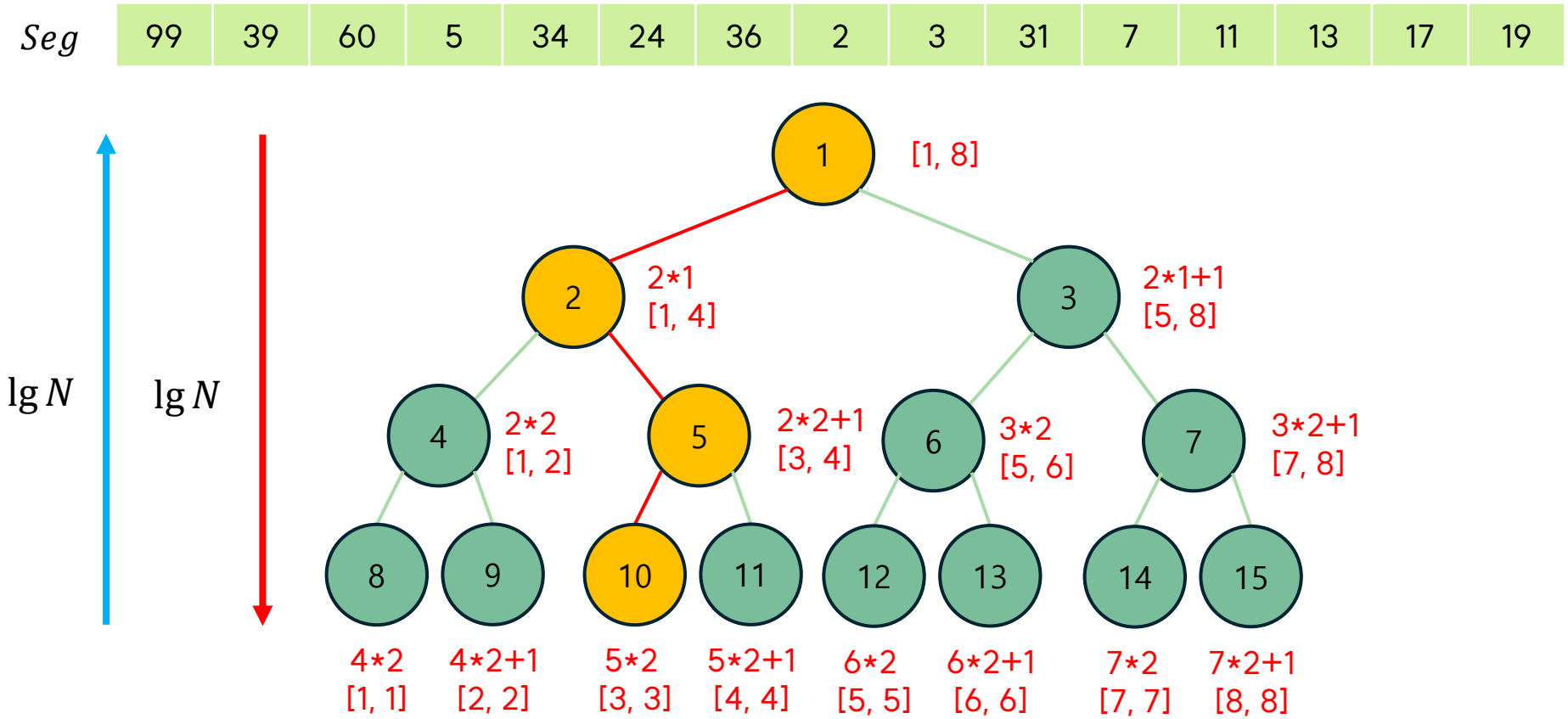
Introduction to Segment Tree

Seg	99	39	60	5	34	24	36	2	3	31	7	11	13	17	19



- 이제 업데이트시의 시간 복잡도를 알아봅시다.
- 먼저 root에서 시작해서 index i에 접근하는 데에 $\lg N$의 시간이 걸립니다. (트리의 높이)

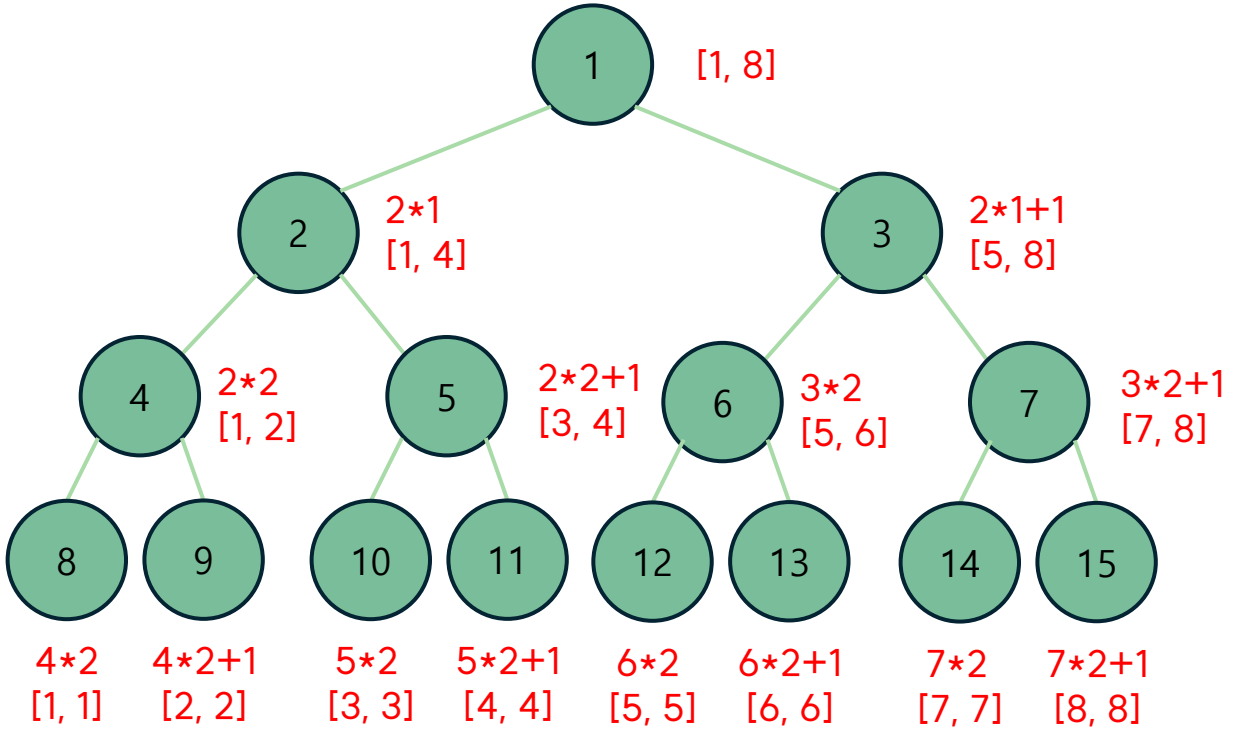
Introduction to Segment Tree



- 그 다음, 다시 올라가면서 부모 노드들의 값을 업데이트 하게 됩니다.
- 그때에도 트리의 높이만큼 연산이 필요하므로 $\lg N$의 시간이 걸립니다. 따라서 업데이트에는 $O(2 \lg N) = O(\lg N)$입니다.

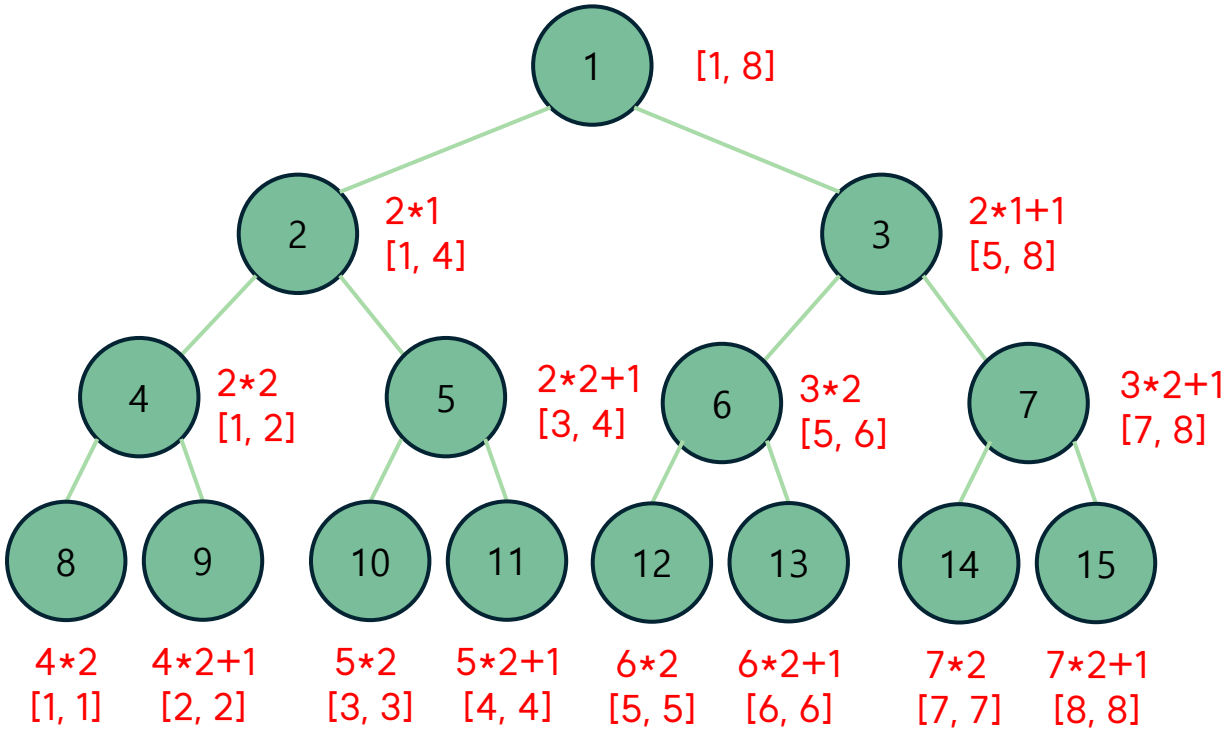
Introduction to Segment Tree

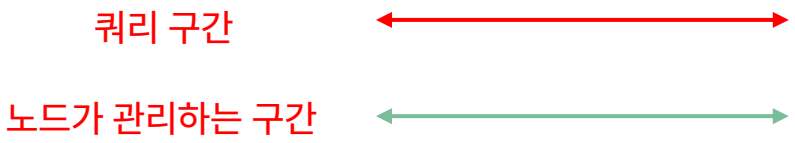
Seg	99	39	60	5	34	24	36	2	3	31	7	11	13	17	19



- 이제 쿼리가 주어질 때 시간 복잡도를 계산해봅시다.

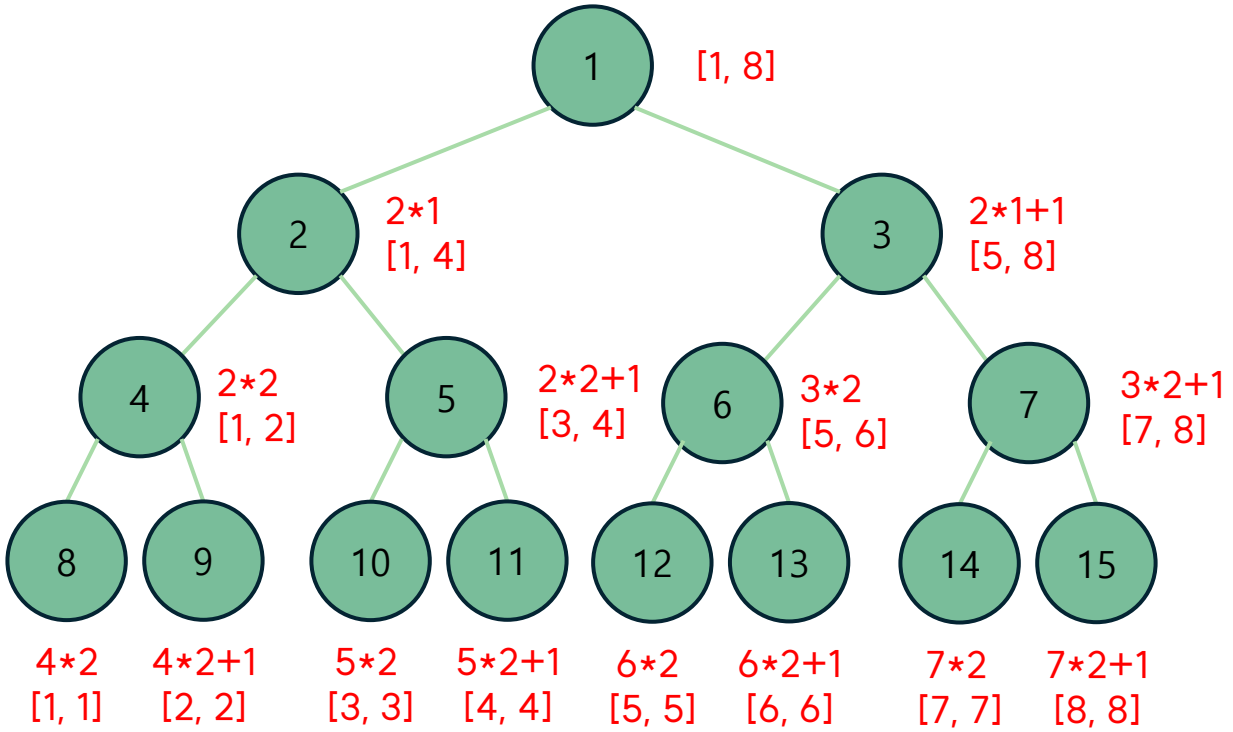
Introduction to Segment Tree





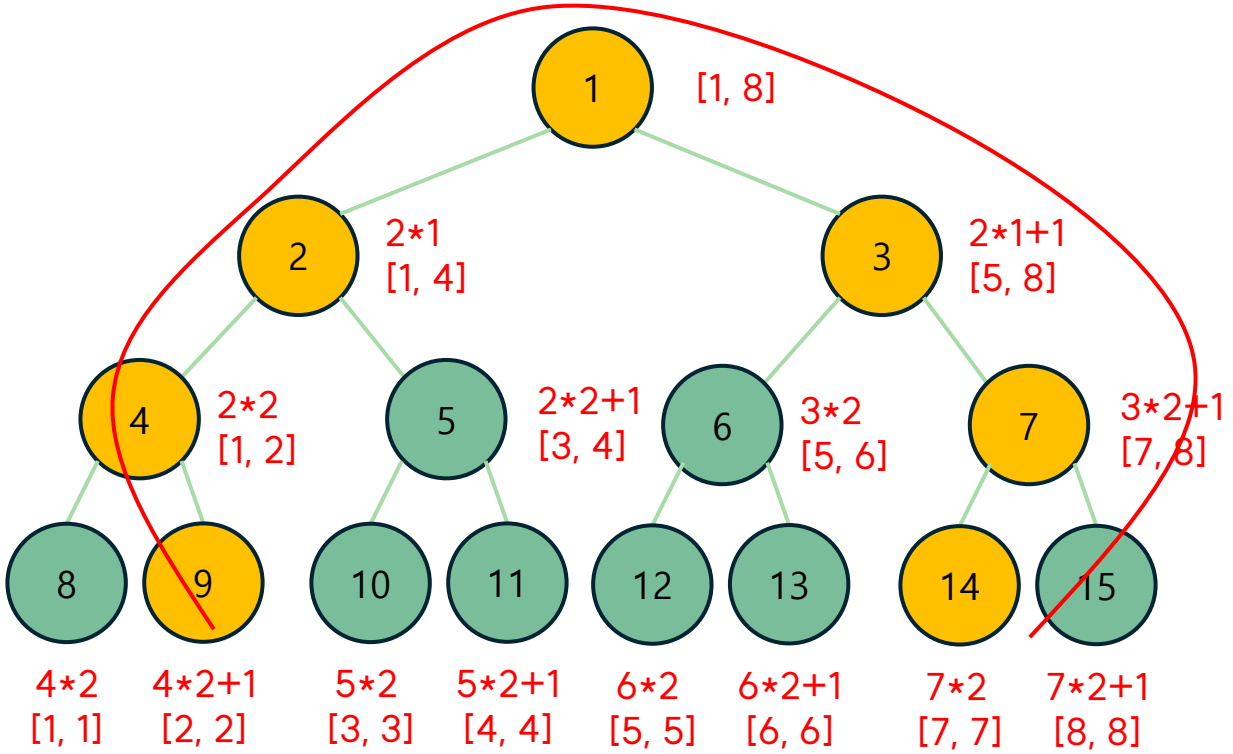
- 먼저 경우의 수를 나눠야 합니다. 먼저, 쿼리 구간과 노드가 관리하는 구간이 같은 경우를 봅시다.
- 이때는 단순히 해당 노드에 저장된 값을 return하면 되므로 $O(1)$에 풀 수 있습니다.

Introduction to Segment Tree



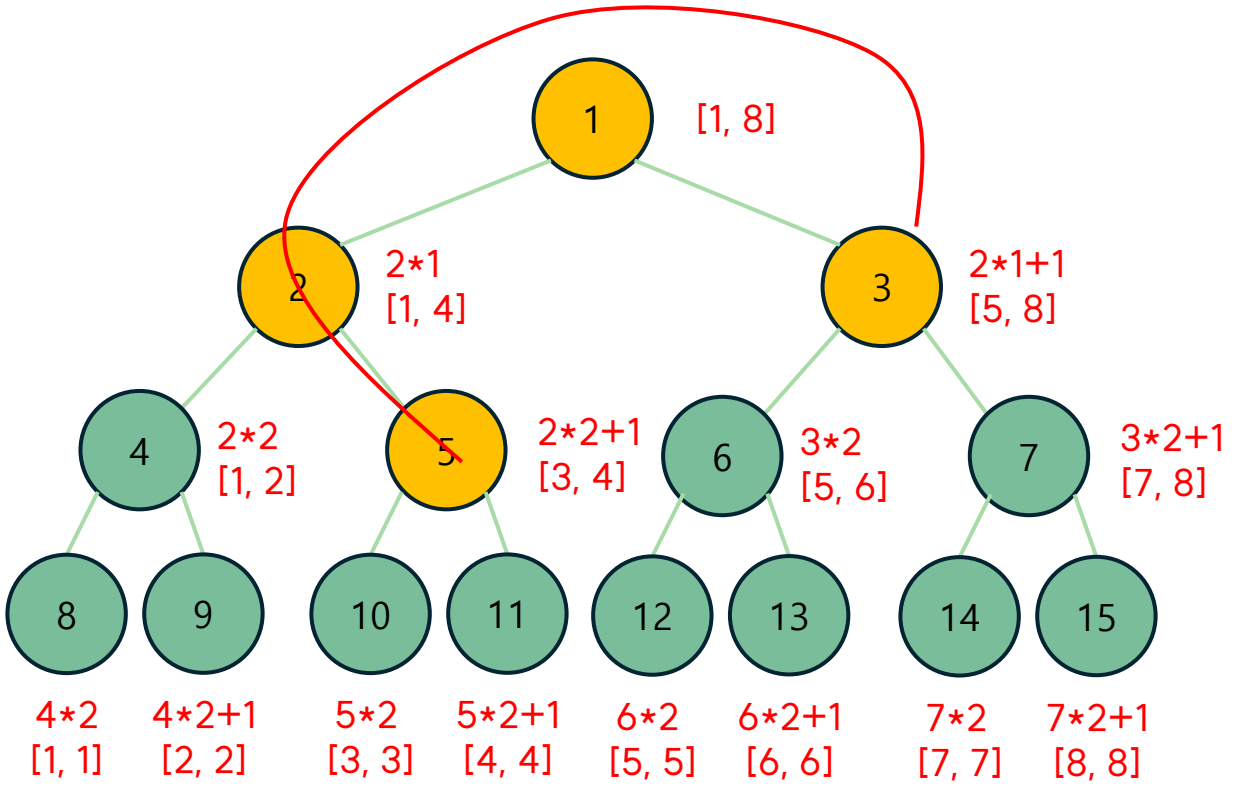
- 그 다음, 이외의 경우를 봅시다.
- 쿼리 구간이 어떻게 되든 거쳐야 하는 노드들의 형태는 ∩ 형태를 띄고 있습니다.

Introduction to Segment Tree



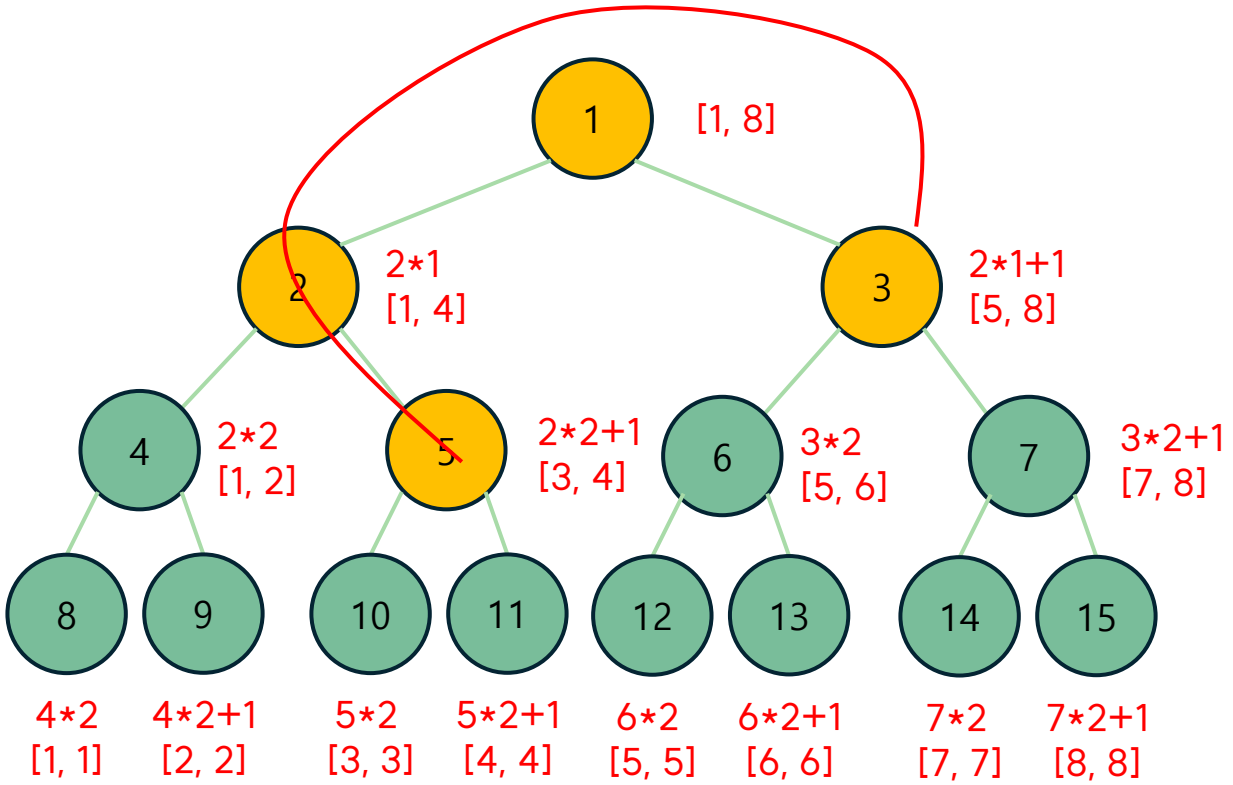
- 예를 들어, [2, 7]의 경우 다음과 같습니다.
- 중간의 노드 5, 6은 단순히 해당 값을 리턴하면 되므로 더 탐색할 필요가 없기 때문입니다.

Introduction to Segment Tree



- 예를 들어, [3, 8]의 경우 다음과 같습니다.

Introduction to Segment Tree



- 따라서 왼쪽에서 긴 부분이 최대 $O(\lg N)$ 이고, 오른쪽도 비슷하게 $O(\lg N)$ 이므로 $O(2 \lg N) = O(\lg N)$ 입니다.

BOJ 2042

BOJ 2042

- 배운 내용을 그대로 적용하면 됩니다.

BOJ 2042

```cpp
ll tree[4000005];
class SegmentTree {
public:
    void init(int n, int s, int e, const vector<ll> &arr) {
        if(s == e) {
            tree[n] = arr[s];
            return;
        }
        int m = (s+e)/2;
        init(2*n, s, m, arr);
        init(2*n+1, m+1, e, arr);
        tree[n] = tree[2*n] + tree[2*n+1];
    }

    ll query(int n, int s, int e, int l, int r) {
        if(l <= s && e <= r) {
            return tree[n];
        }
        if(r < s || e < l) {
            return 0;
        }
        int m = (s+e)/2;
        ll left = query(2*n, s, m, l, r);
        ll right = query(2*n+1, m+1, e, l, r);
        return left + right;
    }

    void change(int n, int s, int e, int idx, ll val) {
        if(idx < s || e < idx) {
            return;
        }
        if(s==e) {
            tree[n] = val;
            return;
        }
        int m = (s+e)/2;
        change(2*n, s, m, idx, val);
        change(2*n+1, m+1, e, idx, val);
        tree[n] = tree[2*n] + tree[2*n+1];
    }
};
```

BOJ 2357

BOJ 2357

- Segment Tree를 2개 만들면서 min을 관리하는 Segment Tree, max를 관리하는 Segment Tree를 만들면 됩니다.

BOJ 12837

BOJ 12837

- Segment Tree에 delta값을 저장해서 구간 합 쿼리를 수행하면 됩니다.

BOJ 6549

BOJ 6549

- Segment Tree가 구간의 min값을 관리하게 하고, 그때의 각 노드마다의 넓이는 $(R - L + 1) \times Seg[node]$임을 이용하면 됩니다.

수고하셨습니다!