SAMSUNG
SW
ACADEMY
FOR
YOUTH

함께가요 미래로!
Enabling People

삼성 청년 SW 아카데미

자료구조

목차

목차

자료구조

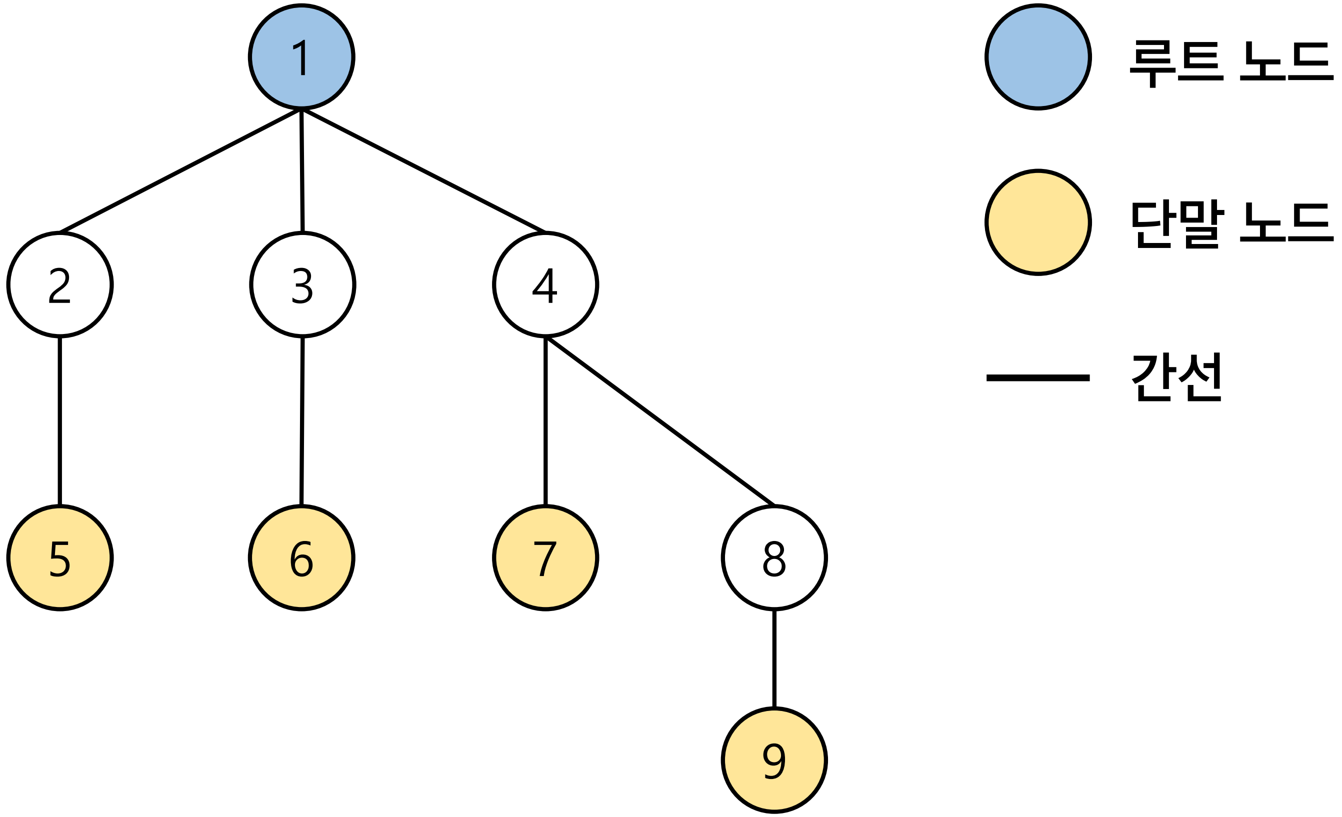
- Tree
- Segment Tree
- Fenwick Tree
- 업데이트
- Lazy Propagation

Tree

Tree

Tree 란?

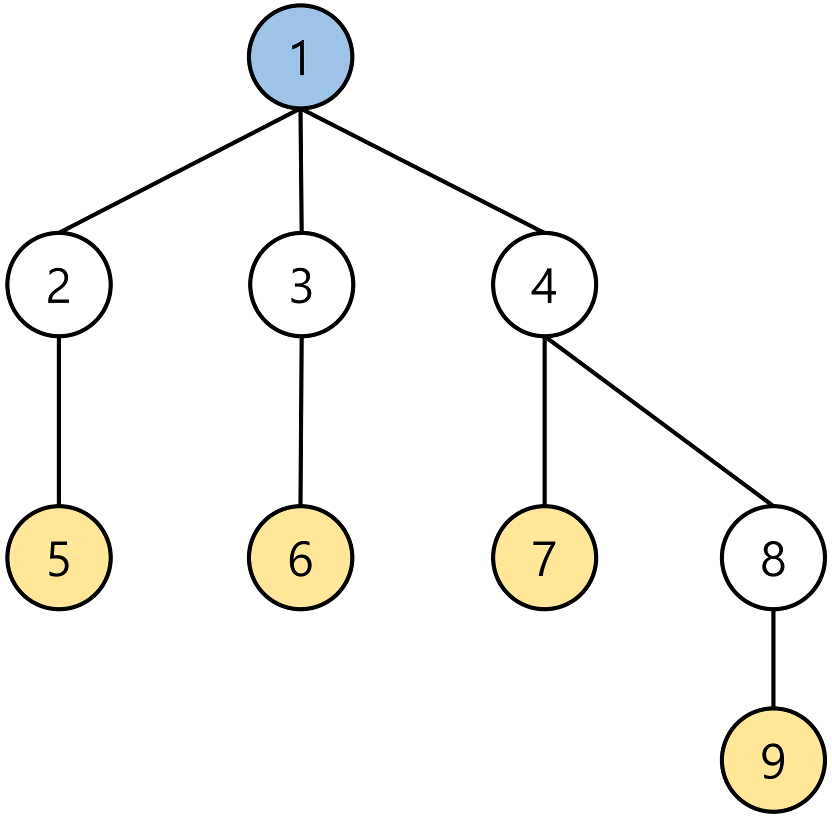
노드 간 계층과 1:N 관계를 갖는 자료구조.



Tree

Tree - Degree (차수)

노드에 연결된 자식 노드의 개수.



2번 노드의 차수는 1

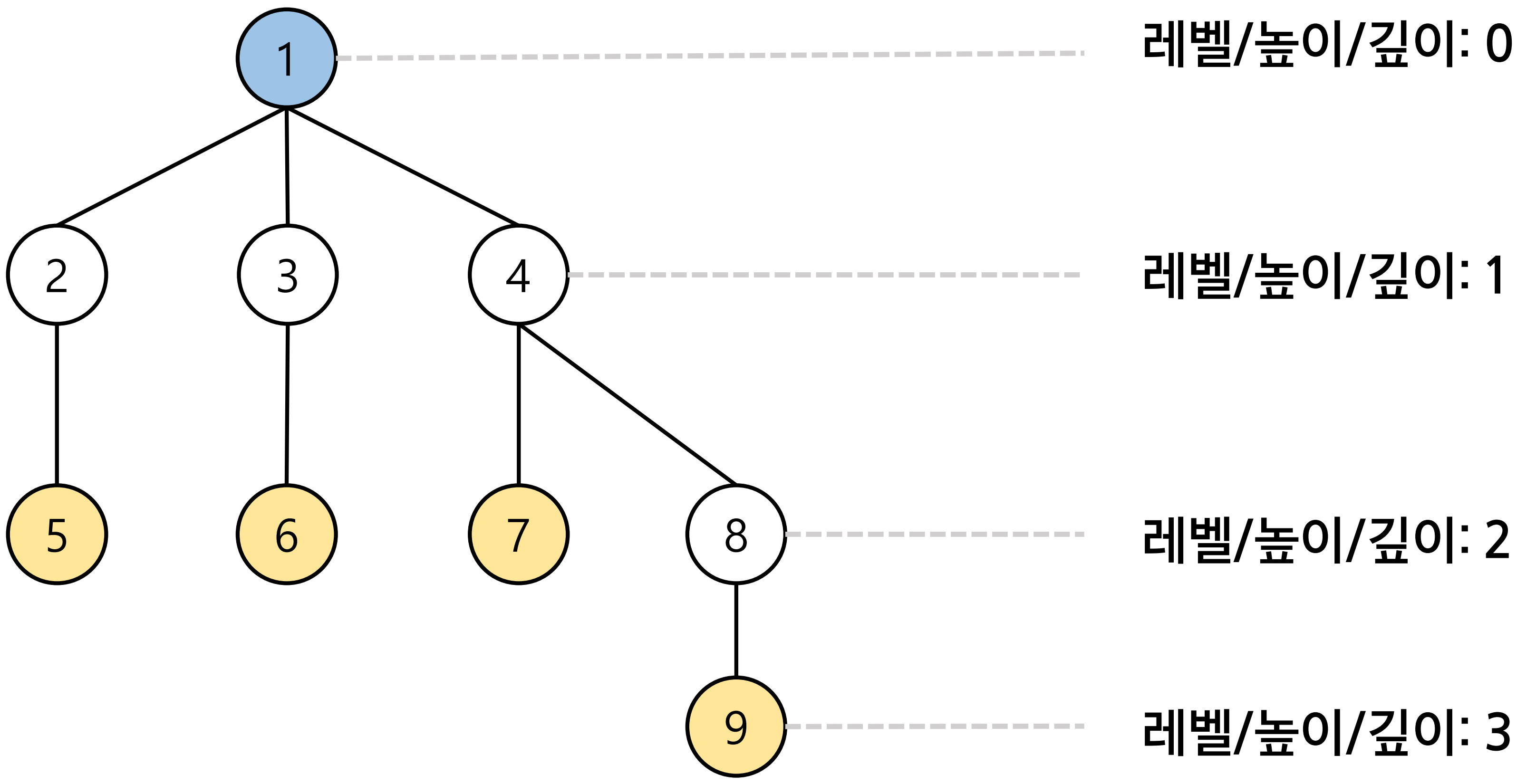
4번 노드의 차수는 2

9번 노드의 차수는 ?

Tree

Tree - Level, Height, Depth (레벨, 높이, 깊이)

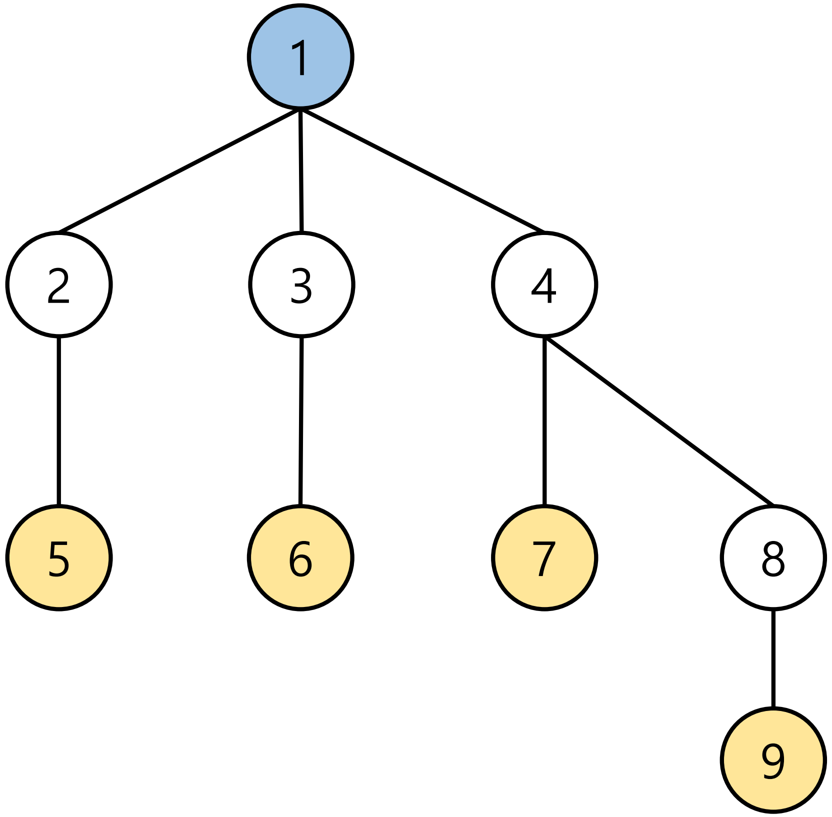
일반적으로 루트 노드부터 가장 높은 레벨을 말한다.



Binary Tree

Binary Tree (이진 트리) 란?

모든 노드가 최대 2개의 자식 노드를 가질 수 있는 트리.

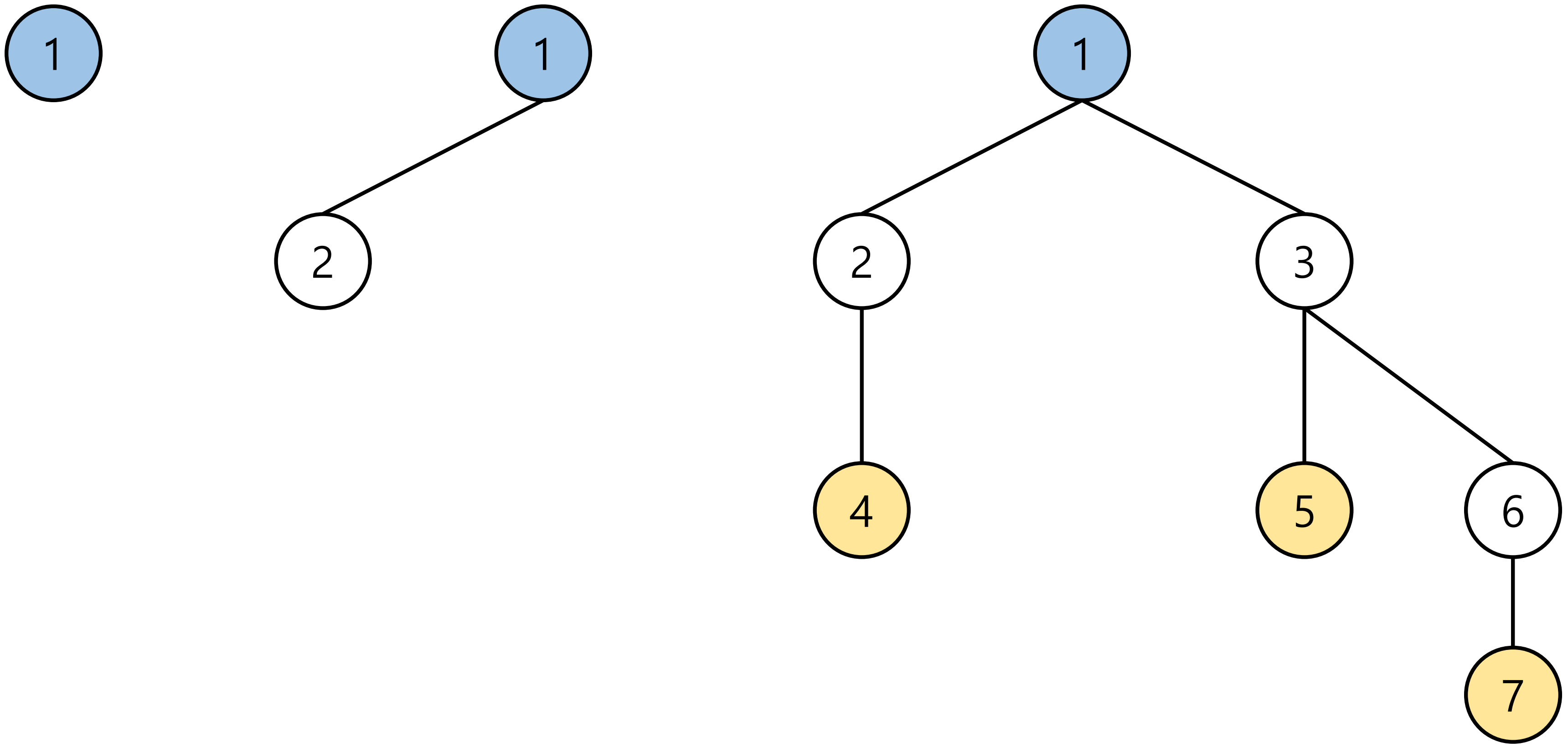


이 트리는 Binary Tree 일까?

Binary Tree

Binary Tree (이진 트리) 란?

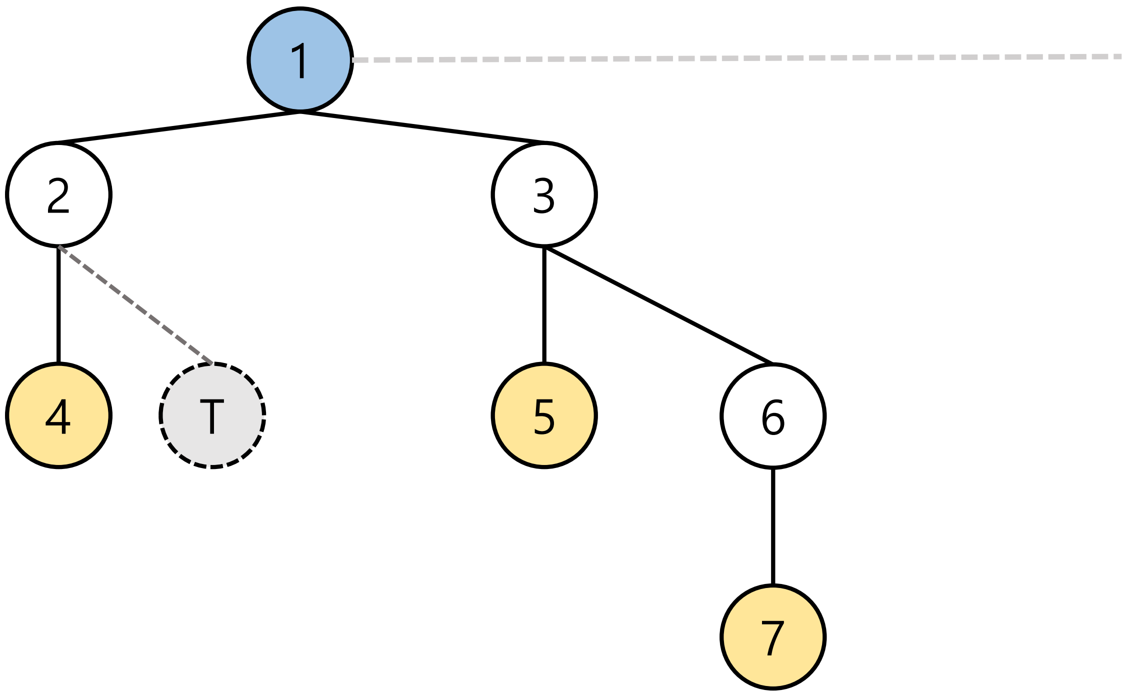
모든 노드가 최대 2개의 자식 노드를 가질 수 있는 트리.



Binary Tree

Binary Tree - 노드의 개수 구하기

Q. 특정 레벨에서의 최대 노드의 개수는?



2^0

2^1

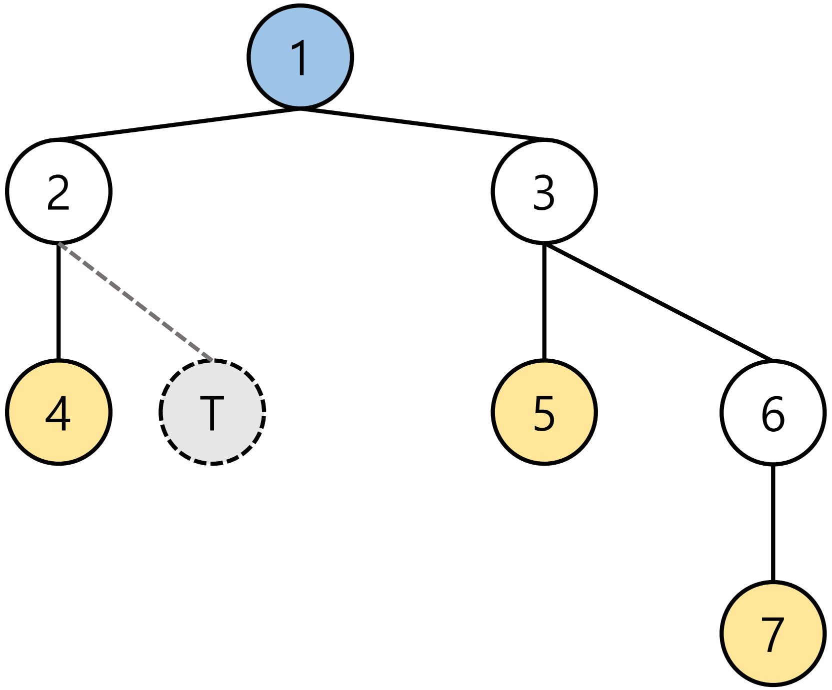
2^2

2^3

Binary Tree

Binary Tree - 노드의 개수 구하기

Q. 높이가 H일 때, Binary Tree가 가질 수 있는 노드의 수는?



레벨/높이: 0 => 최소: 1개 / 최대 1개

레벨/높이: 1 => 최소: 2개 / 최대 3개

레벨/높이: 2 => 최소: 3개 / 최대 7개

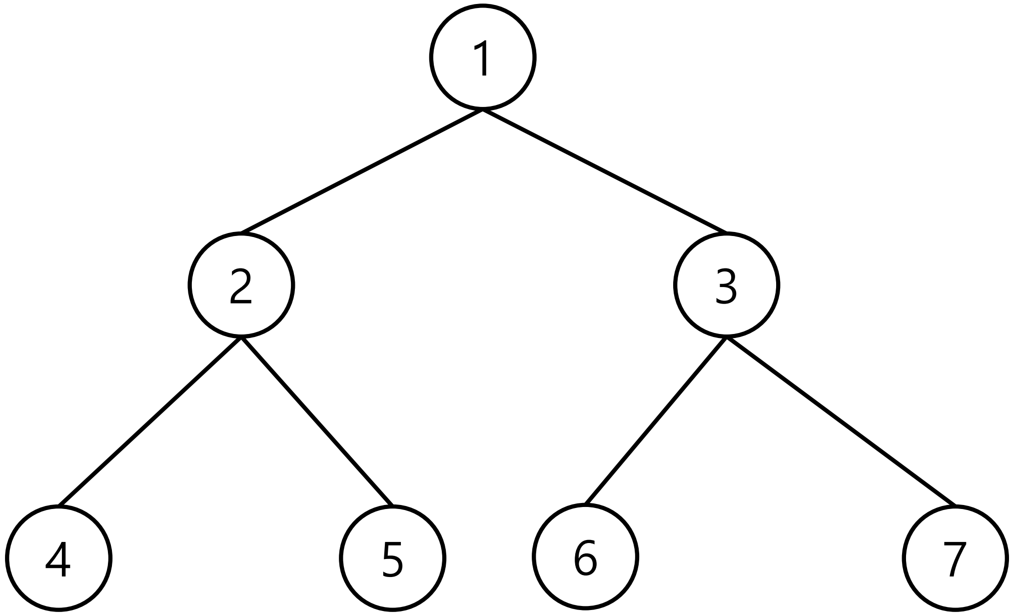
레벨/높이: 3 => 최소: 4개 / 최대 15개

레벨/높이: H => 최소: H+1개 / 최대 $2^{H+1} - 1$개

Binary Tree

Binary Tree - Full Binary Tree (포화 이진 트리)

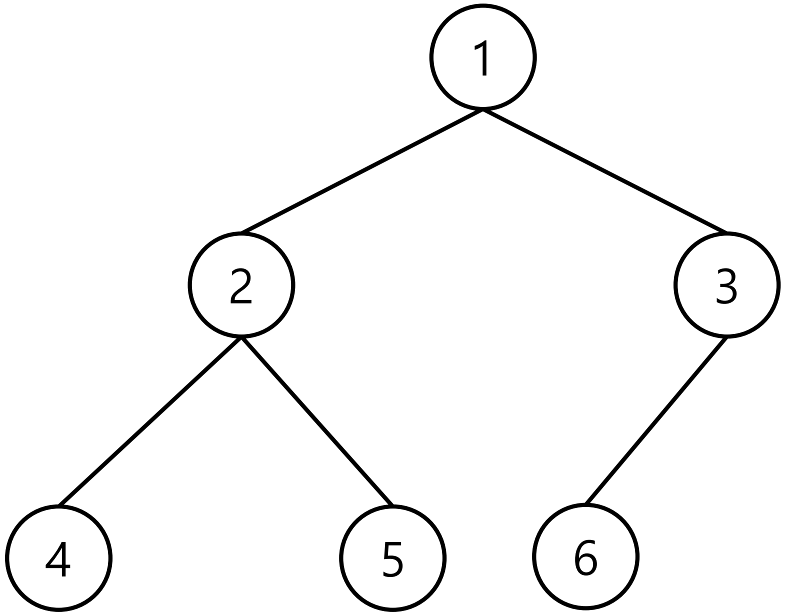
모든 노드가 2개의 노드를 가지고 있는 트리



Q. Full Binary Tree의 노드의 개수는?

Binary Tree - Complete Binary Tree (완전 이진 트리)

높이가 H이고 노드의 개수가 N일 때, 빈 자리 없이 채워지는 트리.

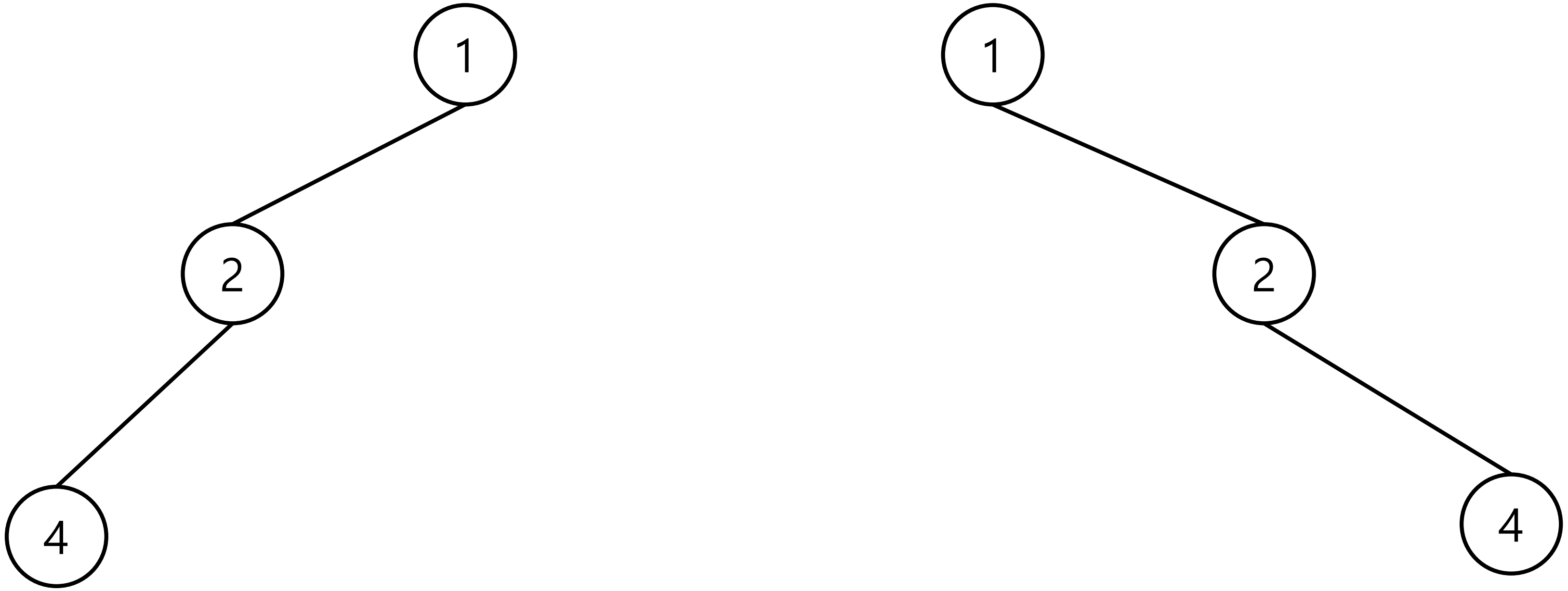


※ 단, 노드의 개수 <= N < 2^{H+1} - 1

Binary Tree

Binary Tree - Skewed Binary Tree (편향 이진 트리)

높이가 H일 때, 최소 노드 개수를 가지면서 한쪽 방향으로 노드는 가지는 트리.



Binary Tree

Binary Tree - Traversal (순회)

Traversal (순회)란 트리의 각 노드를 중복되지 않게 전부 방문하는 것.

비 선형구조로 선형구조처럼 선후 연결 관계를 알 수 없다.

Binary Tree

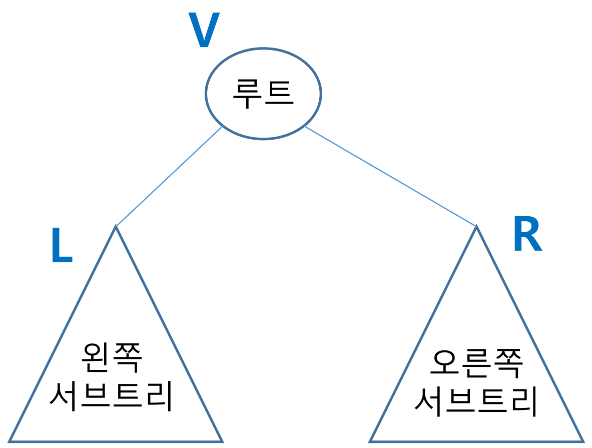
Binary Tree - Traversal (순회)

트리의 순회 방법

- 전위 순회 : 부모 노드 방문 후, 좌/우 순서로 방문.

- 중위 순회 : 왼쪽 자식 노드 방문 후 부모 노드/오른쪽 자식 노드 방문.

- 후위 순회 : 좌/우 자식노드를 방문하고 부모 노드 방문.



Binary Tree - 전위 순회

전위 순회 방법

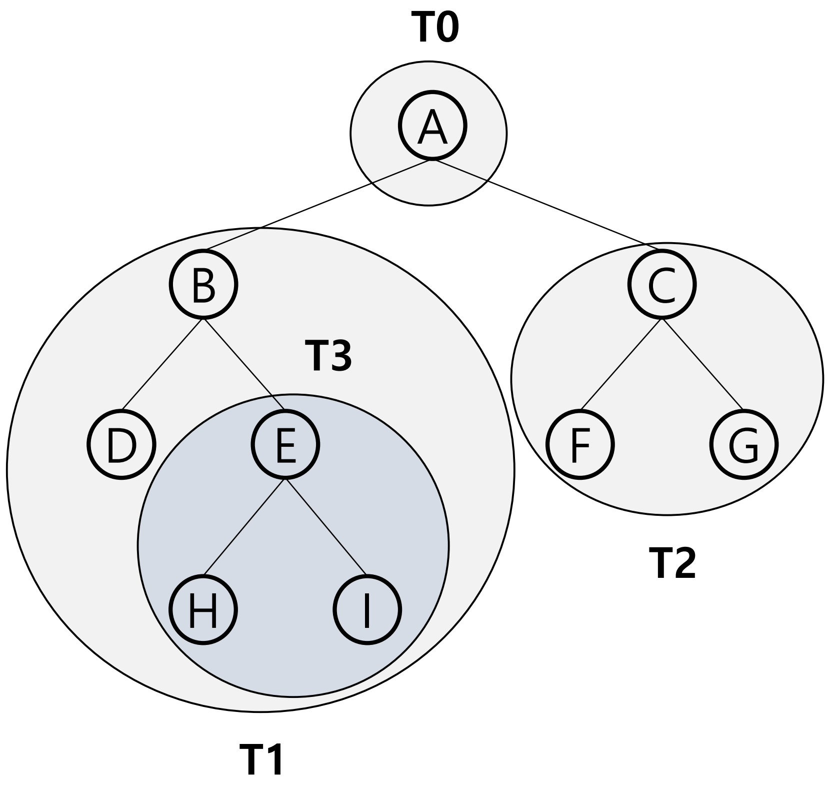
- 현재 노드 n을 방문하여 처리한다. -> V
- 현재 노드 n의 왼쪽 서브 트리로 이동한다. -> L
- 현재 노드 n의 오른쪽 서브 트리로 이동한다. -> R

```
preorder_traverse(T) {
    if (T is not null) {
        visit(T)
        preorder_traverse(T.left)
        preorder_traverse(T.right)
    }
}
```

Binary Tree

Binary Tree - 전위 순회

전위 순회 예시



순서1 : T0 → T1 → T2

순서2 : A → B D (T3) → C F G

총 순서 : A B D E H I C F G

Segment Tree

Segment Tree

Segment Tree 란?

어떤 데이터가 존재할 때, 특정 구간의 결과값을 구하는데 사용하는 자료구조.

Q. 아래와 같이 주어진 배열이 존재할 때, 3~5번째 구간의 합을 구하시오.

인덱스	0	1	2	3	4	5	6	7	8	9
값	1	2	3	4	5	6	7	8	9	10

A. 배열을 돌면서 3~5번째인 경우에 더해주면 되겠다!

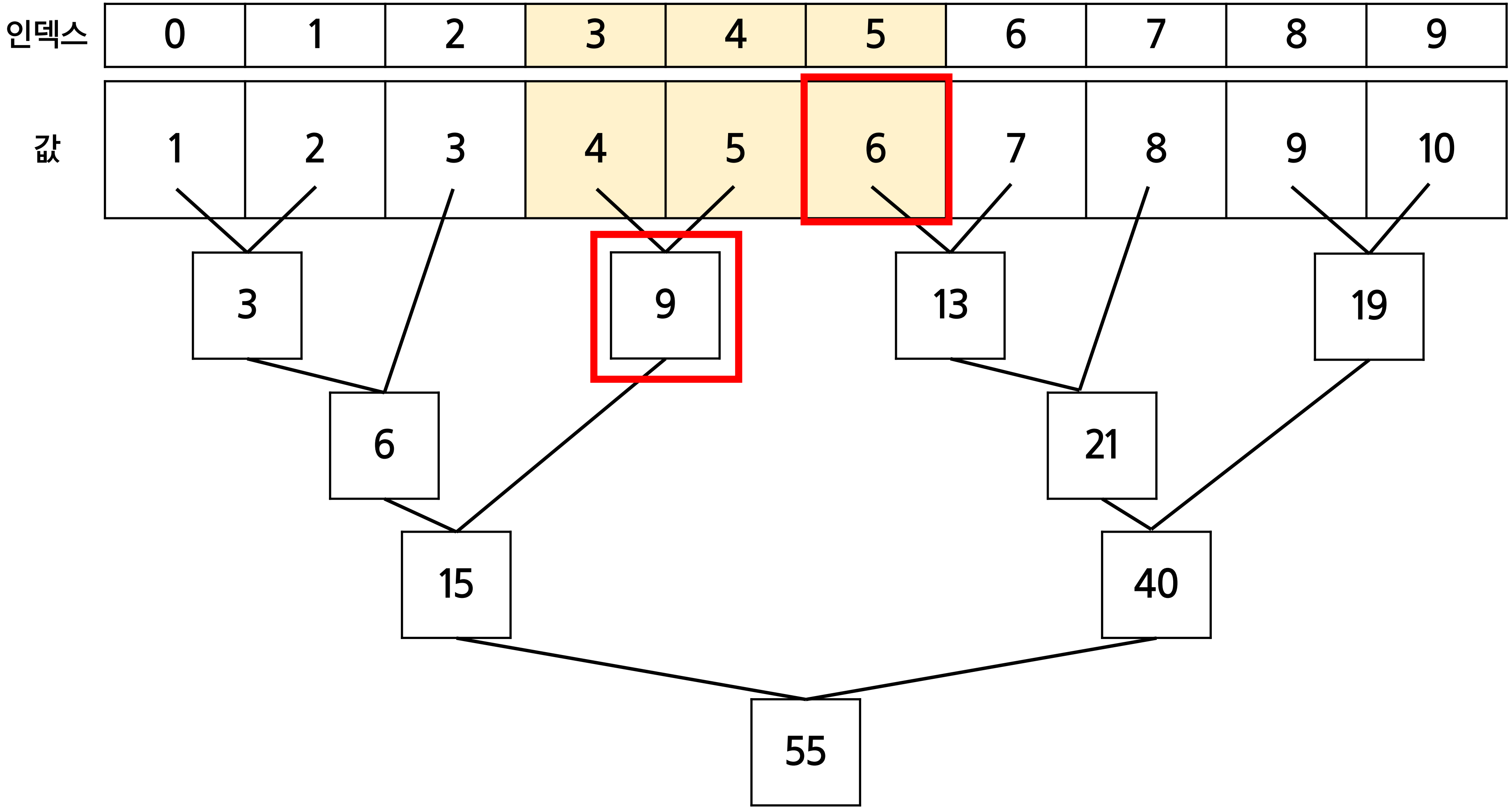
배열의 크기가 커지면 어떻게 해야할까?

구간별 합을 구해 저장해두면 빠르게 찾을 수 있다.

Segment Tree

Segment Tree 란?

Segment Tree는 Binary Tree(이진 트리) 구조를 가지고 있다.



Segment Tree

Segment Tree 란?

인덱스	0	1	2	3	4	5	6	7	8	9
값	1	2	3	4	5	6	7	8	9	10

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15															30
55	15	40	6	9	21	19	3	3	4	5	13	8	9	10	1	2	-	-	-	-	-	-	6	7	-	-	-	-	-	-

0번 인덱스 (55)는 1번 인덱스 (15)와 2번 인덱스 (40)를 더한 값

1번 인덱스 (15)는 3번 인덱스 (6)와 4번 인덱스 (9)를 더한 값

배열의 크기를 1칸만 더 늘려준다면?

Segment Tree

Segment Tree 란?

인덱스	0	1	2	3	4	5	6	7	8	9
값	1	2	3	4	5	6	7	8	9	10

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15															30	31
	55	15	40	6	9	21	19	3	3	4	5	13	8	9	10	1	2	-	-	-	-	-	-	6	7	-	-	-	-	-	-

1번 인덱스 (55)는 2번 인덱스 (15)와 3번 인덱스 (40)를 더한 값

2번 인덱스 (15)는 4번 인덱스 (6)와 5번 인덱스 (9)를 더한 값

왼쪽 노드는 x2 / 오른쪽 노드는 x2 + 1 로 접근 가능!

Segment Tree

Segment Tree 란?

Q. Segment Tree를 구성하는데 필요한 배열의 크기는 어떻게 구할까?

A. Full Binary Tree라고 생각해보자.

Full Binary Tree 의 경우 최대 노드 개수는 $2^{H+1} - 1$ 개 => 2^{H+1}

Ceil(Log2(N)) -> H

Segment Tree

Segment Tree - 재귀

left

0	1	2	3	4	5	6	7	8	9
1	2	3	4	5	6	7	8	9	10

Start

Mid

End

index

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Segment Tree

Segment Tree - 재귀

left	0	1	2	3	4	5	6	7	8	9
	1	2	3	4	5	6	7	8	9	10

Start Mid End

index

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Segment Tree

Segment Tree - 재귀

left	0	1	2	3	4	5	6	7	8	9
	1	2	3	4	5	6	7	8	9	10

Start Mid End

index

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Segment Tree

Segment Tree - 재귀

left	0	1	2	3	4	5	6	7	8	9
	1	2	3	4	5	6	7	8	9	10

Start Mid End

index

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Segment Tree

Segment Tree - 재귀

left	0	1	2	3	4	5	6	7	8	9
	1	2	3	4	5	6	7	8	9	10

Start End

Mid

index

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Segment Tree - 재귀

right	0	1	2	3	4	5	6	7	8	9
	1	2	3	4	5	6	7	8	9	10

Start End

index

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Segment Tree - 재귀

sum	0	1	2	3	4	5	6	7	8	9
	1	2	3	4	5	6	7	8	9	10

Start End

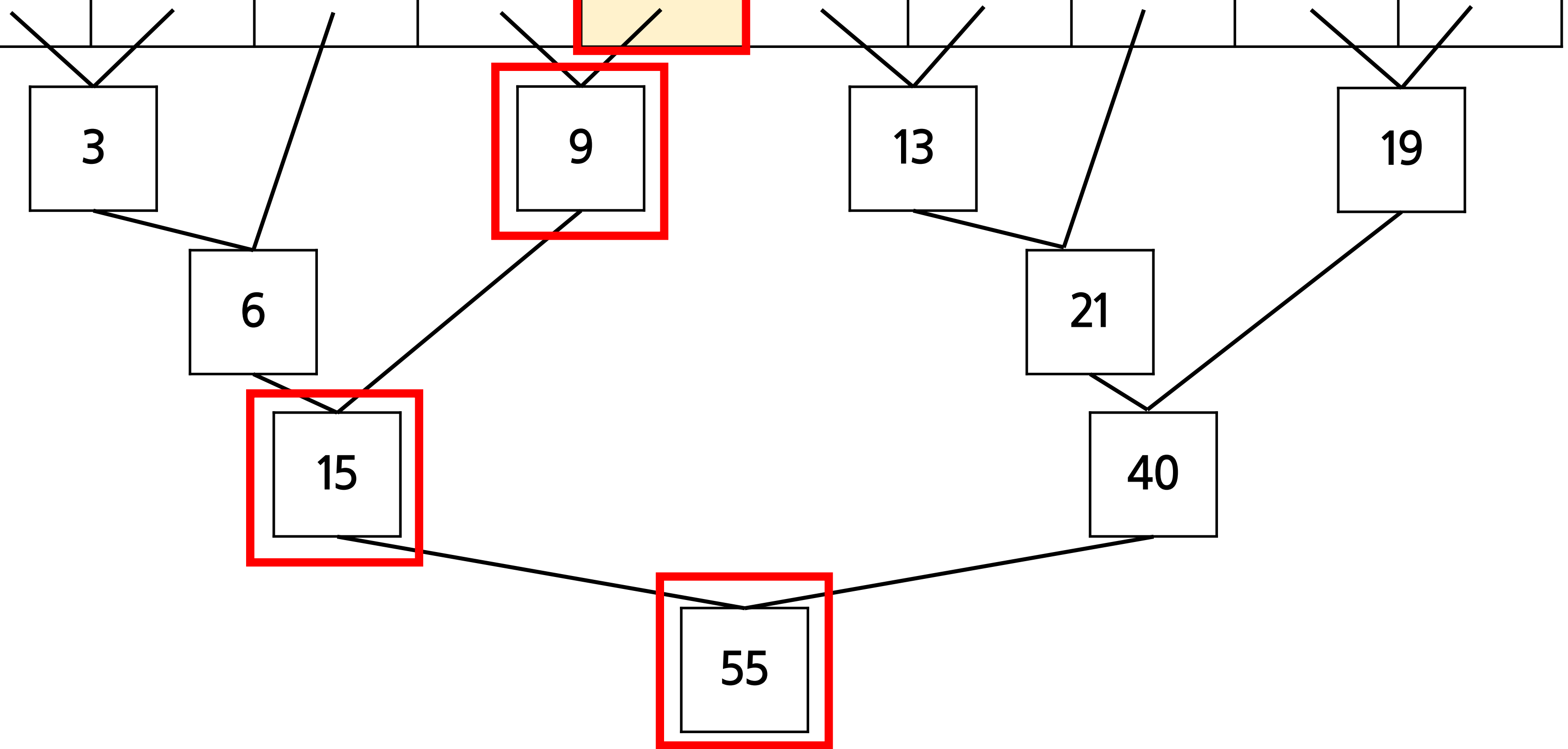
index

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Segment Tree
업데이트

Segment Tree - 업데이트

인덱스	0	1	2	3	4	5	6	7	8	9
값	1	2	3	4	7	6	7	8	9	10



Segment Tree - 업데이트

Update_index = 4

start	0	1	2	3	4	5	6	7	8	9
	1	2	3	4	5 -> 7	6	7	8	9	10

Start

End

index

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	55	15	40	6	9	21	19	3	3	4	5	13	8	9	10	1	2	-	-	-	-	-	-	6	7	-	-	-	-	-	-

Segment Tree - 업데이트

Update_index = 4

left	0	1	2	3	4	5	6	7	8	9
	1	2	3	4	5 -> 7	6	7	8	9	10

Start

End

index

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	55	15	40	6	9	21	19	3	3	4	5	13	8	9	10	1	2	-	-	-	-	-	-	6	7	-	-	-	-	-	-

Segment Tree - 업데이트

Update_index = 4

left	0	1	2	3	4	5	6	7	8	9
	1	2	3	4	5 -> 7	6	7	8	9	10

Start (0) End (2)

index (4)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	55	15	40	6	9	21	19	3	3	4	5	13	8	9	10	1	2	-	-	-	-	-	-	6	7	-	-	-	-	-	-

Segment Tree - 업데이트

Update_index = 4

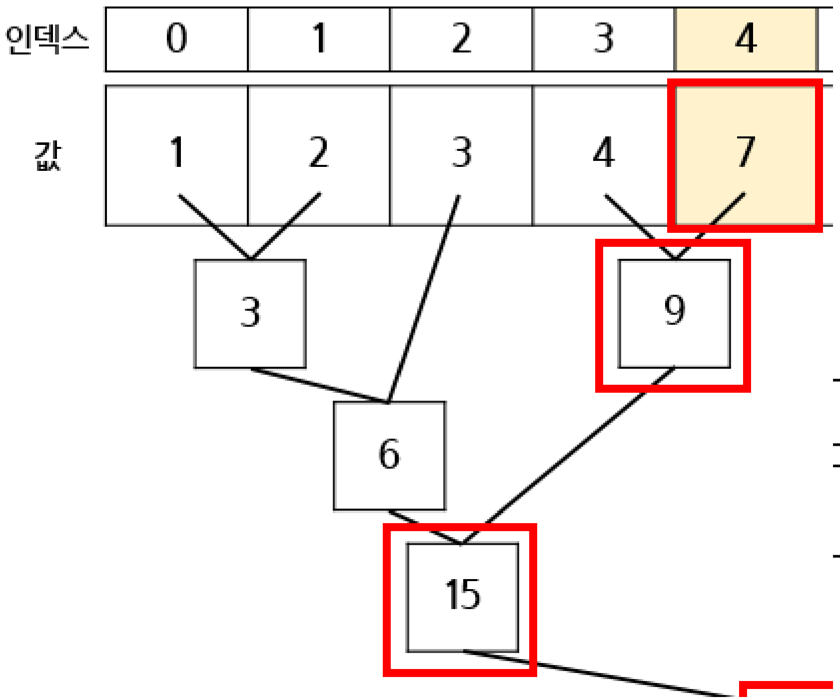
left	0	1	2	3	4	5	6	7	8	9
	1	2	3	4	5 -> 7	6	7	8	9	10

Start

End

index

0	1	2	3	4	5	6	7	8	9	10	11	12	1:
0	55	15	40	6	9	21	19	3	3	4	5	13	8



25	26	27	28	29	30	31
7	-	-	-	-	-	-

Segment Tree - 업데이트

Update_index = 4

right	0	1	2	3	4	5	6	7	8	9
	1	2	3	4	5 -> 7	6	7	8	9	10

Start

End

index

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	55	15	40	6	9	21	19	3	3	4	5	13	8	9	10	1	2	-	-	-	-	-	-	6	7	-	-	-	-	-	-

Segment Tree - 업데이트

Update_index = 4

left	0	1	2	3	4	5	6	7	8	9
	1	2	3	4	5 -> 7	6	7	8	9	10

Start End

index

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	55	15	40	6	9	21	19	3	3	4	5	13	8	9	10	1	2	-	-	-	-	-	-	6	7	-	-	-	-	-	-

Segment Tree - 업데이트

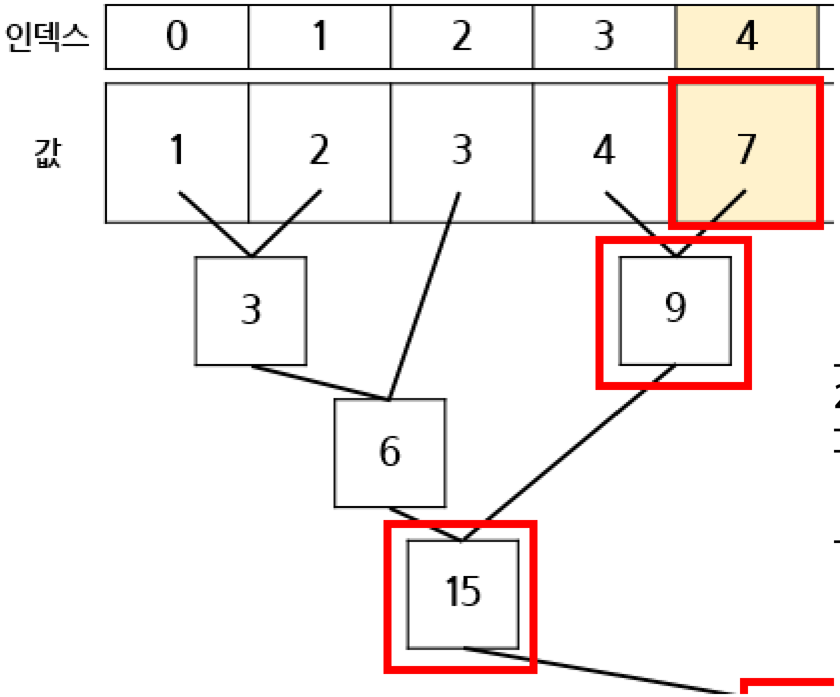
Update_index = 4

left	0	1	2	3	4	5	6	7	8	9
	1	2	3	4	5 -> 7	6	7	8	9	10

Start End

index

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	55	15	40	6	9	21	19	3	3	4	5	13	8	9



26	27	28	29	30	31
-	-	-	-	-	-

Segment Tree

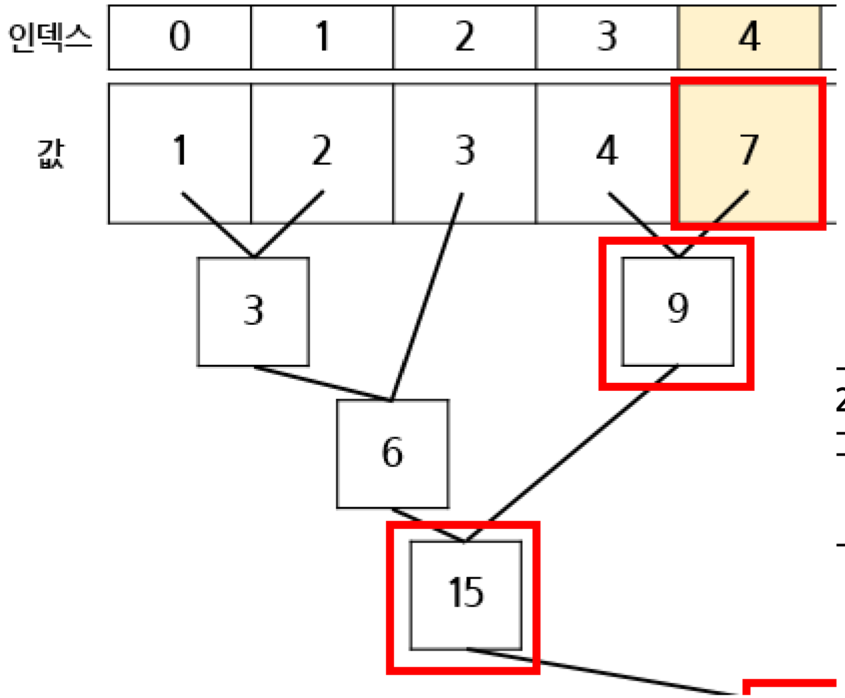
Segment Tree - 업데이트

Update_index = 4

right	0	1	2	3	4	5	6	7	8	9
	1	2	3	4	5 -> 7	6	7	8	9	10

Start End

업데이트 가능!



index

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
0	55	15	40	6	9	21	19	3	3	4	7	13	8	9	10	1

28	29	30	31
-	-	-	-

Segment Tree - 업데이트

Update_index = 4

right									
0	1	2	3	4	5	6	7	8	9
1	2	3	4	5 -> 7	6	7	8	9	10

Start End

자식 노드가 업데이트 되었으니,

부모 노드도 업데이트!

index

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	55	15	40	6	11	21	19	3	3	4	7	13	8	9	10	1	2	-	-	-	-	-	-	6	7	-	-	-	-	-	-

Segment Tree - 업데이트

Update_index = 4

left	0	1	2	3	4	5	6	7	8	9
	1	2	3	4	5 -> 7	6	7	8	9	10

Start

End

index

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	55	17	40	6	11	21	19	3	3	4	7	13	8	9	10	1	2	-	-	-	-	-	-	6	7	-	-	-	-	-	-

Segment Tree - 업데이트

Update_index = 4

right	0	1	2	3	4	5	6	7	8	9
	1	2	3	4	5 -> 7	6	7	8	9	10

Start (index 5)   End (index 9)

index (3)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	55	17	40	6	11	21	19	3	3	4	7	13	8	9	10	1	2	-	-	-	-	-	-	6	7	-	-	-	-	-	-

Segment Tree

Segment Tree - 업데이트

Update_index = 4

right	0	1	2	3	4	5	6	7	8	9
	1	2	3	4	5 -> 7	6	7	8	9	10

Start

End

index

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	57	17	40	6	11	21	19	3	3	4	7	13	8	9	10	1	2	-	-	-	-	-	-	6	7	-	-	-	-	-	-

Fenwick Tree

Fenwick Tree

Fenwick Tree 란?

Segment Tree 처럼 구간에 대한 연산을 저장하는 트리.

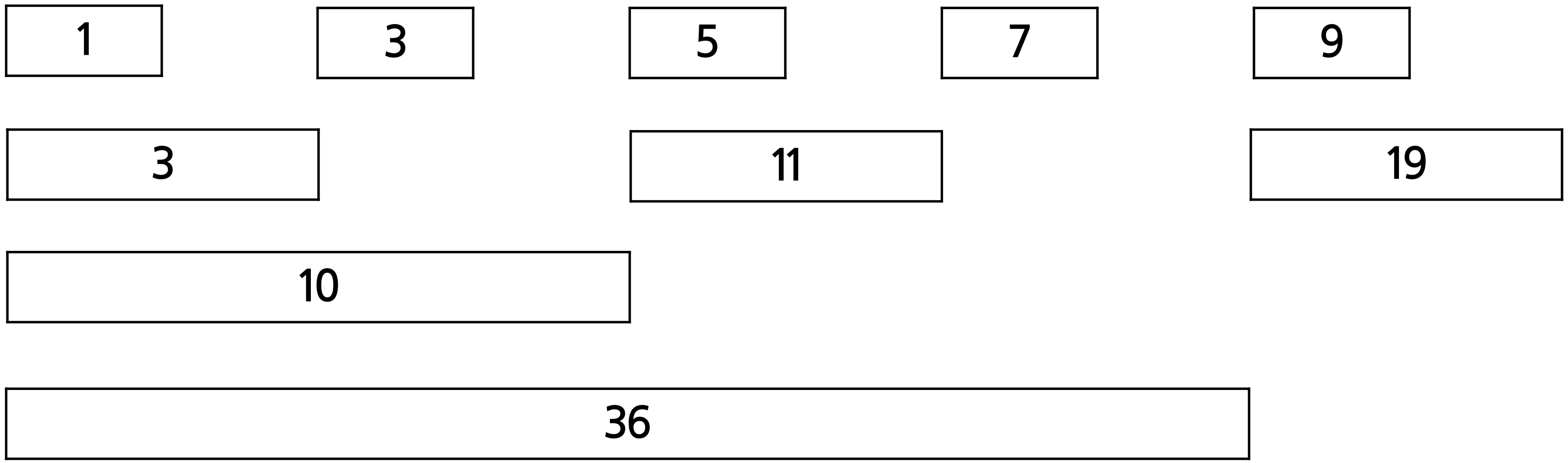
Segment Tree 보다 적은 메모리로 사용가능하다.

비트를 이용한 구간 연산을 진행.

비트는 0과 1만 사용되는 만큼 **1**의 의미가 중요하다.

Fenwick Tree 란?

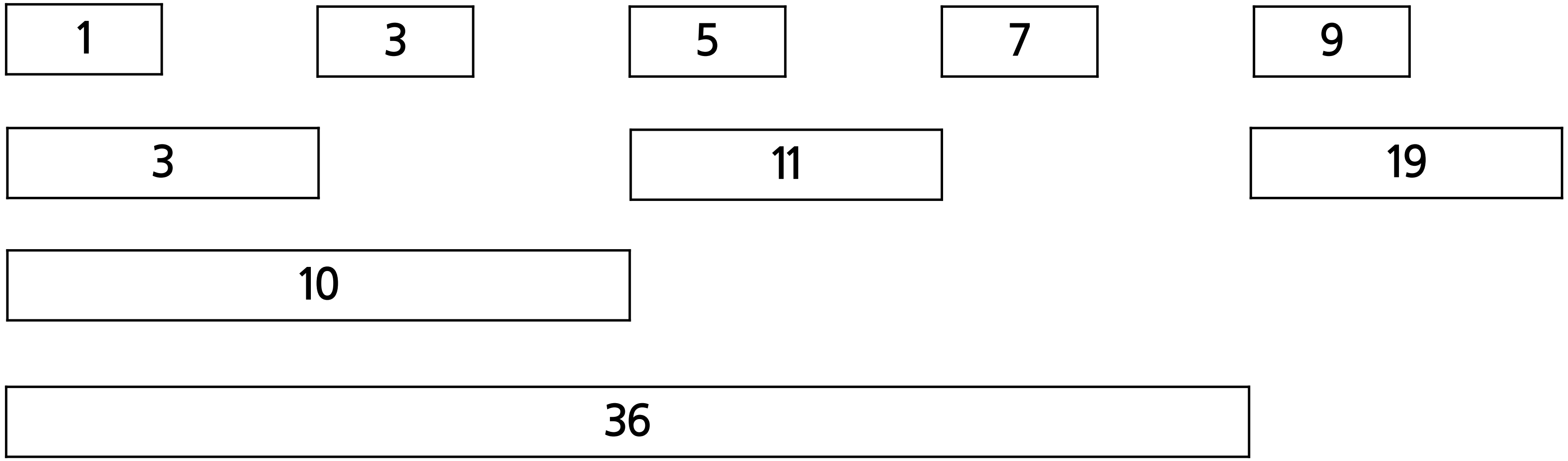
인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010



Fenwick Tree

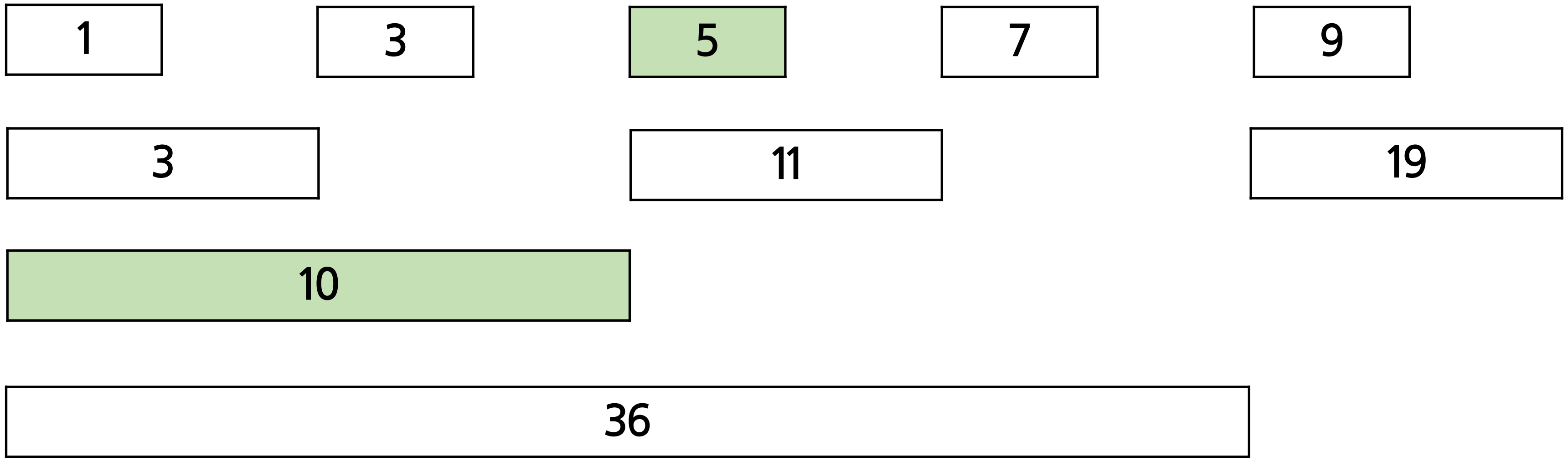
Fenwick Tree 란?

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010



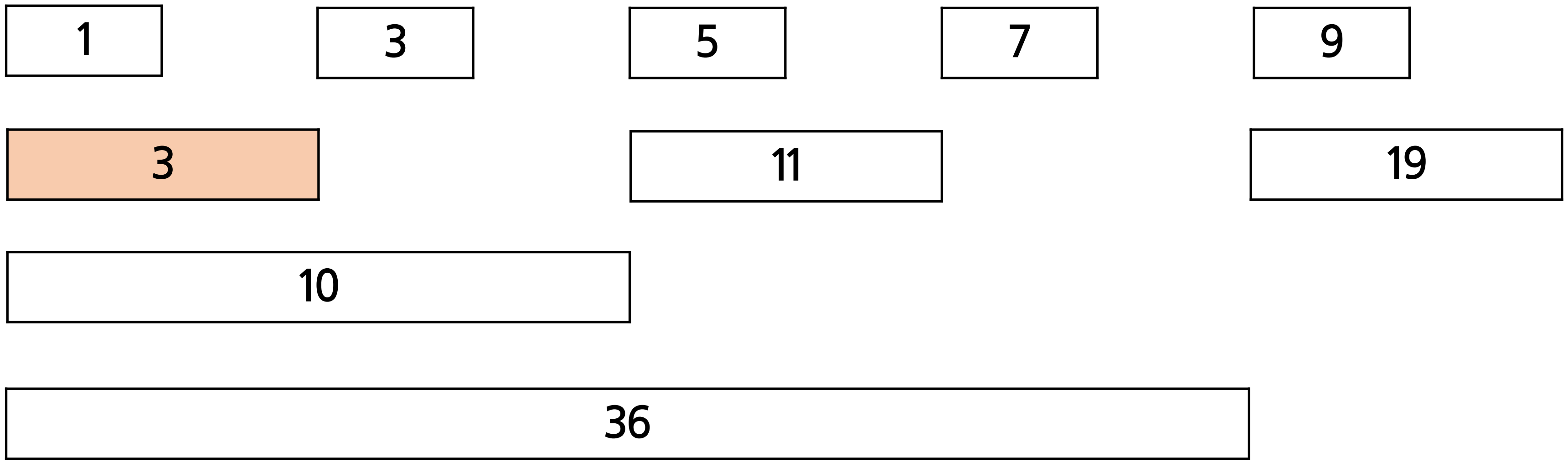
Fenwick Tree 란?

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010



Fenwick Tree 란?

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

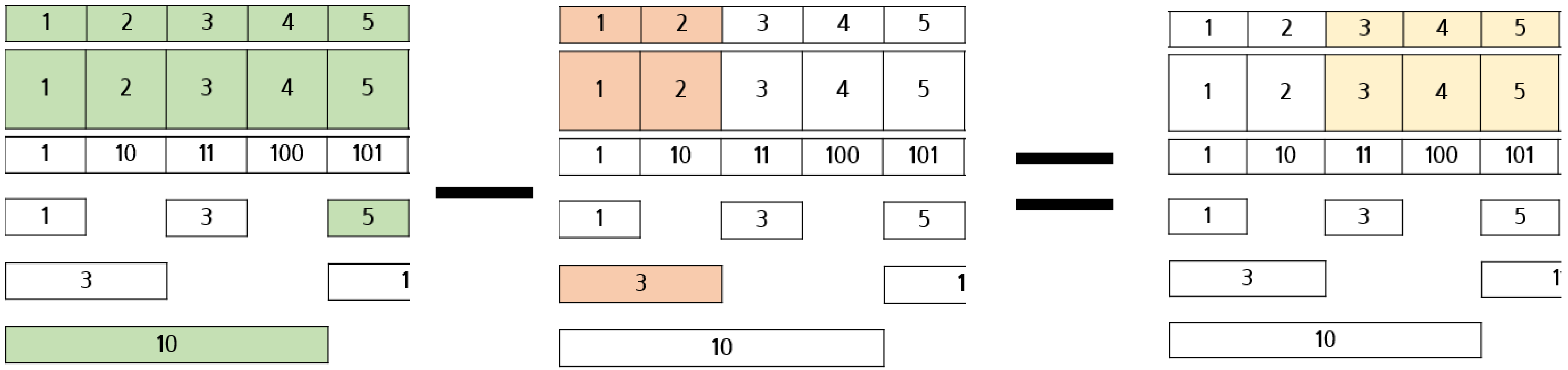


Fenwick Tree

Fenwick Tree 란?

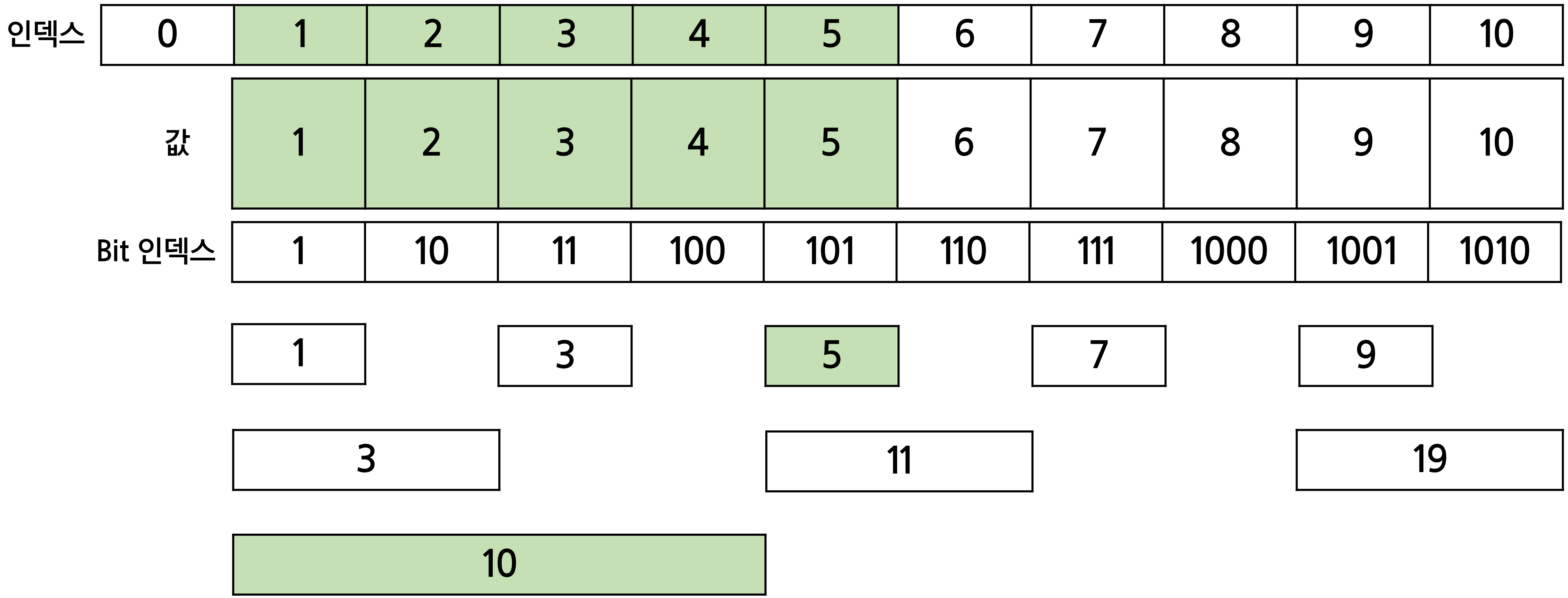
Q. 3 ~ 5번째 인덱스 구간 합을 어떻게 구할까?

A. 두 구간의 차를 구하면 된다.



Fenwick Tree

Fenwick Tree 란?



5번 인덱스까지의 값을 구하려면 2진수 101 인덱스 값과 100 인덱스 값을 더하면 된다.

101인덱스에서 마지막 1의 값을 제거해주면 100이 된다.

Fenwick Tree

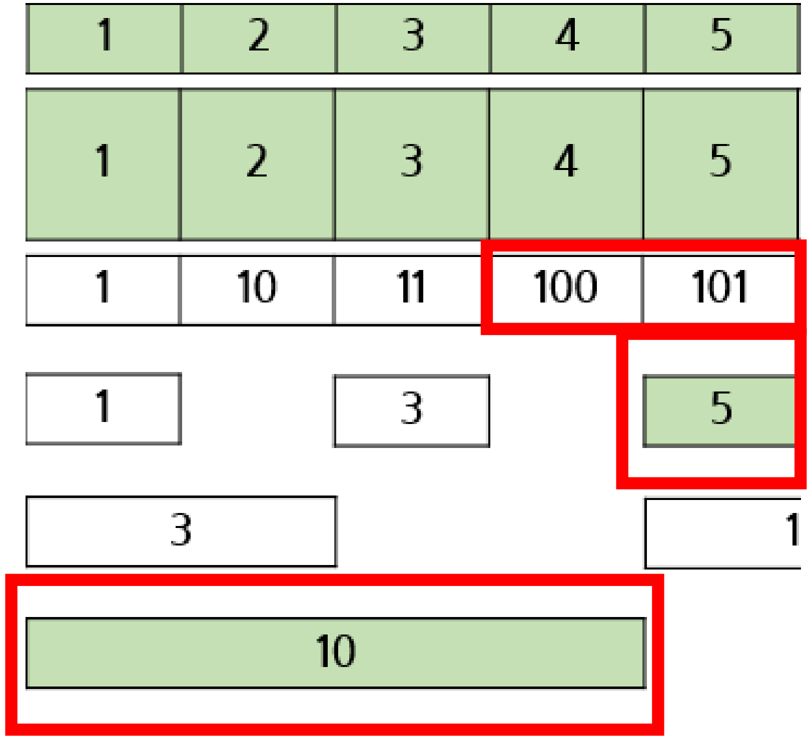
Fenwick Tree 란?

Q. 마지막 2진수 1의 값을 제거하는 방법은?

A. $N = N - (N \,\&\, -N)$

$5 - (5 \,\&\, -5) = 4$

$4 - (4 \,\&\, -4) = 0$



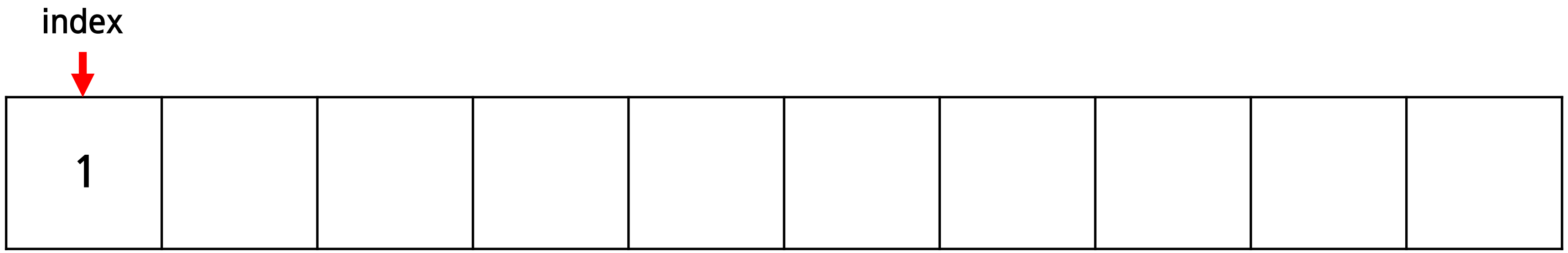
Fenwick Tree

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 1

index = 1 -> 1 += (1 & -1) = 2



Fenwick Tree

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 1

index = 2 -> 2 += (2 & -2) = 4

	index								
1	1								

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 1

index = 4 -> 4 += (4 & -4) = 8

index

1	1		1						

Fenwick Tree

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 1

index = 8 -> 8 += (8 & -8) = 16 트리의 길이보다 index의 값이 크기 때문에 중지.

index

1	1		1				1		

Fenwick Tree

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 2

index = 2 -> 2 += (2 & -2) = 4

index

1	3		1				1		

Fenwick Tree

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 2

index = 4 -> 4 += (4 & -4) = 8

index

1	3		3				1		

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 2

index = 8 -> 8 += (8 & -8) = 16 트리의 길이보다 index의 값이 크기 때문에 중지.

index

1	3		3				3		

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 3

index = 3 -> 3 += (3 & -3) = 4

index

1	3	3	3				3		

Fenwick Tree

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 3

index = 4 -> 4 += (4 & -4) = 8

index

1	3	3	6				3		

Fenwick Tree

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 3

index = 8 -> 8 += (8 & -8) = 16   트리의 길이보다 index의 값이 크기 때문에 중지.

index

1	3	3	6				6		

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 4

index = 4 -> 4 += (4 & -4) = 8

index

1	3	3	10				6		

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 4

index = 8 -> 8 += (8 & -8) = 16   트리의 길이보다 index의 값이 크기 때문에 중지.

index

1	3	3	10				10		

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 5

index = 5 -> 5 += (5 & -5) = 6

index

1	3	3	10	5			10		

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 5

index = 6 -> 6 += (6 & -6) = 8

index

1	3	3	10	5	5		10		

Fenwick Tree

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 5

index = 8 -> 8 += (8 & -8) = 16　　트리의 길이보다 index의 값이 크기 때문에 중지.

index

1	3	3	10	5	5		15		

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 6

index = 6 -> 6 += (6 & -6) = 8

index

1	3	3	10	5	11		15		

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 6

index = 8 -> 8 += (8 & -8) = 16   트리의 길이보다 index의 값이 크기 때문에 중지.

							index ↓		
1	3	3	10	5	11		21		

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 7

index = 7 -> 7 += (7 & -7) = 8

index

1	3	3	10	5	11	7	21		

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 7

index = 8 -> 8 += (8 & -8) = 16 트리의 길이보다 index의 값이 크기 때문에 중지.

							index		
1	3	3	10	5	11	7	28		

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 8

index = 8 -> 8 += (8 & -8) = 16 트리의 길이보다 index의 값이 크기 때문에 중지.

index

1	3	3	10	5	11	7	36		

Fenwick Tree

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 9

index = 9 -> 9 += (9 & -9) = 10

								index	
1	3	3	10	5	11	7	36	9	

Fenwick Tree

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 9

index = 10 -> 10 += (10 & -10) = 12 트리의 길이보다 index의 값이 크기 때문에 중지.

index ↓

1	3	3	10	5	11	7	36	9	9

Fenwick Tree - 생성

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010

Val = 10

index = 10 -> 10 += (10 & -10) = 12 트리의 길이보다 index의 값이 크기 때문에 중지.

index

1	3	3	10	5	11	7	36	9	19

Fenwick Tree

Fenwick Tree - 생성

<table>
<tr><td>인덱스</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td></tr>
<tr><td>값</td><td></td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td></tr>
<tr><td>Bit 인덱스</td><td></td><td>1</td><td>10</td><td>11</td><td>100</td><td>101</td><td>110</td><td>111</td><td>1000</td><td>1001</td><td>1010</td></tr>
<tr><td></td><td></td><td>1</td><td></td><td>3</td><td></td><td>5</td><td></td><td>7</td><td></td><td>9</td><td></td></tr>
<tr><td></td><td></td><td colspan="2">3</td><td></td><td></td><td colspan="2">11</td><td></td><td></td><td colspan="2">19</td></tr>
<tr><td></td><td></td><td colspan="4">10</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td colspan="8">36</td><td></td><td></td></tr>
<tr><td></td><td></td><td>1</td><td>3</td><td>3</td><td>10</td><td>5</td><td>11</td><td>7</td><td>36</td><td>9</td><td>19</td></tr>
</table>

Segment Tree - 업데이트

주어진 배열의 구간 합을 생성해 두었는데….

데이터가 변경이 되었다면?

인덱스	0	1	2	3	4	5	6	7	8	9
값	1	2	3	4	5 -> 7	6	7	8	9	10

다시 Segment Tree를 생성할까?

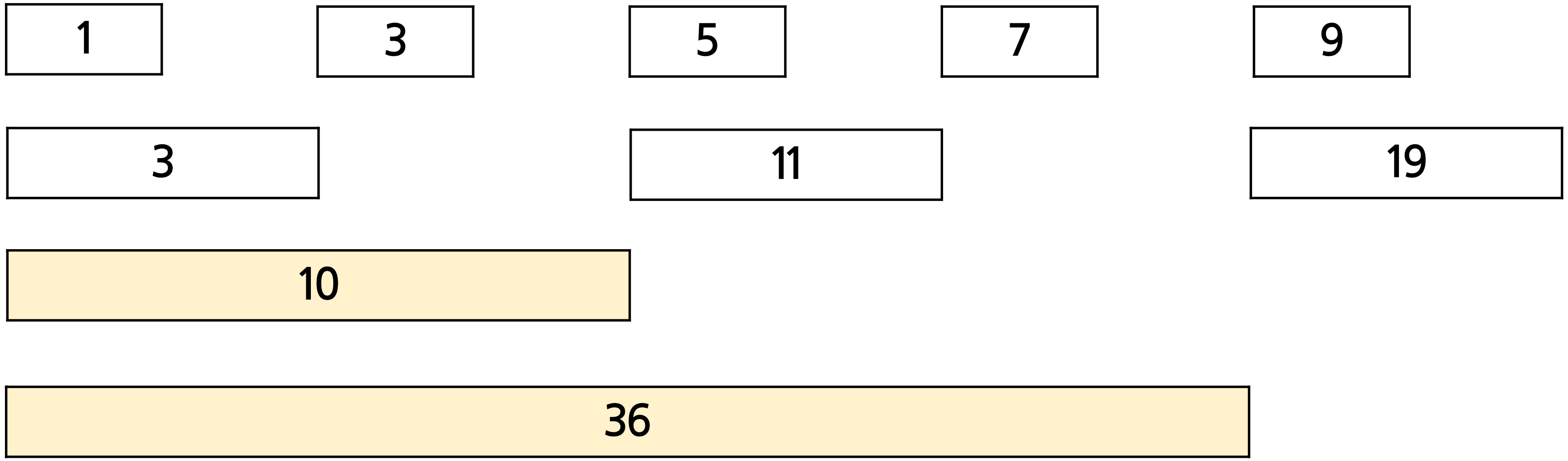
특정 구간만 수정할 수 없을까?

Fenwick Tree
업데이트

Fenwick Tree

Fenwick Tree - 업데이트

인덱스	0	1	2	3	4	5	6	7	8	9	10
값		1	2	3	4	5	6	7	8	9	10
Bit 인덱스		1	10	11	100	101	110	111	1000	1001	1010



1	3	3	10	5	11	7	36	9	19

Fenwick Tree - 업데이트

인덱스	0	1	2	3	4
값		1	2	3	4->7 +3
Bit 인덱스		1	10	11	100

1 3

3

13 +3

39 +3

1	3	3	13	5	11	7	39	9	19

4번 인덱스의 값을 7로 변경한다면?

4번 인덱스는 비트 인덱스로 100

Fenwick Tree를 생성할 때 사용한 연산 그대로 사용.

100 -> 1000 -> 10000

Fenwick Tree를 처음 생성할 때 사용하는 방식과 동일.

Segment Tree / Fenwick Tree

Segment Tree / Fenwick Tree

- 구간 합
- 구간 최댓값
- 구간 최대공약수

SAMSUNG
SW
ACADEMY
FOR
YOUTH