RSQ

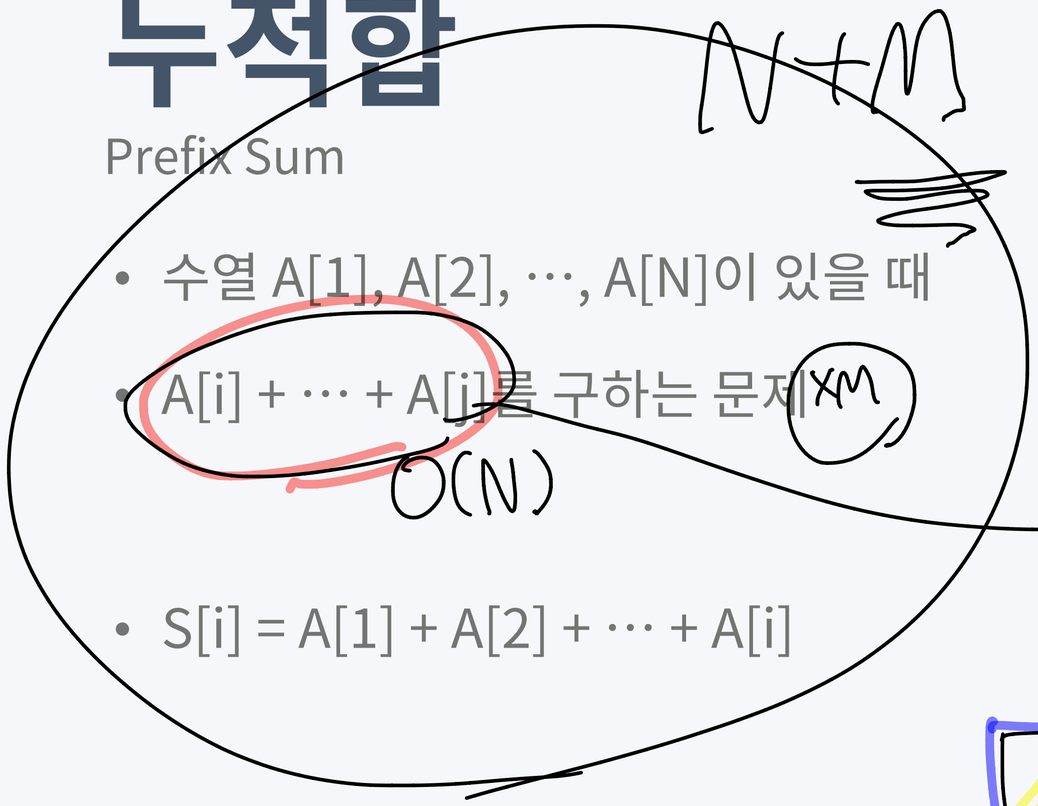
최백준 choi@startlink.io

누적합

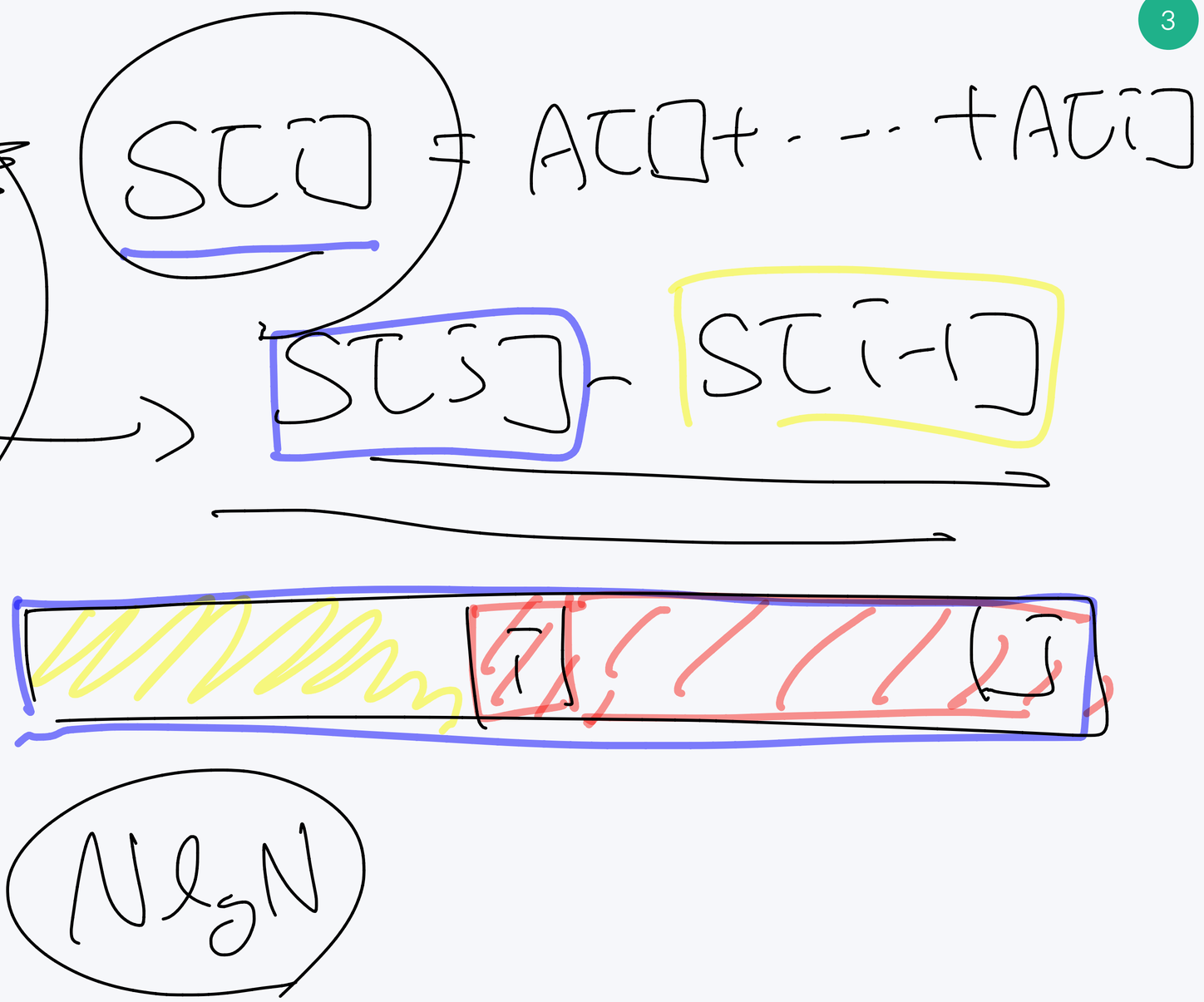
누적합

Prefix Sum

- 수열 A[1], A[2], …, A[N]이 있을 때
- A[i] + … + A[j]를 구하는 문제



- S[i] = A[1] + A[2] + … + A[i]



누적합

Prefix Sum

- A[i] + ⋯ + A[j]를 구하는 문제

- S[j] = A[1] + A[2] + ⋯ + A[i-1] + A[i] + ⋯ + A[j]
- S[i-1] = A[1] + A[2] + ⋯ + A[i-1]
- S[j] – S[i-1] = A[i] + ⋯ + A[j]

구간 합 구하기 4

https://www.acmicpc.net/problem/11659

- 수 N개가 주어졌을 때, i번째 수부터 j번째 수까지 합을 구하는 문제

구간 합 구하기 4

https://www.acmicpc.net/problem/11659

- 소스: http://boj.kr/213b7873c0744899bf0b135fb7b342d6

나머지 합

https://www.acmicpc.net/problem/10986

- 수 N개 A[1], A[2], ..., A[N]이 주어진다.
- 연속된 부분 구간의 합이 M으로 나누어 떨어지는 구간의 개수를 구하는 문제
- 즉, A[i] + ... + A[j] (i ≤ j) 의 합이 M으로 나누어 떨어지는 (i, j) 쌍의 개수를 구해야 한다.

나머지 합

https://www.acmicpc.net/problem/10986

- S[i] = A[1] + … + A[i] 라고 하자
- A[i] + … + A[j] = S[j] – S[i-1]
- (A[i] + … + A[j]) % M = (S[j] – S[i-1]) % M

- (A[i] + … + A[j]) % M == 0 인 것의 개수를 구해야 한다
- (S[j] – S[i-1]) % M == 0 와 같다
- 나눈 나머지가 0이 되려면
- S[j] % M == S[i-1] % M 이 되어야 한다

나머지 합

https://www.acmicpc.net/problem/10986

- 이 문제는
- S[j] % M == S[i-1] % M 이 되어야 한다
- 를 만족하는 (i, j) 쌍의 개수를 구하는 문제가 된다.
- cnt[k]를 S[i] % M == k 인 i의 개수라고 하면
- $0 \leq k < M$인 k에 대해서
- cnt[k] * (cnt[k] - 1) / 2 의 합을 구하면 된다.

나머지 합

https://www.acmicpc.net/problem/10986

- 소스: http://boj.kr/bc4c380173d94b858411591acbb25edb

$$S[i] = \frac{S[i-1]}{1 \sim i-1} + A[i]$$

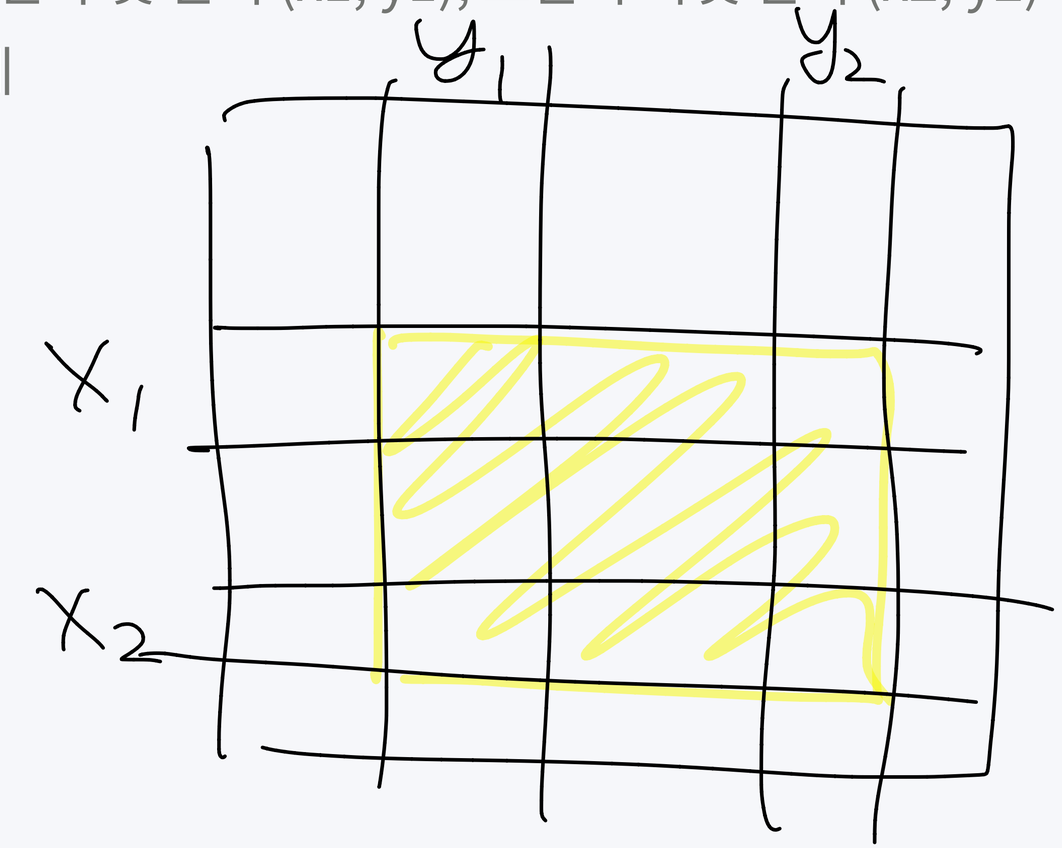
2차원 누적합

구간 합 구하기 5

https://www.acmicpc.net/problem/11660

O(1)

- 2차원 배열에서 왼쪽 윗 칸이 (x1, y1), 오른쪽 아랫 칸이 (x2, y2) 인 직사각형에 들어있는 수의 합을 구하는 문제



구간 합 구하기 5

https://www.acmicpc.net/problem/11660

- 합을 효율적으로 구하는 방법
- S[i][j] = (1, 1) ~ (i, j)까지 합
- S[i][j] = S[i-1][j] + S[i][j-1] – S[i-1][j-1] + A[i][j]

구간 합 구하기 5

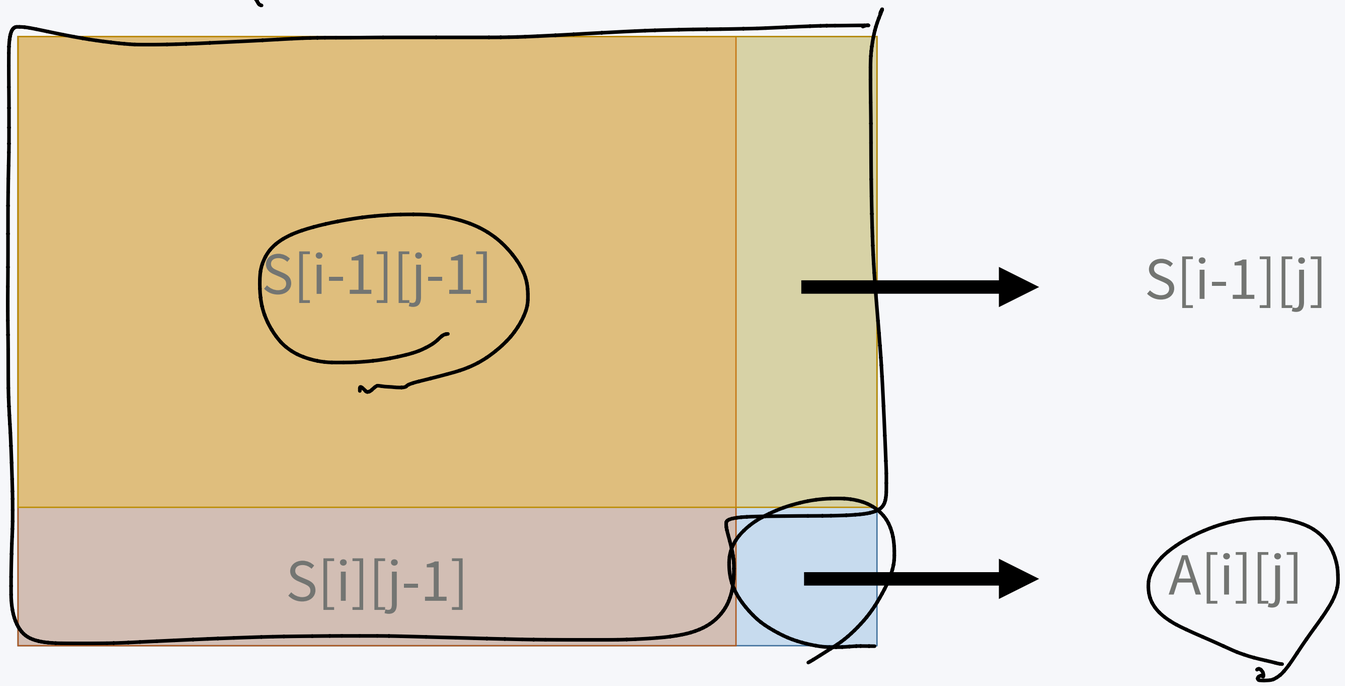
https://www.acmicpc.net/problem/11660

- 합을 효율적으로 구하는 방법
- S[i][j] = (1, 1) ~ (i, j)까지 합
- S[i][j] = S[i-1][j] + S[i][j-1] – S[i-1][j-1] + A[i][j]

구간 합 구하기 5

https://www.acmicpc.net/problem/11660

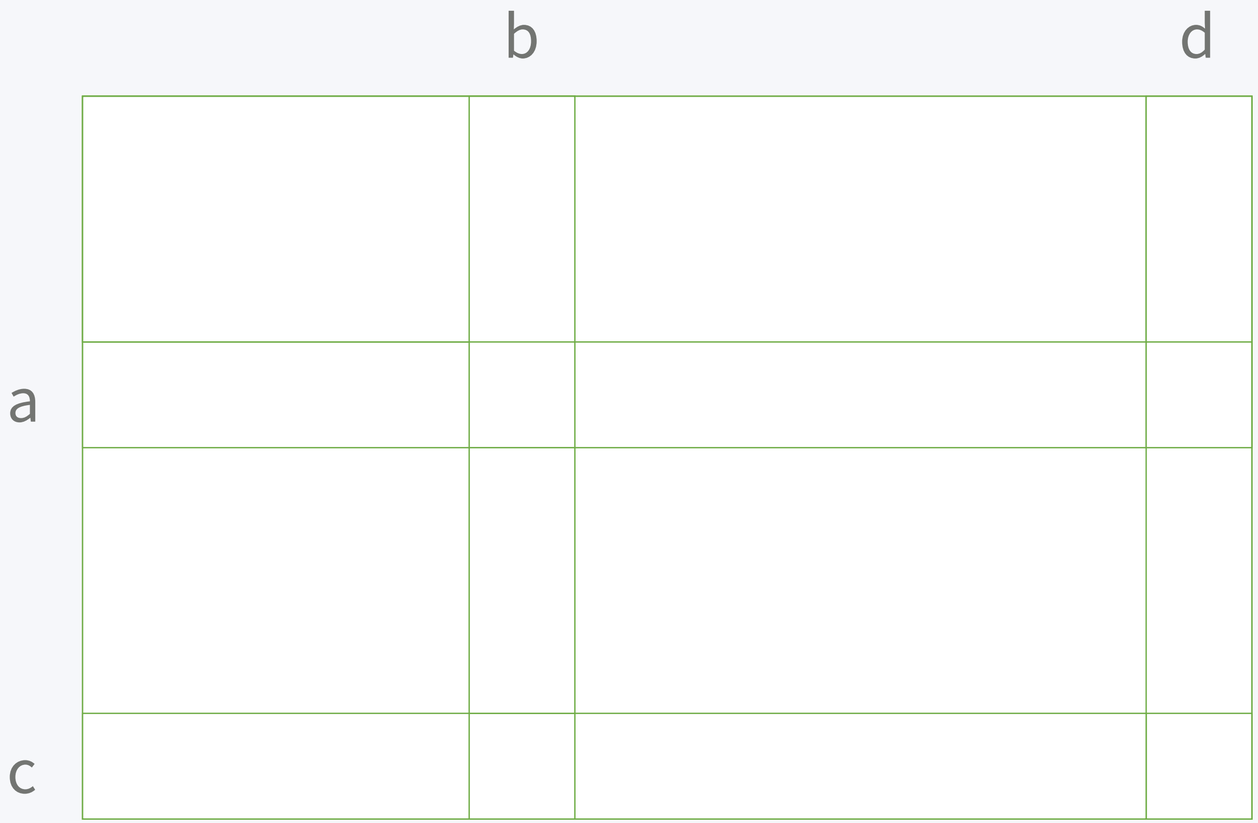
- 합을 효율적으로 구하는 방법
- S[i][j] = (1, 1) ~ (i, j)까지 합
- S[i][j] = S[i-1][j] + S[i][j-1] – S[i-1][j-1] + A[i][j]



구간 합 구하기 5

https://www.acmicpc.net/problem/11660

- (a,b) ~ (c,d) 합 구하기



구간 합 구하기 5

https://www.acmicpc.net/problem/11660

- (a,b) ~ (c,d) 합 구하기

구간 합 구하기 5

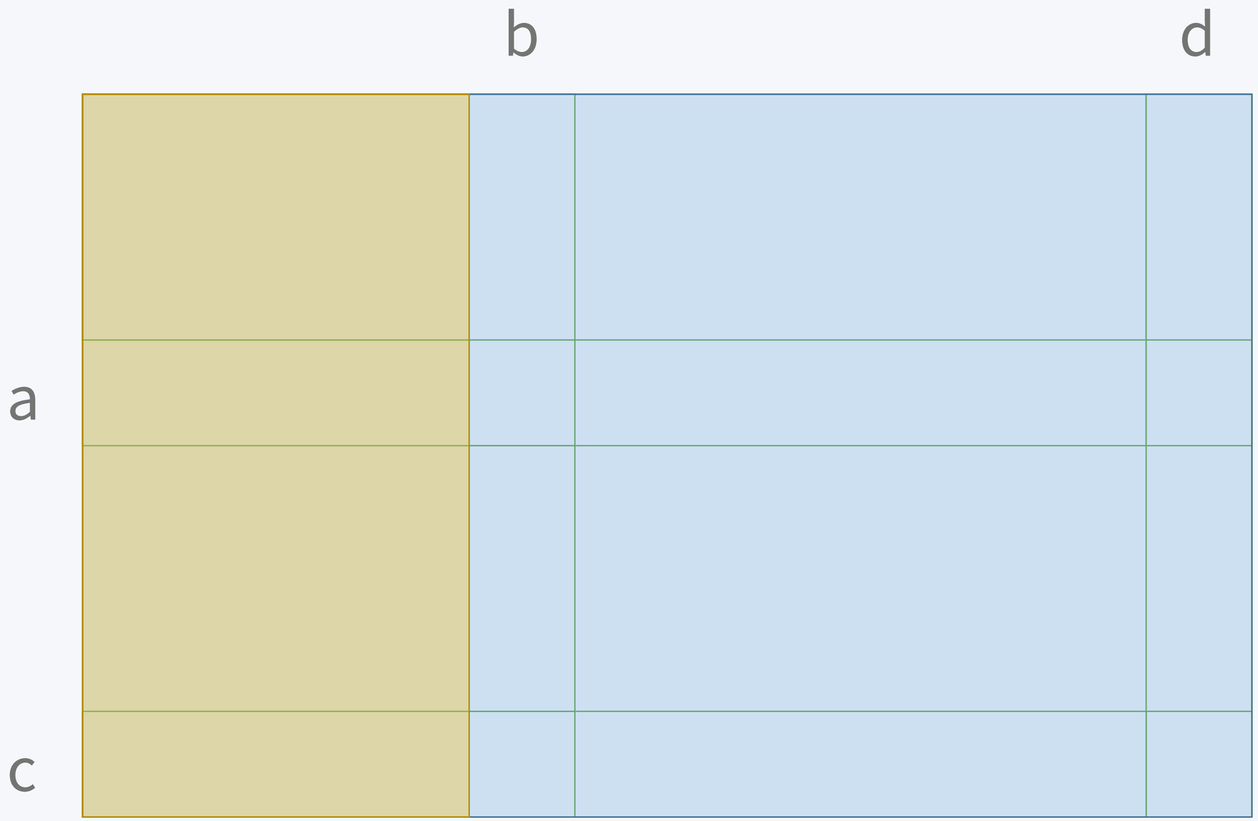
https://www.acmicpc.net/problem/11660

- S[c][d]

구간 합 구하기 5

https://www.acmicpc.net/problem/11660

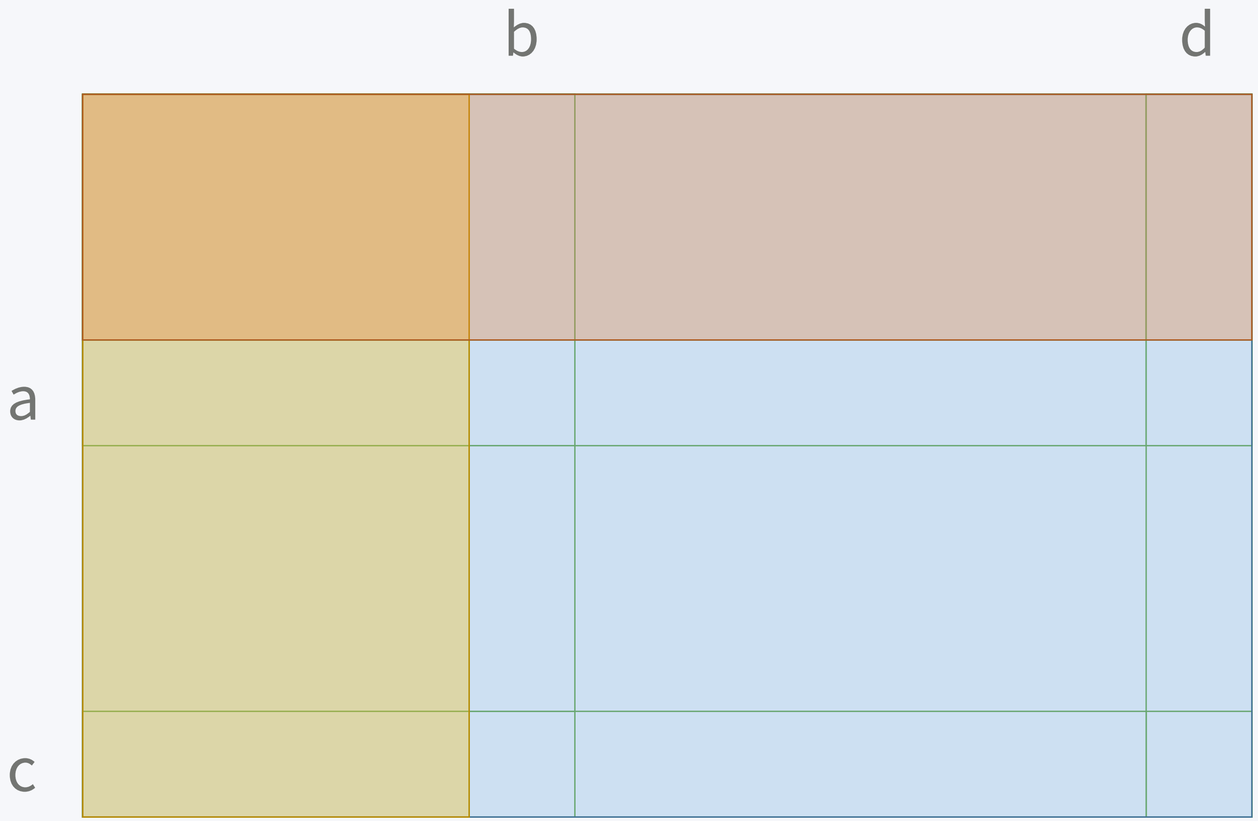
- S[c][d] – S[c][b-1]



구간 합 구하기 5

https://www.acmicpc.net/problem/11660

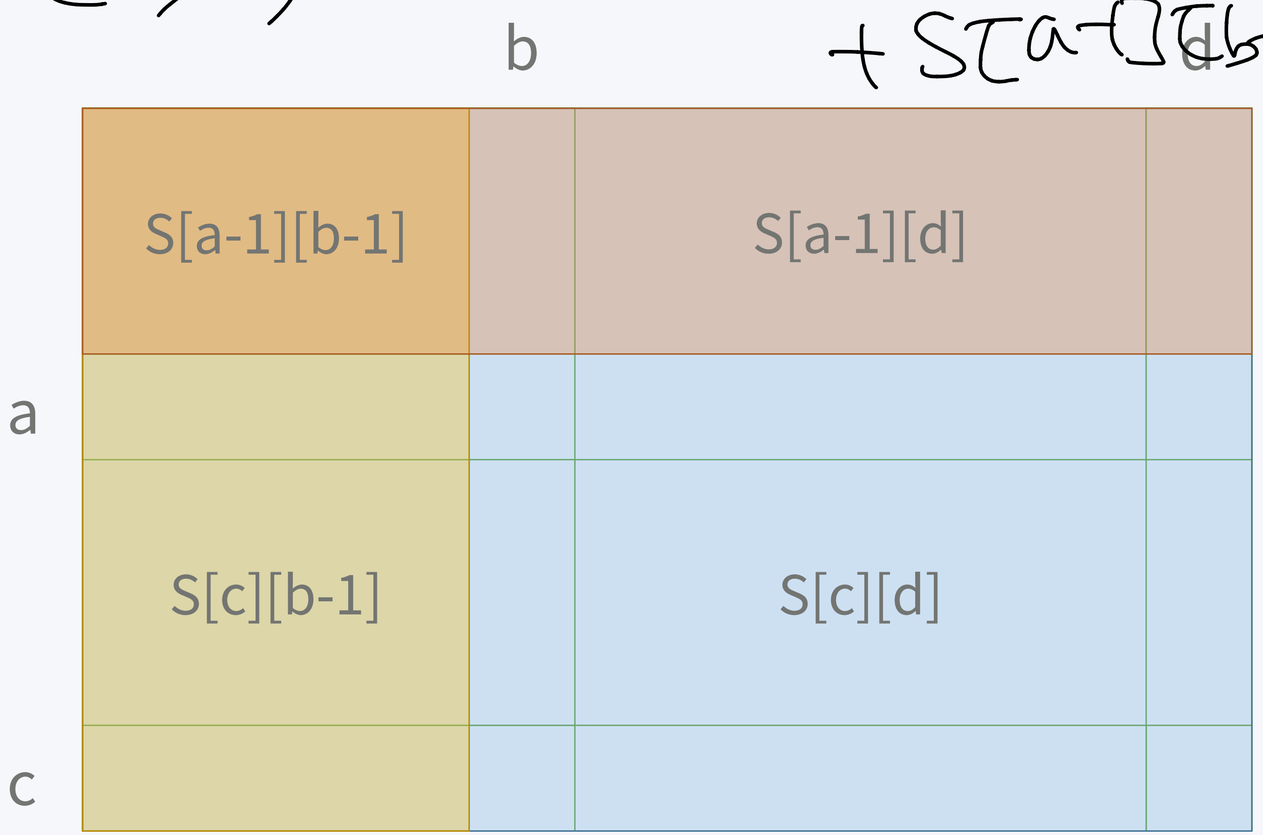
- S[c][d] – S[c][b-1] – S[a-1][d]



구간 합 구하기 5

https://www.acmicpc.net/problem/11660

- S[c][d] – S[c][b-1] – S[a-1][d] + S[a-1][b-1]



구간 합 구하기 5

3차원 누적합

https://www.acmicpc.net/problem/11660

- 소스: http://boj.kr/e44f76af1d2a4184afc762c9f596906d

$$S[i] = S[i-1] + A[i]$$

$$S[i][j] = S[i-1][j] + S[i][j-1] - S[i-1][j-1] + A[i][j]$$

$$S[i][j][k] = S[i-1][j][k] + S[i][j-1][k] + S[i][j][k-1] - S[i-1][j-1][k] - S[i-1][j][k-1] - S[i][j-1][k-1] + S[i-1][j-1][k-1] + A[i][j][k]$$

S[5] - S[i-1]

(a,b) - (c,d)

세그먼트 트리

구간 합 구하기

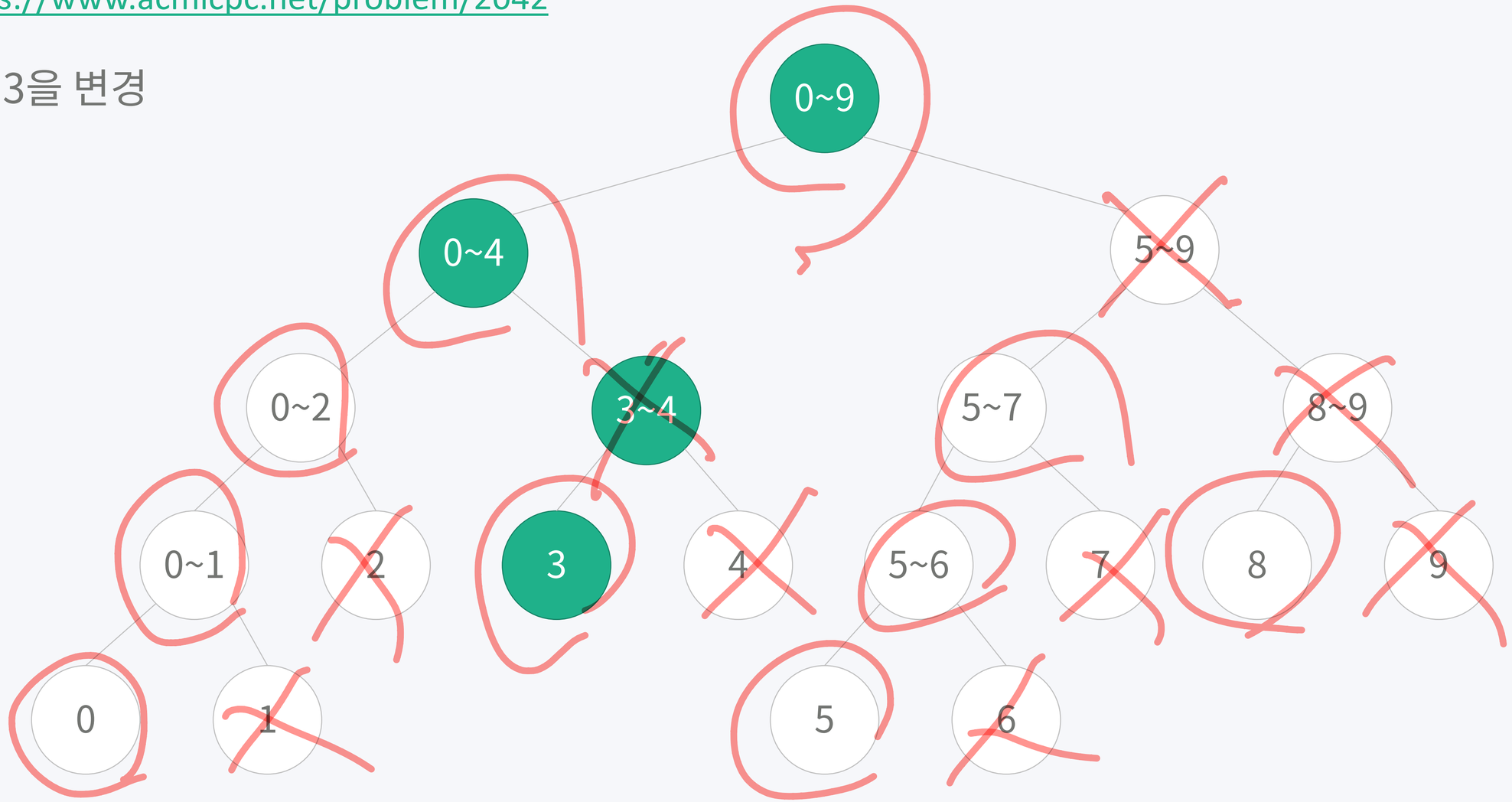
https://www.acmicpc.net/problem/2042

- 구간의 최소값이 아니고 합을 구하는 경우에는
- min 대신 +를 하면 된다

구간 합 구하기

https://www.acmicpc.net/problem/2042

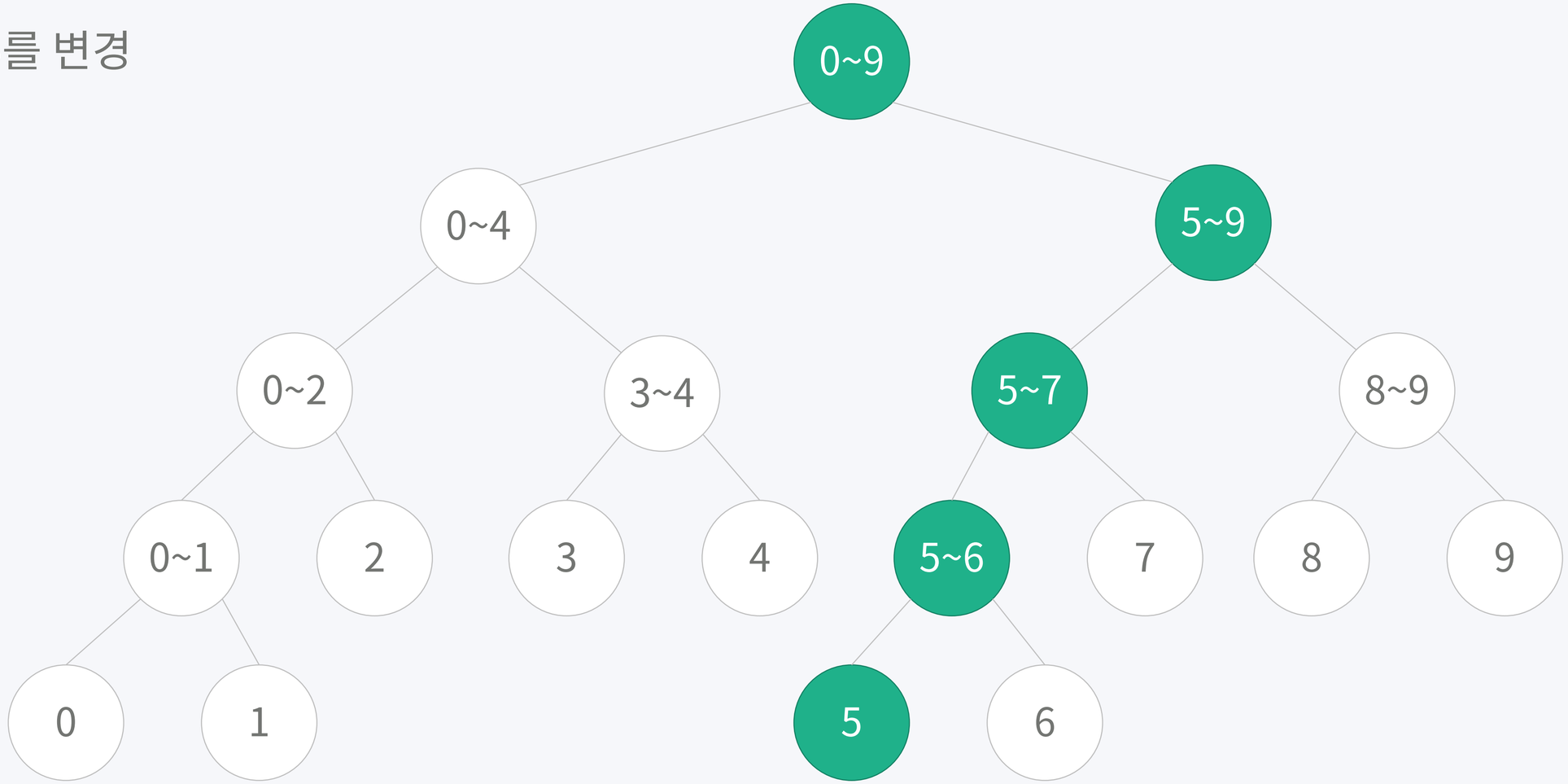
- 3을 변경



구간 합 구하기

https://www.acmicpc.net/problem/2042

- 5를 변경



구간 합 구하기

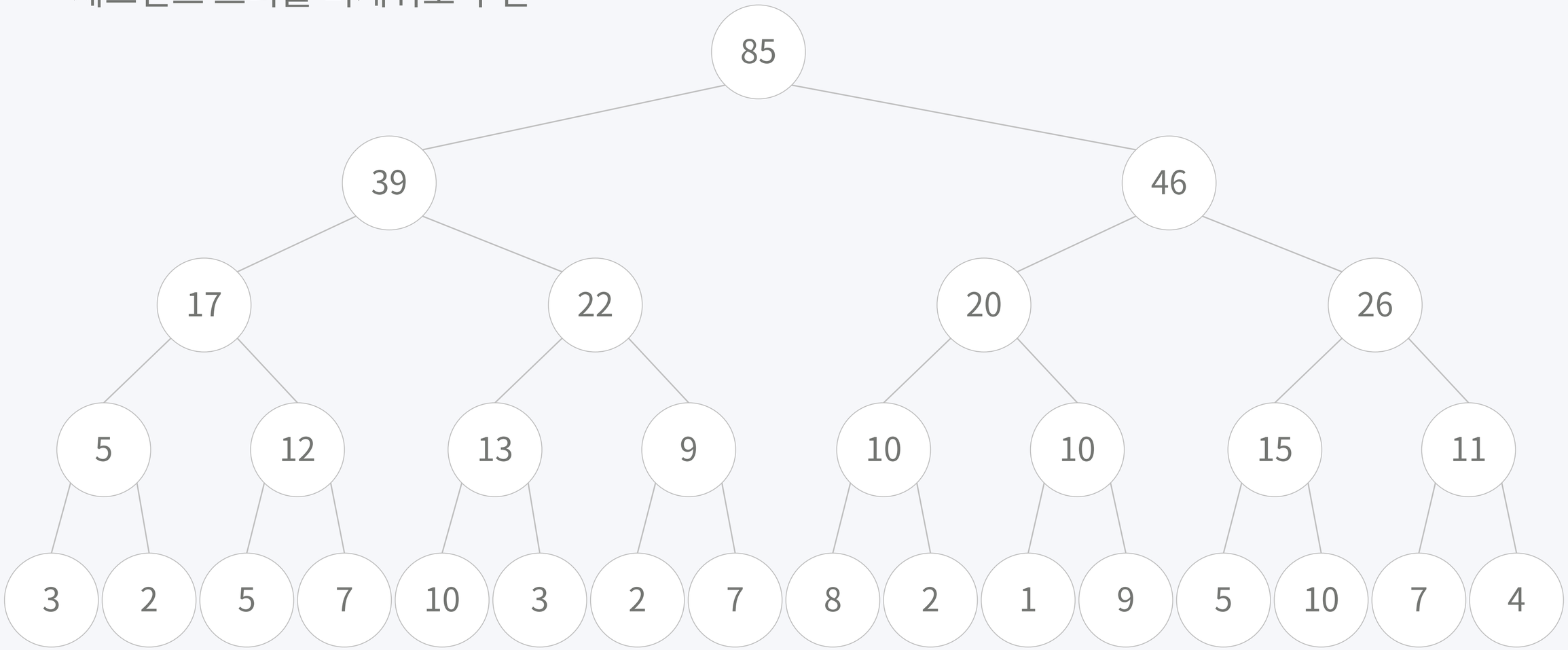
https://www.acmicpc.net/problem/2042

- 소스: http://boj.kr/be44f5abe25a4e3b81d28670da52b3e5

세그먼트 트리 비재귀 구현

구간의 합 구하기

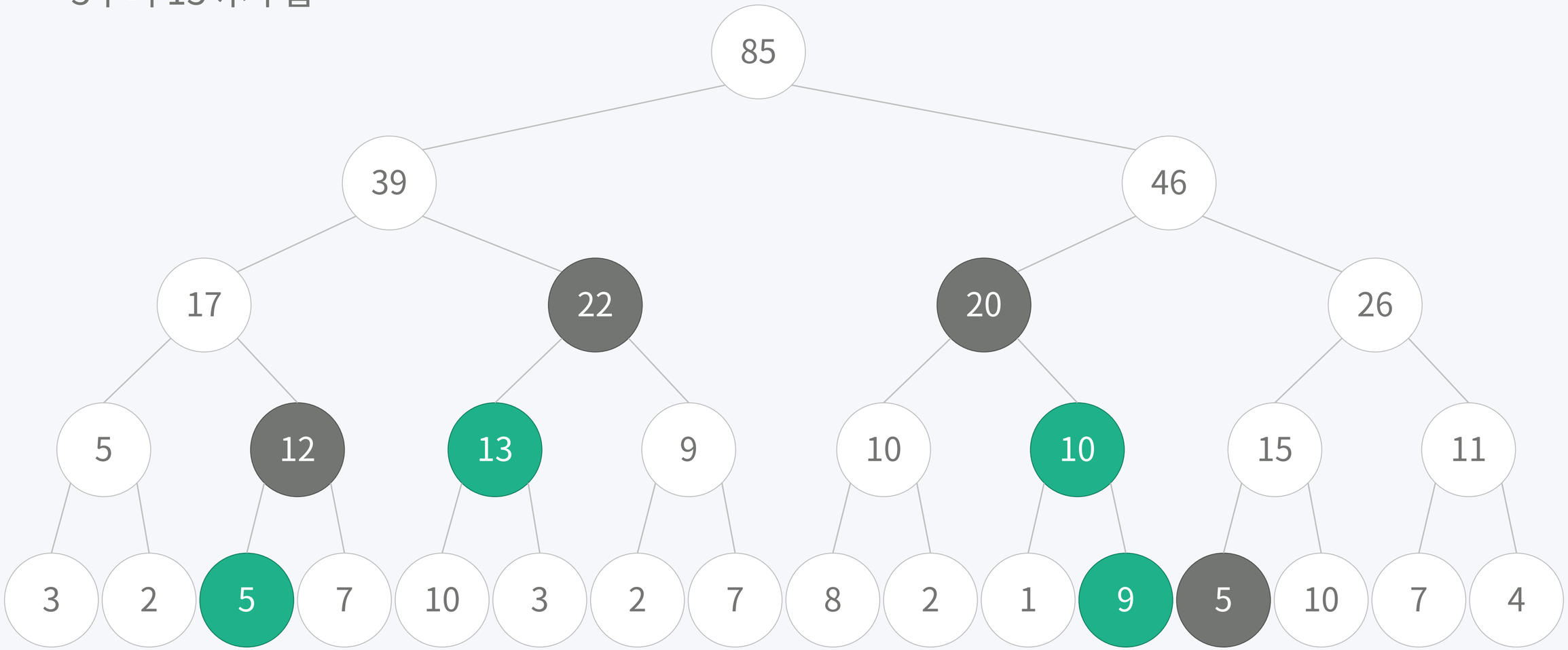
- 세그먼트 트리를 비재귀로 구현



세그먼트 트리 비재귀 구현

구간의 합 구하기

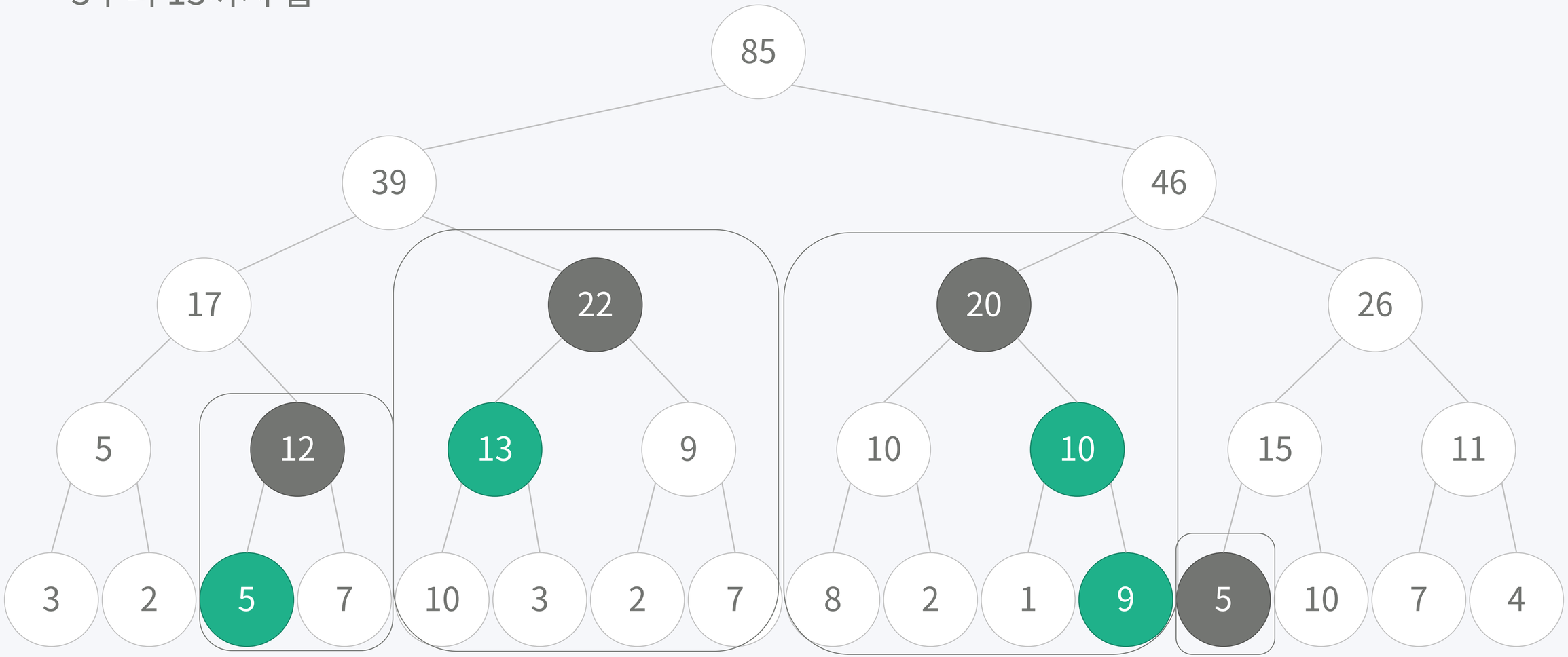
- 3부터 13까지 합



세그먼트 트리 비재귀 구현

구간의 합 구하기

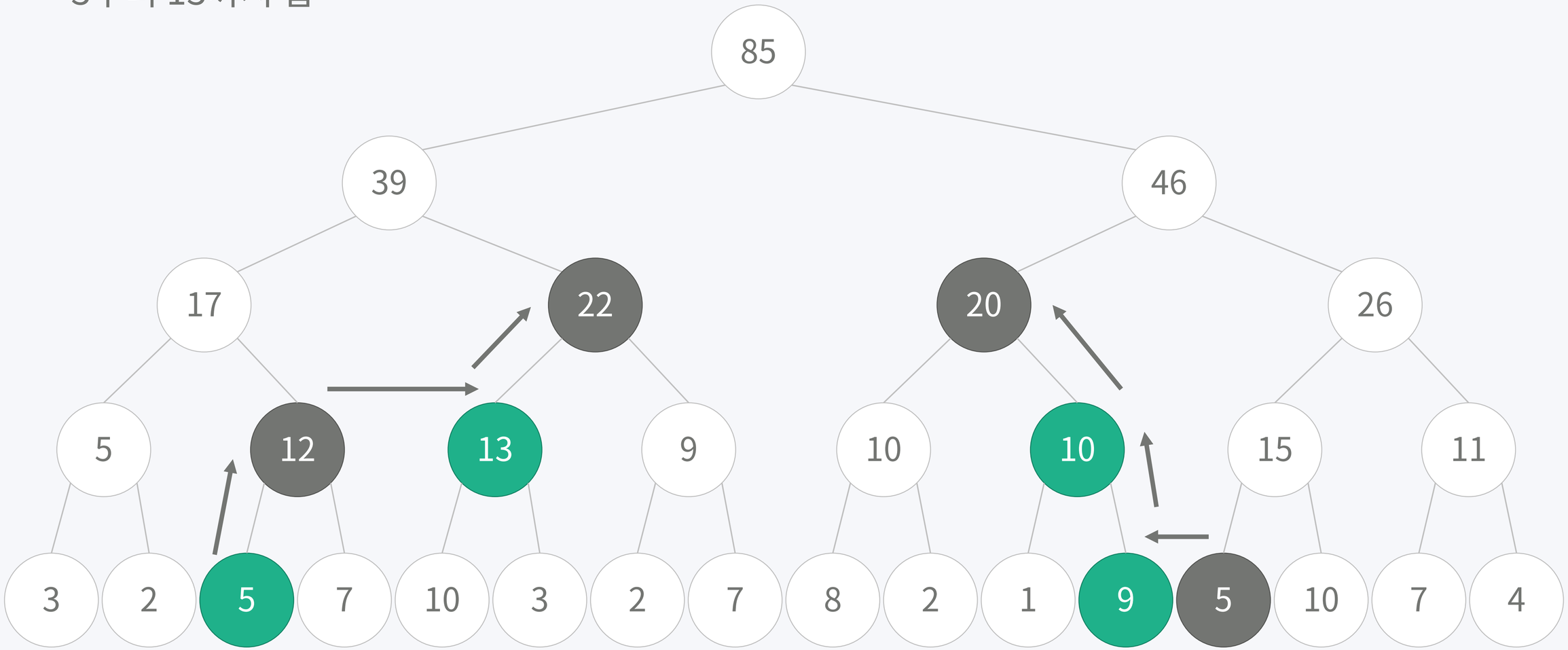
- 3부터 13까지 합



세그먼트 트리 비재귀 구현

구간의 합 구하기

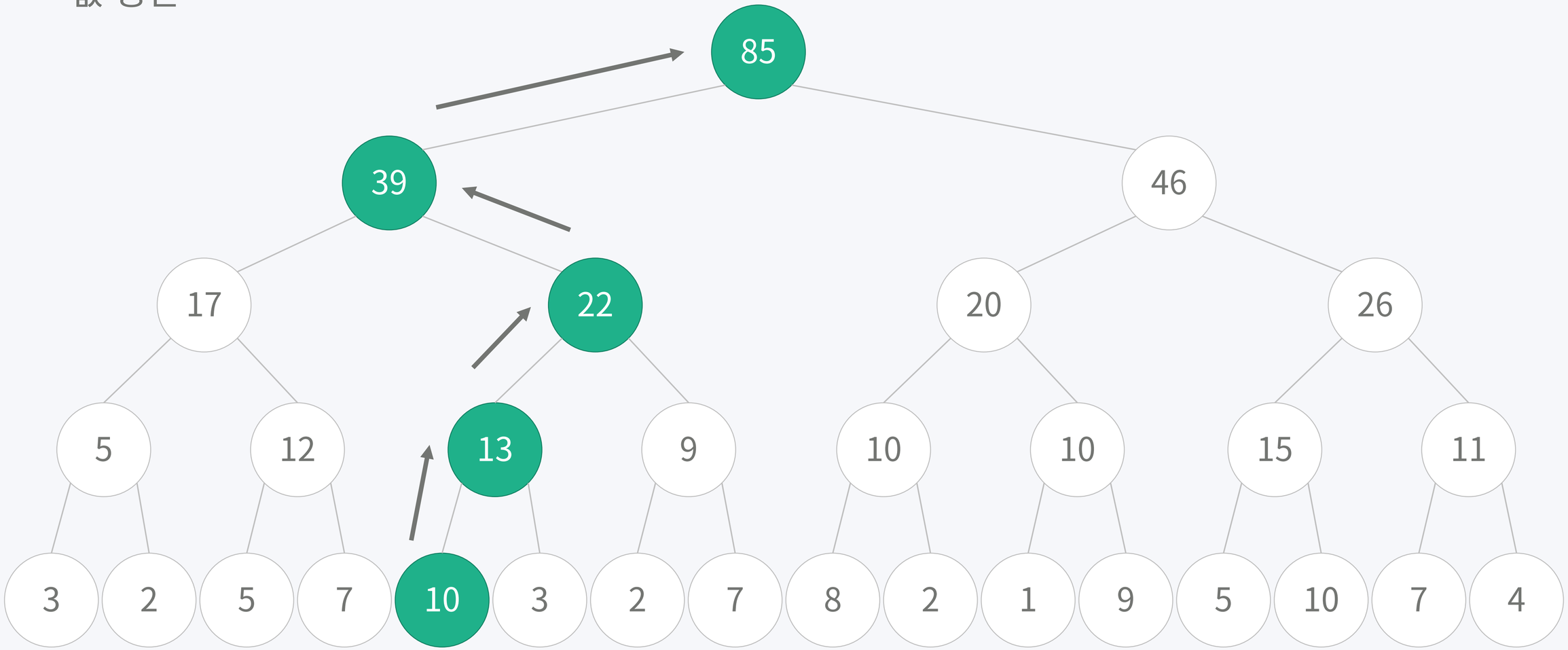
- 3부터 13까지 합



세그먼트 트리 비재귀 구현

구간의 합 구하기

- 값 갱신



세그먼트 트리 비재귀 구현

구간의 합 구하기

- 합 구하는 방법
- 왼쪽
  - 왼쪽 자식이면 올라간다
  - 오른쪽 자식이면 답을 더하고, 오른쪽 칸으로 이동
- 오른쪽
  - 오른쪽 자식이면 올라간다
  - 왼쪽 자식이면 답을 더하고, 왼쪽 칸으로 이동

세그먼트 트리 비재귀 구현

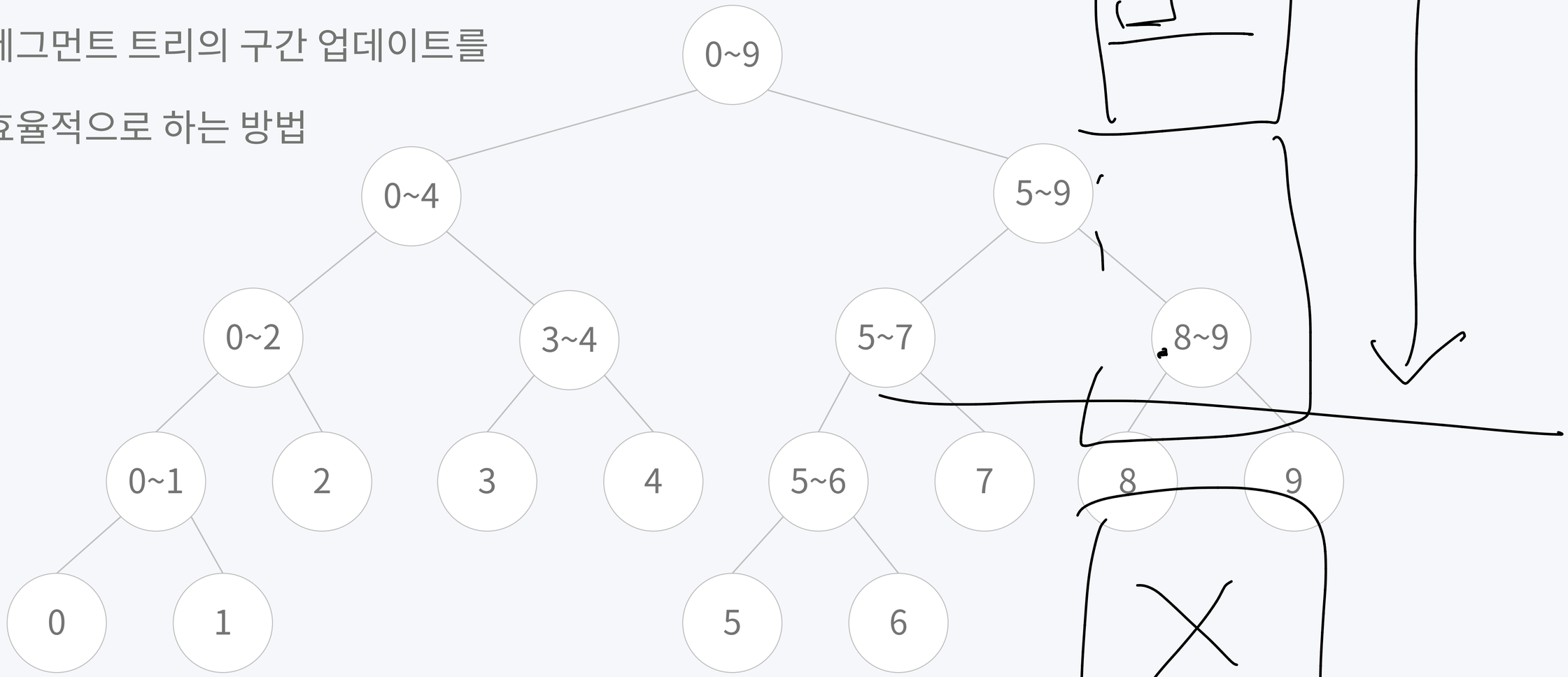
구간의 합 구하기

- 소스: http://boj.kr/1fdc0adda6e345b6af1b777af7f93fc9

Lazy Propagation

구간의 합구하기

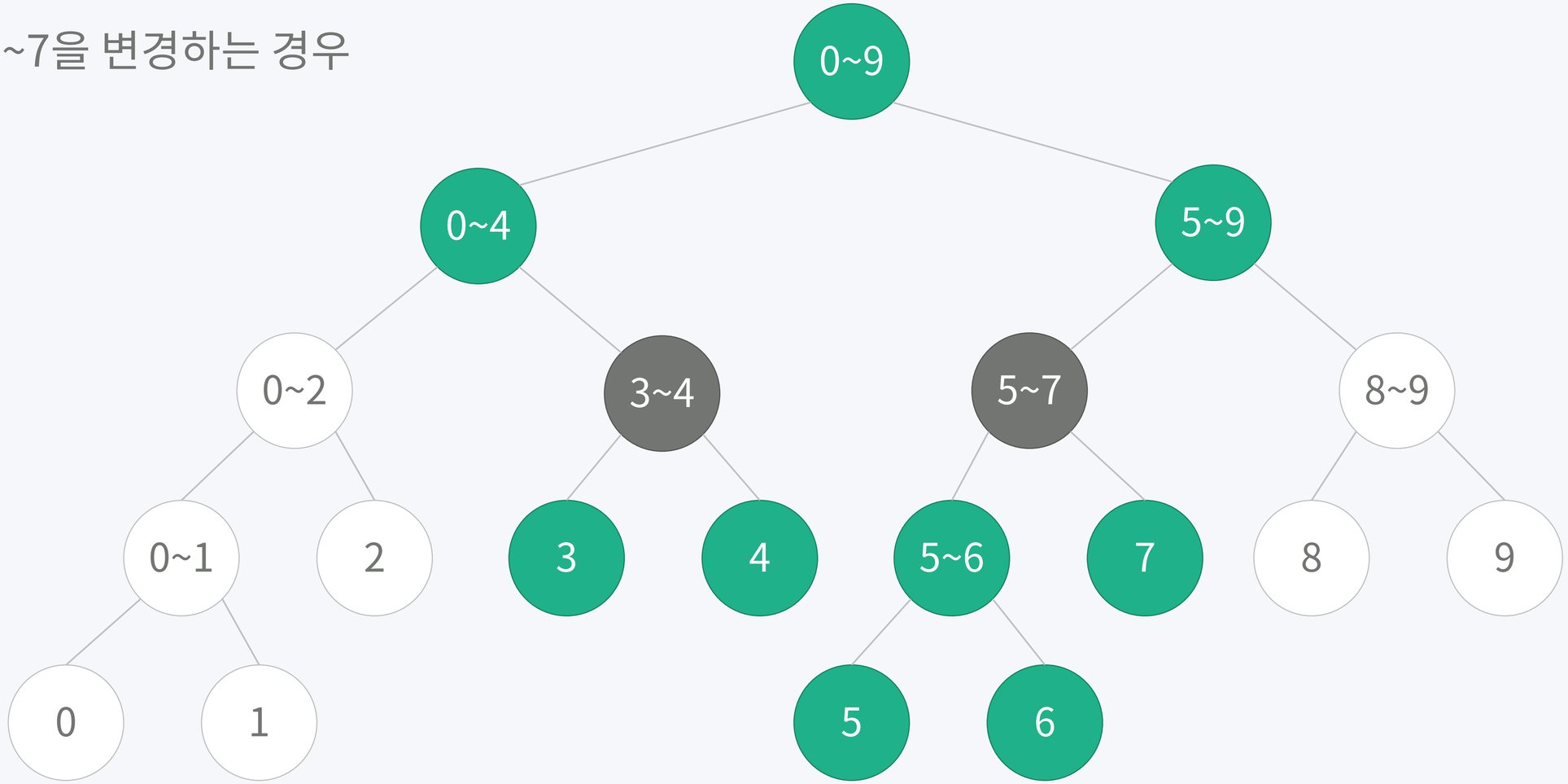
- 세그먼트 트리의 구간 업데이트를
- 효율적으로 하는 방법



Lazy Propagation

구간의 합구하기

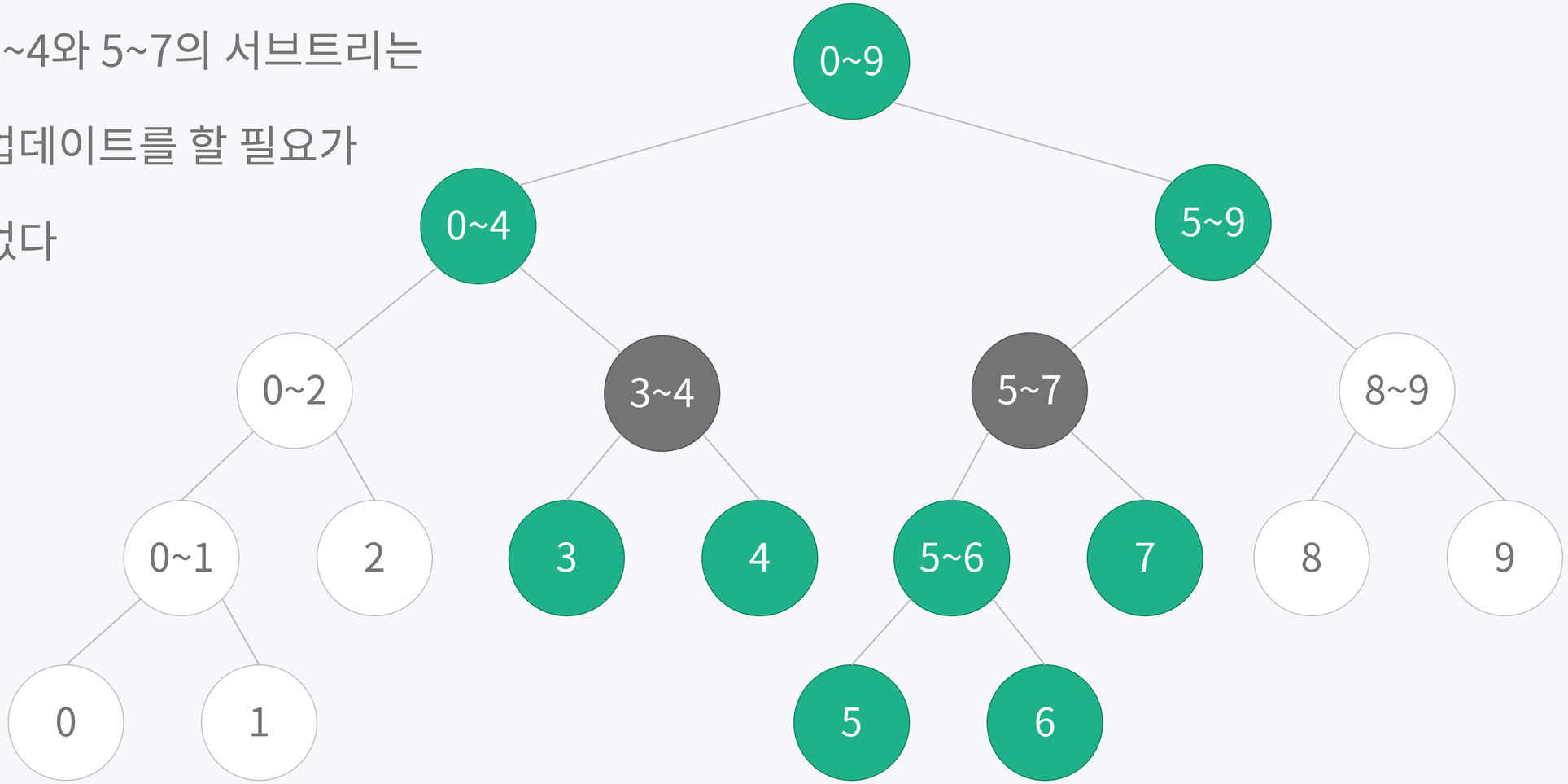
- 3~7을 변경하는 경우



Lazy Propagation

구간의 합 구하기

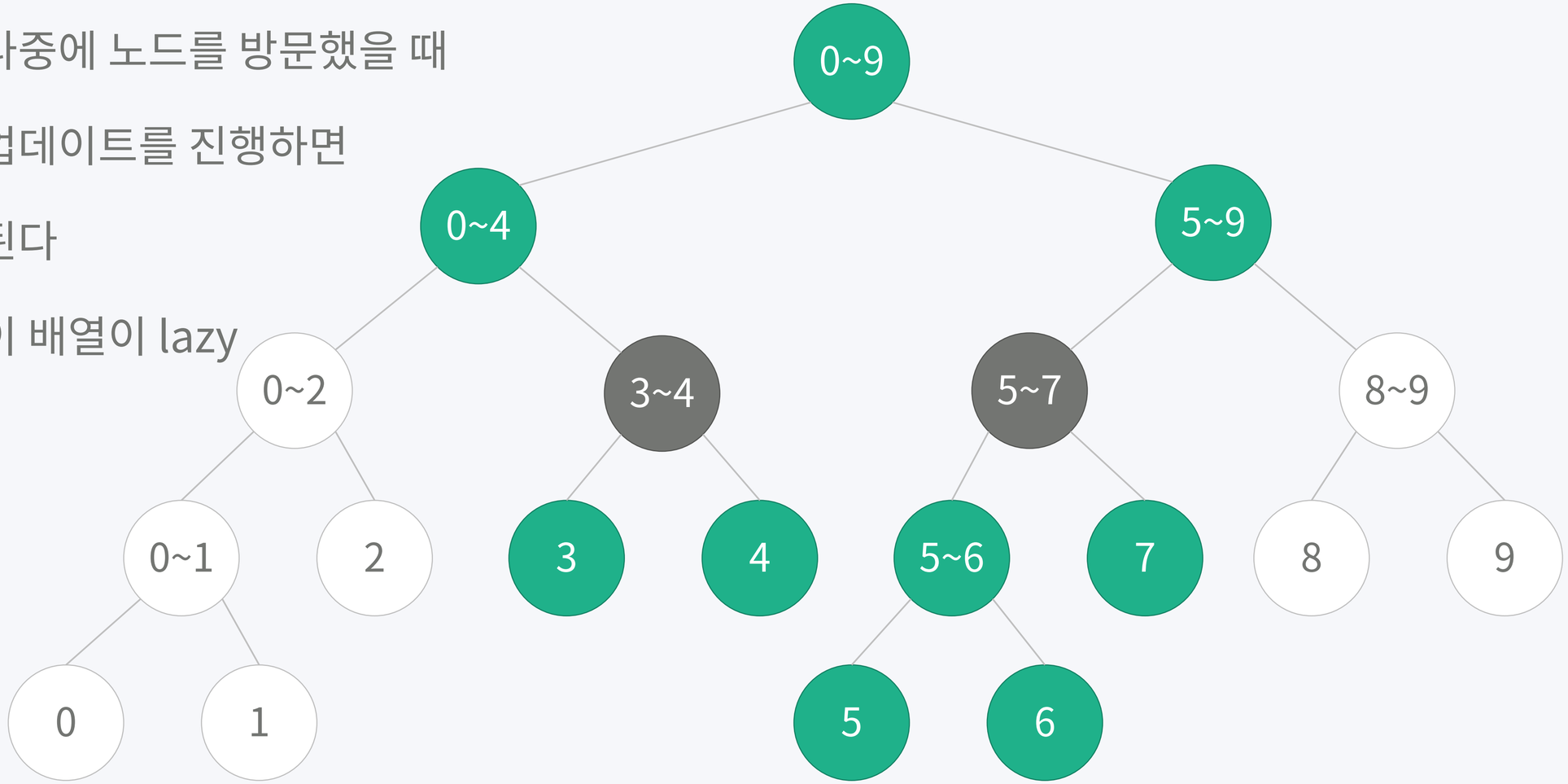
- 3~4와 5~7의 서브트리는
- 업데이트를 할 필요가
- 없다



Lazy Propagation

구간의 합 구하기

- 나중에 노드를 방문했을 때
- 업데이트를 진행하면
- 된다
- 이 배열이 lazy



Lazy Propagation

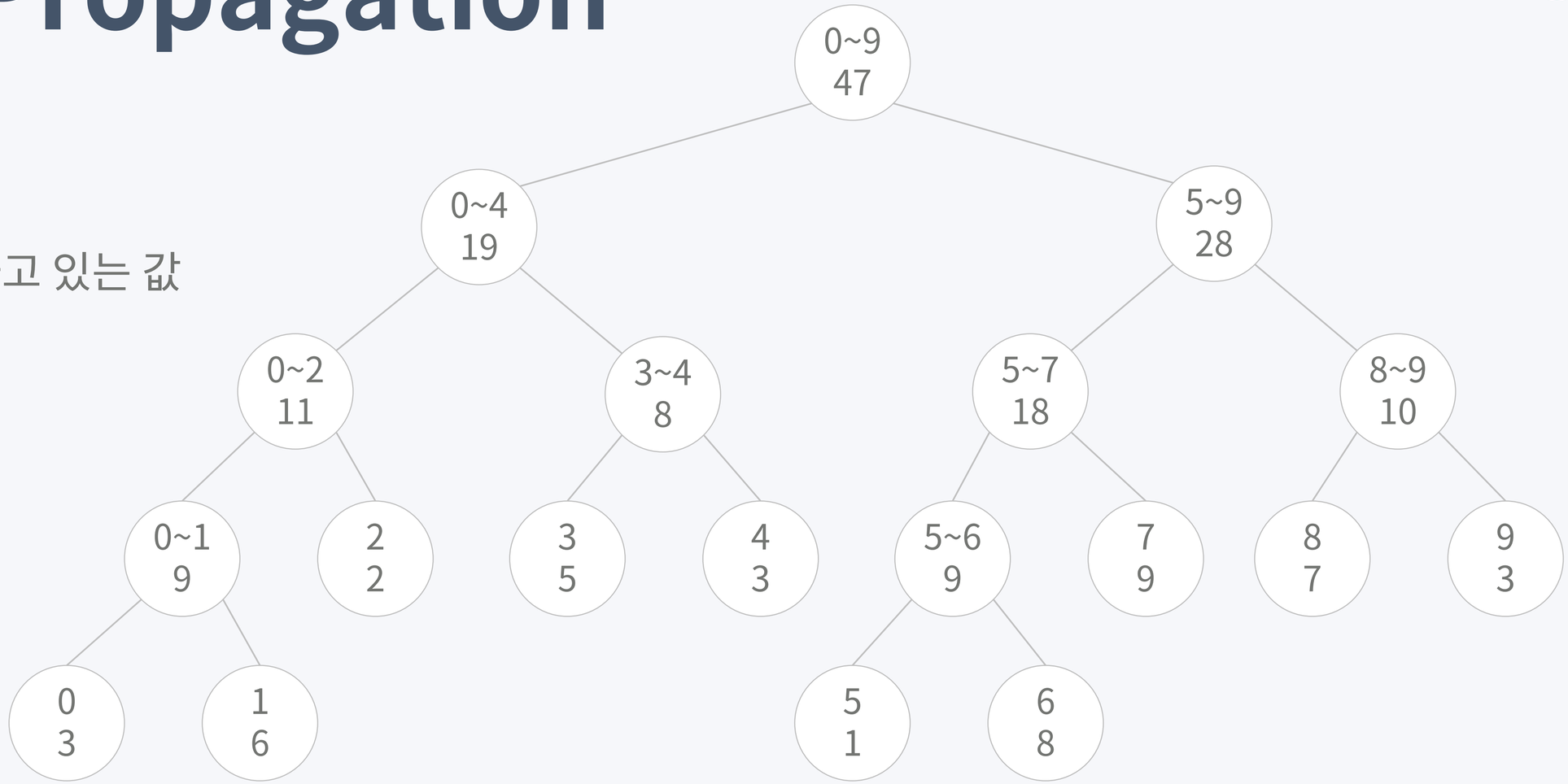
구간의 합 구하기

- 3~4를 나타내는 노드의 lazy에 10이 저장되어 있으면
- 3번째 수와 4번째 수에 10을 더해야 하는데, 나중에 10을 더하겠다는 의미
- 5~7의 lazy에 20이 저장되어 있다면
- 5, 6, 7번째 수에 20을 더해야 하지만, 지금은 더하지 않고 나중에 더하겠다는 의미

Lazy Propagation

구간의 합 구하기

- 위: 범위
- 아래: 저장하고 있는 값

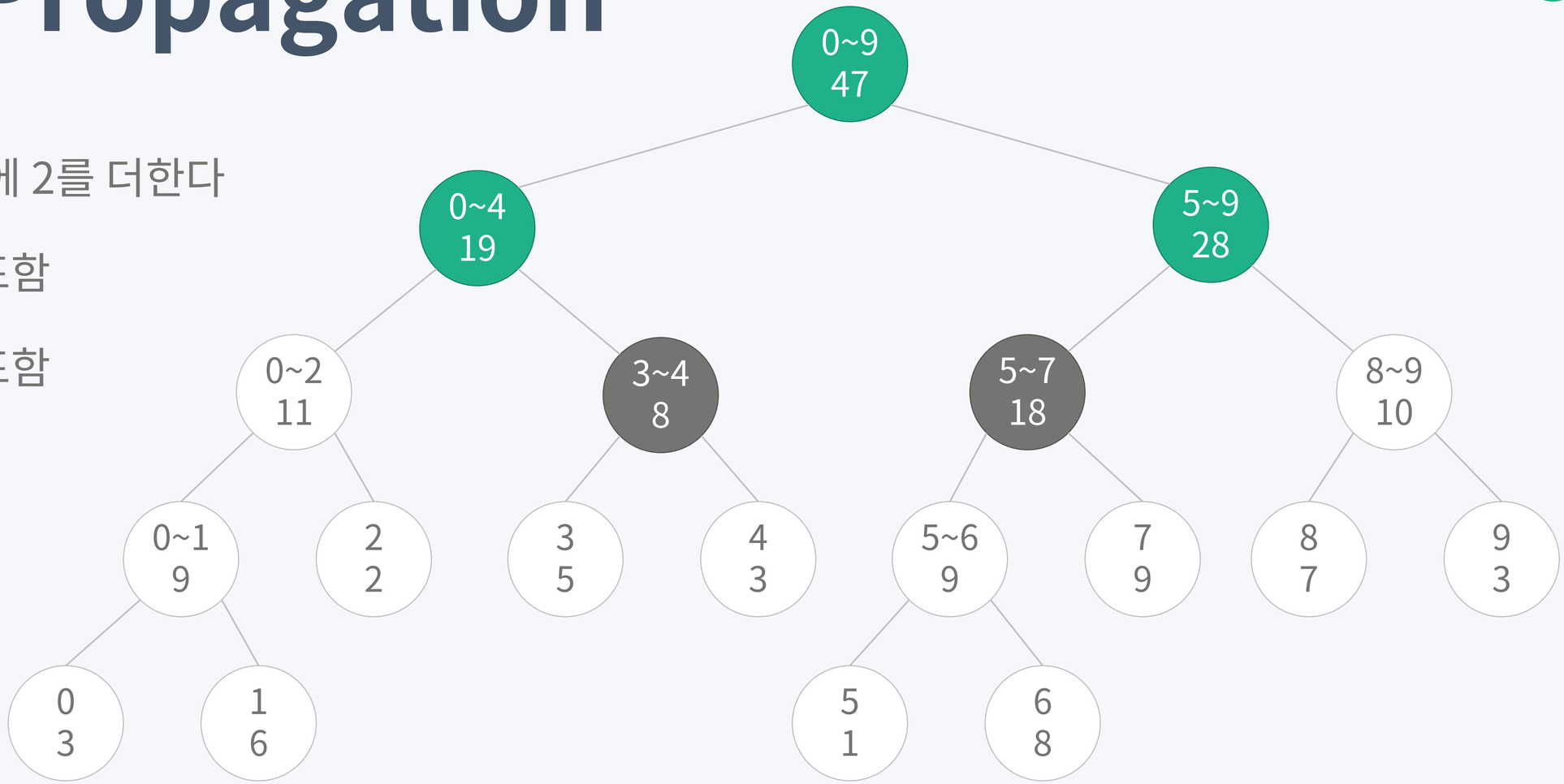


A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
3	6	2	5	3	1	8	9	7	3

Lazy Propagation

구간의 합 구하기

- 3~7번째 수에 2를 더한다
- 초록: 일부 포함
- 검정: 전체 포함

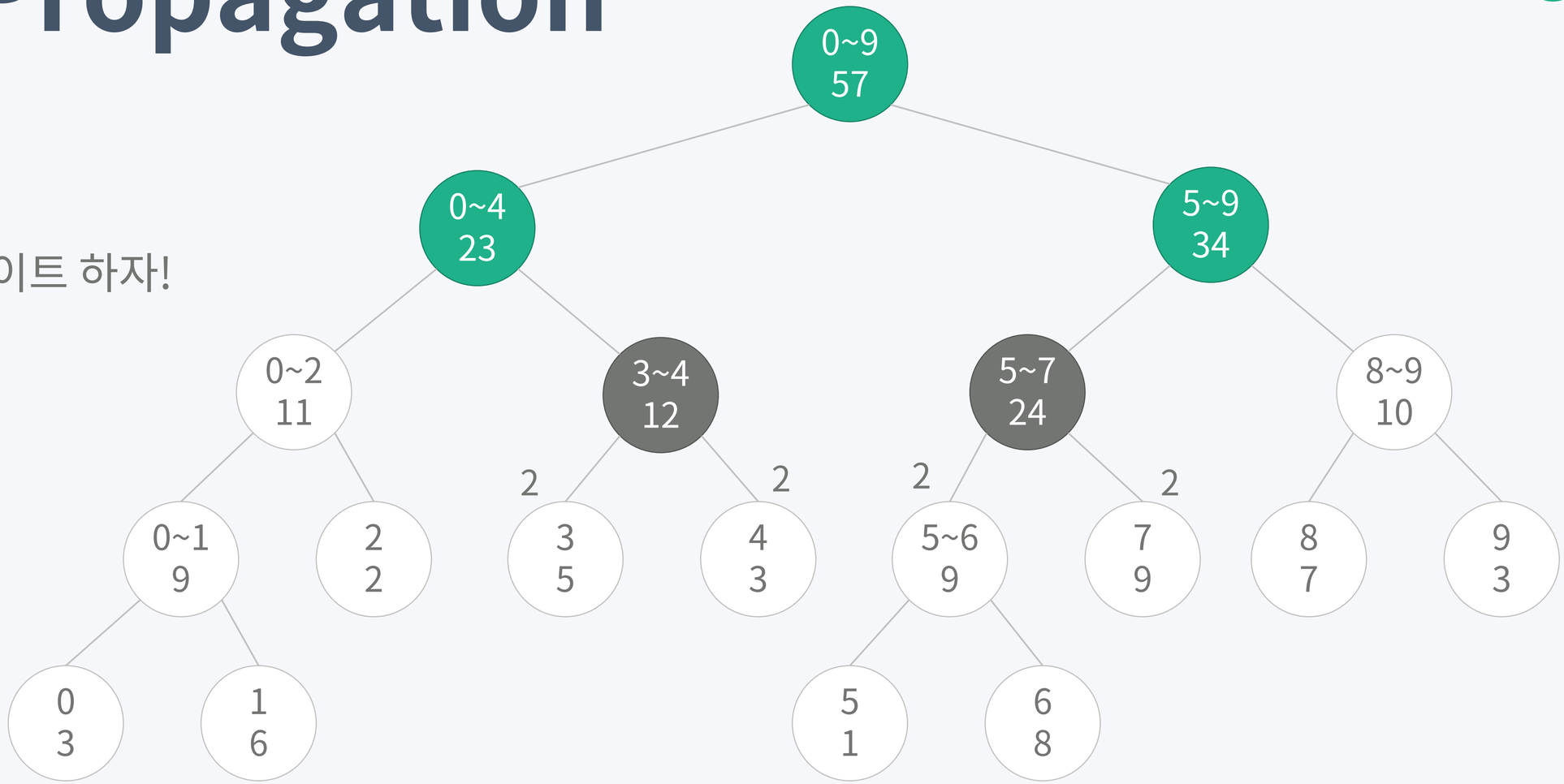


A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
3	6	2	5	3	1	8	9	7	3

Lazy Propagation

구간의 합 구하기

- 검정 아래는
- 나중에 업데이트 하자!

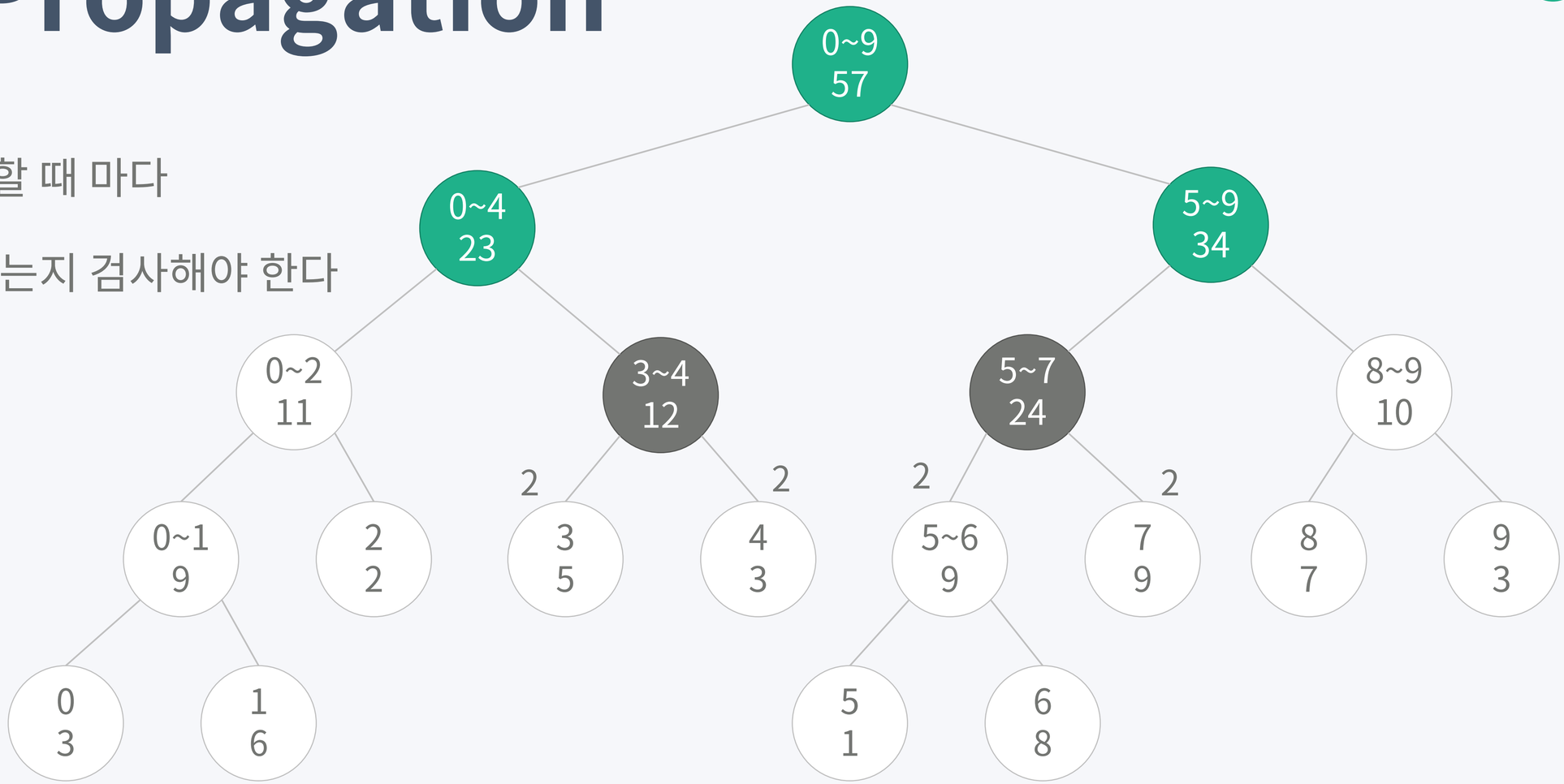


A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
3	6	2	7	5	3	10	11	7	3

Lazy Propagation

구간의 합 구하기

- 노드를 방문할 때 마다
- lazy 값이 있는지 검사해야 한다

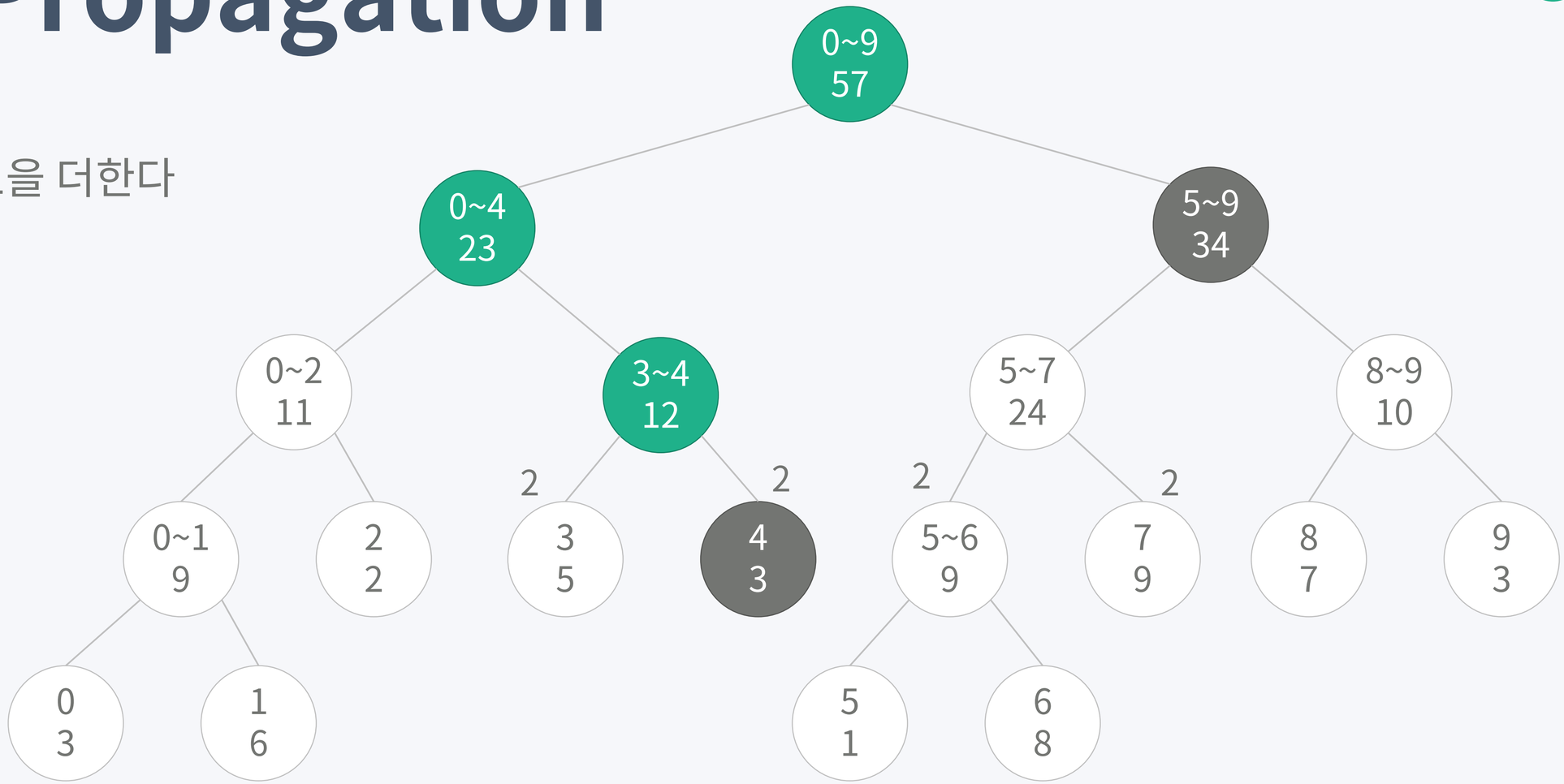


A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
3	6	2	7	5	3	10	11	7	3

Lazy Propagation

구간의 합 구하기

- 4~9번째에 1을 더한다

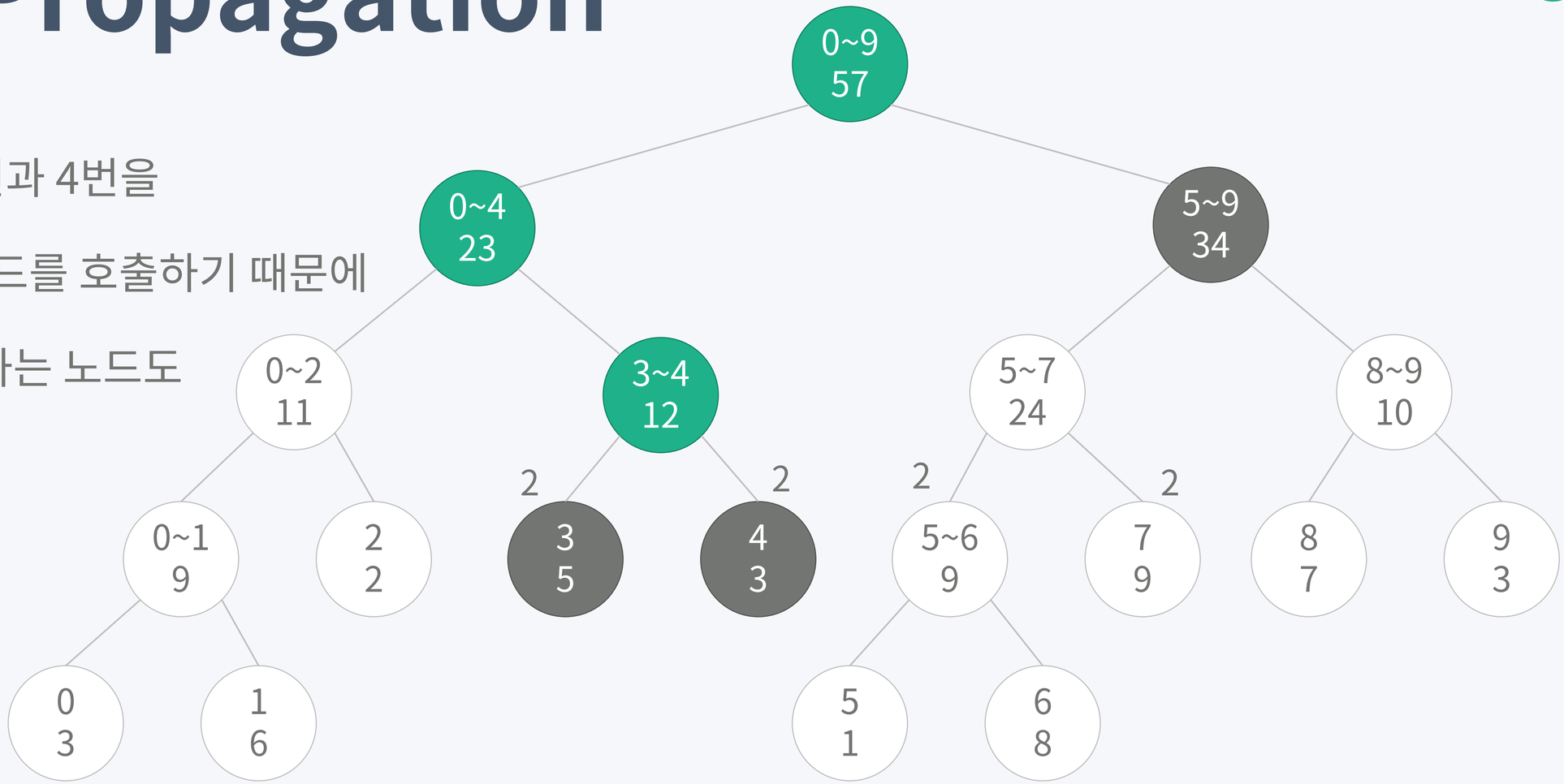


A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
3	6	2	7	5	3	10	11	7	3

Lazy Propagation

구간의 합 구하기

- 3~4에서 3번과 4번을
- 담당하는 노드를 호출하기 때문에
- 3번만 담당하는 노드도
- 호출

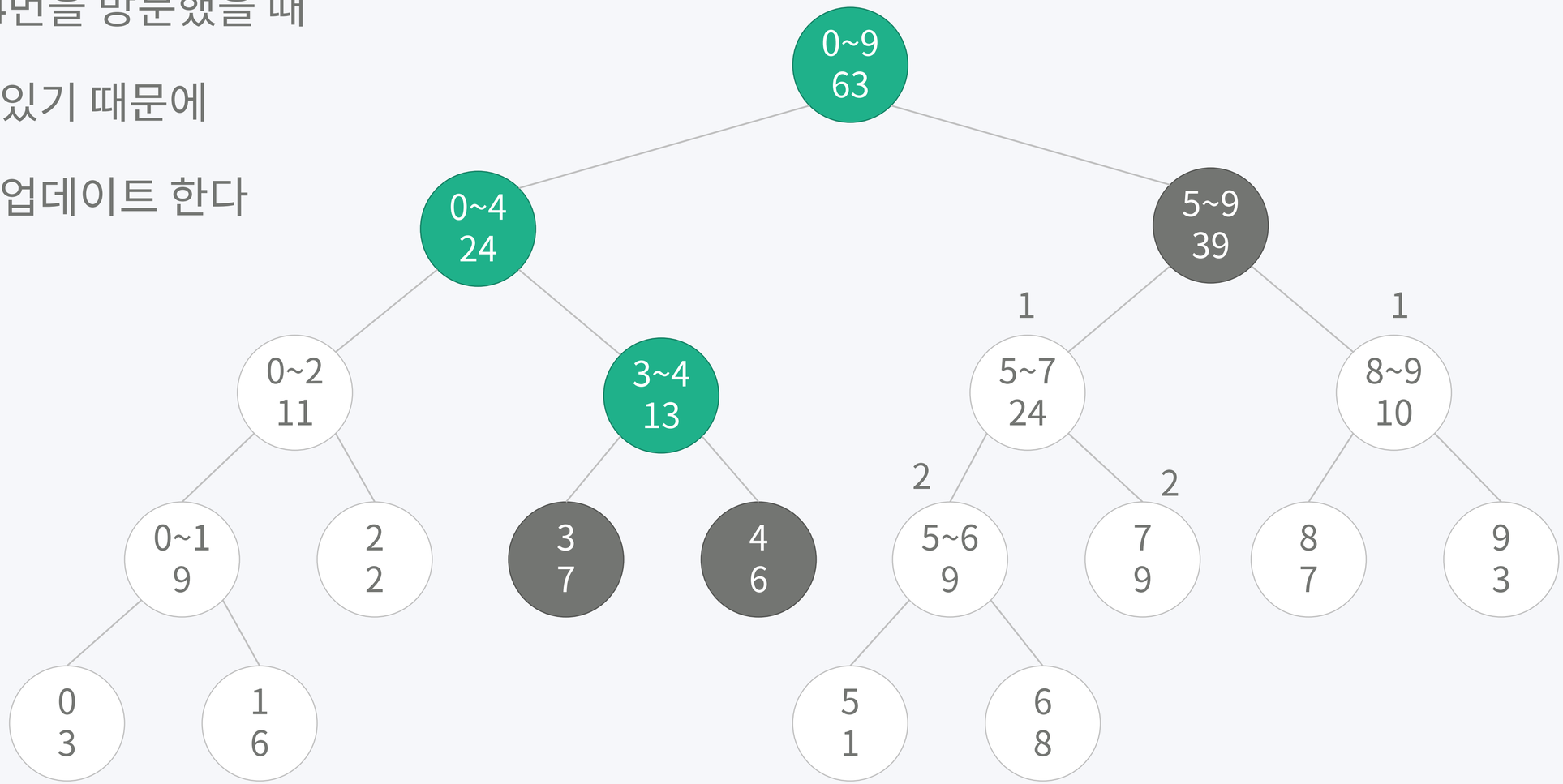


A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]
3	6	2	7	5	3	10	11	7	3

Lazy Propagation

구간의 합 구하기

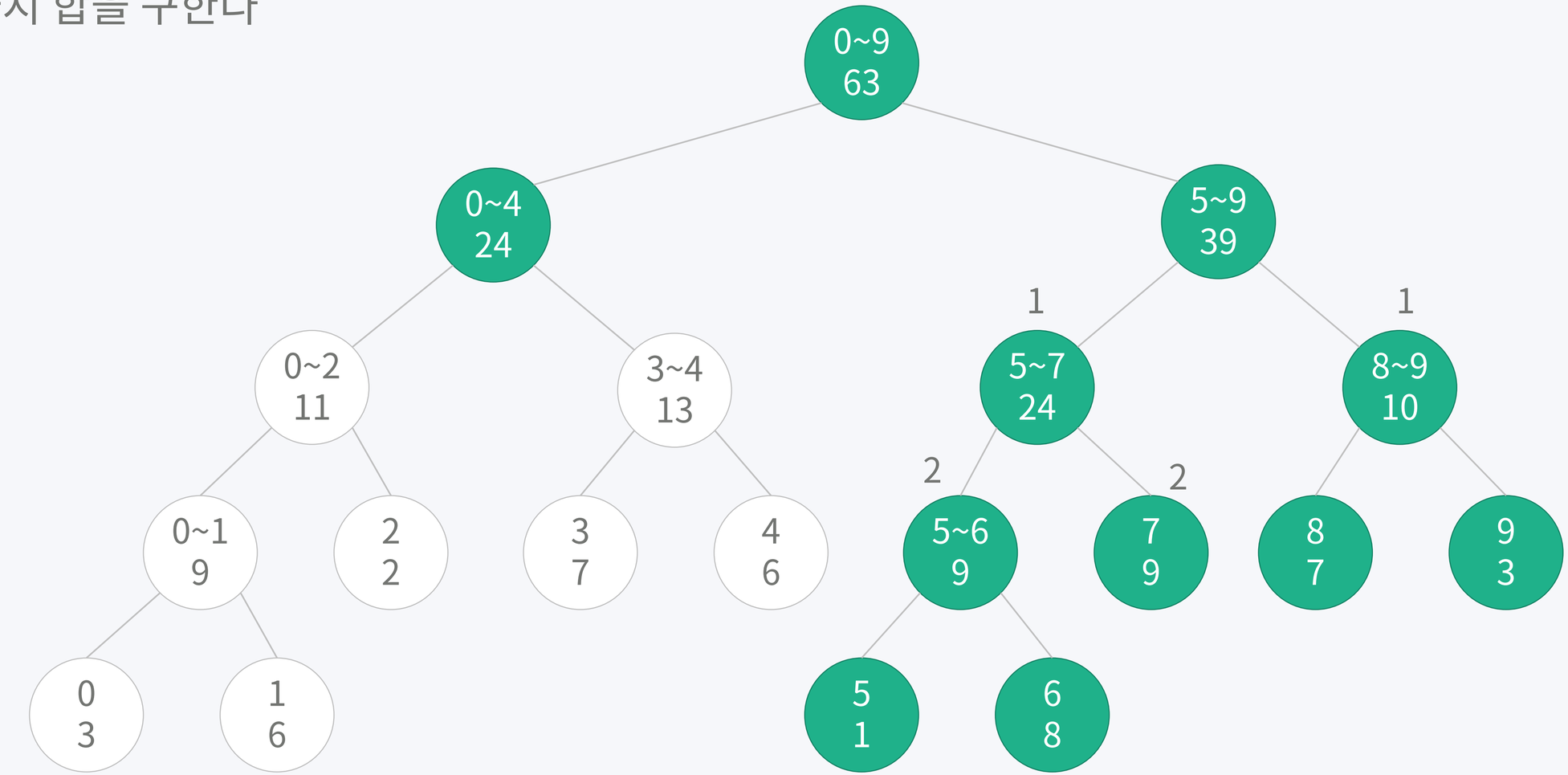
- 3번과 4번을 방문했을 때
- lazy가 있기 때문에
- lazy를 업데이트 한다



Lazy Propagation

구간의 합 구하기

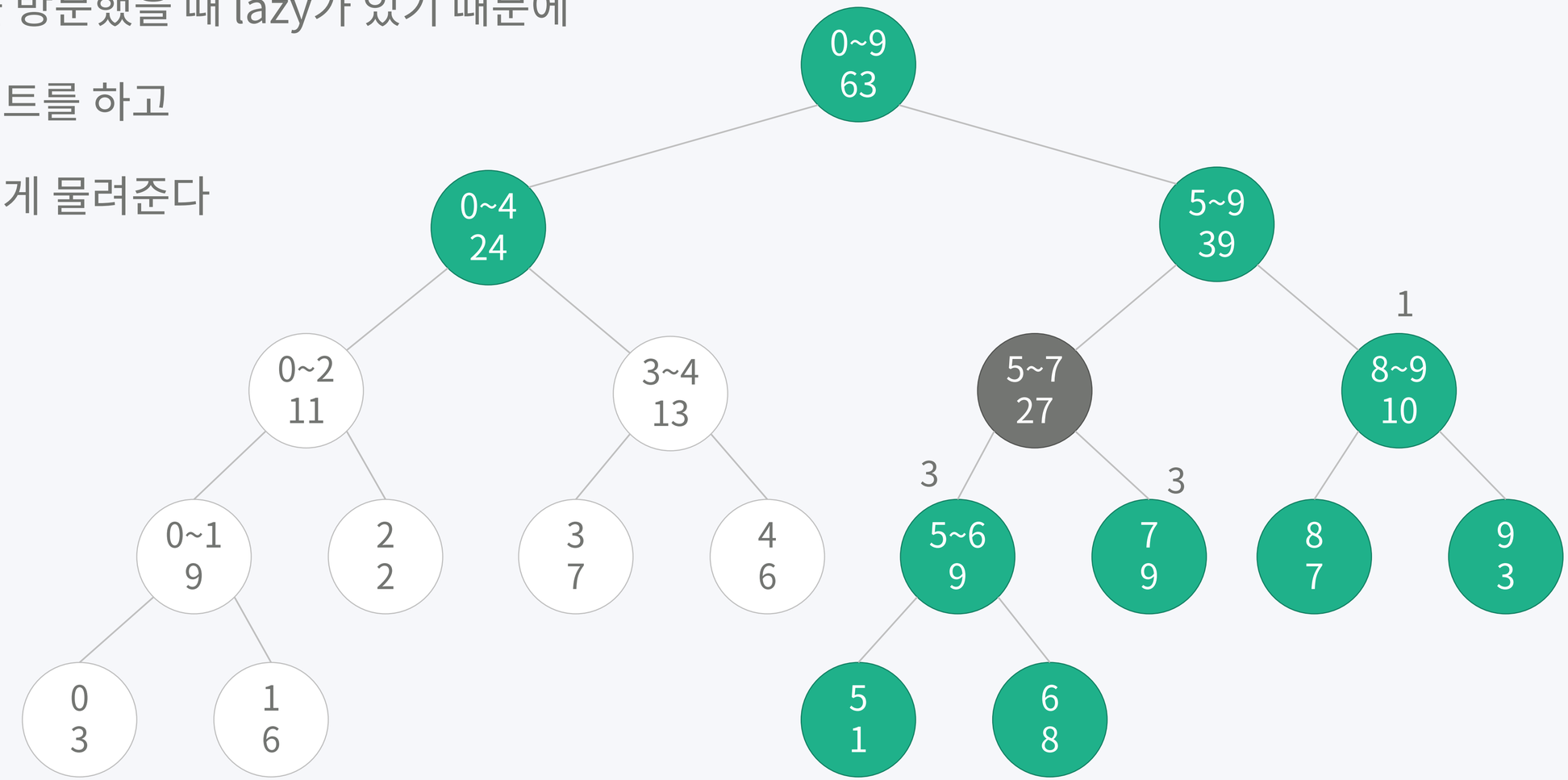
- 6~8까지 합을 구한다



Lazy Propagation

구간의 합 구하기

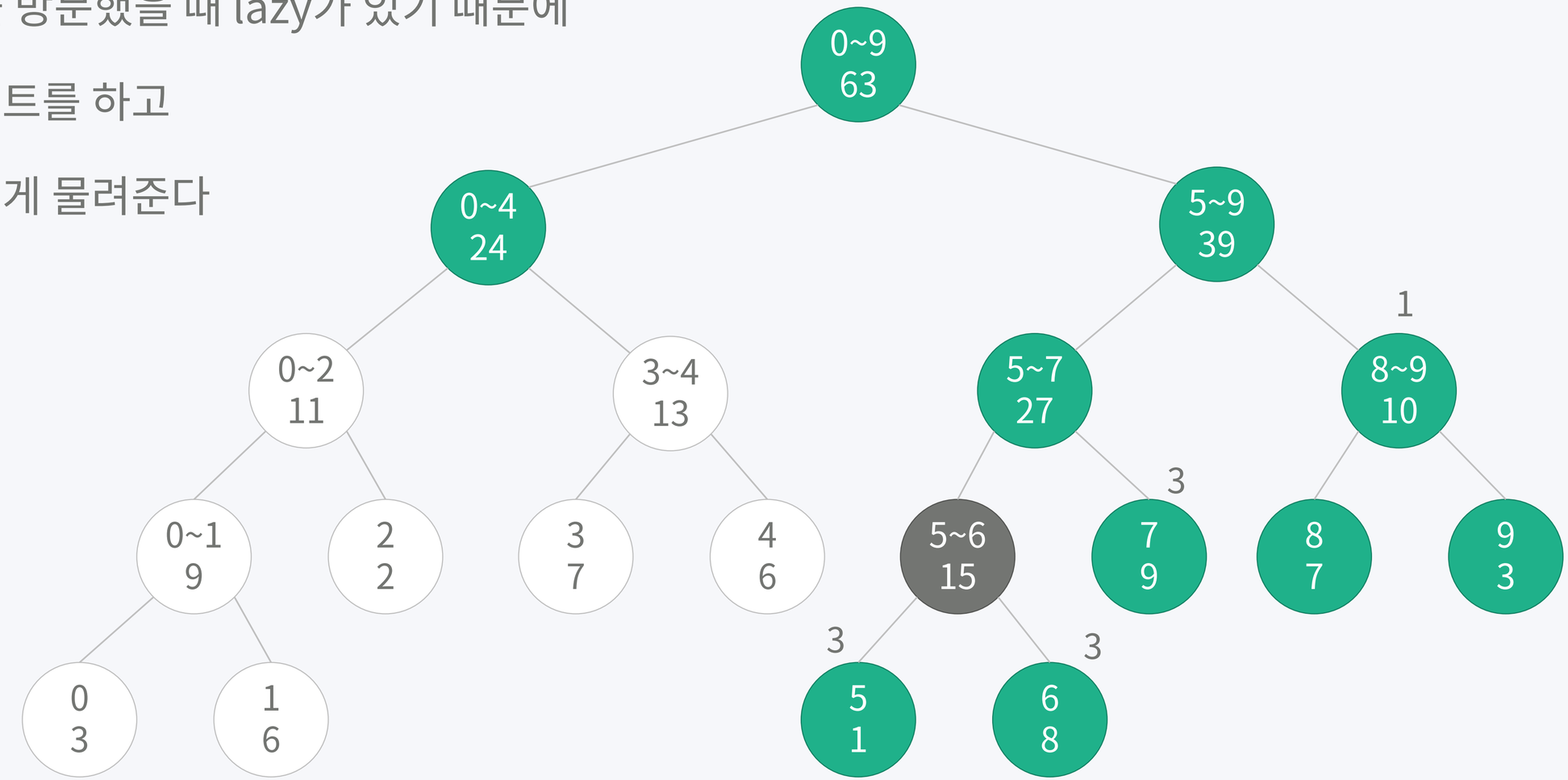
- 5~7 을 방문했을 때 lazy가 있기 때문에
- 업데이트를 하고
- 자식에게 물려준다



Lazy Propagation

구간의 합 구하기

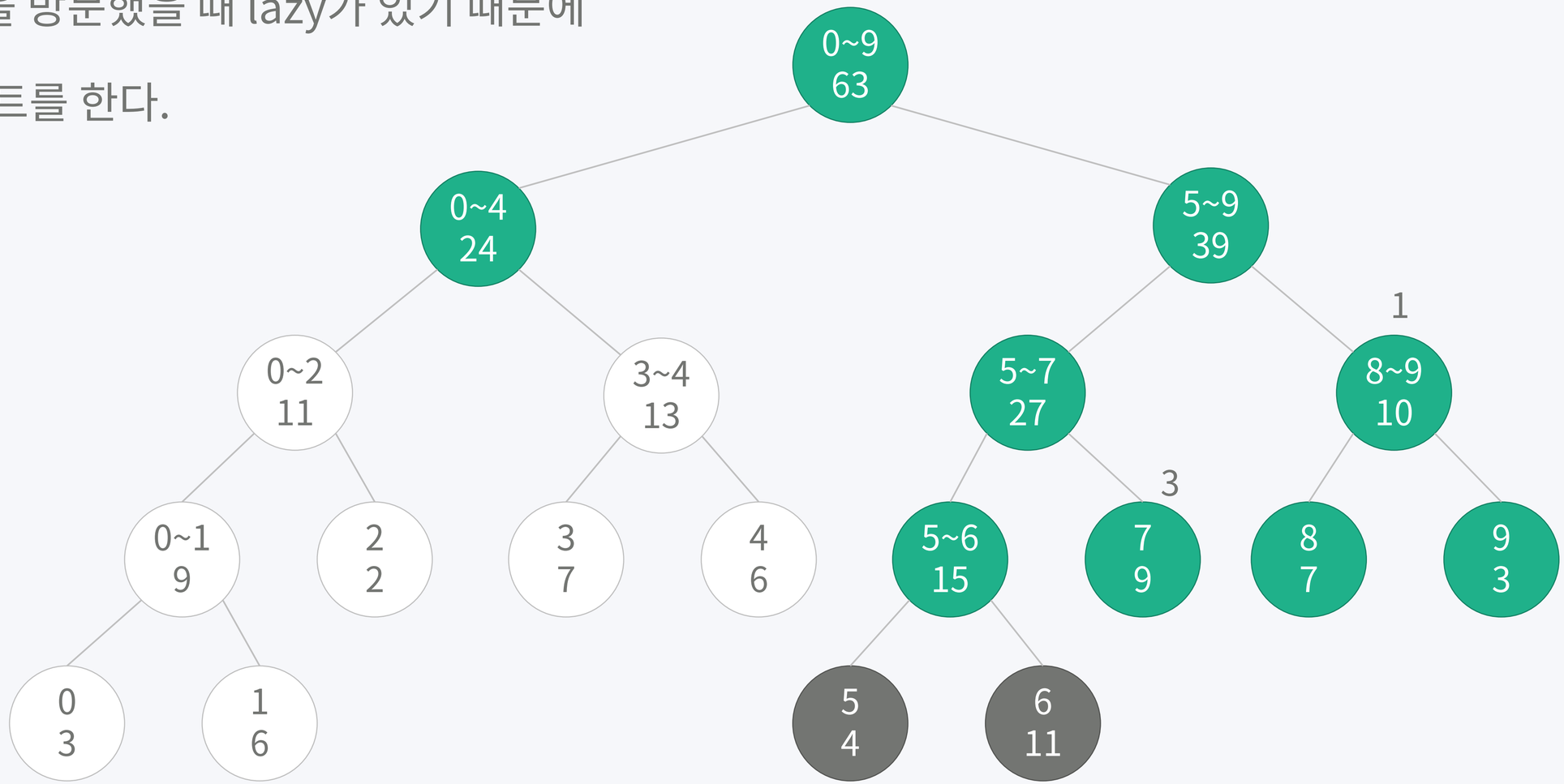
- 5~6 을 방문했을 때 lazy가 있기 때문에
- 업데이트를 하고
- 자식에게 물려준다



Lazy Propagation

구간의 합 구하기

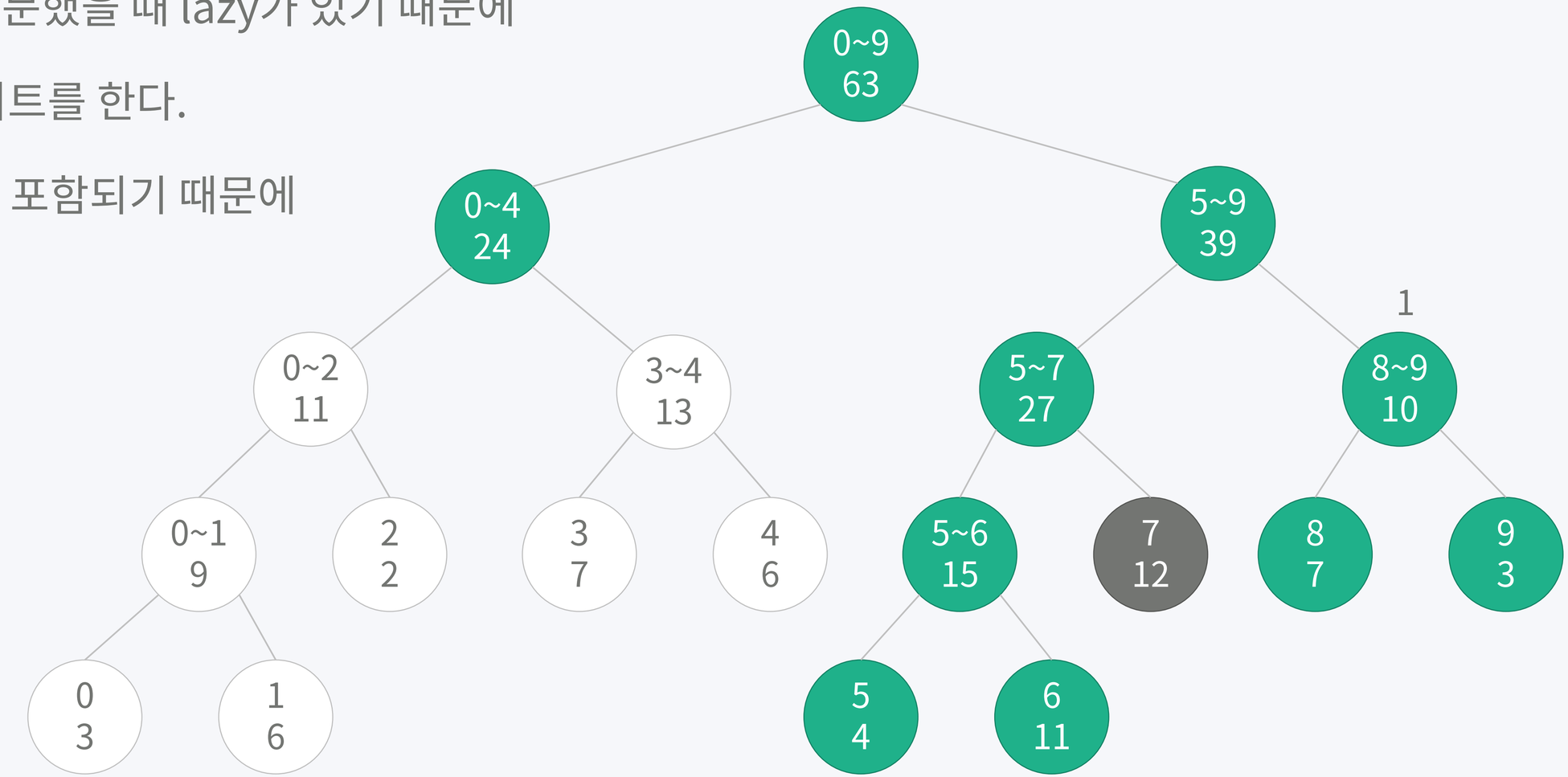
- 5와 6 을 방문했을 때 lazy가 있기 때문에
- 업데이트를 한다.



Lazy Propagation

구간의 합 구하기

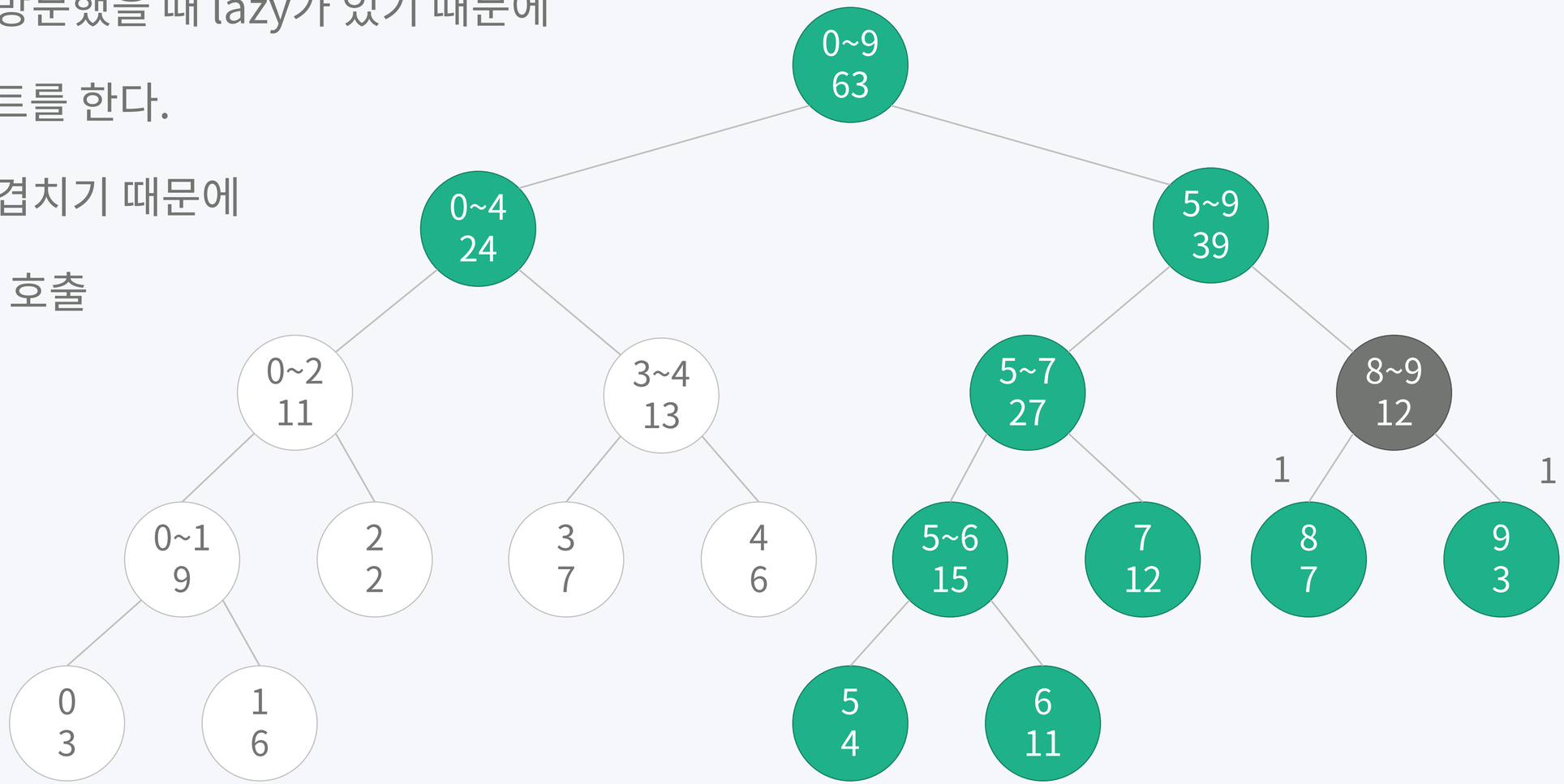
- 7을 방문했을 때 lazy가 있기 때문에
- 업데이트를 한다.
- 6~8에 포함되기 때문에
- 리턴



Lazy Propagation

구간의 합 구하기

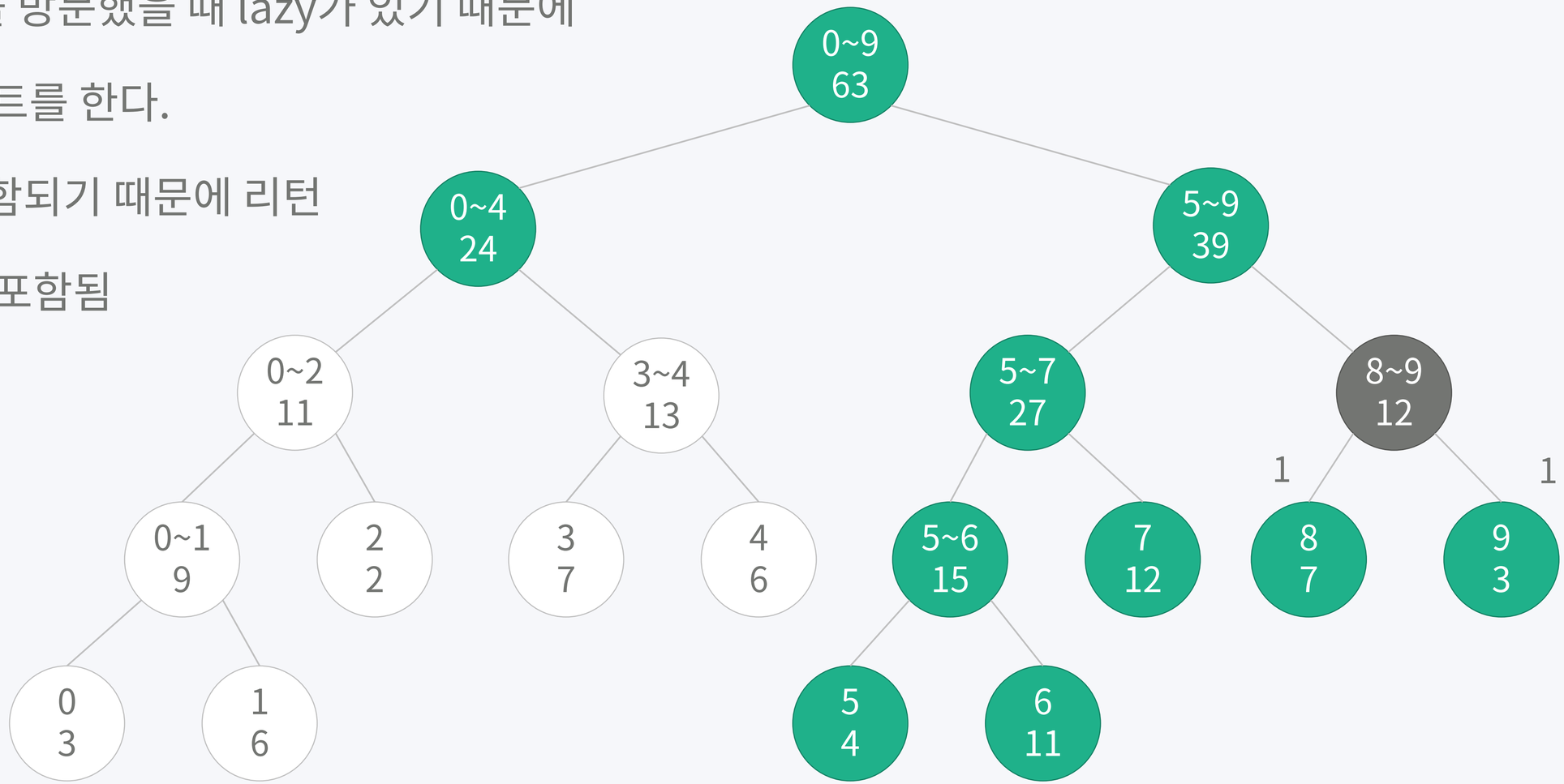
- 8~9을 방문했을 때 lazy가 있기 때문에
- 업데이트를 한다.
- 6~8과 겹치기 때문에
- 두 자식 호출



Lazy Propagation

구간의 합 구하기

- 8과 9을 방문했을 때 lazy가 있기 때문에
- 업데이트를 한다.
- 8은 포함되기 때문에 리턴
- 9는 안 포함됨



Lazy Propagation

구간의 합 구하기

```
void update_lazy(vector<long long> &tree, vector<long long> &lazy,
int node, int start, int end) {
    if (lazy[node] != 0) {
        tree[node] += (end-start+1)*lazy[node];
        // leaf가 아니면
        if (start != end) {
            lazy[node*2] += lazy[node];
            lazy[node*2+1] += lazy[node];
        }
        lazy[node] = 0;
    }
}
```

구간 합 구하기 2

https://www.acmicpc.net/problem/10999

- 소스: http://boj.kr/b3196c338b0c466aa1f65a919f5a7fbe

스위치

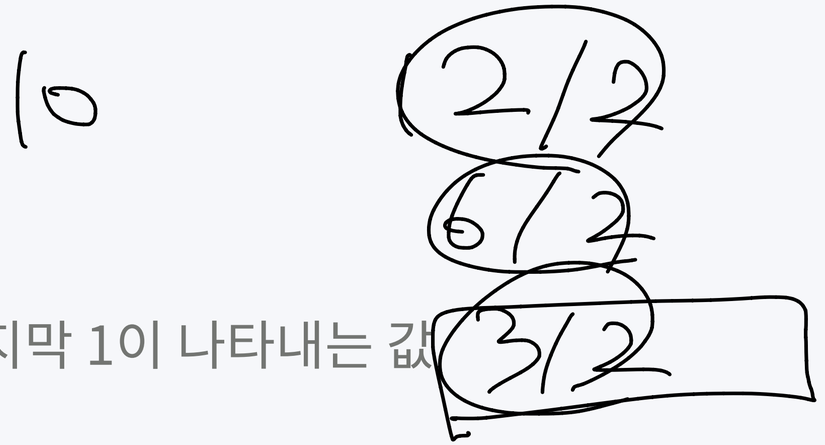
https://www.acmicpc.net/problem/1395

- 소스: http://boj.kr/c2341cb98de54df8bde74782c6c0ed41

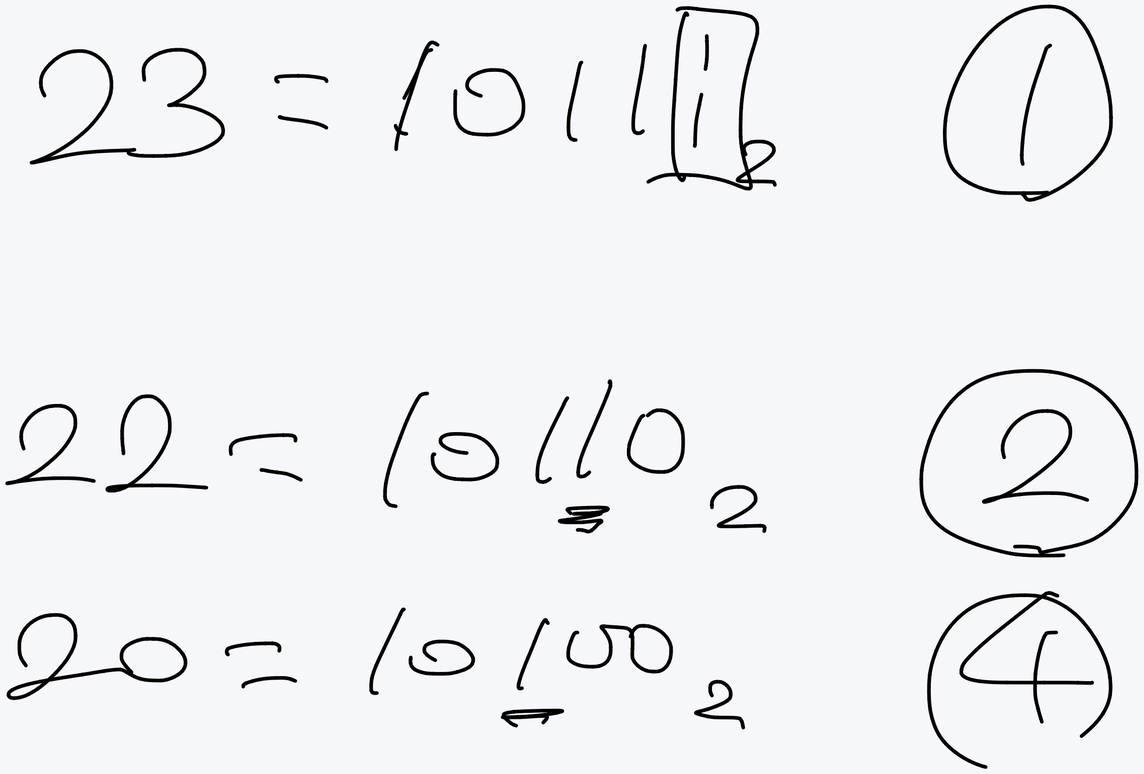
Fenwick Tree

Fenwick Tree

Fenwick Tree (BIT)



- i의 마지막 비트: i를 2진수로 나타냈을 때, 가장 마지막 1이 나타내는 값
- $3 = 1\mathbf{1}_2$
- $5 = 10\mathbf{1}_2$
- $6 = 1\mathbf{1}0_2$
- $8 = \mathbf{1}000_2$
- $9 = 100\mathbf{1}_2$
- $10 = 10\mathbf{1}0_2$
- $11 = 101\mathbf{1}_2$
- $12 = 1\mathbf{1}00_2$
- $16 = \mathbf{1}0000_2$



Fenwick Tree

Fenwick Tree (BIT)

- `-num = ~num + 1`
- `num = 10011010111010110000000000`
- `~num = 01100101000101001111111111`
- `-num = 01100101000101010000000000`
- `num & -num = 00000000000000010000000000`

if (N&(N-1))==0

N이 2^k 이

Fenwick Tree

Fenwick Tree (BIT)

6 & -6 = 2

12 = 1100₂

14 = 1110

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
tree 1	2	1	4	1	2	1	8	1	2	1	4	1	2	1	16

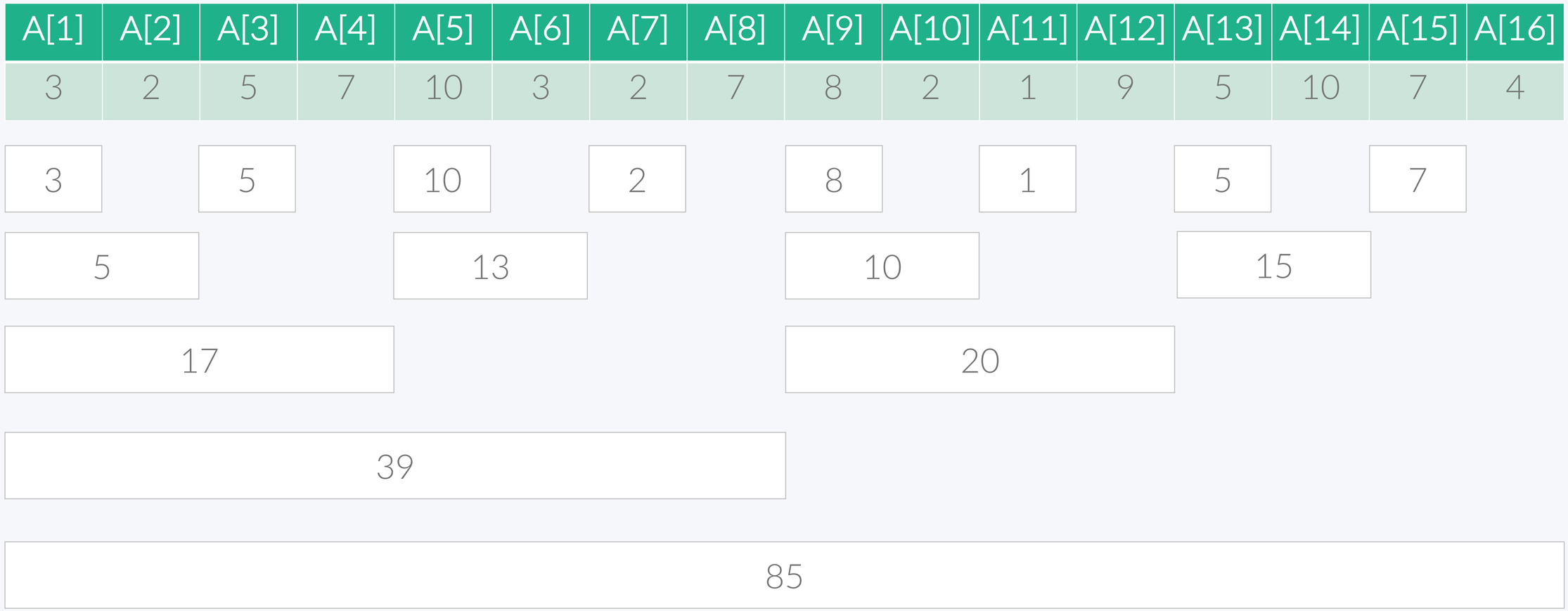
1		3		5		7		9		11		13		15	
	2				6				10				14		
			4								12				
							8								
															16

tree[12] = A[9] - A[12]

Fenwick Tree

Fenwick Tree (BIT)

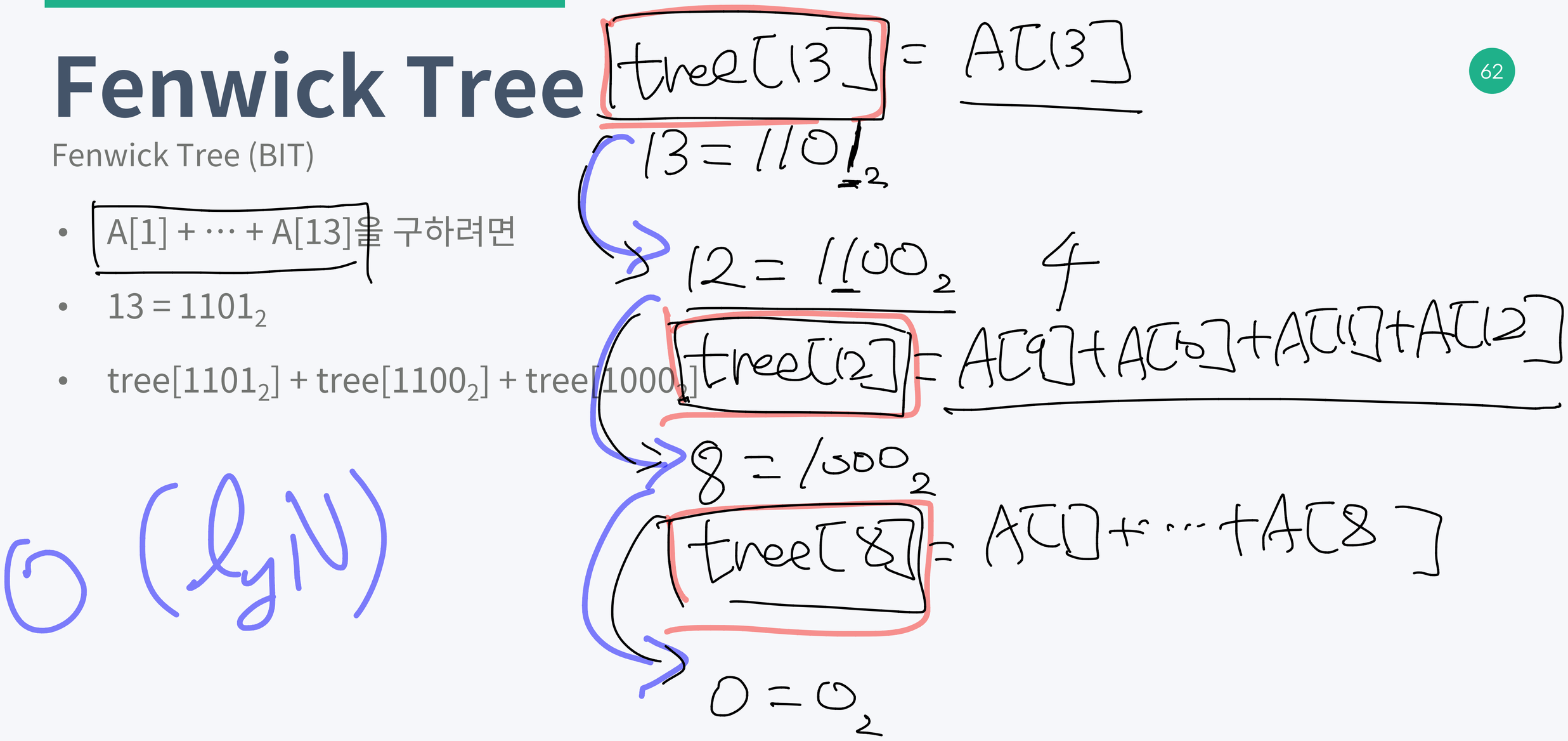
A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]	A[11]	A[12]	A[13]	A[14]	A[15]	A[16]
3	2	5	7	10	3	2	7	8	2	1	9	5	10	7	4



Fenwick Tree

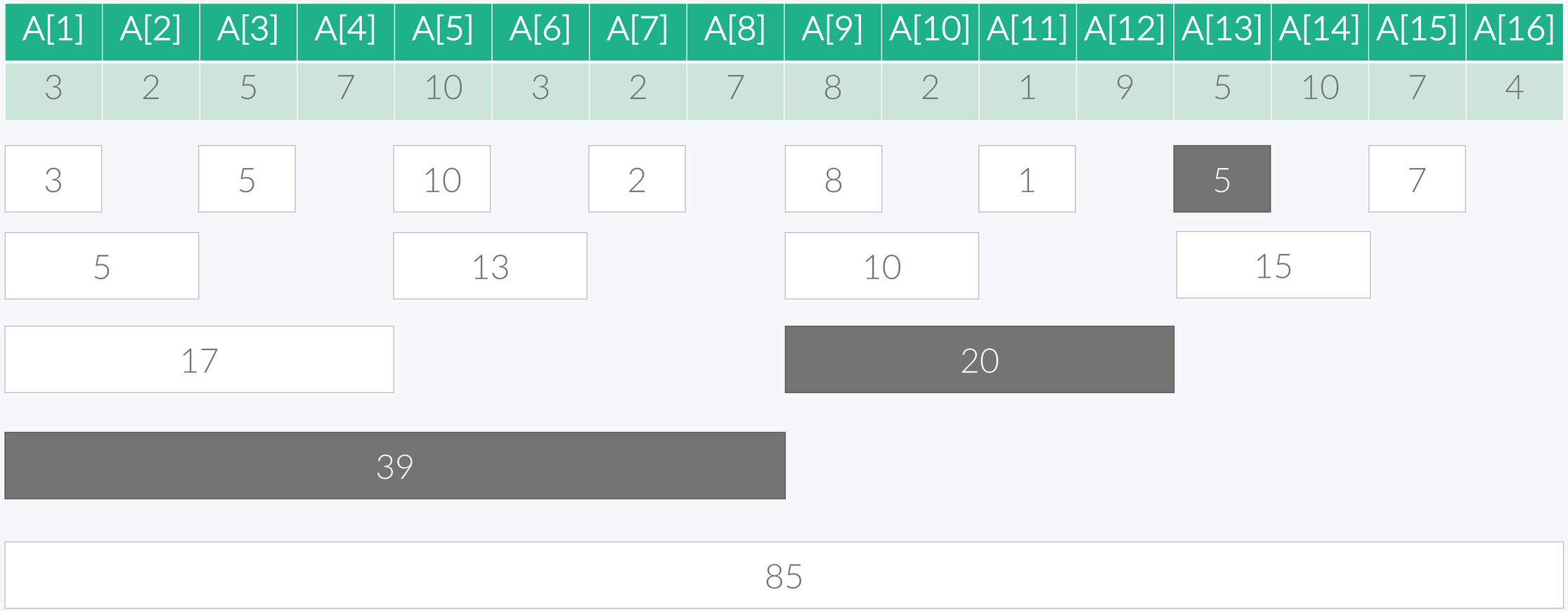
Fenwick Tree (BIT)

- A[1] + … + A[13]을 구하려면
- $13 = 1101_2$
- $tree[1101_2] + tree[1100_2] + tree[1000_2]$



Fenwick Tree

Fenwick Tree (BIT)



Fenwick Tree

Fenwick Tree (BIT)

```
int sum(int i) {
    int ans = 0;
    while (i > 0) {
        ans += tree[i];
        i -= (i & -i);
    }
    return ans;
}
```

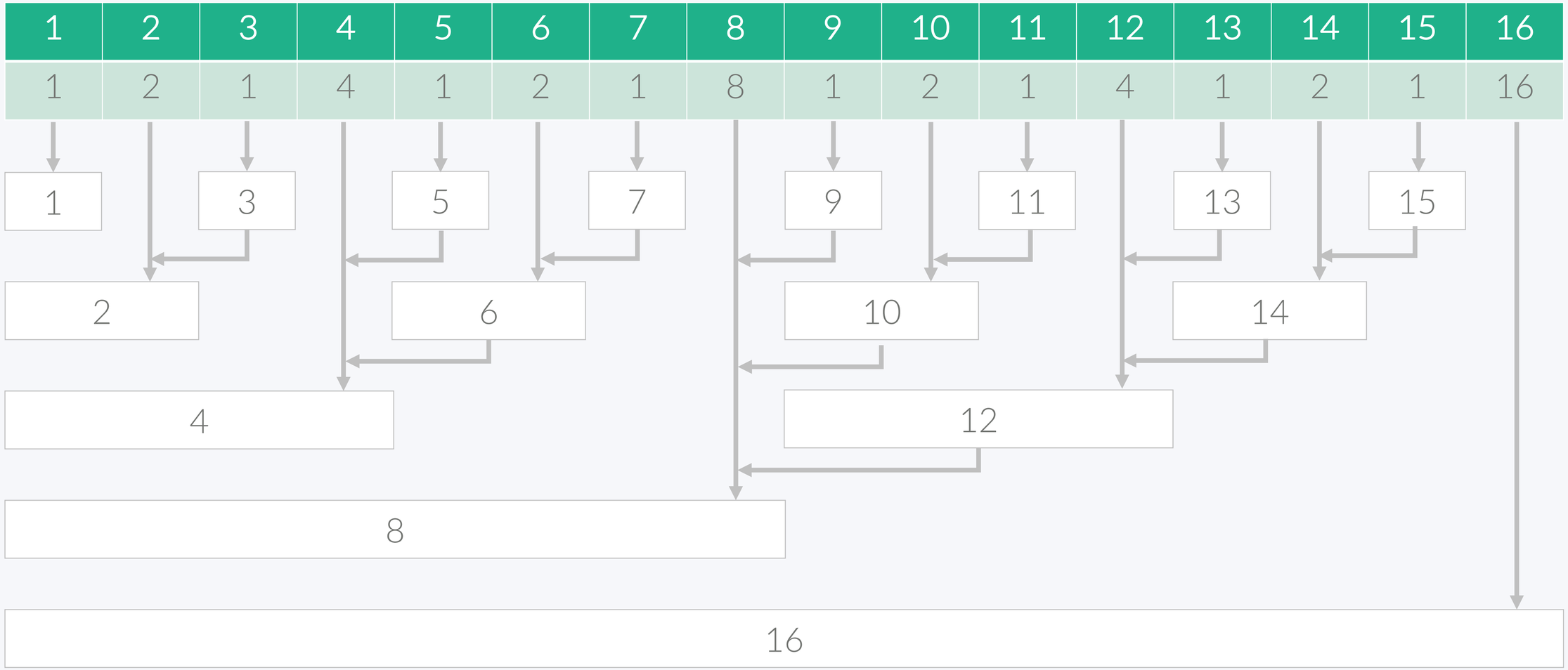
1 ~ i 까지의 합

[~]

Sum(j) - Sum(i-1)

Fenwick Tree

Fenwick Tree (BIT)



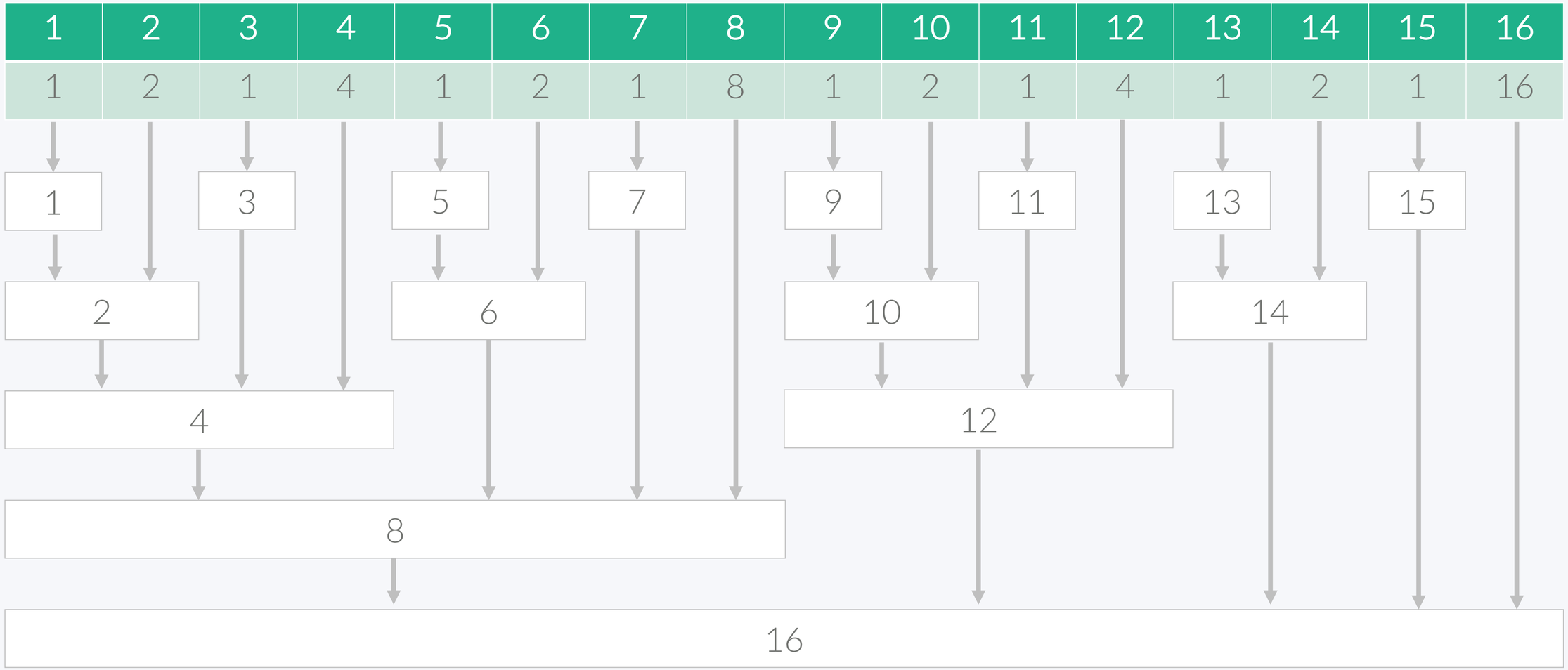
Fenwick Tree

Fenwick Tree (BIT)

```
int update(int i, int num) {
    while (i <= n) {
        tree[i] += num;
        i += (i & -i);
    }
}
```

Fenwick Tree

Fenwick Tree (BIT)



구간 합 구하기

https://www.acmicpc.net/problem/2042

일차원 배열
이차원 배열 : 일차원 배열의 일차원

- 소스 1: http://boj.kr/b4340d2a7bd946fe9bddf9244b43b650
- 소스 2: http://boj.kr/bcddf78aee4748989991ae35fdb8b8f0

1차원 A[X1] ~ A[X2]

```
for(i= X1; i<=X2; i++) {
    sum += A[i];
}
```

2차원 A[X1][Y1] ~ A[X2][Y2]

```
for(i= X1; i<=X2; i++) {
    for(j= Y1; j<=Y2; j++) {
        sum += A[i][j];
    }
}
```

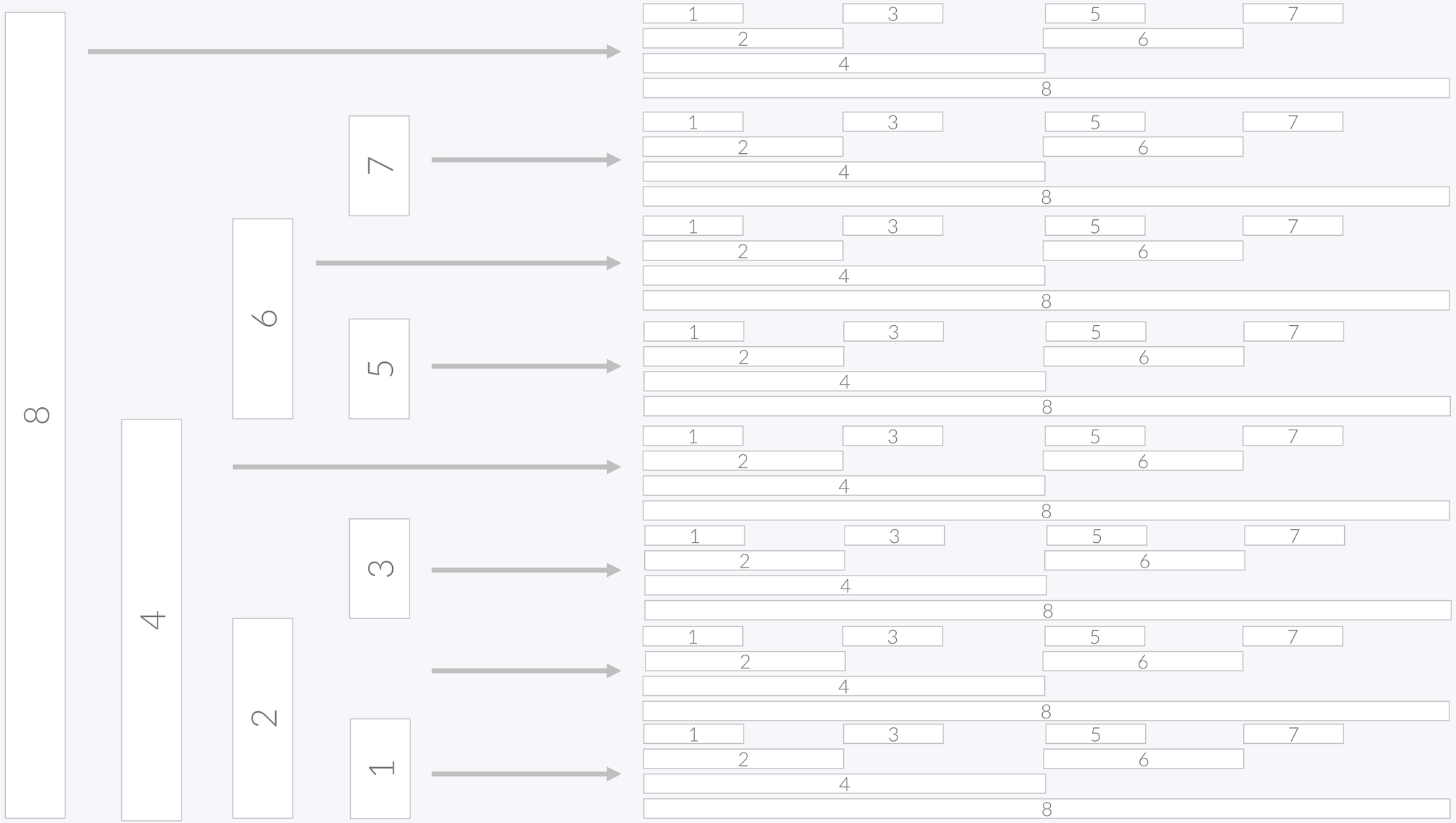
2D Fenwick Tree

2D Fenwick Tree (BIT)

- 1차원을 2차원으로 확장해서 풀면 된다.
- x에 대해서 그리고 y에 대해서 트리를 만들면 된다

2D Fenwick Tree

2D Fenwick Tree (BIT)



2D Fenwick Tree

2D Fenwick Tree (BIT)

```
void update(int x, int y, int val) {
    for (int i=x; i<=n; i+=i&-i) {
        for (int j=y; j<=n; j+=j&-j) {
            tree[i][j] += val;
        }
    }
}
```

2D Fenwick Tree

2D Fenwick Tree (BIT)

```
int sum(int x, int y) {
    int ans = 0;
    for (int i=x; i>0; i-=i&-i) {
        for (int j=y; j>0; j-=j&-j) {
            ans += tree[i][j];
        }
    }
    return ans;
}
```

구간 합 구하기 3

https://www.acmicpc.net/problem/11658

- 소스: http://boj.kr/e1591b97a44344bbad57b8892113ec60