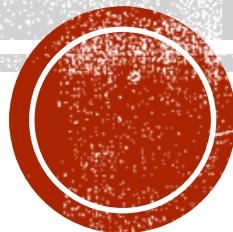


INDEXED TREE

SCCC 서영선 sys7961(nein)
sys7961@gmail.com



INDEXED TREE 란?

- 일단 시작하기에 앞서 indexed tree란 말은 외국에서 사용되지 않는 말임을 언급한다. 검색을 하면 binary indexed tree란 것이 나올텐데, 이는 소개할 자료구조와 다른 내용이다. bottom up segment tree란 말이 그나마 어울리는 말이다. 하지만 국내에 많은 사람들이 인덱스 트리라고 많이 사용하기에 이런 이름을 사용한다. (출처는 아무도 모르지만 KOI쪽에서 나온게 아닐까 추측됨)
- 이 자료구조는 segment tree를 perfect binary tree에서 구현하며 재귀적인 방법이 아닌 반복적인 방법으로 구현한다. 구현 방식에 따라 range/single query, range/single update 를 $O(\lg N)$ 만에 처리할 수 있다.
- segment tree란 정점들이 수열과 같은 것에 대해서 특정 구간의 대표값을 가지고 있는 자료구조이다. 보통 자식들의 구간을 포함한 구간의 대표값을 가진다. segment tree는 이 수열의 특정 구간에 대해서 값을 가져오거나 수정할 수 있다.



INDEXED TREE 구현 방식

- perfect binary tree는 일반적으로 배열을 이용하여 많이 구현한다.
- 루트를 1으로 구현하고 현재 node를 i라고 가정할 때,
 - 왼쪽 자식은 i^*2 ,
 - 오른쪽 자식은 i^*2+1 이다.
- 식을 통하여 배열을 tree로 생각하여 반복적으로 구현한다.



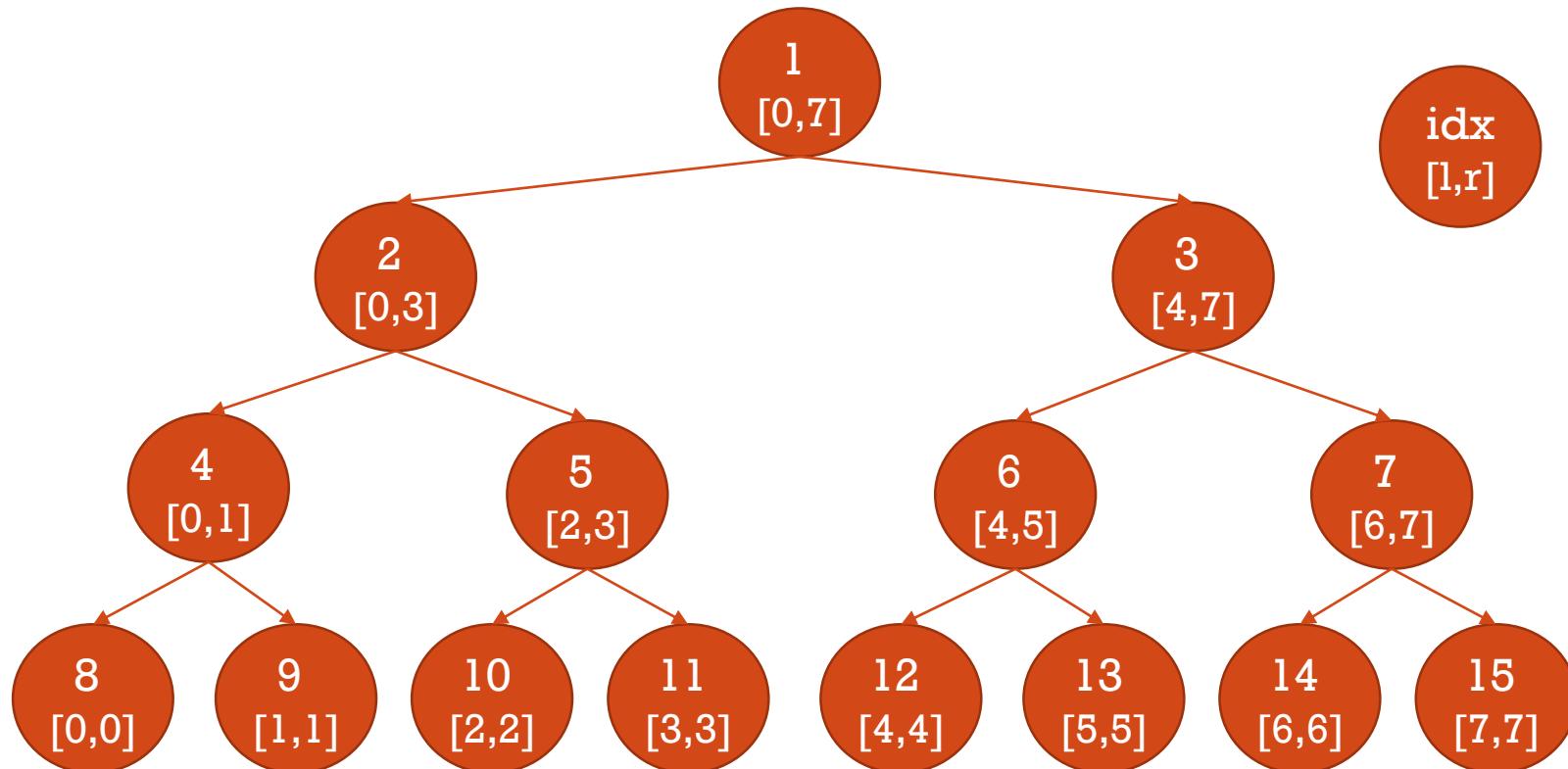
INDEXED TREE 구현 방식

실제 배열

0	1	2	3	4	5	6	7
---	---	---	---	---	---	---	---

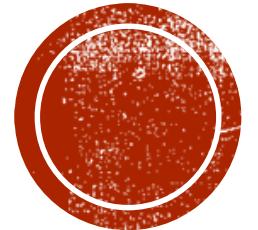
idx
[l,r]

idx : 배열 번호
[l,r] : 커버하는 구간



1 [0,7]	2 [0,3]	3 [4,7]	4 [0,1]	5 [2,3]	6 [4,5]	7 [6,7]	8 [0,0]	9 [1,1]	10 [2,2]	11 [3,3]	12 [4,4]	13 [5,5]	14 [6,6]	15 [7,7]
------------	------------	------------	------------	------------	------------	------------	------------	------------	-------------	-------------	-------------	-------------	-------------	-------------





RANGE QUERY/SINGLE UPDATE



RANGE QUERY / SINGLE UPDATE

- 구간 합을 구하는 sum tree에 대해서 Range Query / Single Update 를 구현하는 경우, 각 구간을 뜻하는 node가 그 구간의 합을 가지게 구현한다.
- Single Update는 leaf node부터 root까지 부모를 올라가며 값을 변경한다.
- Range Query는 $[l,r]$ 의 값을 구한다 할 때, 그 구간을 모두 포함하는 가장 상위 구간들을 더한다.(e.g. $[1,7] = [1,1] + [2,3] + [4,7]$)



SINGLE UPDATE

- RANGE QUERY/SINGLE UPDATE

- Single Update는 다음과 같은 알고리즘으로 작동한다.
 1. leaf node를 구한다.
 2. 그 위치에 값을 더한다.
 3. 부모로 올라간다.
 4. root까지 2~3을 반복.



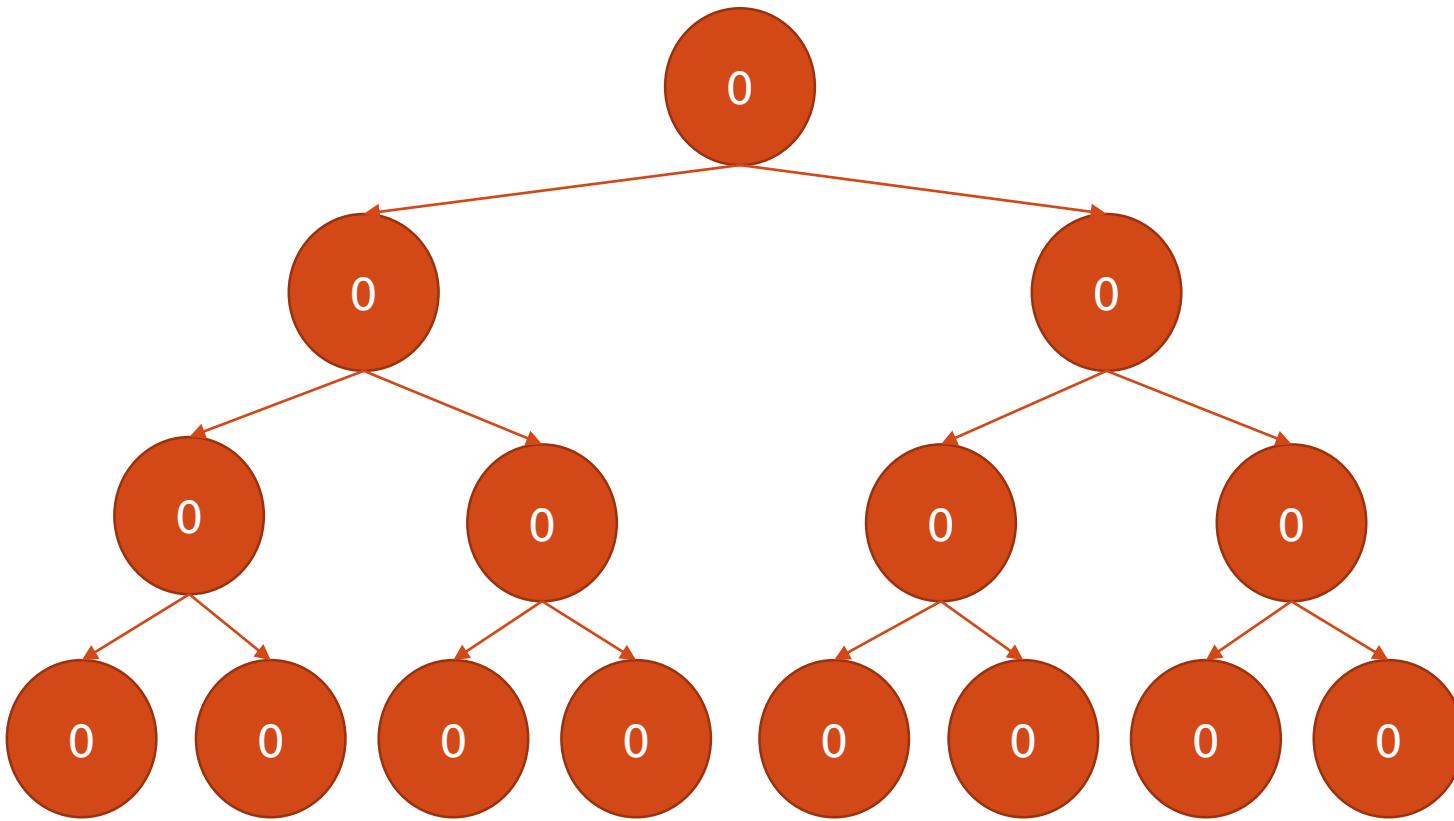
실제 배열

0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---

SINGLE UPDATE

- RANGE QUERY/SINGLE UPDATE

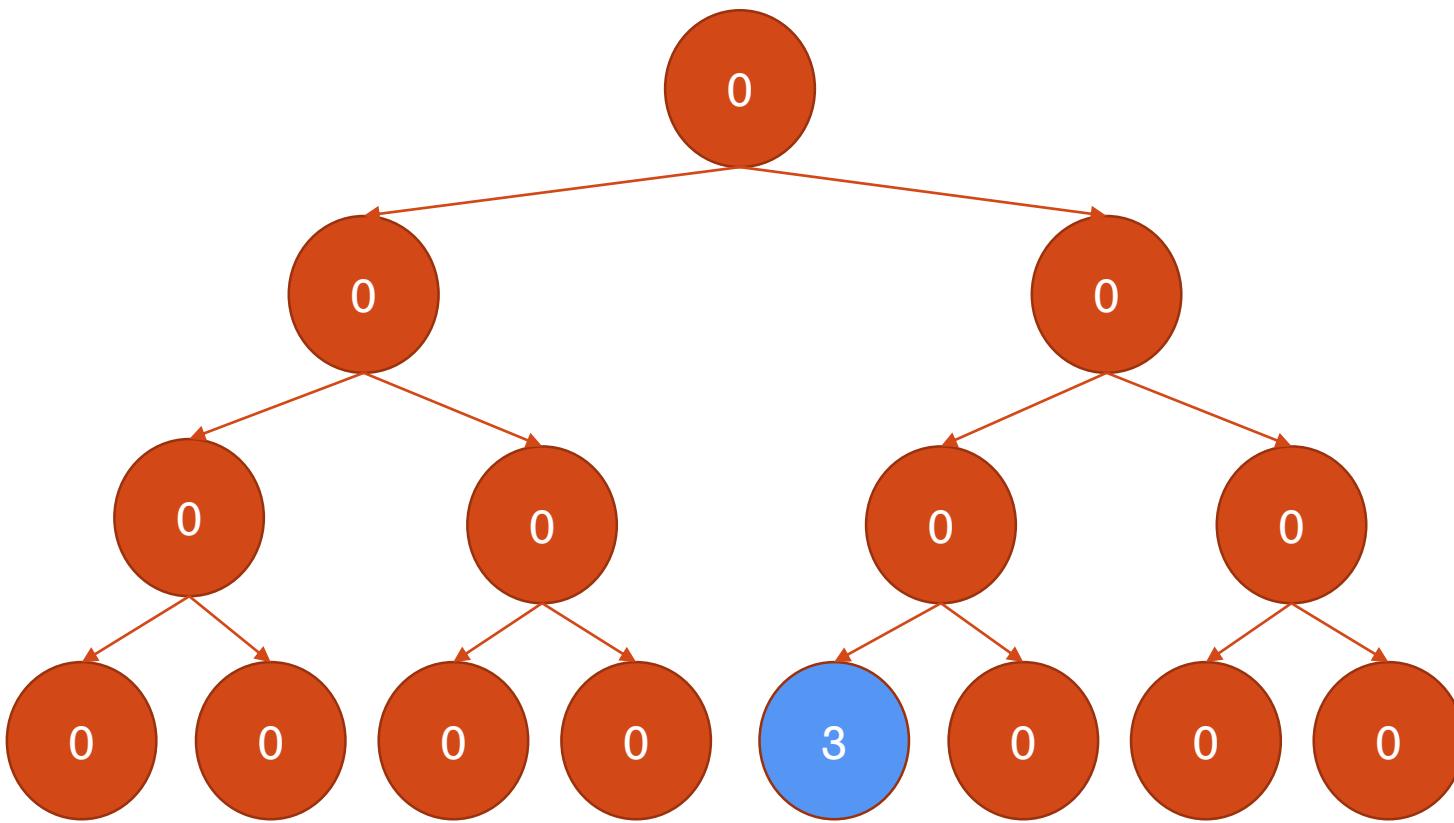
- 4번째 원소에 3을 더한다.



SINGLE UPDATE

- RANGE QUERY/SINGLE UPDATE

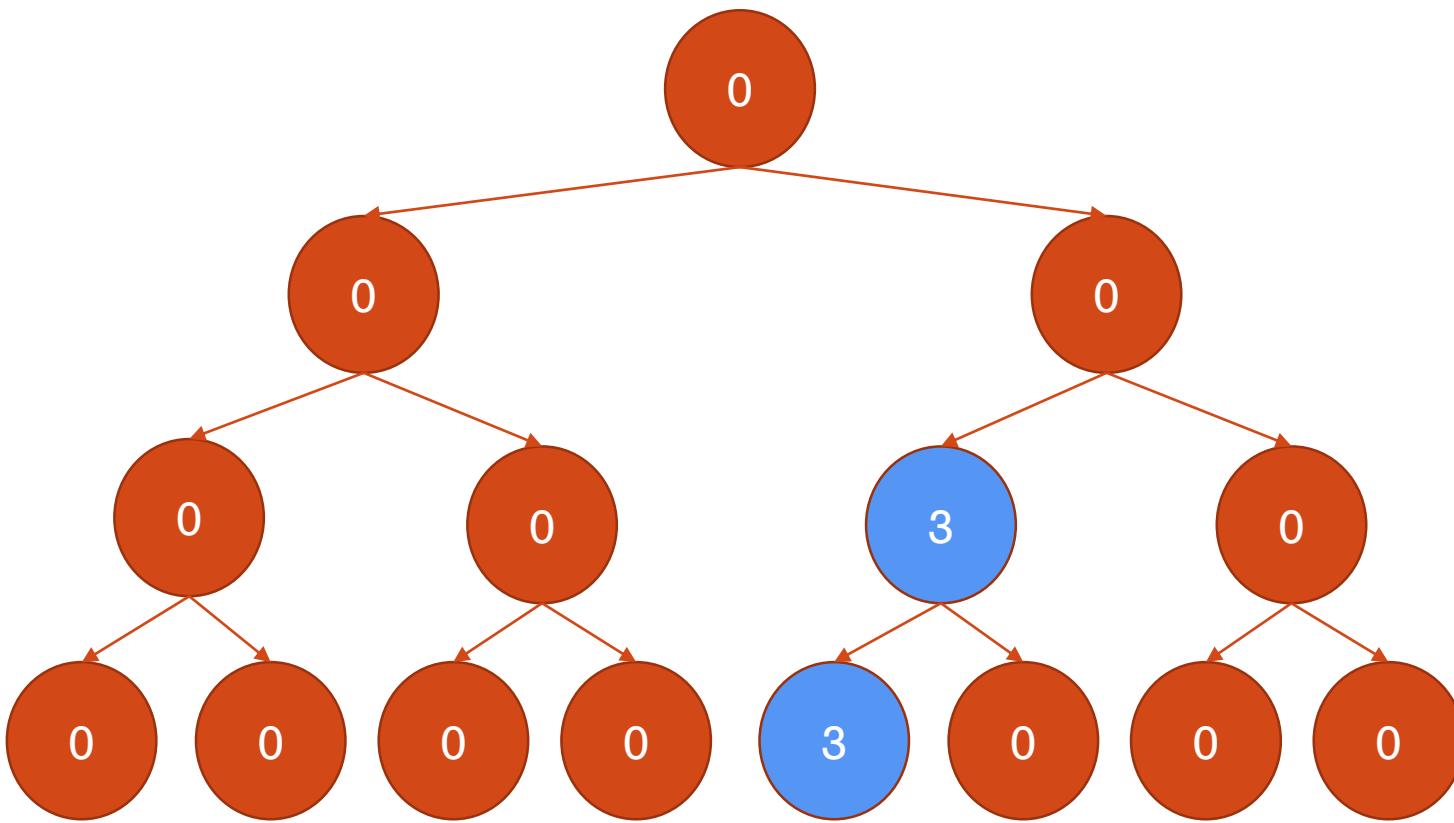
- 4번째 원소에 3을 더한다.



SINGLE UPDATE

- RANGE QUERY/SINGLE UPDATE

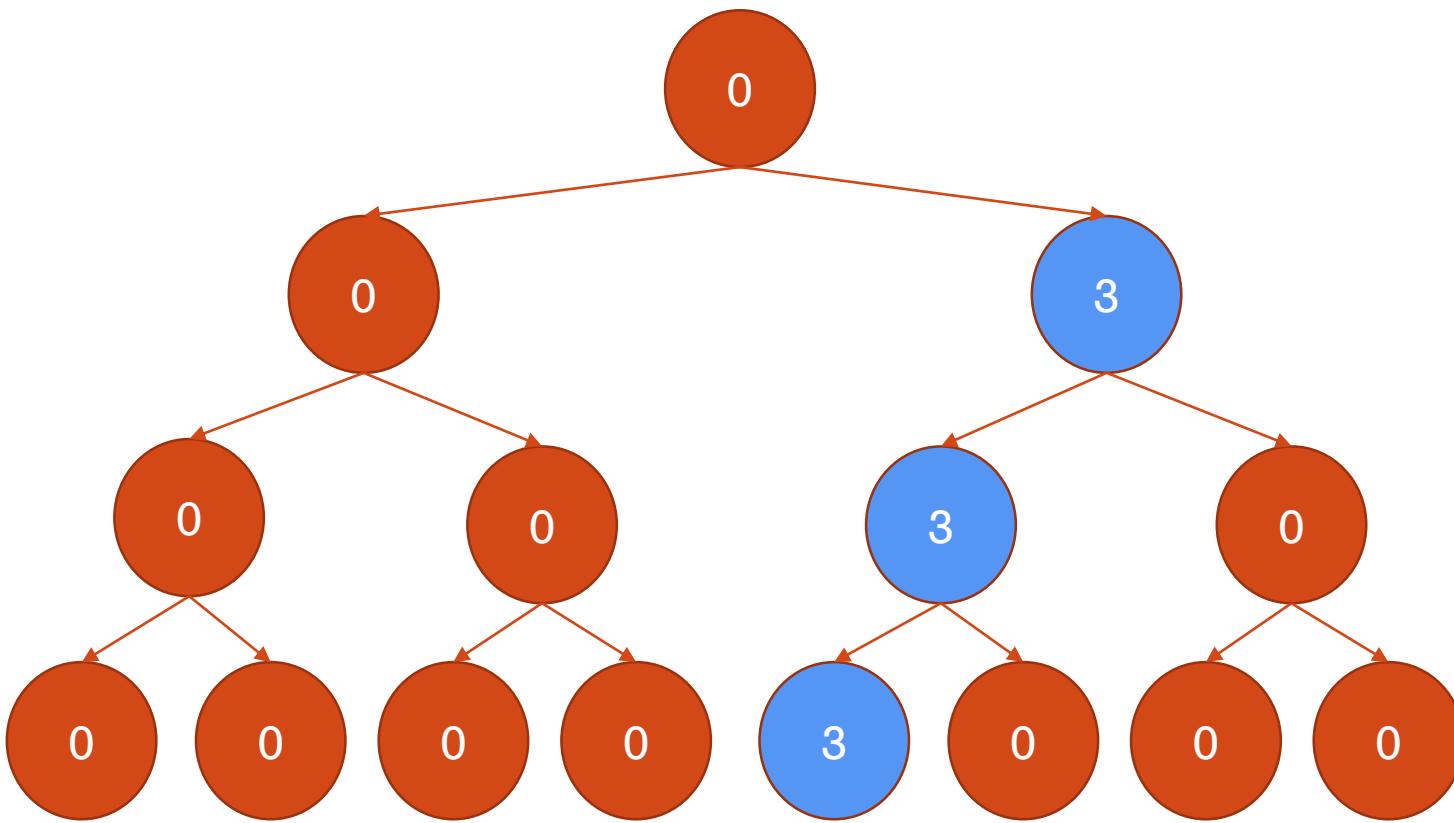
- 4번째 원소에 3을 더한다.



SINGLE UPDATE

- RANGE QUERY/SINGLE UPDATE

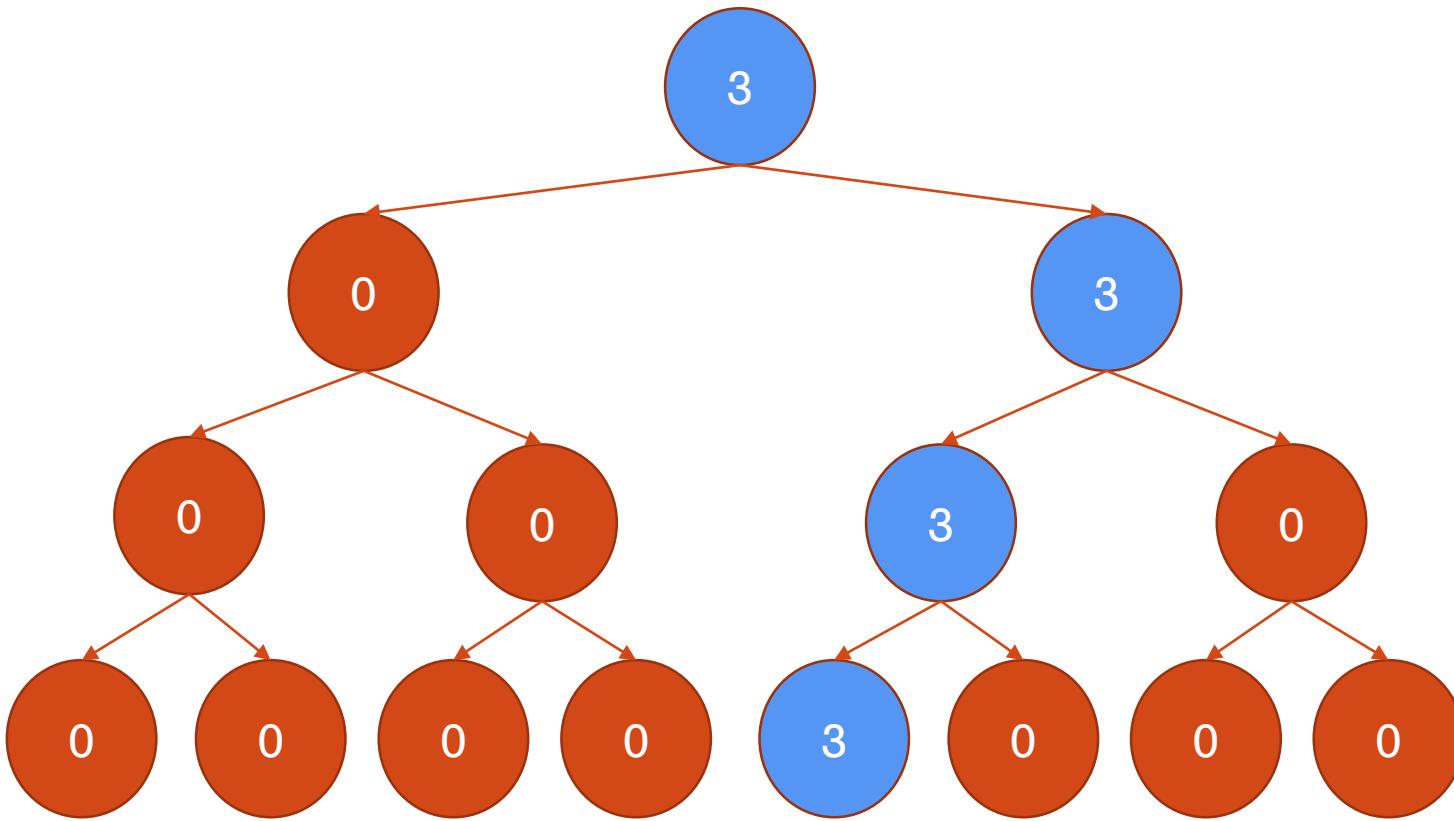
- 4번째 원소에 3을 더한다.



SINGLE UPDATE

- RANGE QUERY/SINGLE UPDATE

- 4번째 원소에 3을 더한다.



실제 배열

0	0	0	0	3	0	0	0
---	---	---	---	---	---	---	---

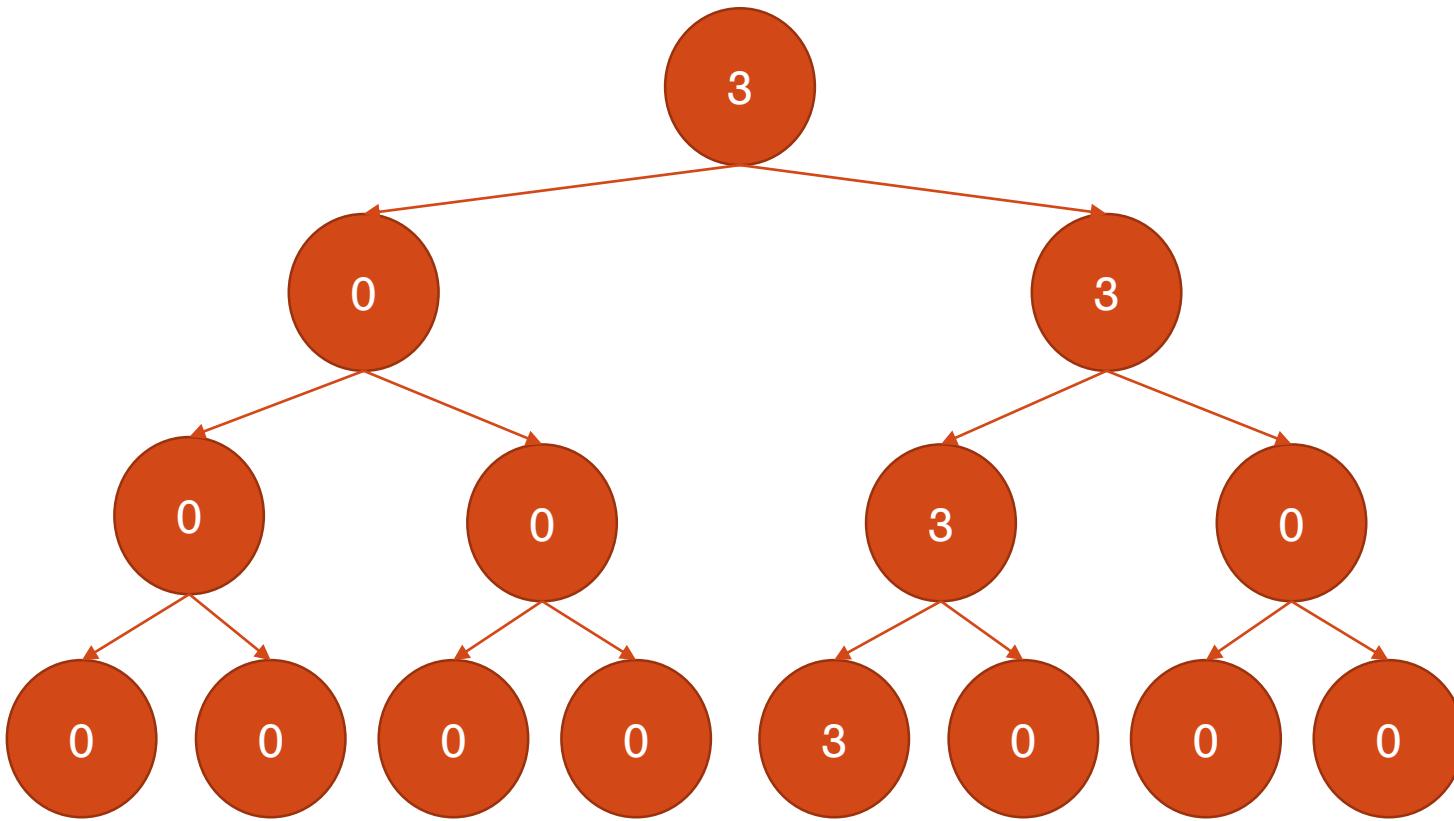
실제 배열

0	0	0	0	3	0	0	0
---	---	---	---	---	---	---	---

SINGLE UPDATE

- RANGE QUERY/SINGLE UPDATE

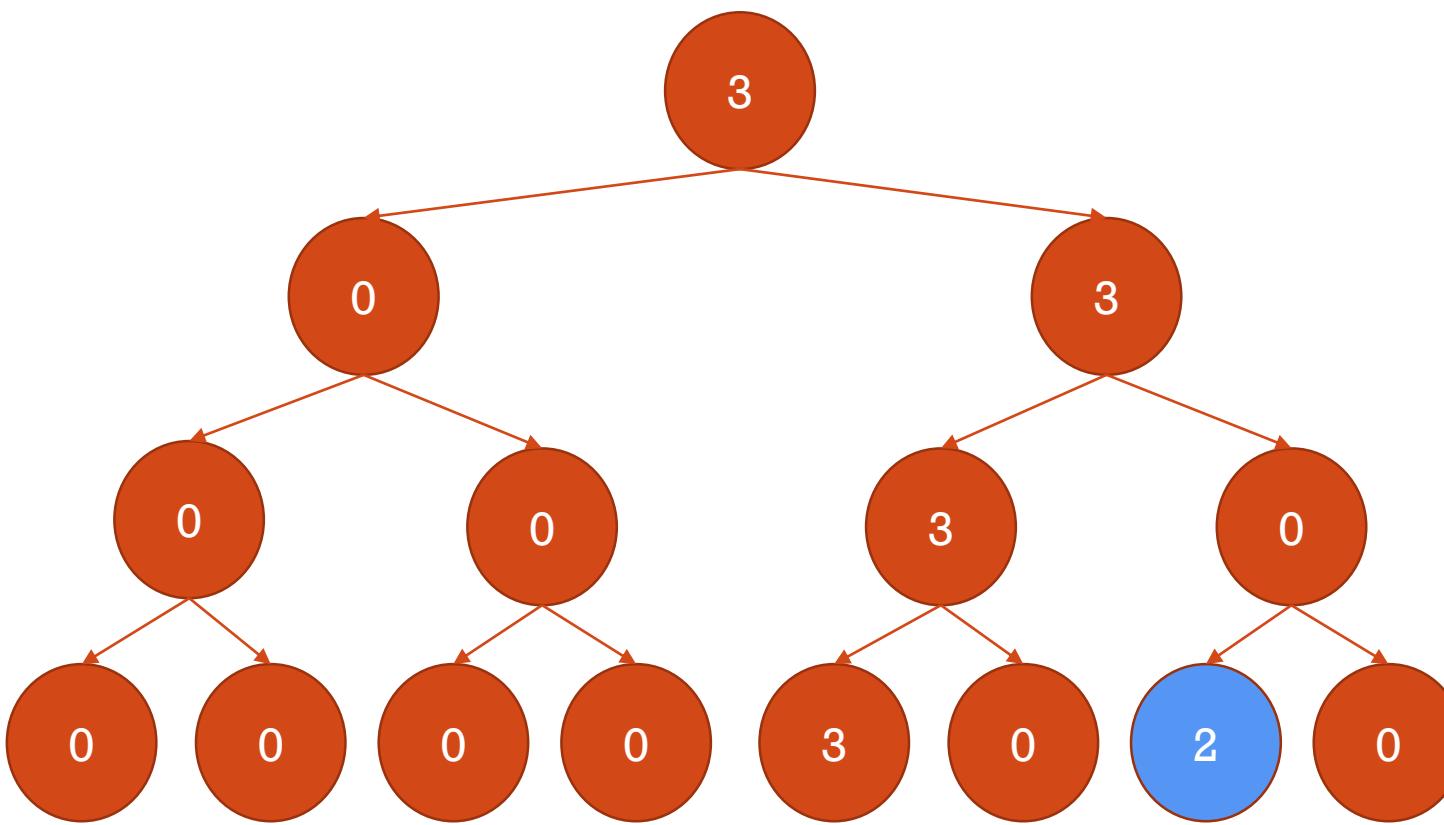
- 6번째 원소에 2를 더한다.



SINGLE UPDATE

- RANGE QUERY/SINGLE UPDATE

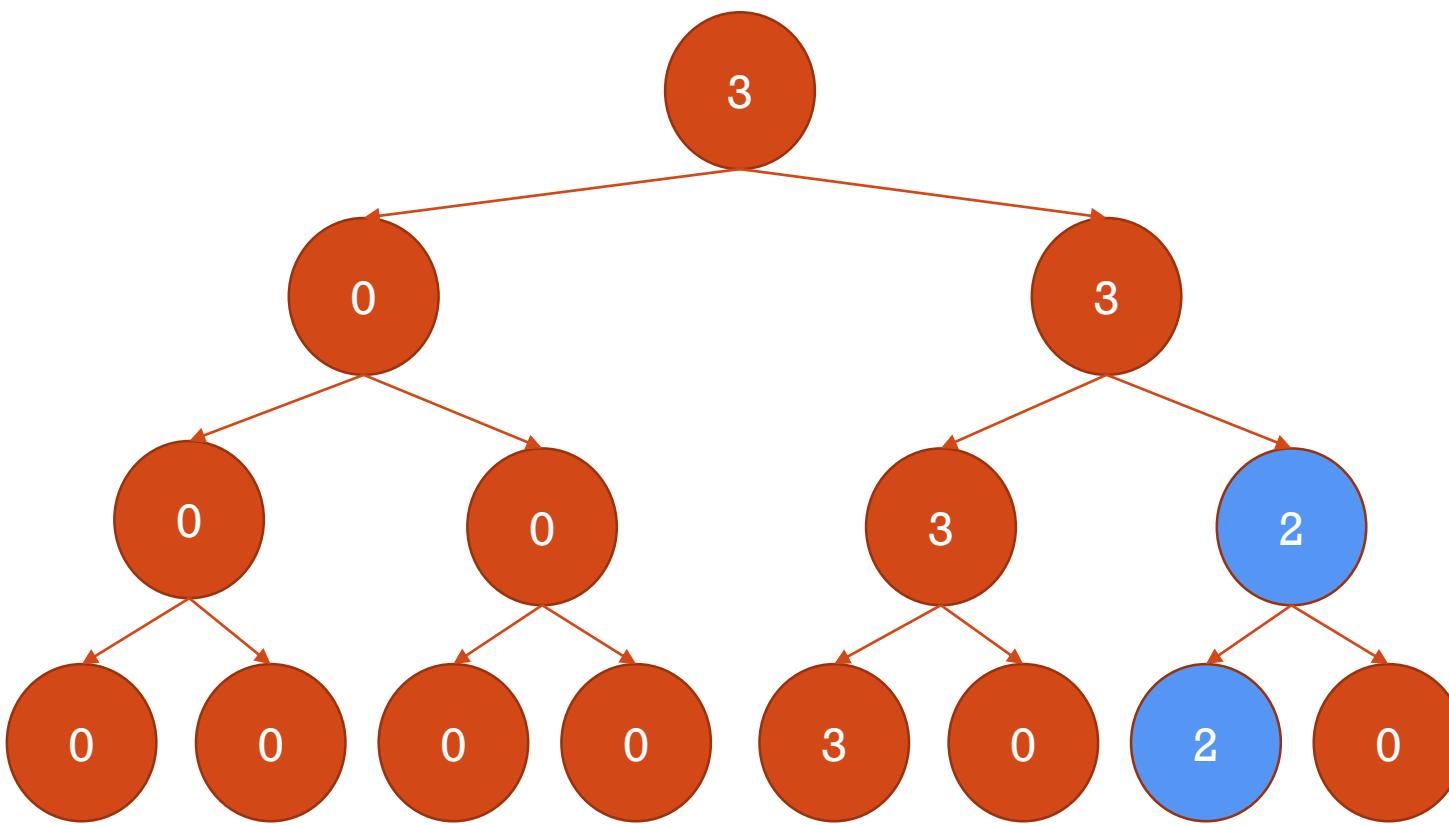
- 6번째 원소에 2를 더한다.



SINGLE UPDATE

- RANGE QUERY/SINGLE UPDATE

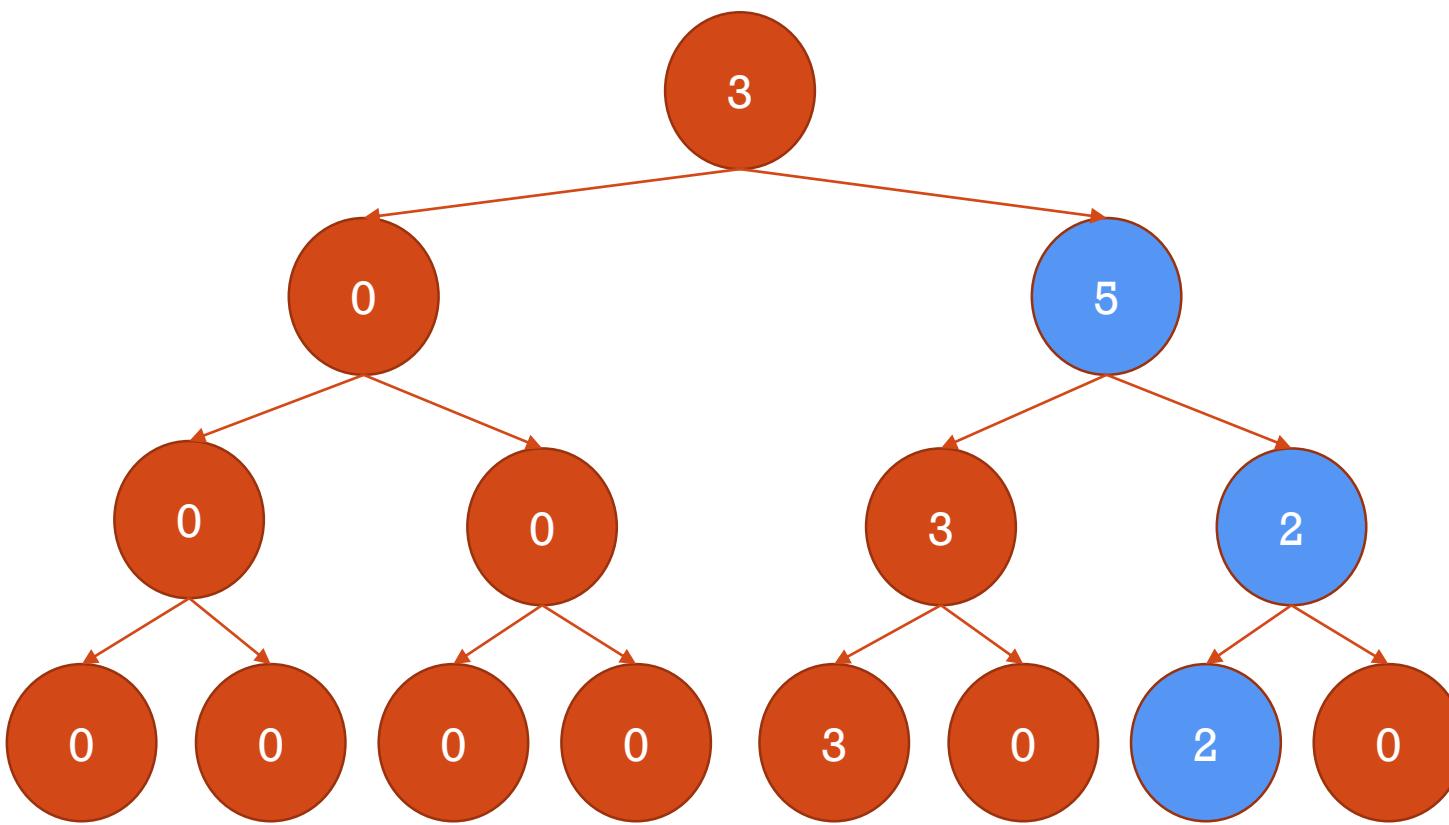
- 6번째 원소에 2를 더한다.



SINGLE UPDATE

- RANGE QUERY/SINGLE UPDATE

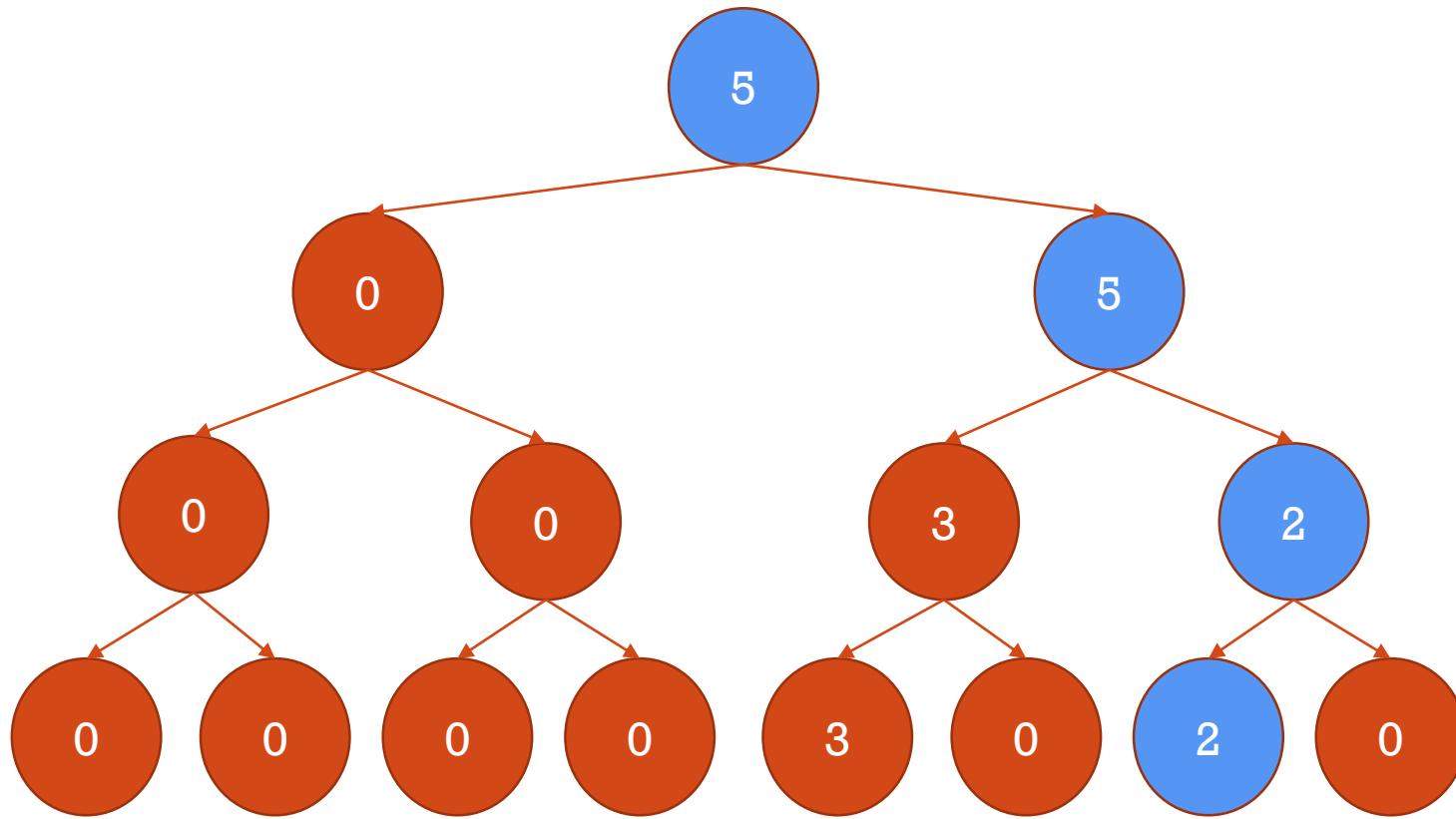
- 6번째 원소에 2를 더한다.



SINGLE UPDATE

- RANGE QUERY/SINGLE UPDATE

- 6번째 원소에 2를 더한다.



실제 배열

0	0	0	0	3	0	2	0
---	---	---	---	---	---	---	---

SINGLE UPDATE

- RANGE QUERY/SINGLE UPDATE

```
void update(int idx,int val){  
    idx += k;                                // k 는 첫번째 leaf node의 배열 번호  
    while(idx){                                // idx!=0, root까지 값을 갱신  
        tree[idx] += val;                      // 트리의 값 update  
        idx /= 2;                               // 부모로 이동  
    }  
}
```



RANGE QUERY

- RANGE QUERY/SINGLE UPDATE

- Range Query는 다음과 같은 알고리즘으로 작동한다.
 - 왼쪽 정점을 기준으로
 1. 현재 정점이 부모 기준으로 오른쪽 자식의 경우, 정답으로서 갱신하고 오른쪽으로 이동한다.
 2. 현재 정점이 부모 기준으로 왼쪽 자식의 경우, 부모가 갱신할 구간을 다 포함하므로 부모로 올라간다.
 3. 오른쪽 정점의 경우 반대로 하여 동시에 이동한다.
 4. 왼쪽 정점과 오른쪽 정점이 엇갈린 경우 종료한다.



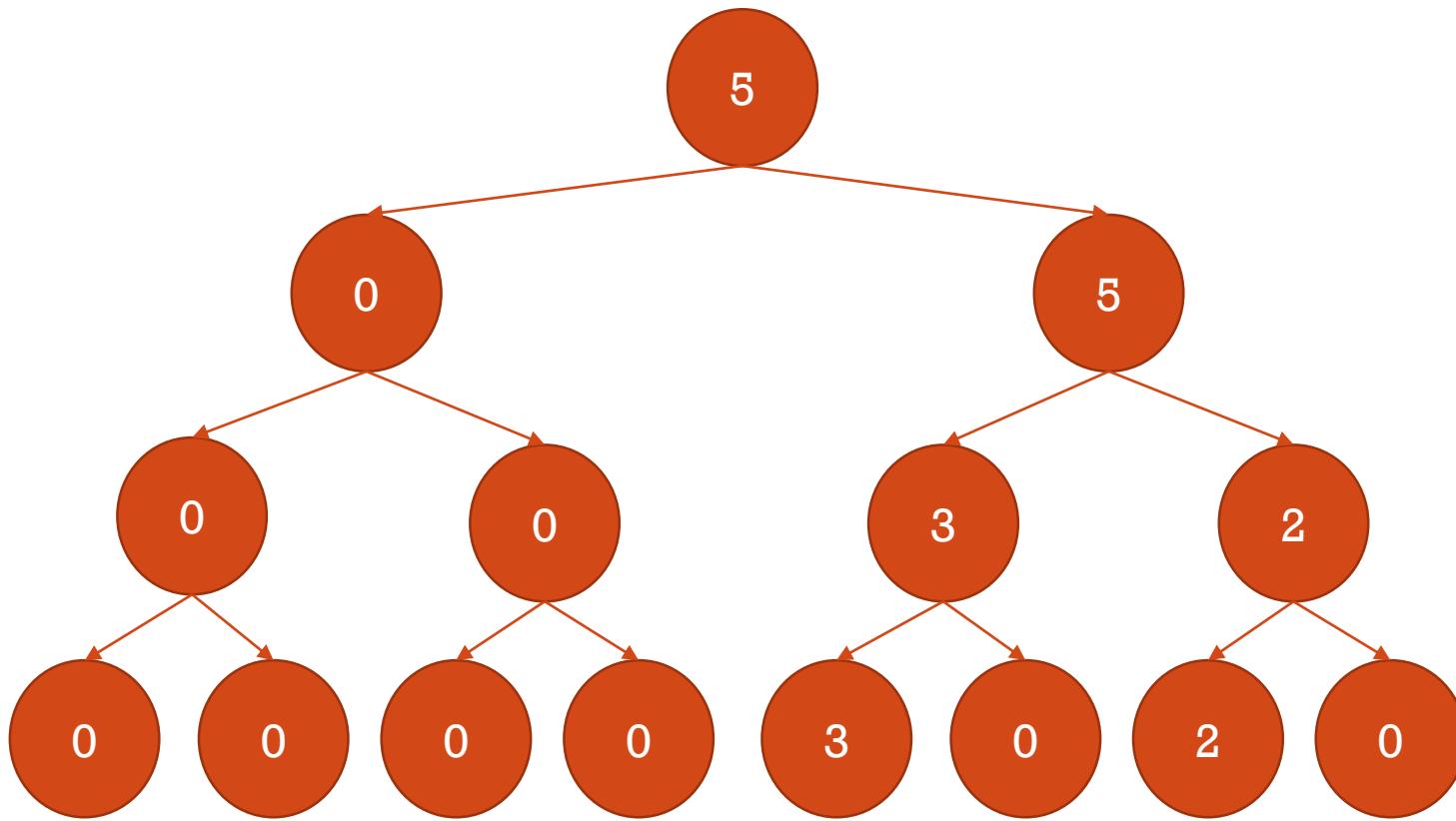
실제 배열

0	0	0	0	3	0	2	0
---	---	---	---	---	---	---	---

RANGE QUERY

- RANGE QUERY/SINGLE UPDATE

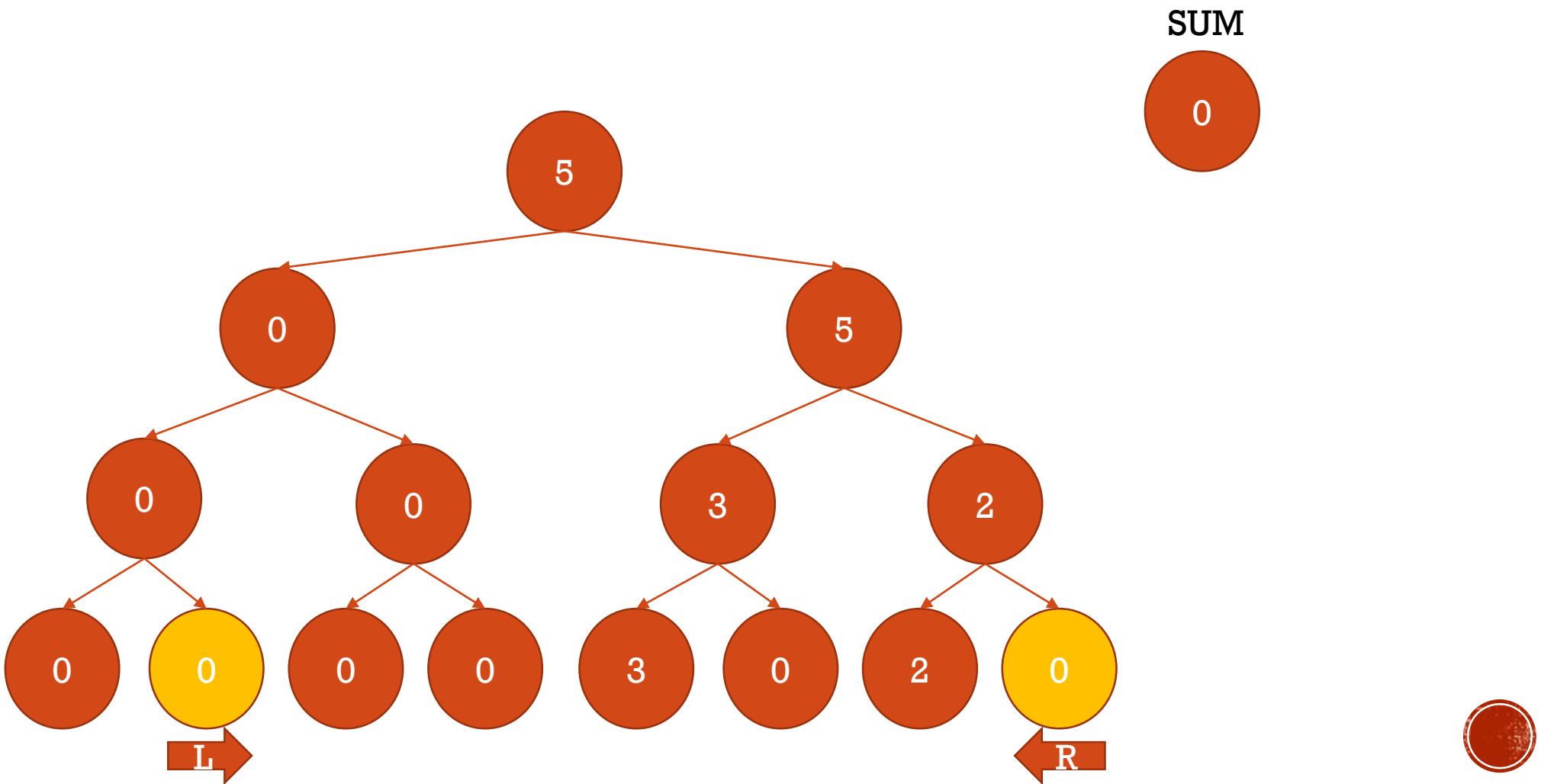
- [1,7]



RANGE QUERY

- RANGE QUERY/SINGLE UPDATE

- [1,7]



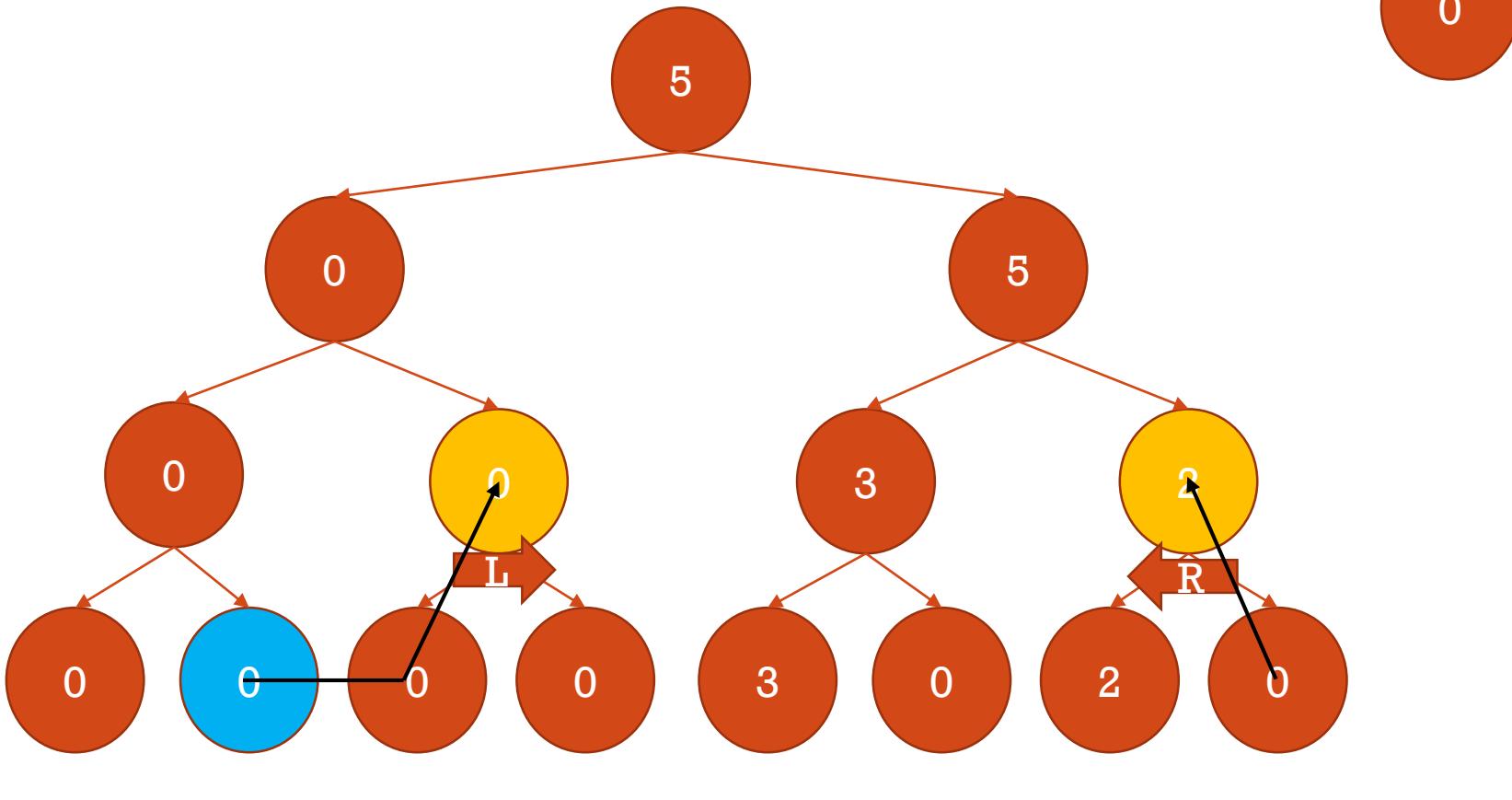
실제 배열

0	0	0	0	3	0	2	0
---	---	---	---	---	---	---	---

RANGE QUERY

- RANGE QUERY/SINGLE UPDATE

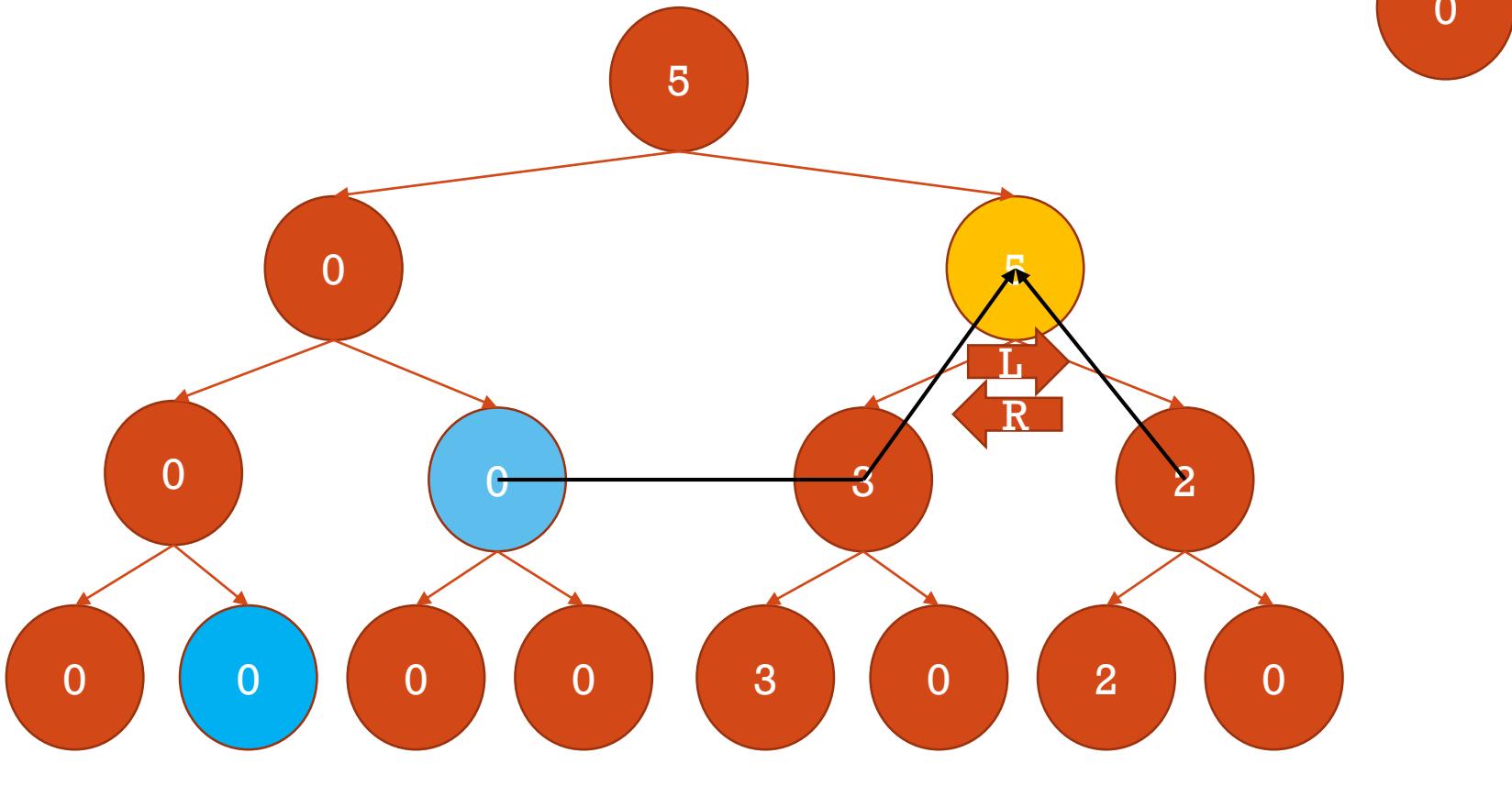
- [1,7] = [1,1]



RANGE QUERY

- RANGE QUERY/SINGLE UPDATE

- $[1,7] = [1,1] + [2,3]$



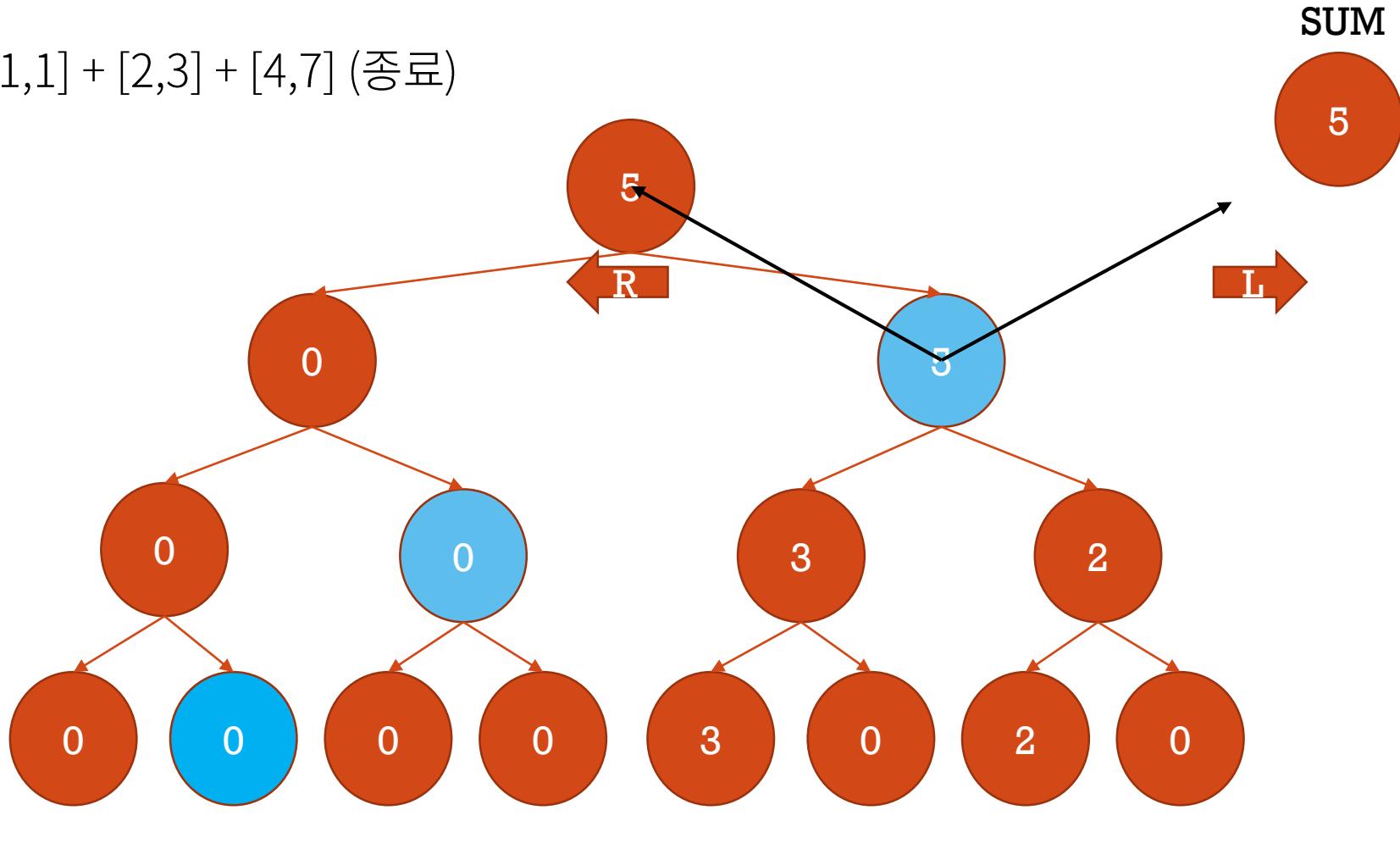
실제 배열

0	0	0	0	3	0	2	0
---	---	---	---	---	---	---	---

RANGE QUERY

- RANGE QUERY/SINGLE UPDATE

- [1,7] = [1,1] + [2,3] + [4,7] (종료)

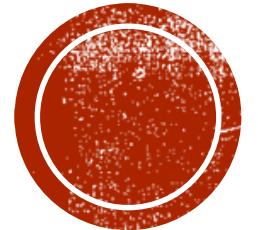


RANGE QUERY

- RANGE QUERY/SINGLE UPDATE

```
int query(int left,int right){  
    int SUM = 0; // 모든 구간의 합을 저장하는 변수  
    left += k; right += k; // leaf 노드  
    while(left <= right){ // 두 정점이 같은 depth를 가지므로 값을 통하여 엇갈렸는지 확인  
        if(left%2 == 1)SUM+=tree[left]; // 왼쪽 정점이 오른쪽 자식인 경우  
        if(right%2 == 0)SUM+=tree[right]; // 오른쪽 정점이 왼쪽 자식인 경우  
        left = (left+1)/2; // 오른쪽으로 이동하고 부모로 이동  
        right = (right-1)/2; // 왼쪽으로 이동하고 부모로 이동  
    }  
    return SUM;  
}
```





SINGLE QUERY/RANGE UPDATE



SINGLE QUERY/RANGE UPDATE

- 구간 합을 구하는 sum tree에 대해서 Single Query / Range Update 를 구현하는 경우, 각 구간을 뜻하는 node에 값이 적절히 나뉘어져 있어, 모두 합하여야 한 위치의 값이 만들어진다.
- Single Query는 leaf node부터 root까지 부모를 올라가며 모든 노드의 합을 구한다.
- Range Update는 $[l,r]$ 에 값을 더한다 할 때, 그 구간을 모두 포함하는 가장 상위 구간들에 더한다.(e.g. $[1,7] = [1,1] + [2,3] + [4,7]$)
- Range Query / Single Update에서 update와 query의 구현을 바꾸어서 코드를 작성하면 된다.



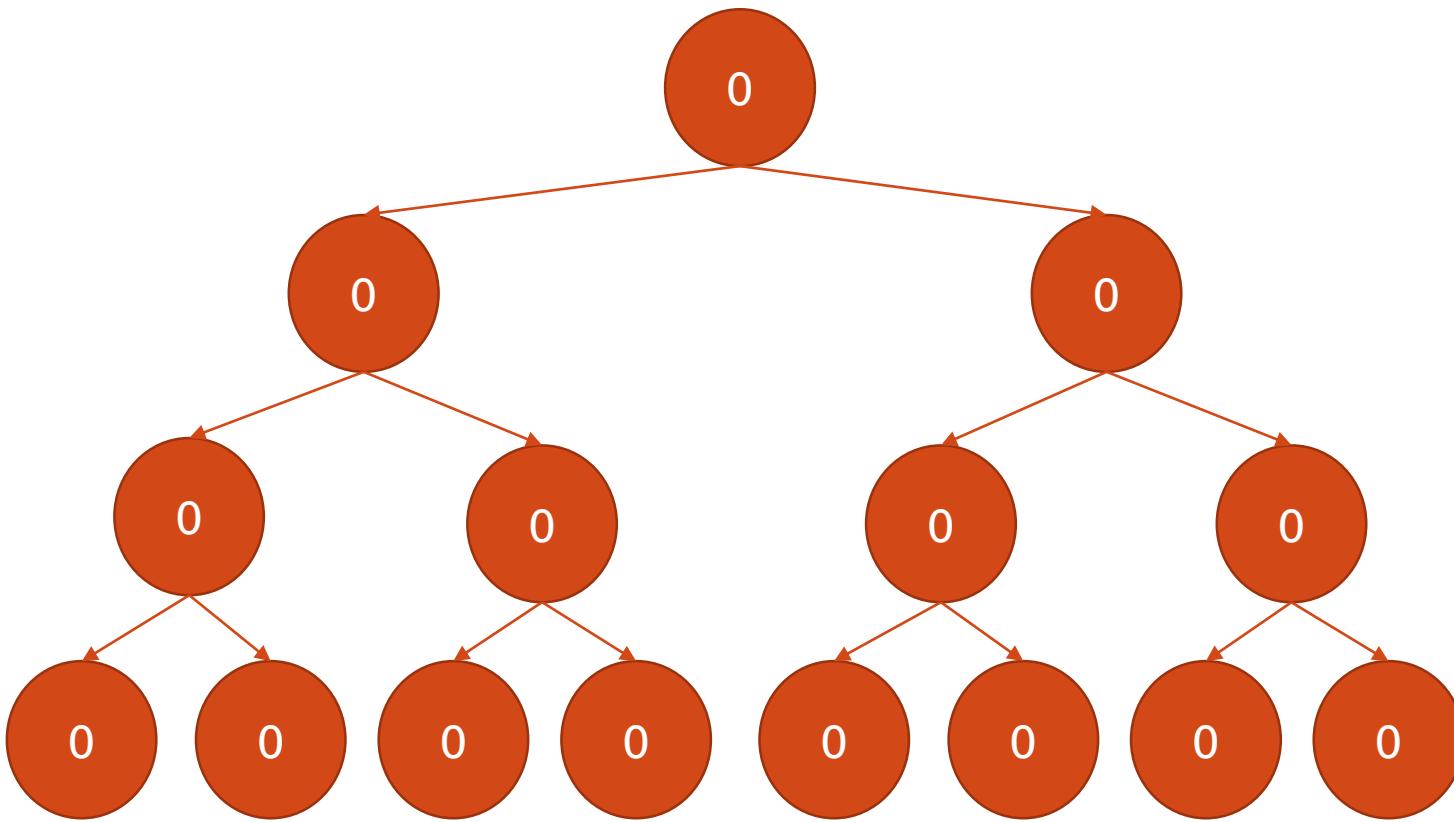
실제 배열

0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---

RANGE UPDATE

- SINGLE QUERY/RANGE UPDATE

- update(range = [1,7],value = 5)



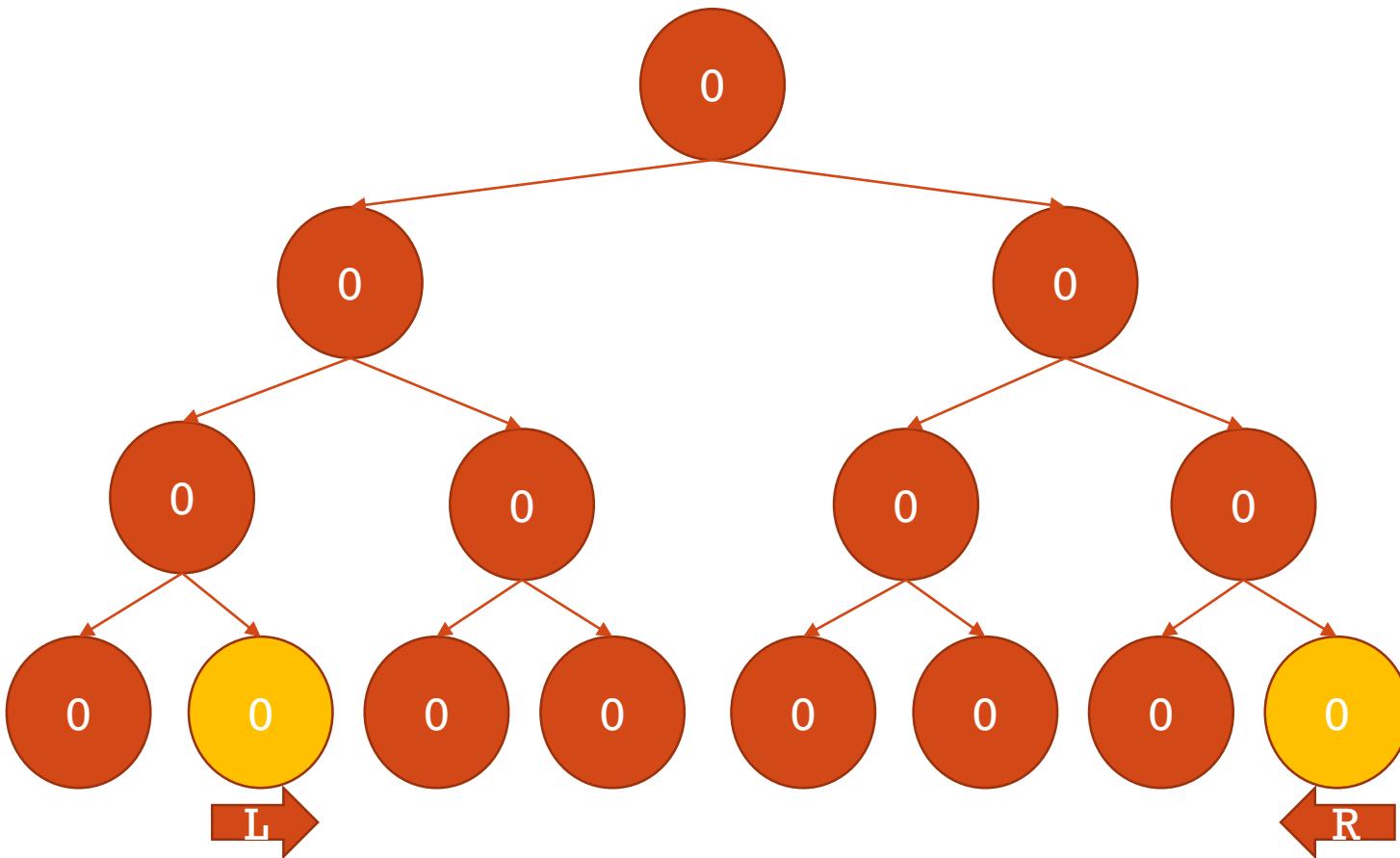
실제 배열

0	0	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---

RANGE UPDATE

- SINGLE QUERY/RANGE UPDATE

- update(range = [1,7],value = 5)



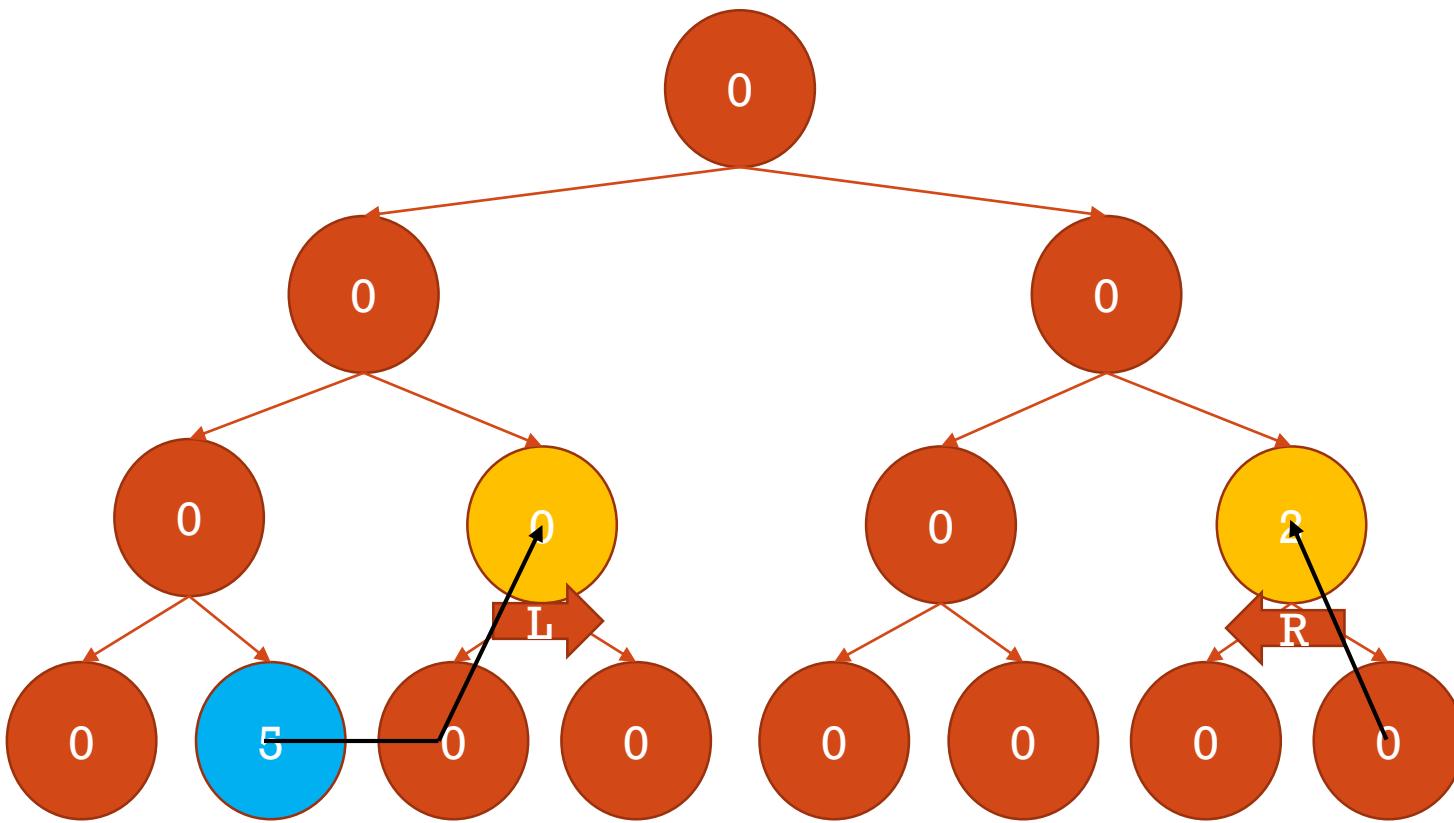
실제 배열

0	5	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---

RANGE UPDATE

- SINGLE QUERY/RANGE UPDATE

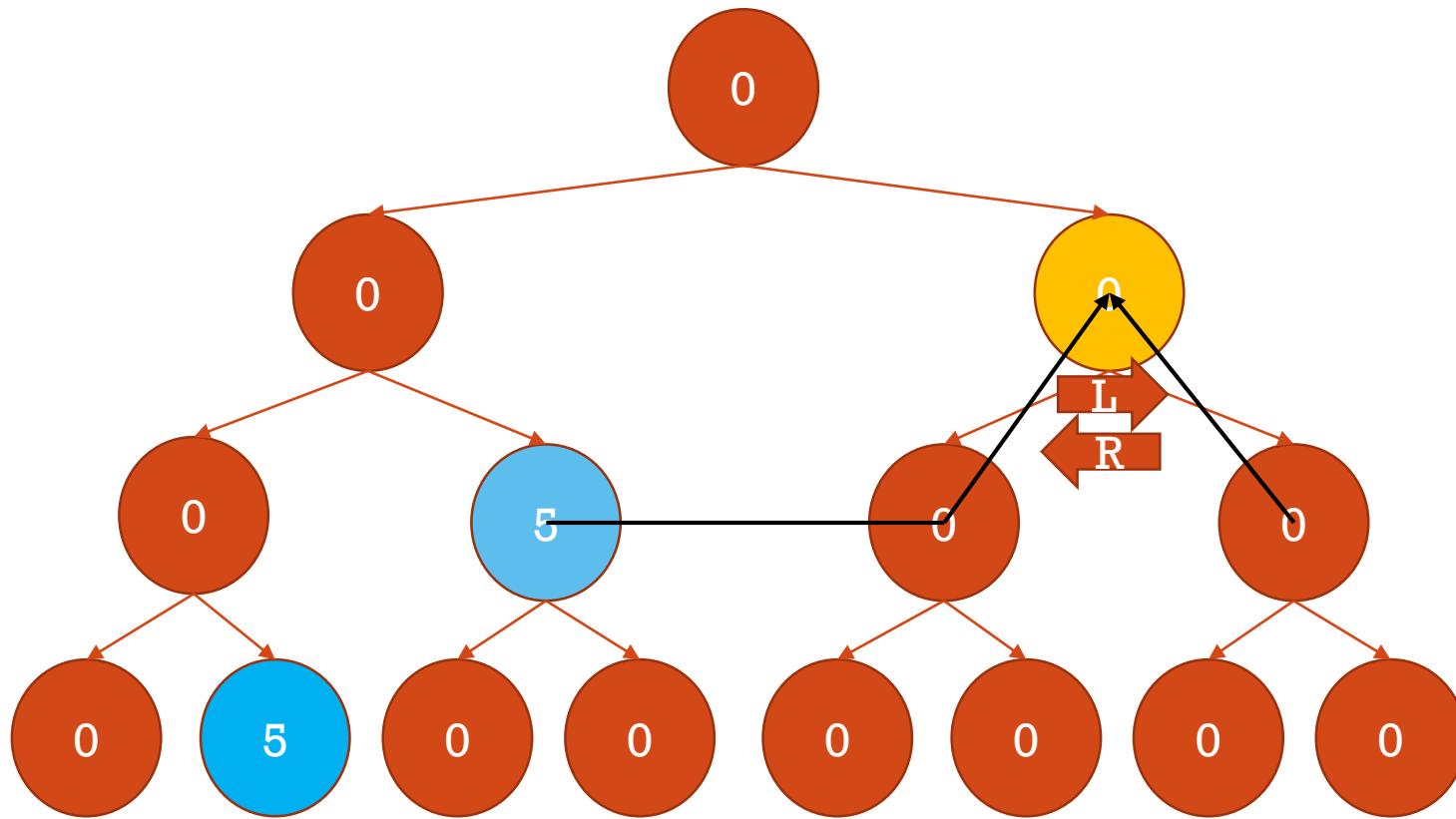
- update(range = [1,7],value = 5)



RANGE UPDATE

- SINGLE QUERY/RANGE UPDATE

- update(range = [1,7],value = 5)



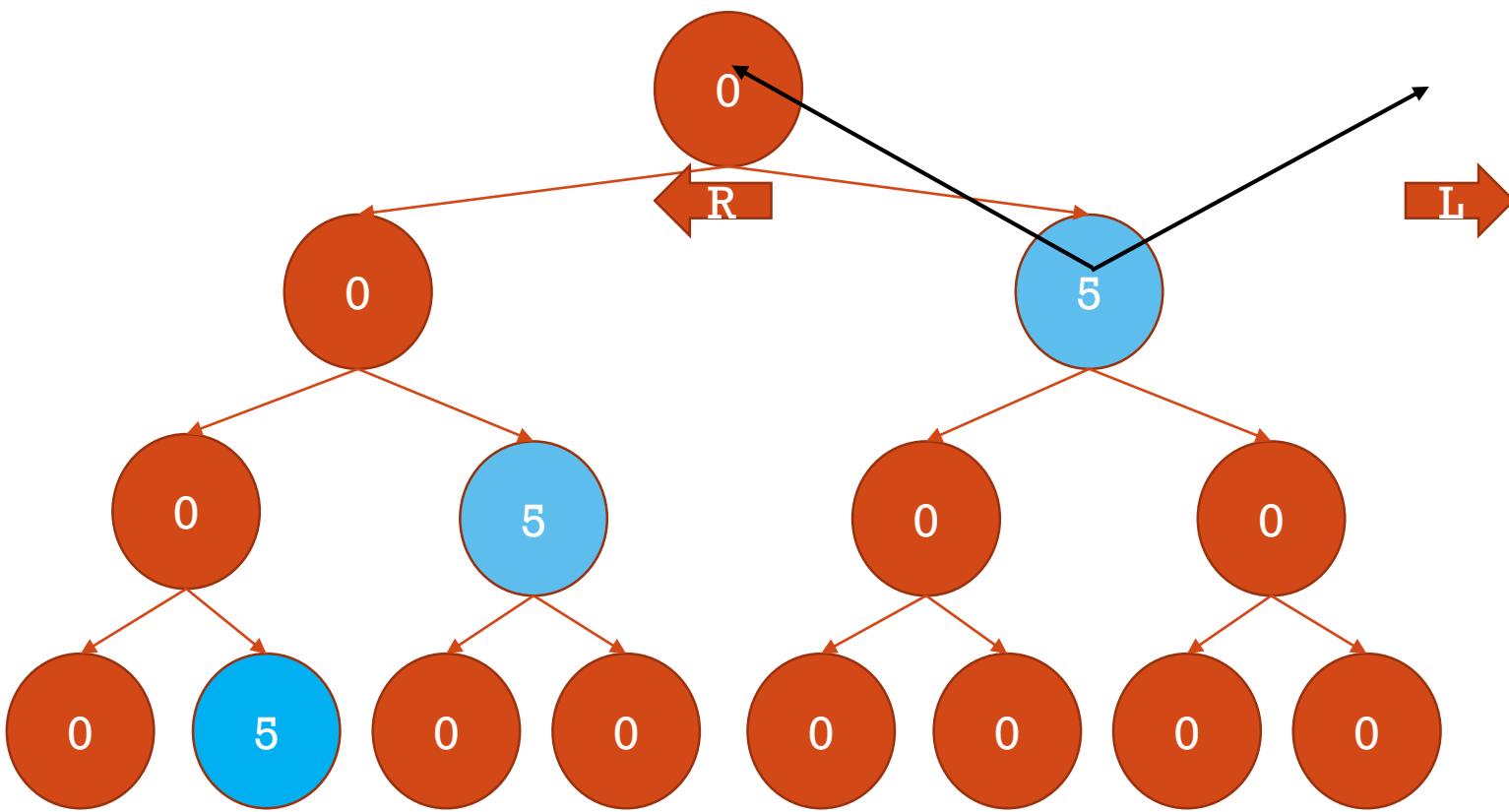
실제 배열

0	5	5	5	5	5	5	5	5
---	---	---	---	---	---	---	---	---

RANGE UPDATE

- SINGLE QUERY/RANGE UPDATE

- update(range = [1,7],value = 5)



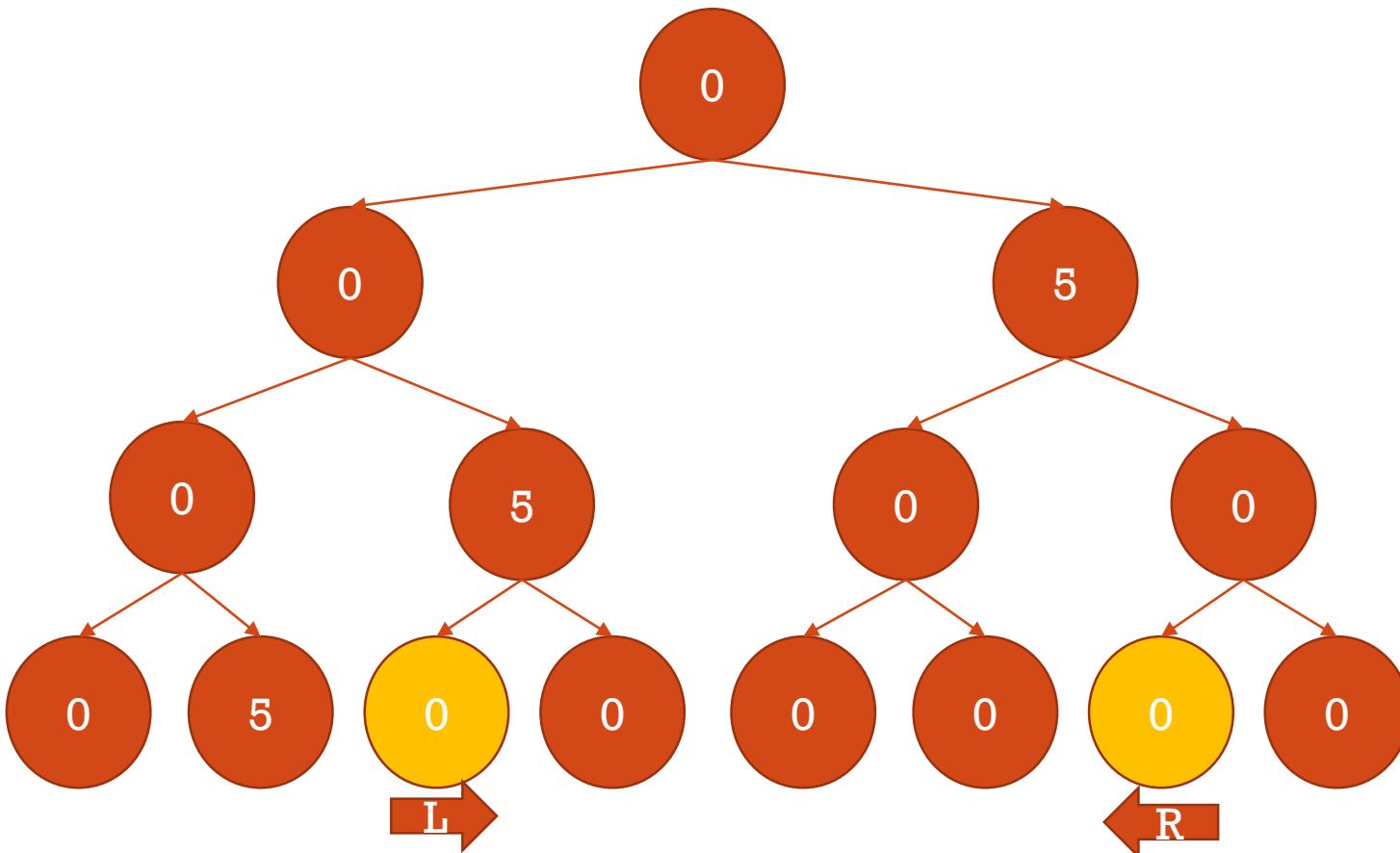
실제 배열

0	5	5	5	5	5	5	5	5
---	---	---	---	---	---	---	---	---

RANGE UPDATE

- SINGLE QUERY/RANGE UPDATE

- update(range = [2,6],value = 7)



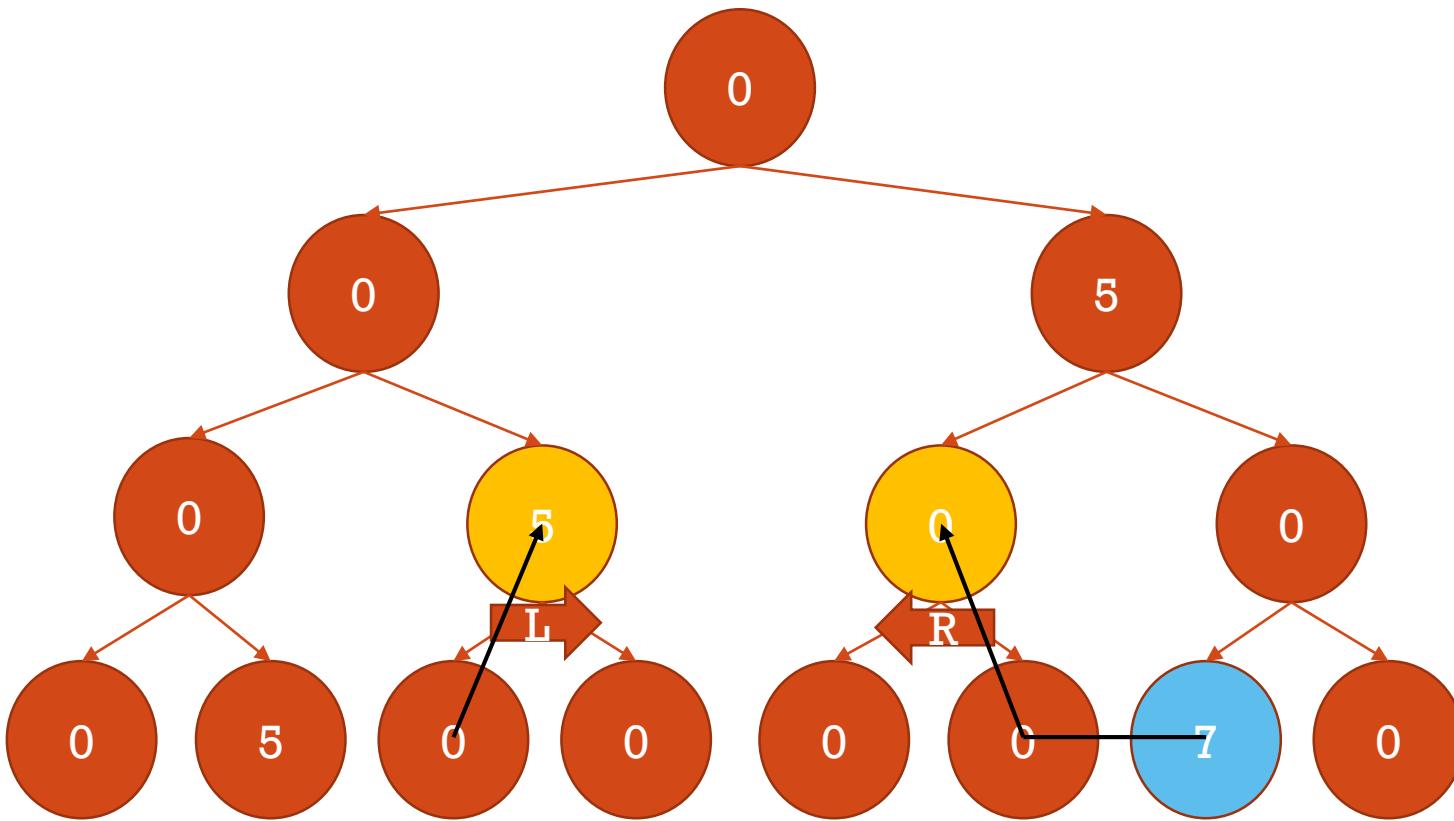
실제 배열

0	5	5	5	5	5	5	12	5
---	---	---	---	---	---	---	----	---

RANGE UPDATE

- SINGLE QUERY/RANGE UPDATE

- update(range = [2,6],value = 7)



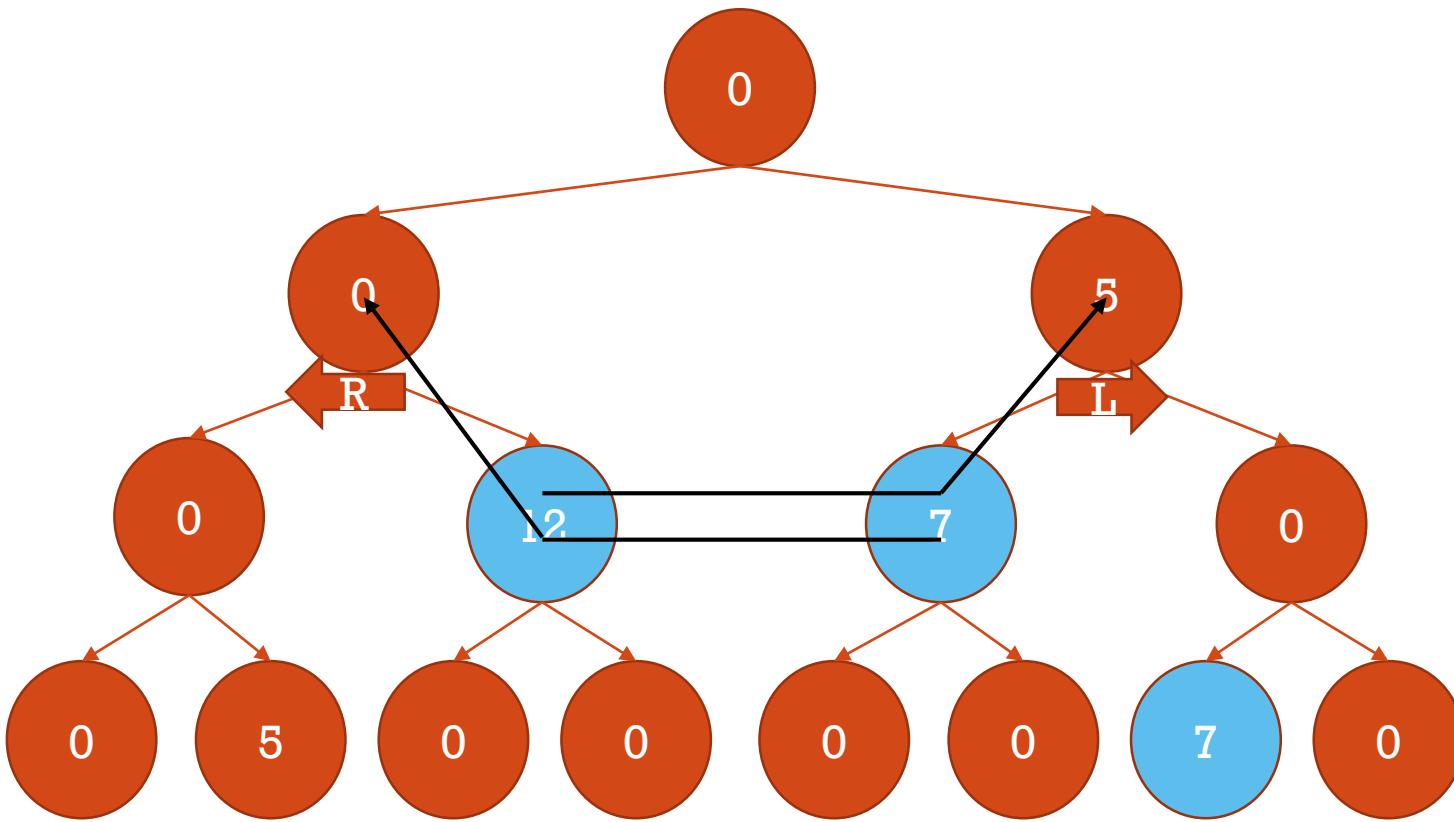
실제 배열

0	5	12	12	12	12	12	5
---	---	----	----	----	----	----	---

RANGE UPDATE

- SINGLE QUERY/RANGE UPDATE

- update(range = [2,6],value = 7)



RANGE UPDATE

- SINGLE QUERY/RANGE UPDATE

```
void update(int left,int right,int val){  
    left += k; right += k; // left 노드  
    while(left <= right){ // 두 정점이 같은 depth를 가지므로 값을 통하여 엇갈렸는지 확인  
        if(left%2 == 1) tree[left] += val; // 왼쪽 정점이 오른쪽 자식인 경우  
        if(right%2 == 0) tree[right] += val; // 오른쪽 정점이 왼쪽 자식인 경우  
        left = (left+1)/2; // 오른쪽으로 이동하고 부모로 이동  
        right = (right-1)/2; // 왼쪽으로 이동하고 부모로 이동  
    }  
}
```



SINGLE QUERY

- SINGLE QUERY/RANGE UPDATE

- Single Query는 다음과 같은 알고리즘으로 작동한다.
 1. leaf node를 구한다.
 2. 그 위치의 값을 더한다.
 3. 부모로 올라간다.
 4. root까지 2~3을 반복.



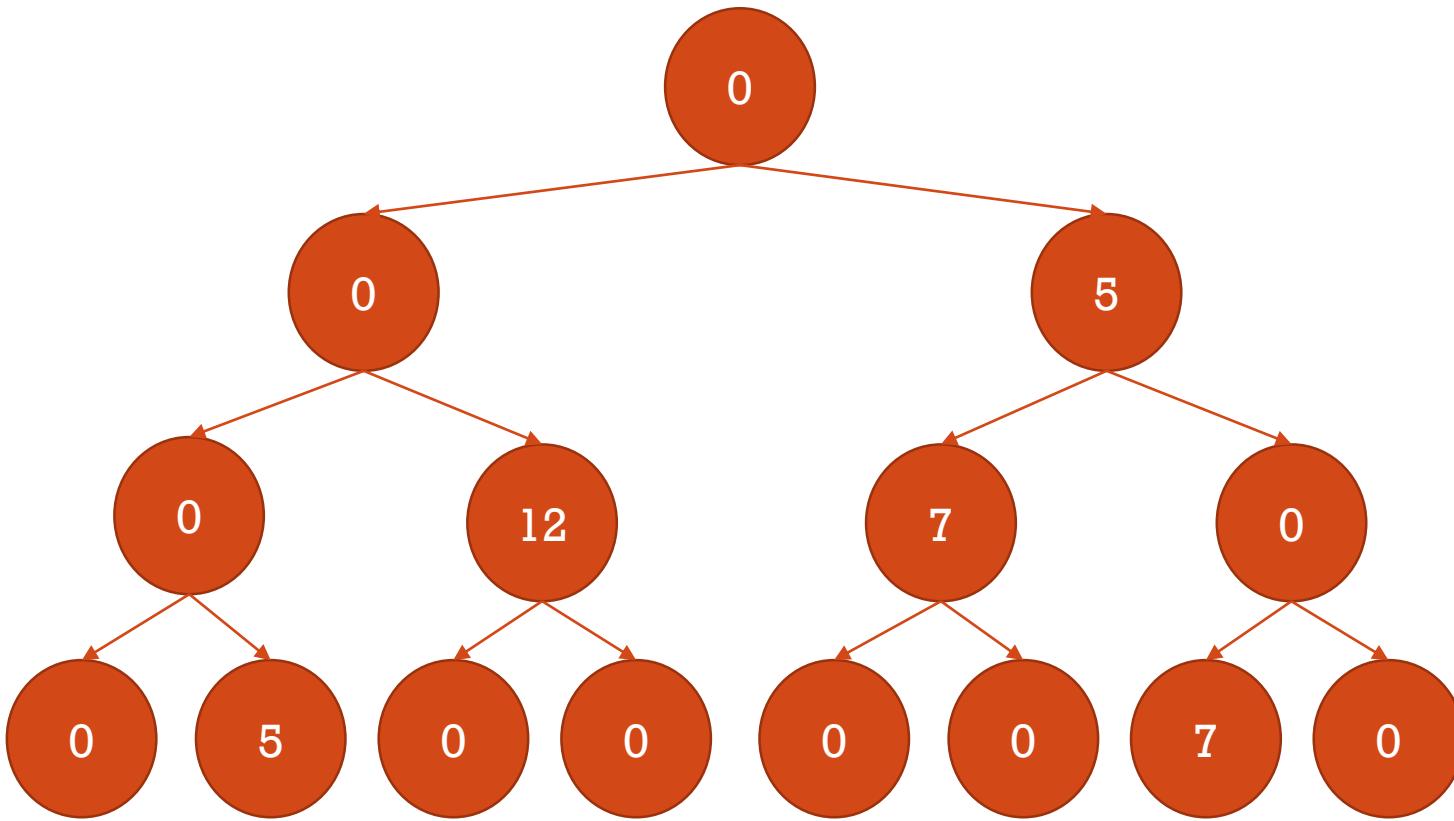
실제 배열

0	5	12	12	12	12	12	5
---	---	----	----	----	----	----	---

SINGLE QUERY

- SINGLE QUERY/RANGE UPDATE

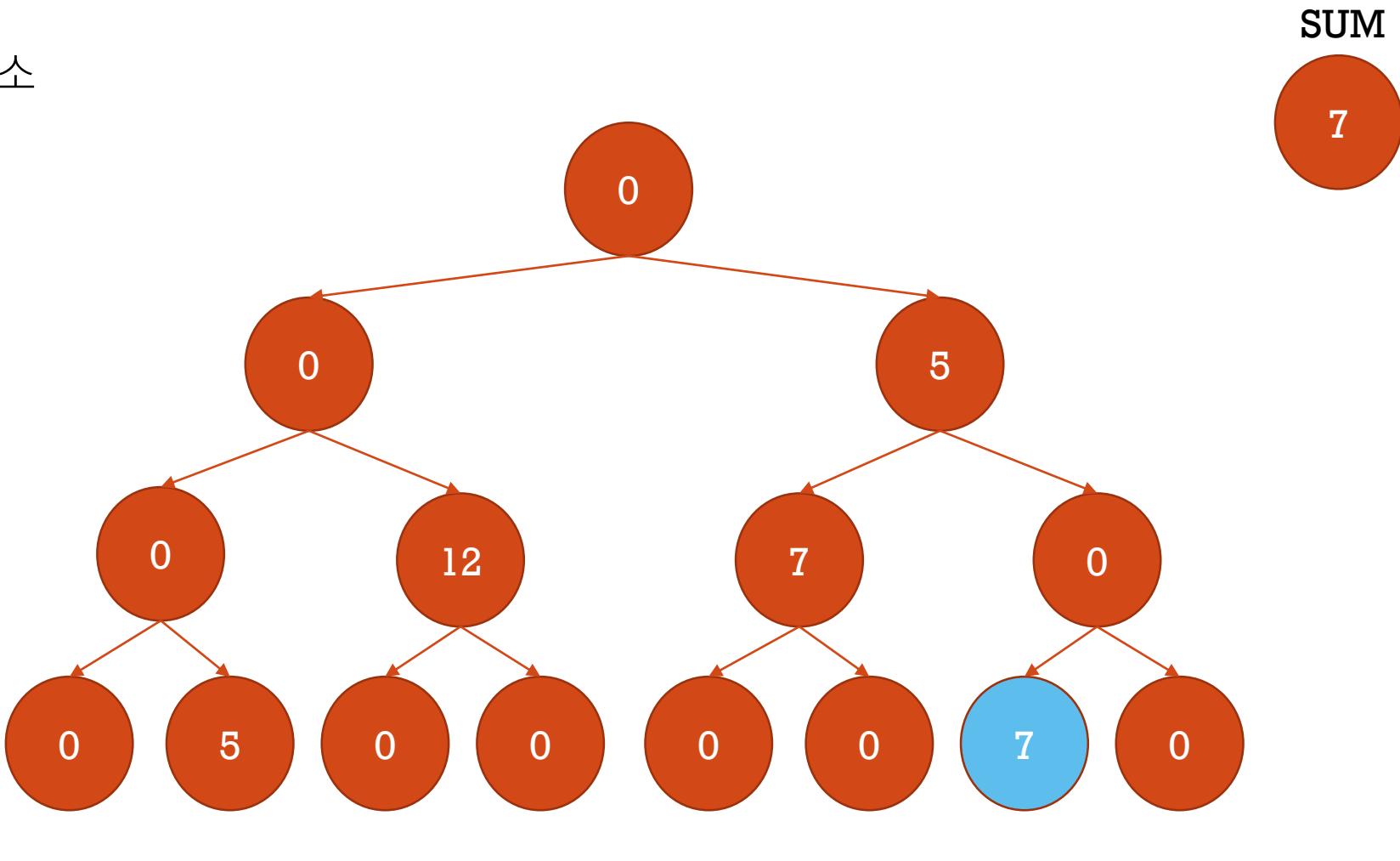
- 6번째 원소



SINGLE QUERY

- SINGLE QUERY/RANGE UPDATE

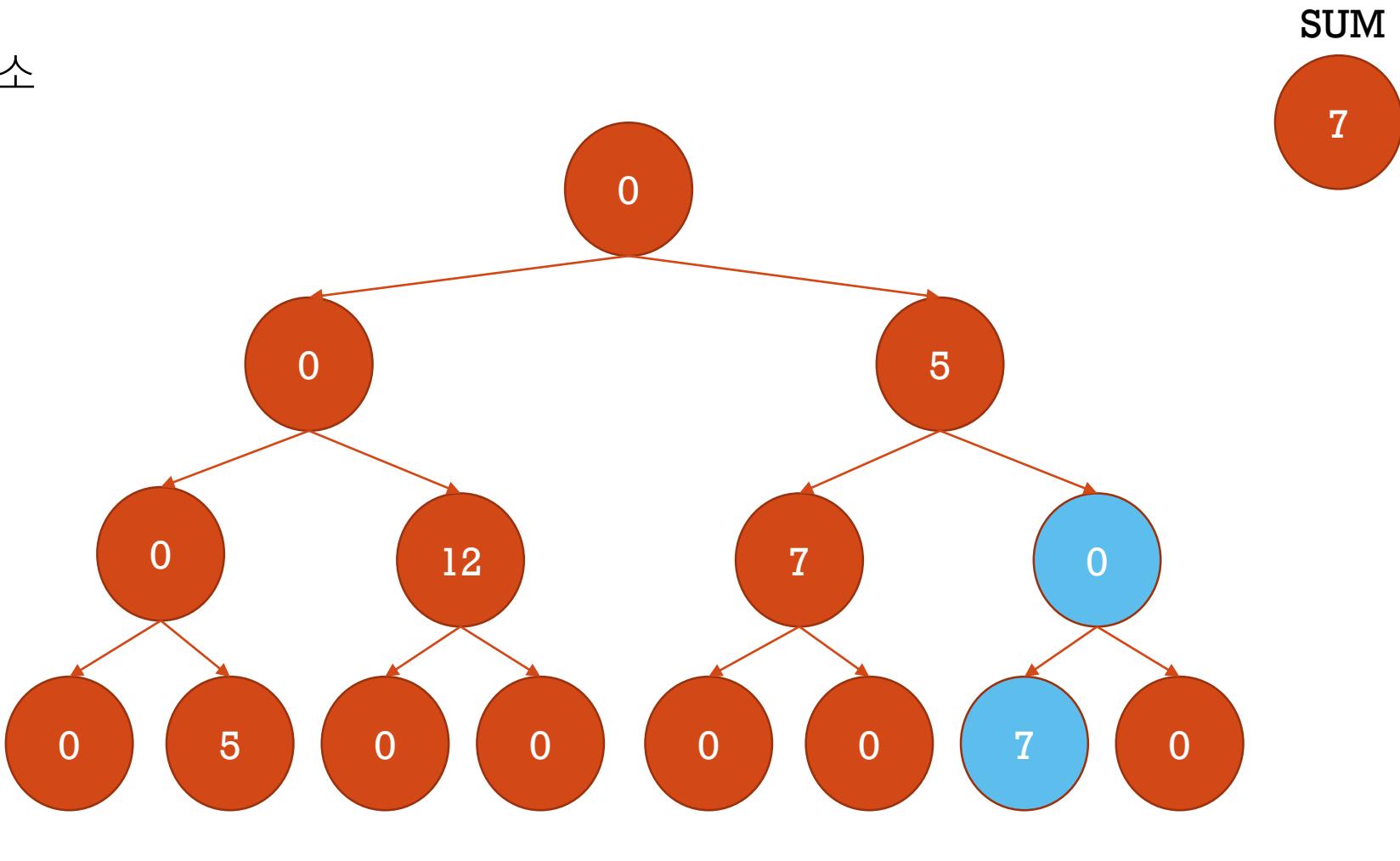
- 6번째 원소



SINGLE QUERY

- SINGLE QUERY/RANGE UPDATE

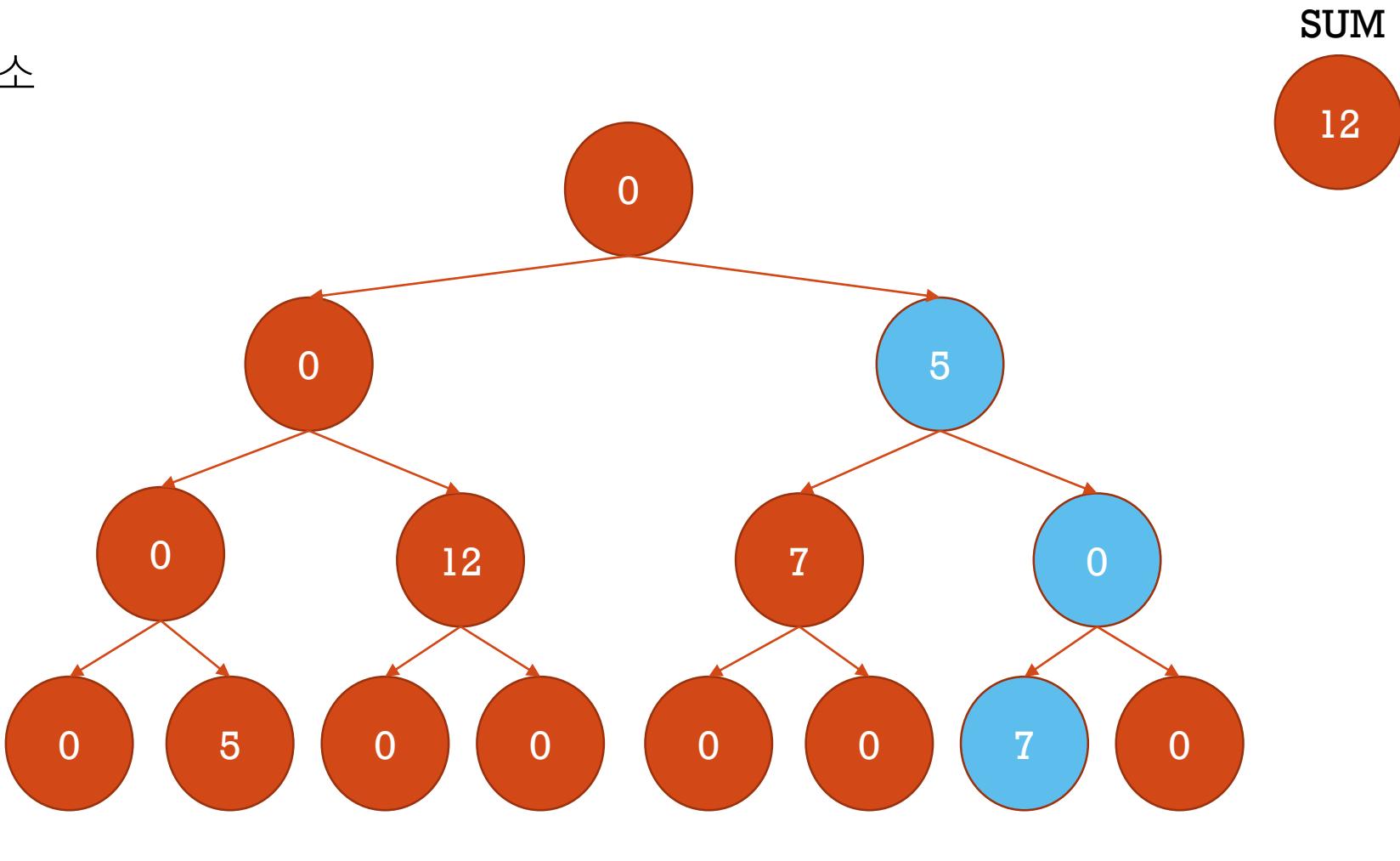
- 6번째 원소



SINGLE QUERY

- SINGLE QUERY/RANGE UPDATE

- 6번째 원소



SINGLE QUERY

- SINGLE QUERY/RANGE UPDATE

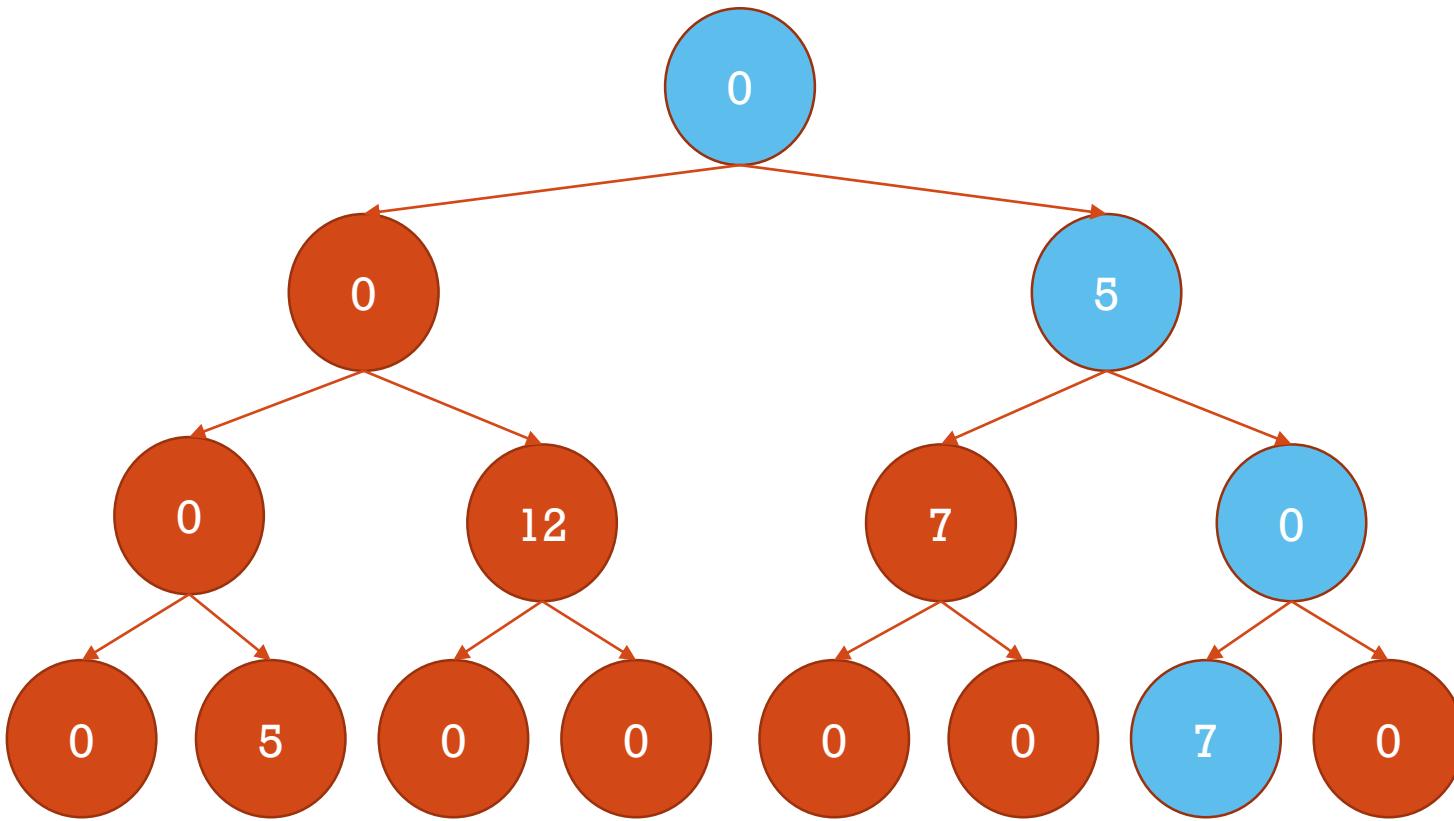
- 6번째 원소

실제 배열

0	5	12	12	12	12	12	5
---	---	----	----	----	----	----	---

SUM

12

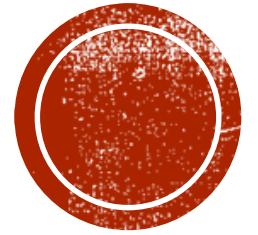


SINGLE QUERY

- SINGLE QUERY/RANGE UPDATE

```
int query(int idx){  
    int SUM = 0; // root까지의 값을 저장하는 변수  
    idx += k; // left 노드  
    while(idx){ // idx!=0, root까지 값을 가져옴  
        SUM += tree[idx]; // 부모의 모든 값을 더한다.  
        idx /= 2; // 부모로 이동  
    }  
    return SUM;  
}
```





RANGE QUERY/RANGE UPDATE



RANGE QUERY/RANGE UPDATE

- 구간 합을 구하는 sum tree에 대해서 Range Query / Range Update 를 구현하는 경우, Single Query / Range Update와 마찬가지로 값이 구간에 적절히 나뉘어져 있다.
- 따라서, 구간의 값을 가져올 때, 그 구간의 부모와 자식에서 더해진 값을 가져와야 한다.
- 구현에서는 원래 그 구간 전부에 더해지는 단위 값을 가진 변수(unit)와, 부모와 자식들의 값을 저장해두는 변수(sum) 2개를 사용한다.



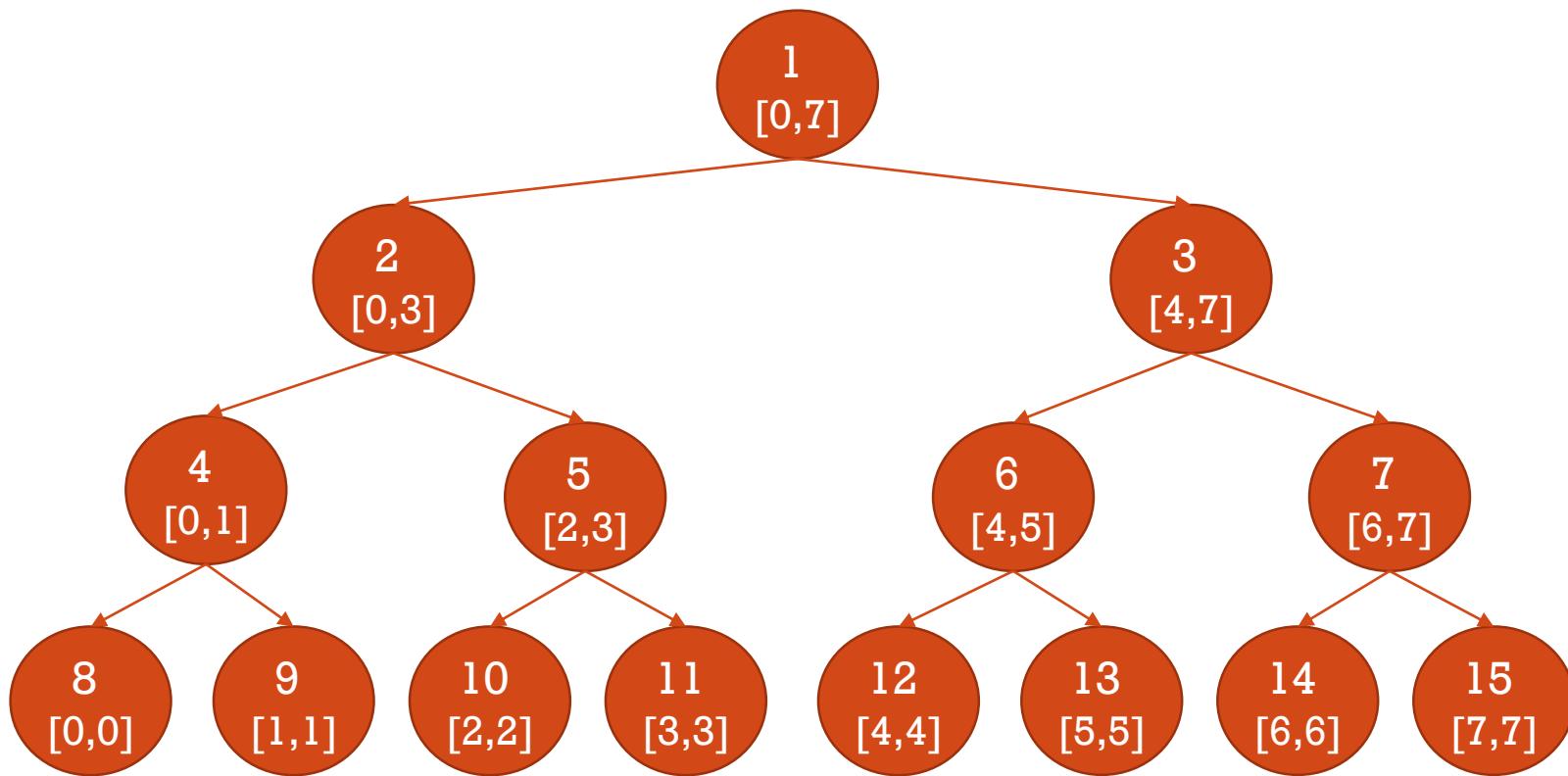
RANGE QUERY/RANGE UPDATE

unit/sum

- 트리의 정점에는 다음과 같이 unit / sum에 대한 값을 가지고 있어야 한다.
- unit은 single query/ range update와 같은 방법으로 값을 가지고 있다.
- sum은 부모와 자식 모두의 unit의 합을 구간 길이에 맞게 들고 있어야 한다.
- 부모 -> 자식으로 값을 찾을 경우 자식의 수가 많아 구현하기 힘들다.
- 따라서 unit의 값을 sum에 갱신할 때 자식의 unit을 부모의 sum에 더하는 식으로 구현한다.
- 하지만 그러면 부모의 unit값을 알 수가 없는데, 이는 query를 날릴 때 부모의 unit값을 가져오는 식으로 구현하면 해결 할 수 있다.
- sum에는 자식의 unit의 합에 대한 것만 가지고, 부모에 대한 값은 가지고 있지 않다.

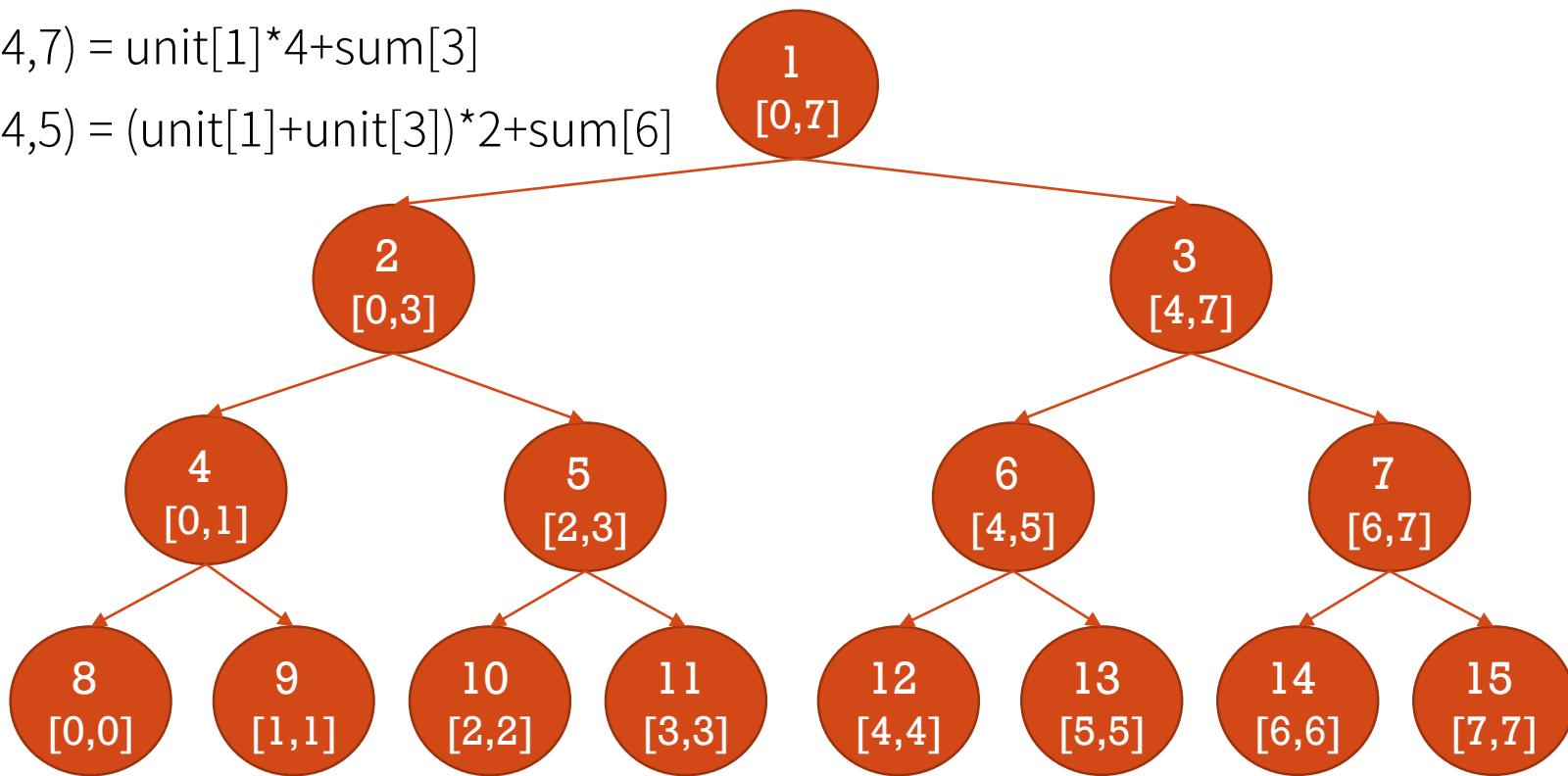
RANGE QUERY/RANGE UPDATE

- single query/ range update 과 같이 저장한다면 다음 위치의 값을 가져와야 한다.
- $\text{query}(4,7) = (\text{unit}[1]+\text{unit}[3]) * 4 + (\text{unit}[6] + \text{unit}[7]) * 2 + (\text{unit}[12]+\text{unit}[13]+\text{unit}[14]+\text{unit}[15]) * 1$



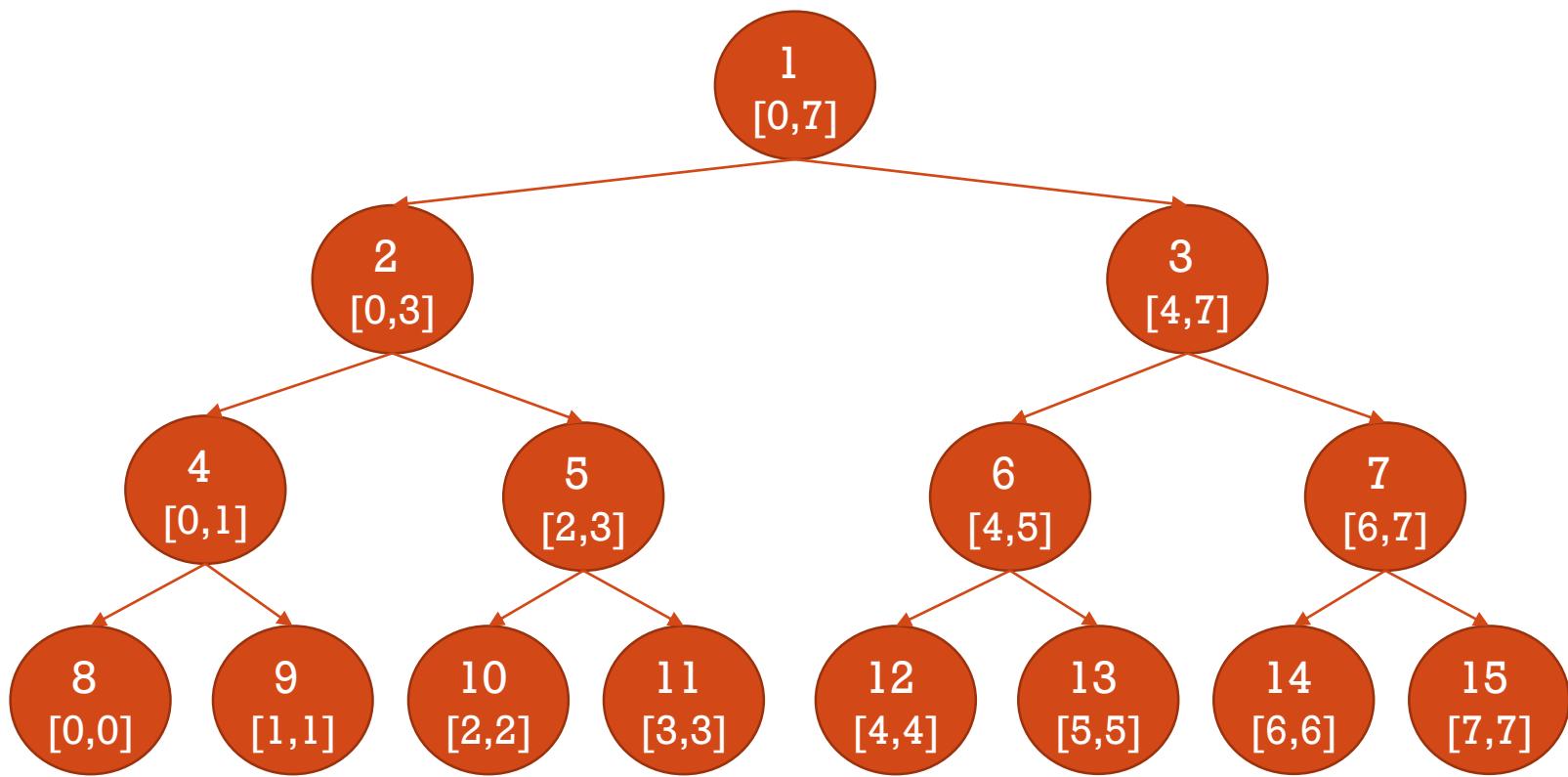
RANGE QUERY/RANGE UPDATE

- 부모에 대한 값은 \log 만에 처리할 수 있다. 따라서 자신과 자식에 대한 값만 모아두면 된다.
- $\text{sum}[3] += \text{unit}[3] * 4 + (\text{unit}[6] + \text{unit}[7]) * 2 + (\text{unit}[12] + \text{unit}[13] + \text{unit}[14] + \text{unit}[15]) * 1$
- $\text{query}(4,7) = \text{unit}[1] * 4 + \text{sum}[3]$
- $\text{query}(4,5) = (\text{unit}[1] + \text{unit}[3]) * 2 + \text{sum}[6]$



RANGE QUERY/RANGE UPDATE

- query(1,7) = query(1,1)+query(2,3)+query(4,7)



RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

- Range Update는 다음과 같은 알고리즘으로 작동한다. $O((\log n)^2)$
- 이전에 사용한 Range 관련 연산에서 사용한 알고리즘을 통하여 구간을 모두 포함하는 가장 상위 구간을 구한다.
 1. 각 구간별로 아래와 같은 연산을 한다.
 1. 해당 구간의 unit에 값을 더한다.
 2. 해당 구간과 그 구간의 모든 조상의 sum에 unit*(해당 구간의 길이)를 더한다.(조상 구간의 길이 x)



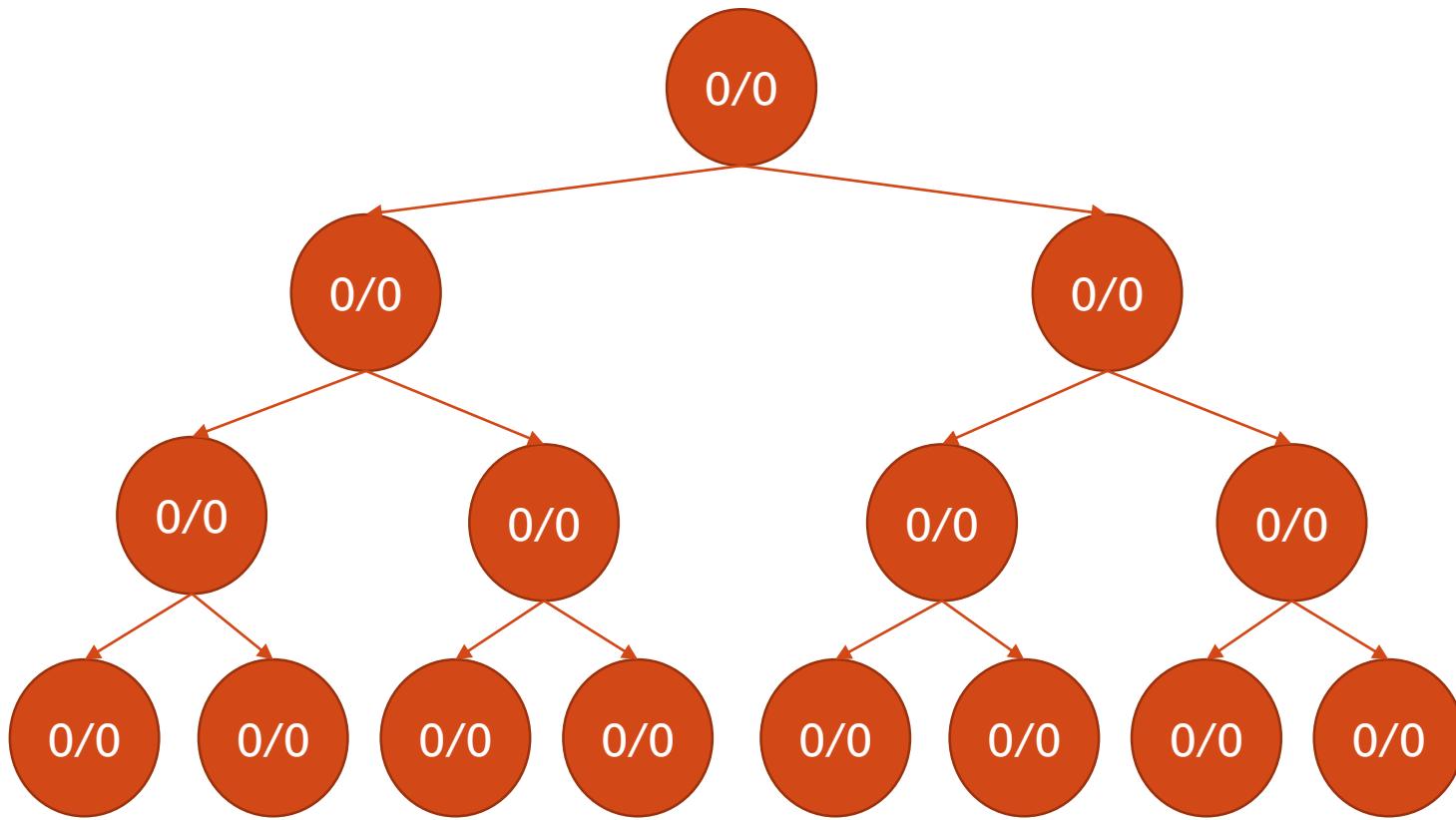
RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

- update(range = [1,7],value = 5)

실제 배열							
0	0	0	0	0	0	0	0

unit/sum



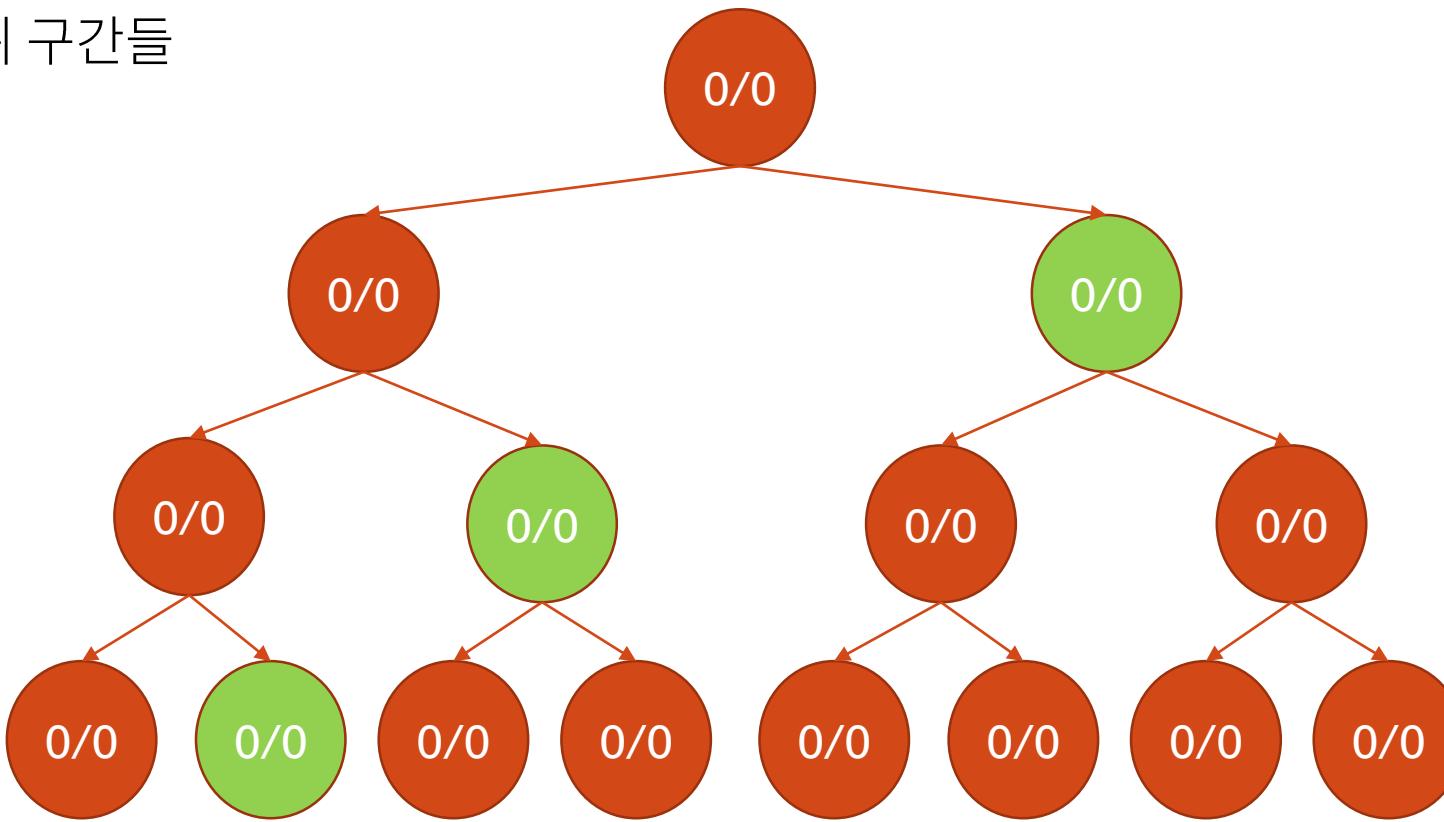
RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

- update(range = [1,7],value = 5)
- 가장 상위 구간들



unit/sum



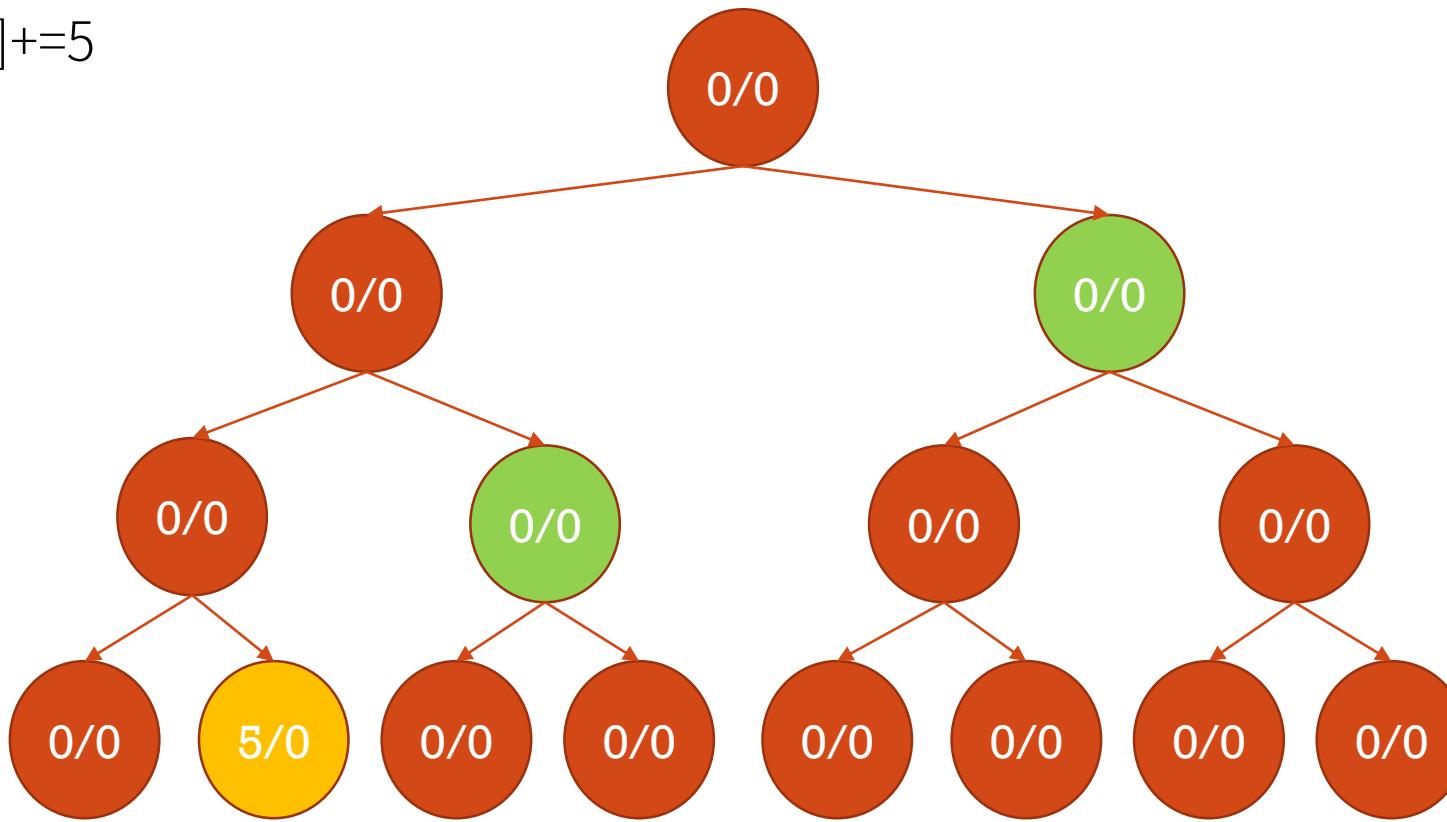
RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

- update(range = [1,7],value = 5)
- unit[1,1]+=5



unit/sum



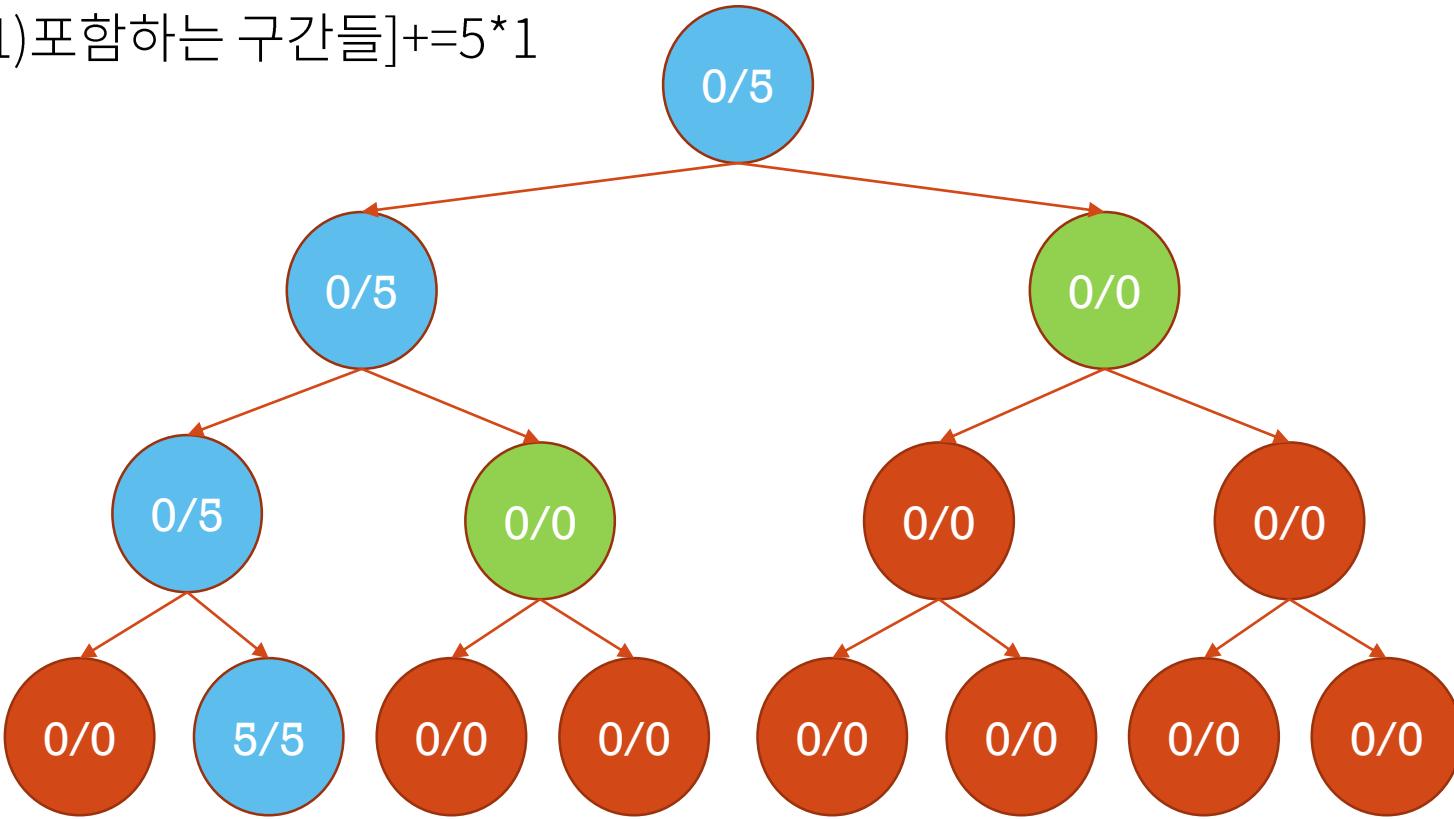
RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

- update(range = [1,7],value = 5)
- sum[(1,1)포함하는 구간들]+=5*1



unit/sum



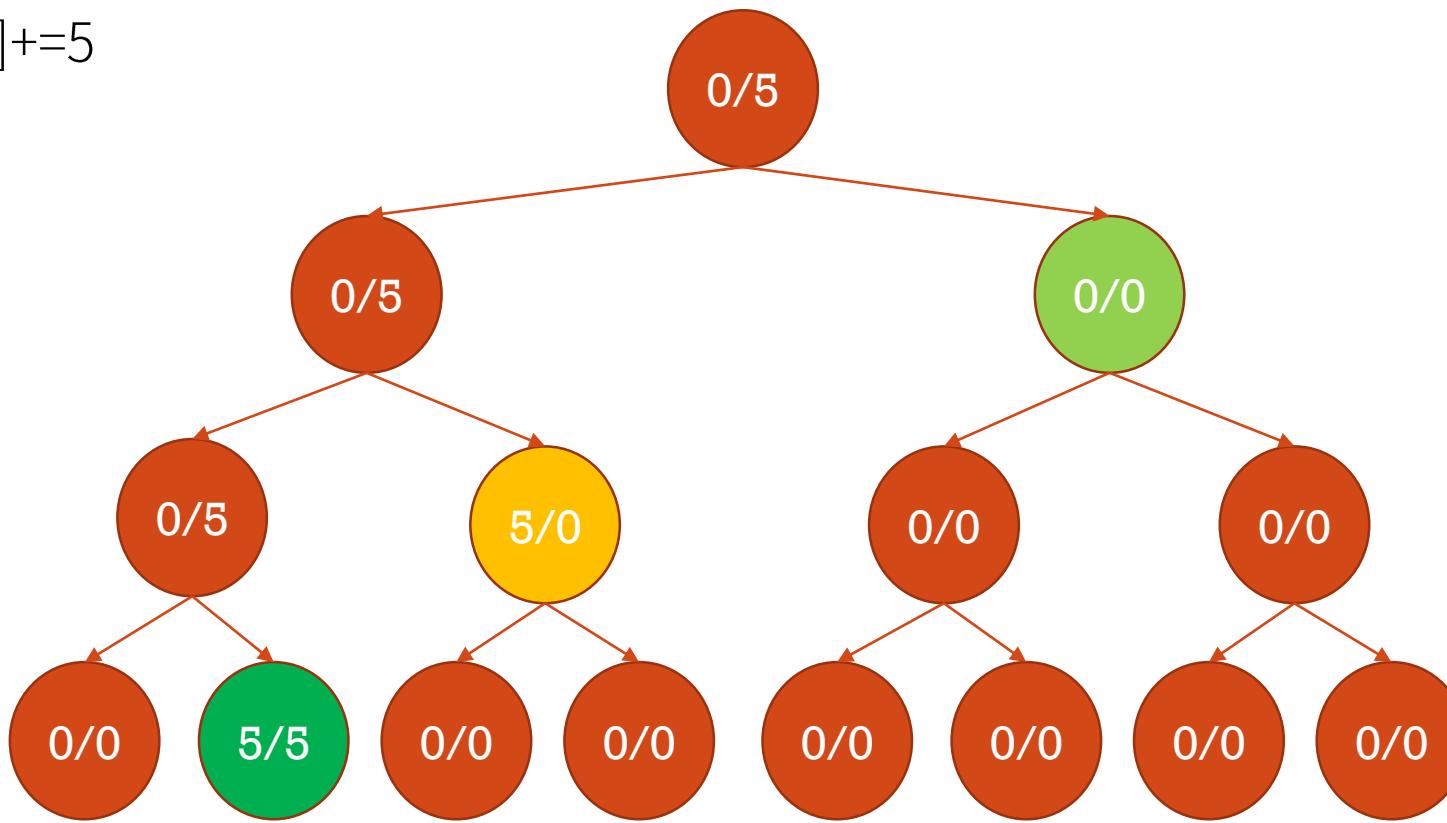
RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

- update(range = [1,7],value = 5)
- unit[2,3]+=5



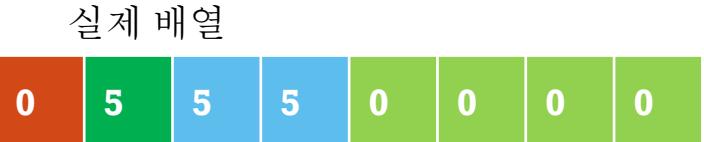
unit/sum



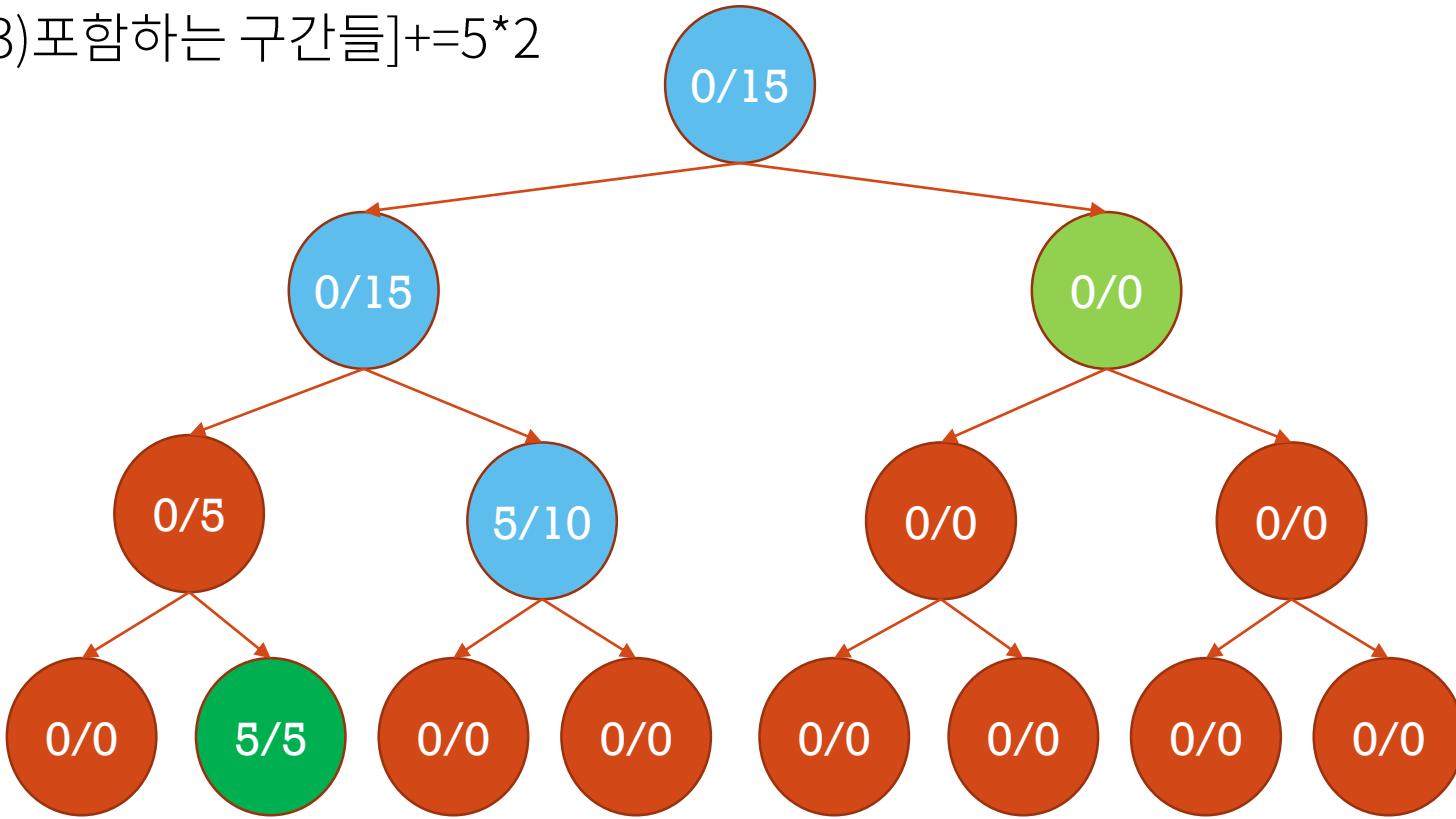
RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

- update(range = [1,7],value = 5)
- sum[(2,3)포함하는 구간들]+=5*2



unit/sum



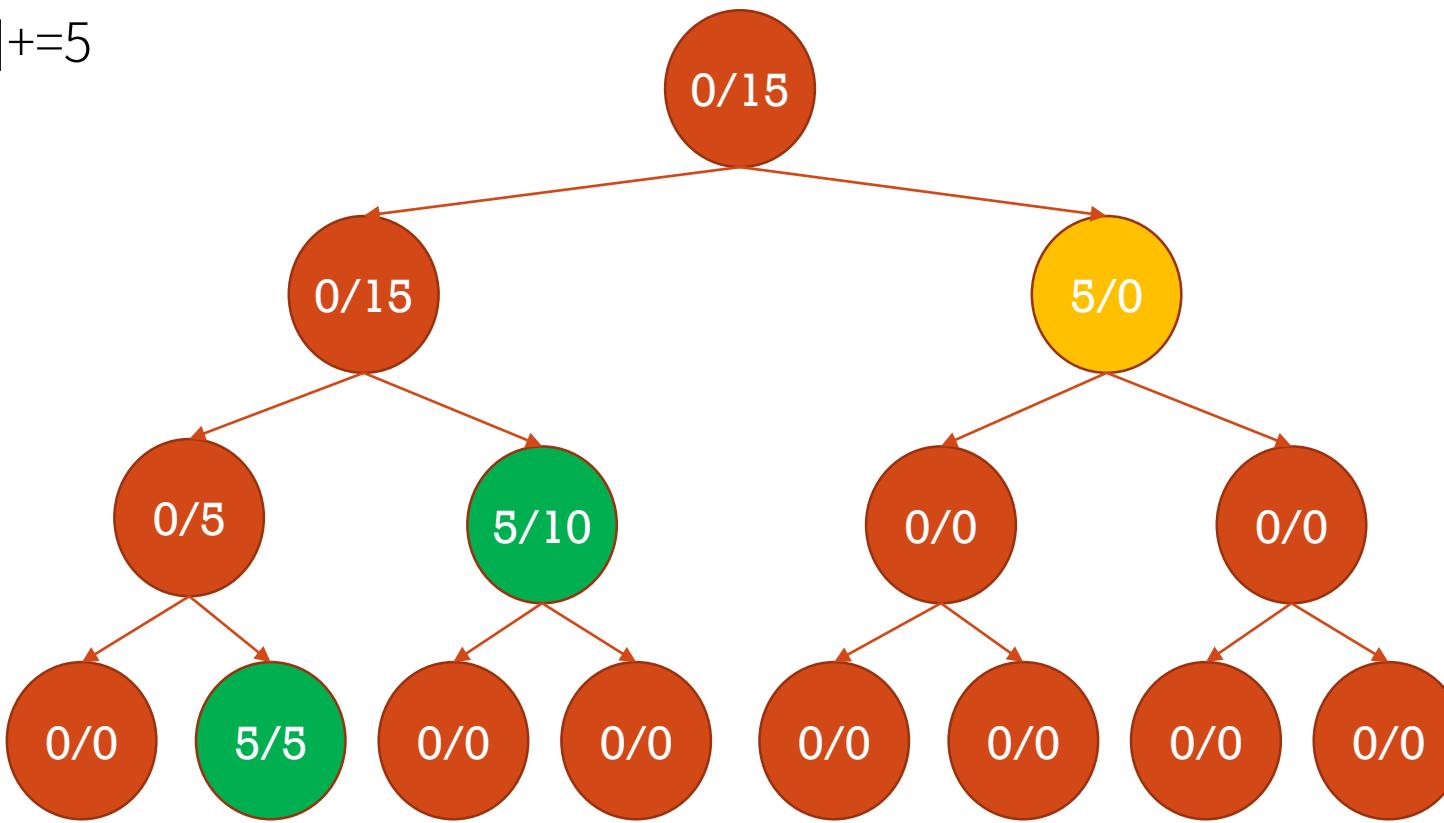
RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

- update(range = [1,7],value = 5)
- unit[4,7]+=5



unit/sum



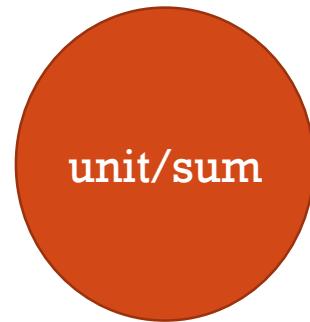
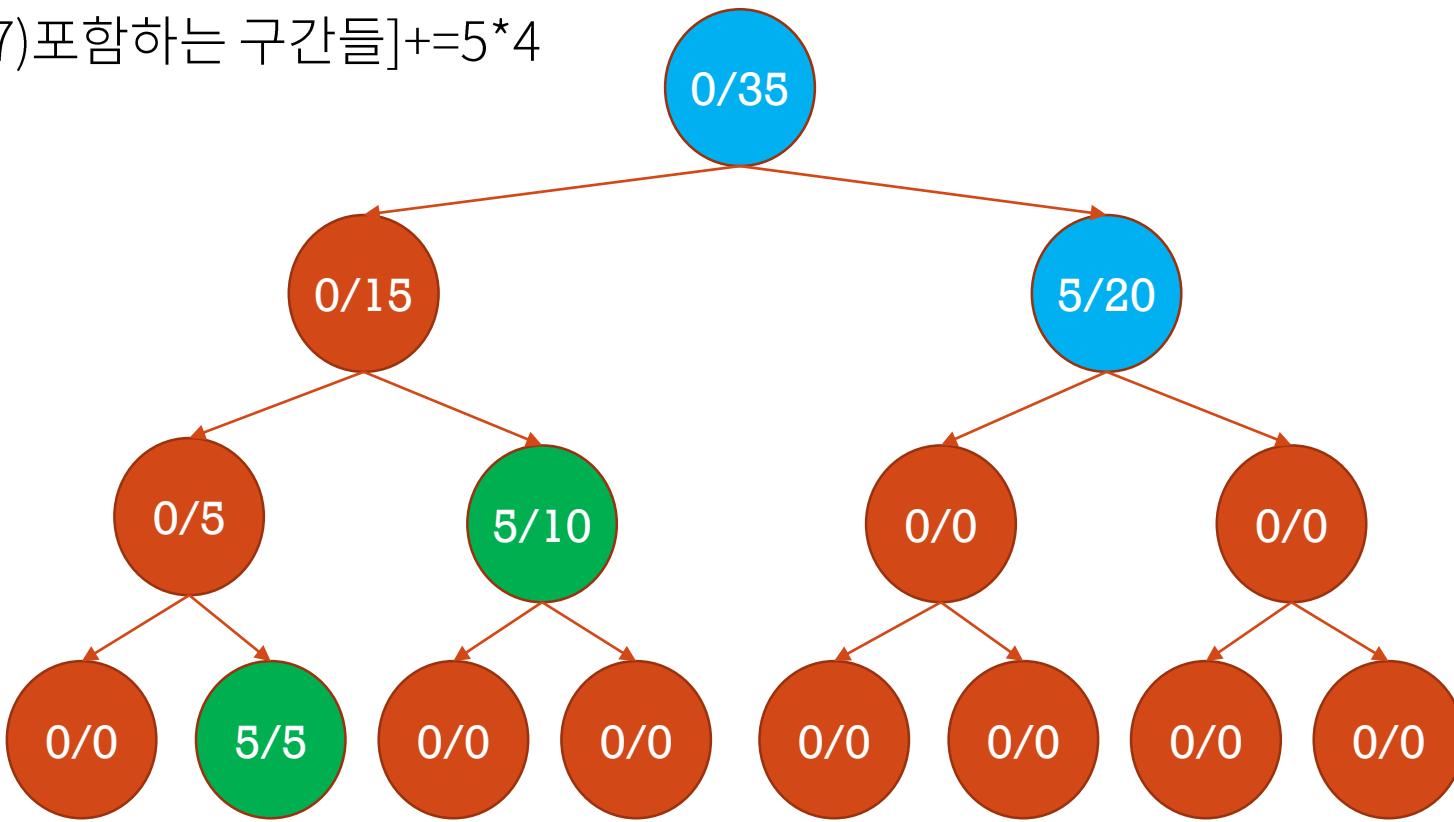
실제 배열

0	5	5	5	5	5	5	5	5
---	---	---	---	---	---	---	---	---

RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

- update(range = [1,7], value = 5)
- sum[(4,7) 포함하는 구간들] += 5 * 4



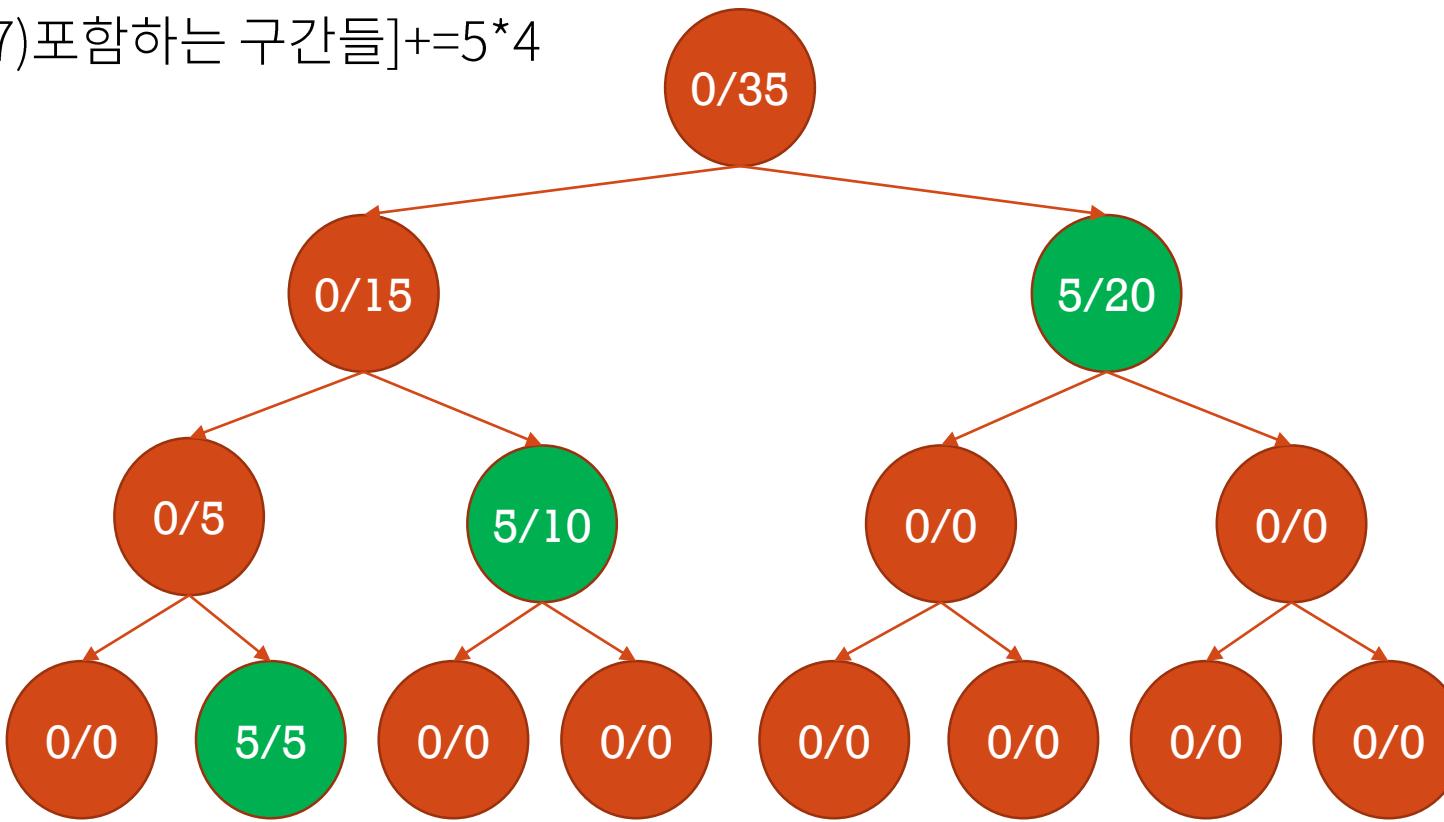
실제 배열

0	5	5	5	5	5	5	5	5
---	---	---	---	---	---	---	---	---

RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

- update(range = [1,7], value = 5)
- sum[(4,7) 포함하는 구간들] += 5 * 4



RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

```
void sumUpdate(int idx,int val){  
    //조상의 sum에 val를 더함  
    while(idx){  
        sum[idx]+=val;  
        idx/=2;  
    }  
}  
  
void update(int left,int right,int val){  
    left+=k; right+=k;  
    int len=1;
```

```
while(left<=right){  
    //가장 상위 구간에 unit에 더하고 조상  
    //의 sum에 더한다.  
    if(left%2==1){unit[left]+=val;sumUp  
    date(left,val*len);}  
    if(right%2==0){unit[right]+=val;su  
    mUpdate(right,val*len);}  
    left=(left+1)/2;  
    right=(right-1)/2;  
    len*=2;  
}
```



RANGE QUERY

- RANGE QUERY/RANGE UPDATE

- Range Query는 다음과 같은 알고리즘으로 작동한다. $O((\log n)^2)$
 1. 이전에 사용한 Range 관련 연산에서 사용한 알고리즘을 통하여 구간을 모두 포함하는 가장 상위 구간을 구한다.
 2. 각 구간별로 아래의 값을 전부 모아 return을 한다.
 1. 해당 구간의 sum을 더한다.
 2. 해당 구간의 모든 조상의 unit에 대해서 $\text{unit}^*(\text{해당 구간의 길이})$ 를 더한다. (조상 구간의 길이 x)

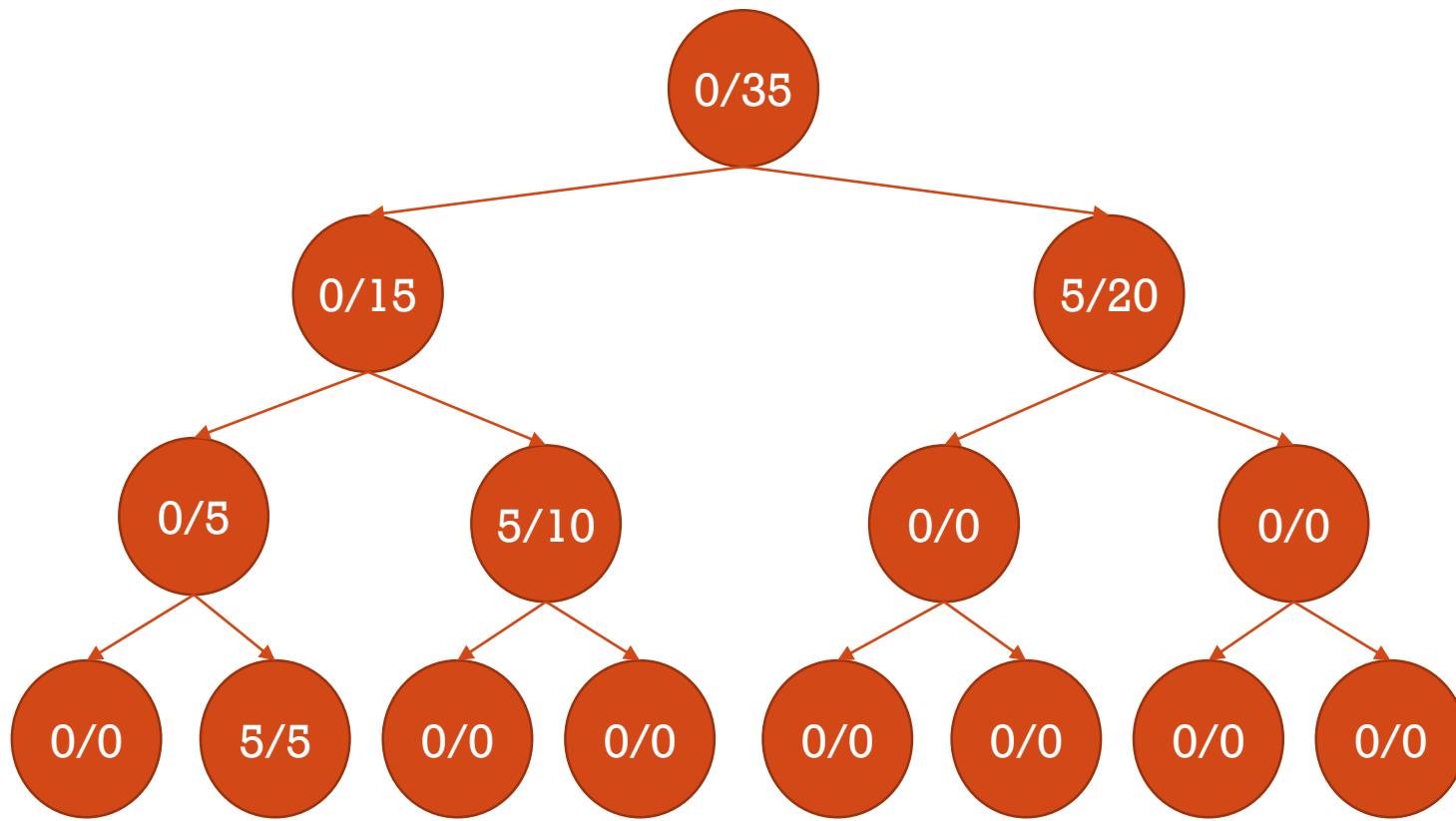


RANGE QUERY

- RANGE QUERY/RANGE UPDATE

- query(2,6)

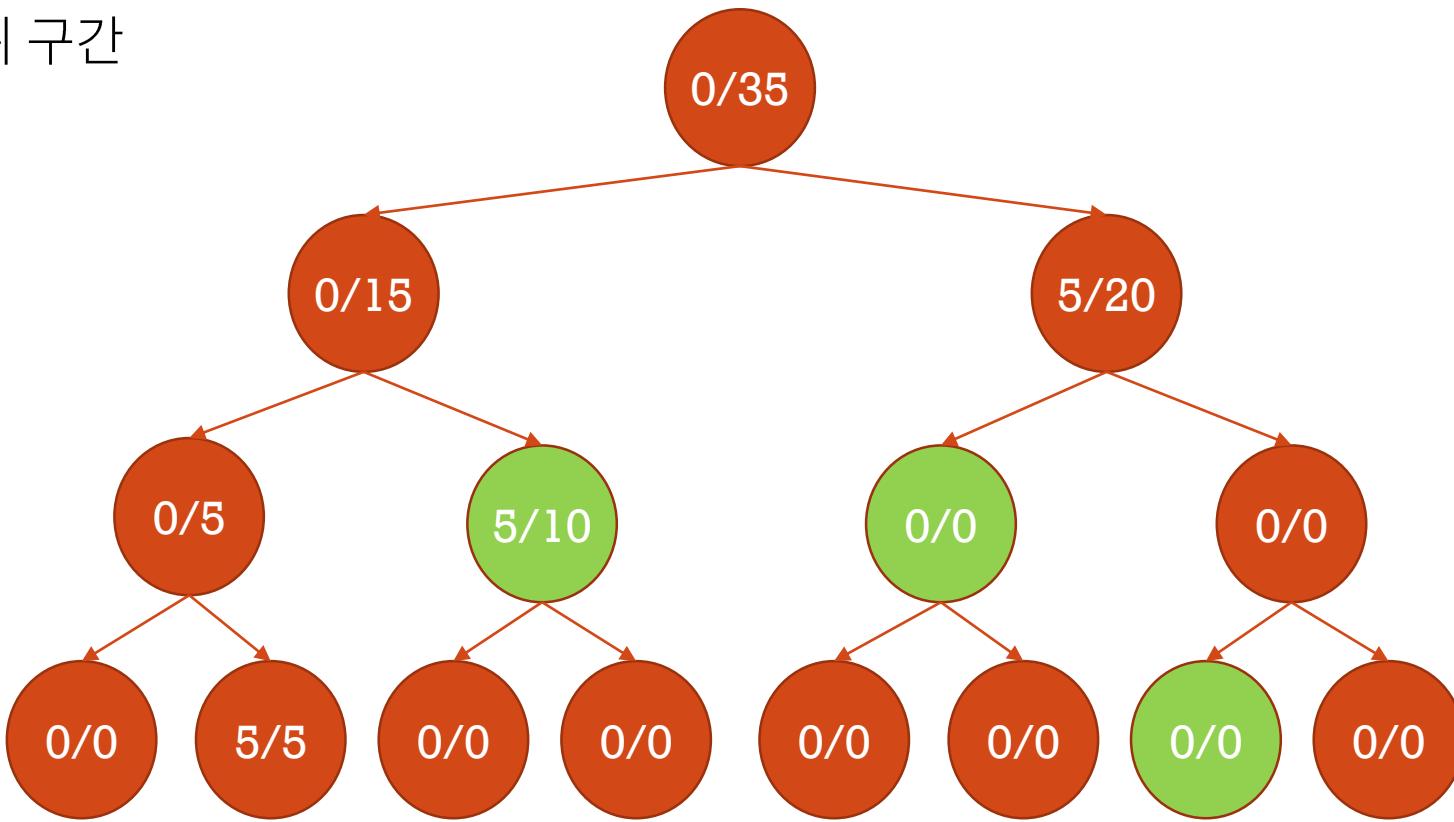
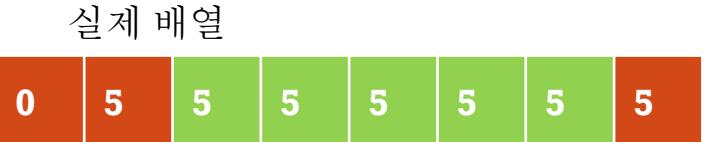
실제 배열								
0	5	5	5	5	5	5	5	5



RANGE QUERY

- RANGE QUERY/RANGE UPDATE

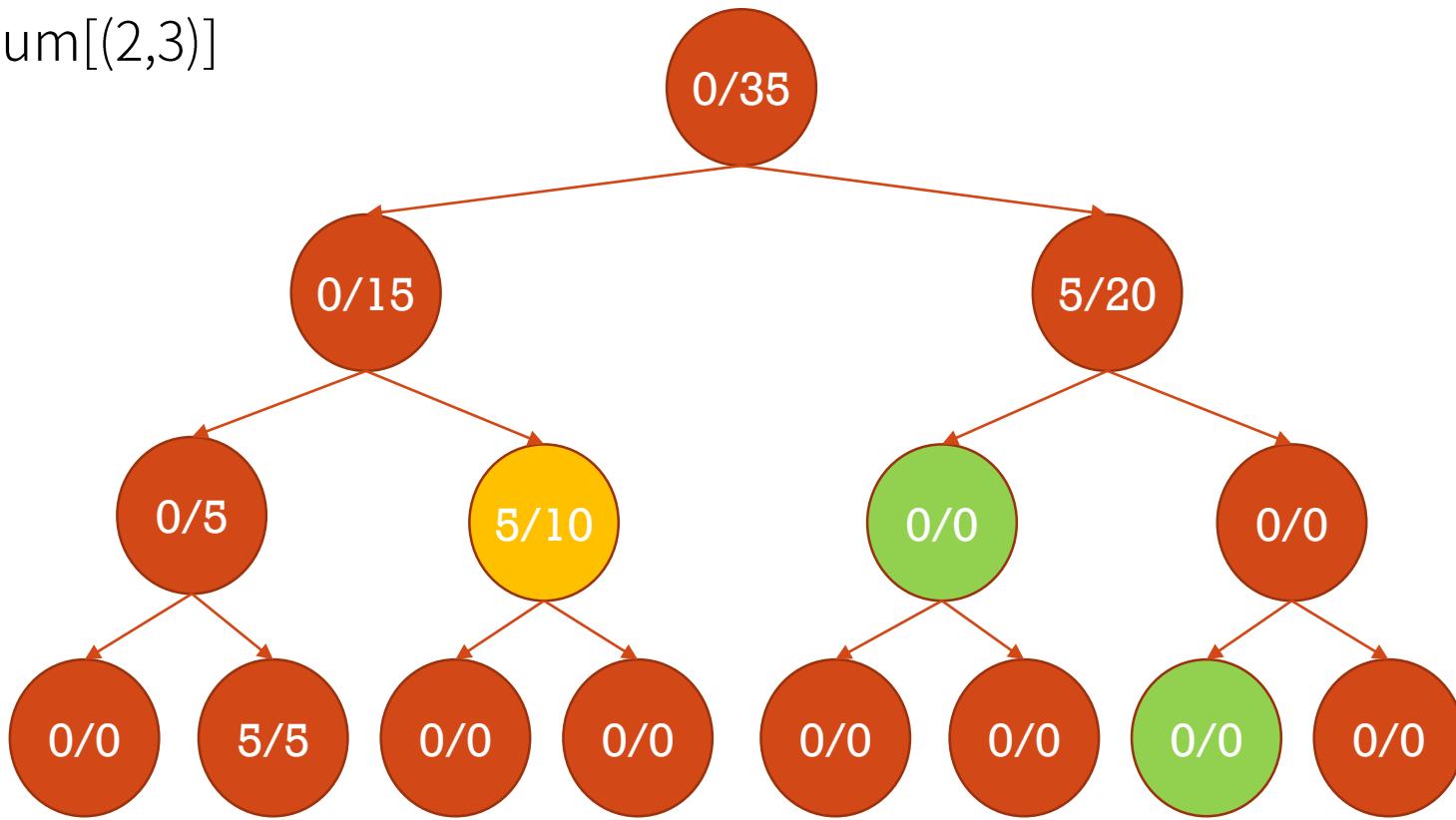
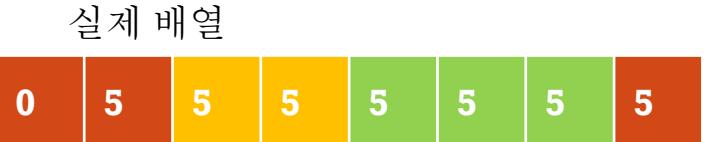
- query(2,6)
- 가상 상위 구간



RANGE QUERY

- RANGE QUERY/RANGE UPDATE

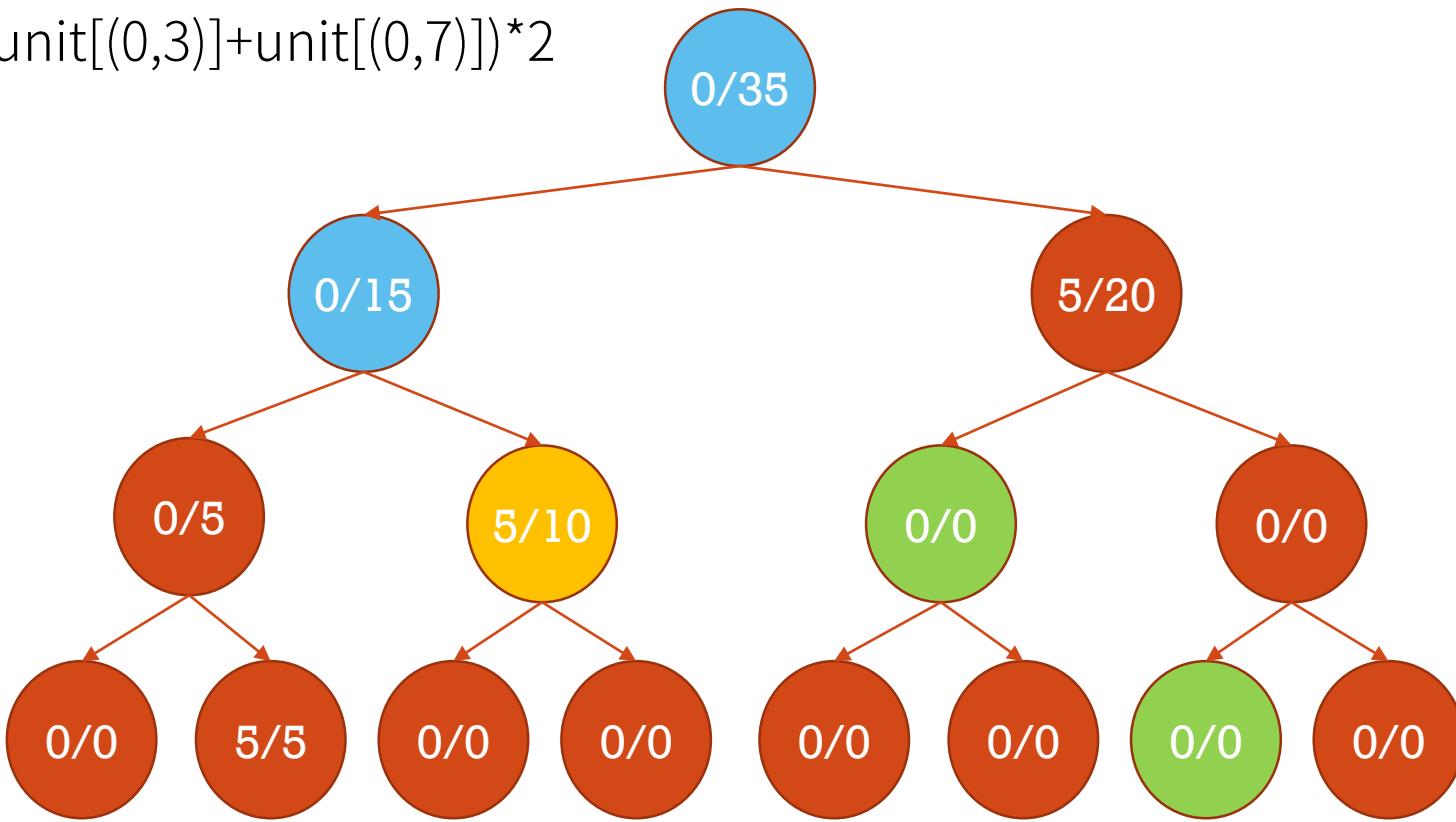
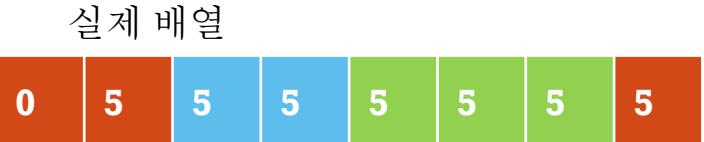
- query(2,6)
- SUM+=sum[(2,3)]



RANGE QUERY

- RANGE QUERY/RANGE UPDATE

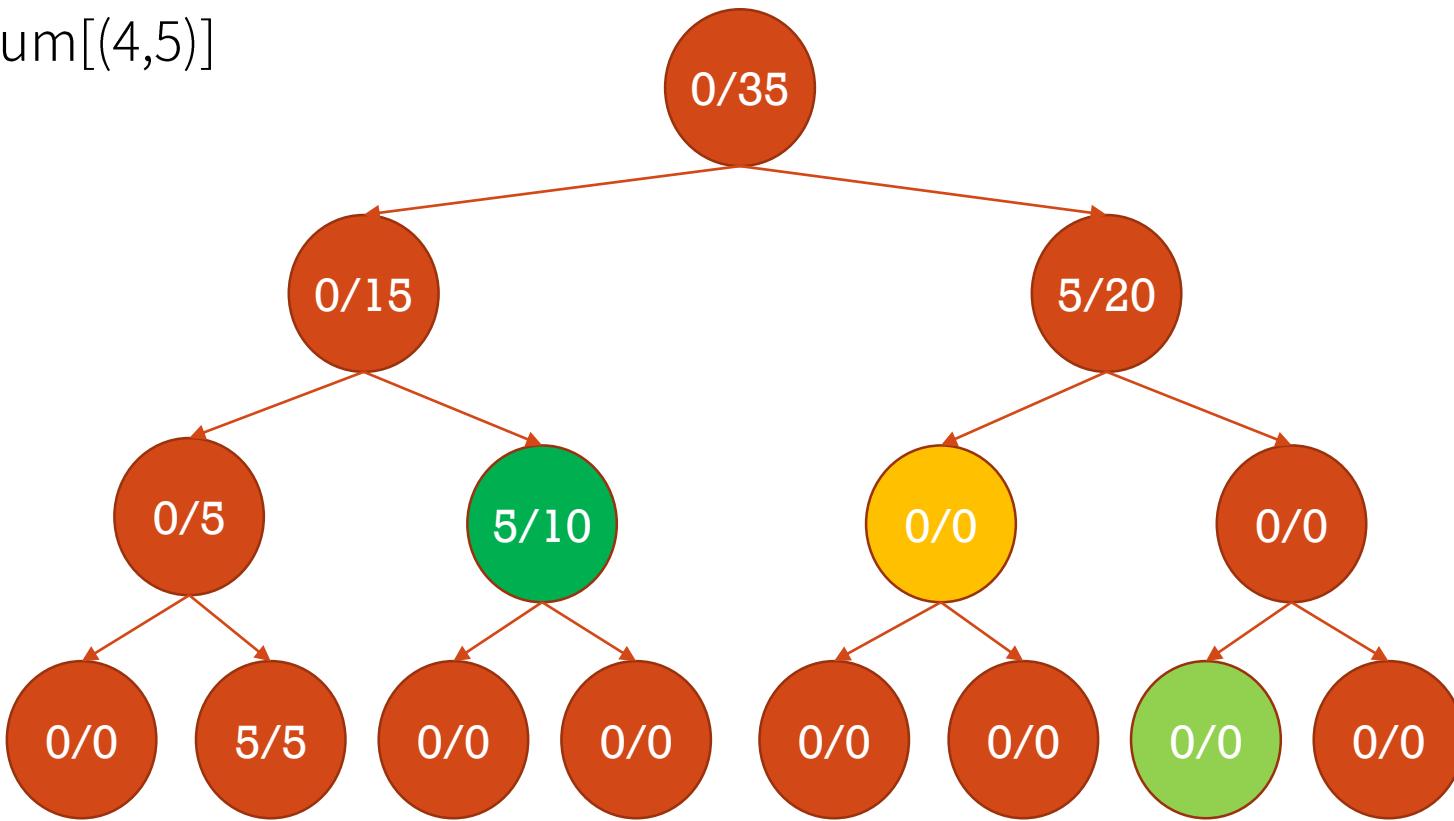
- query(2,6)
- $\text{SUM}+=\text{unit}[(0,3)]+\text{unit}[(0,7)]*2$



RANGE QUERY

- RANGE QUERY/RANGE UPDATE

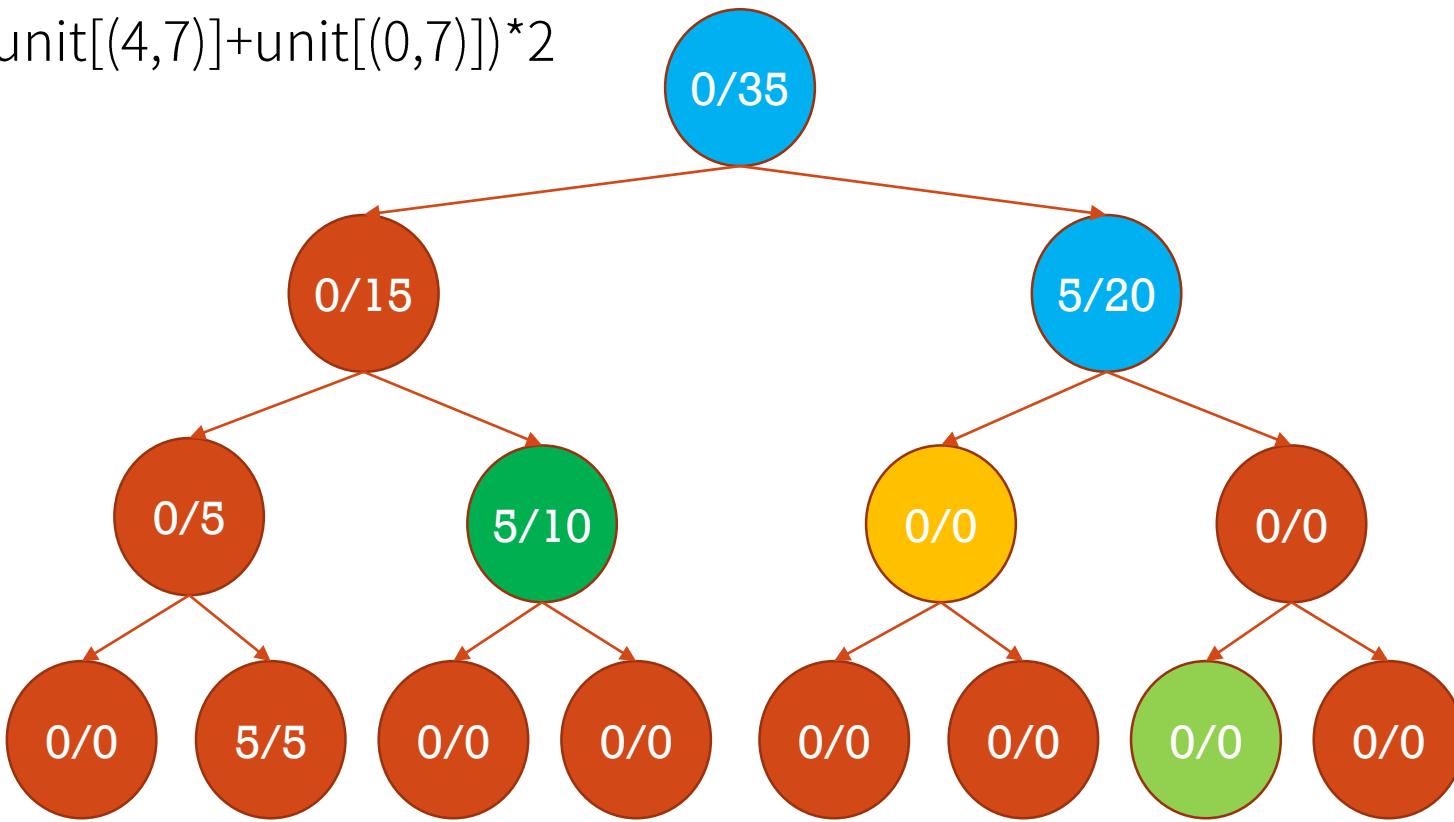
- query(2,6)
- SUM+=sum[(4,5)]



RANGE QUERY

- RANGE QUERY/RANGE UPDATE

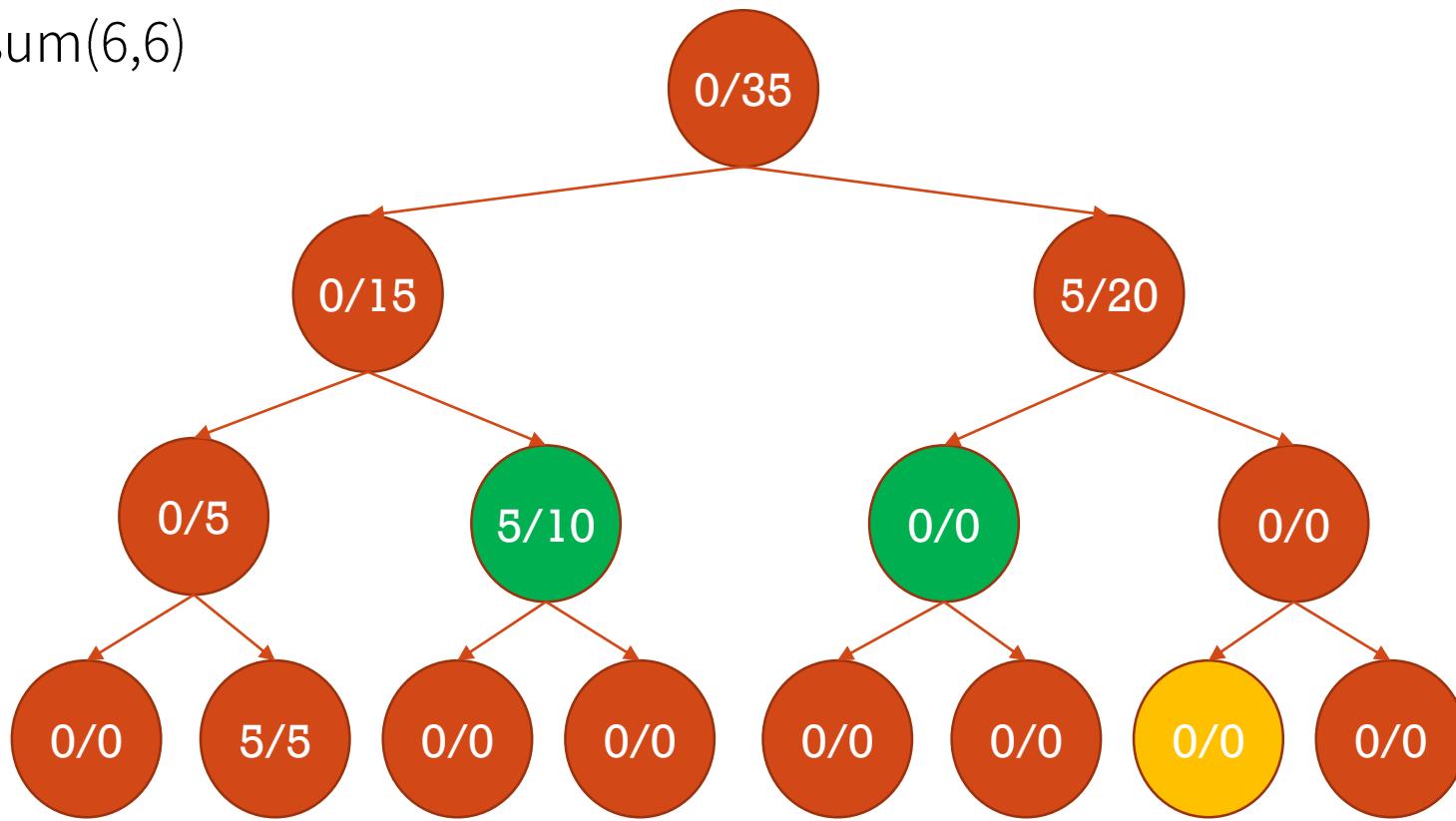
- query(2,6)
- $\text{SUM}+=\text{unit}[(4,7)]+\text{unit}[(0,7)]*2$



RANGE QUERY

- RANGE QUERY/RANGE UPDATE

- query(2,6)
- SUM+=sum(6,6)



unit/sum

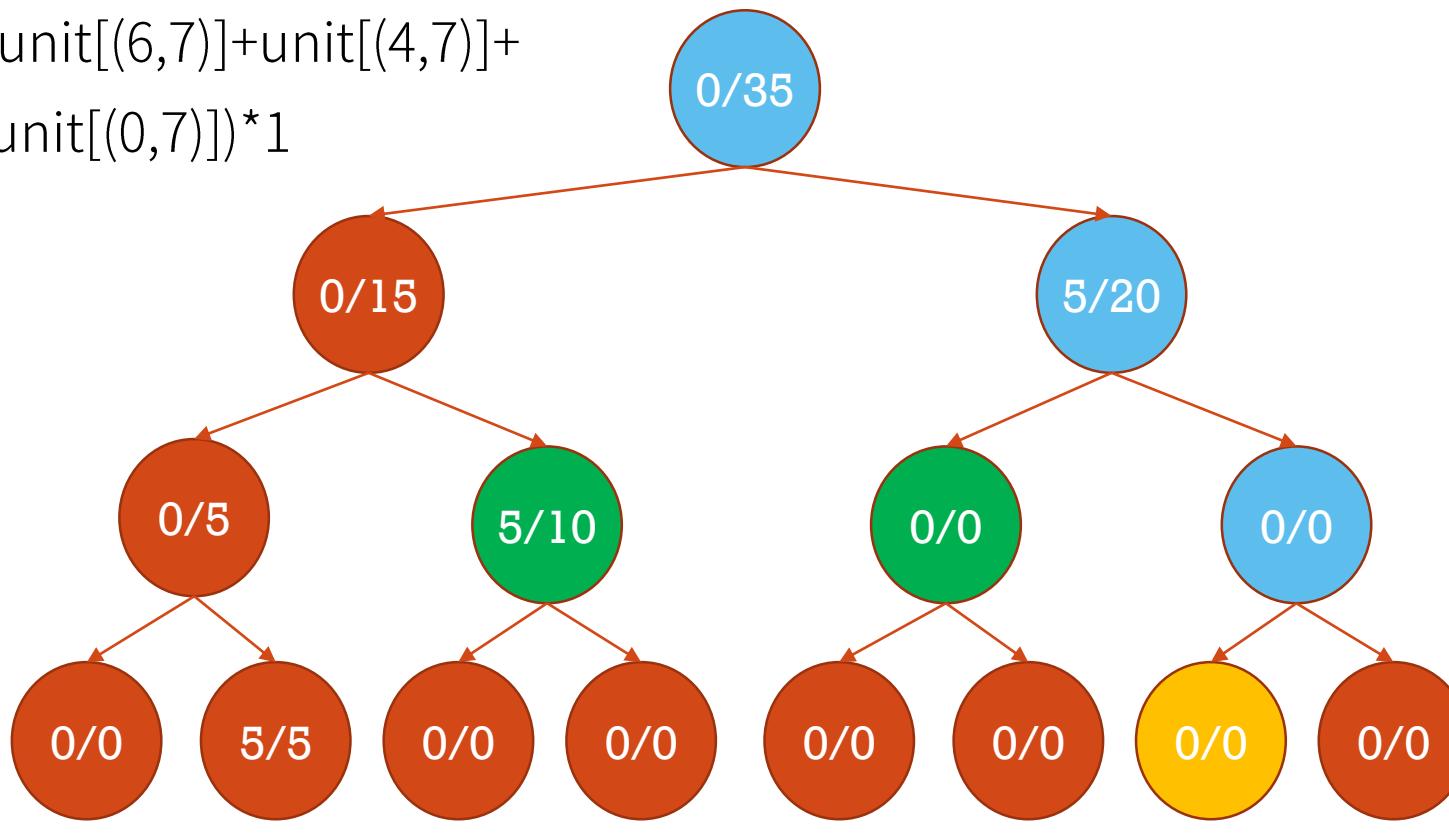
SUM
20



RANGE QUERY

- RANGE QUERY/RANGE UPDATE

- query(2,6)
- $\text{SUM} += (\text{unit}[(6,7)] + \text{unit}[(4,7)] + \text{unit}[(0,7)]) * 1$



실제 배열

0	5	5	5	5	5	5	5	5
---	---	---	---	---	---	---	---	---

unit/sum

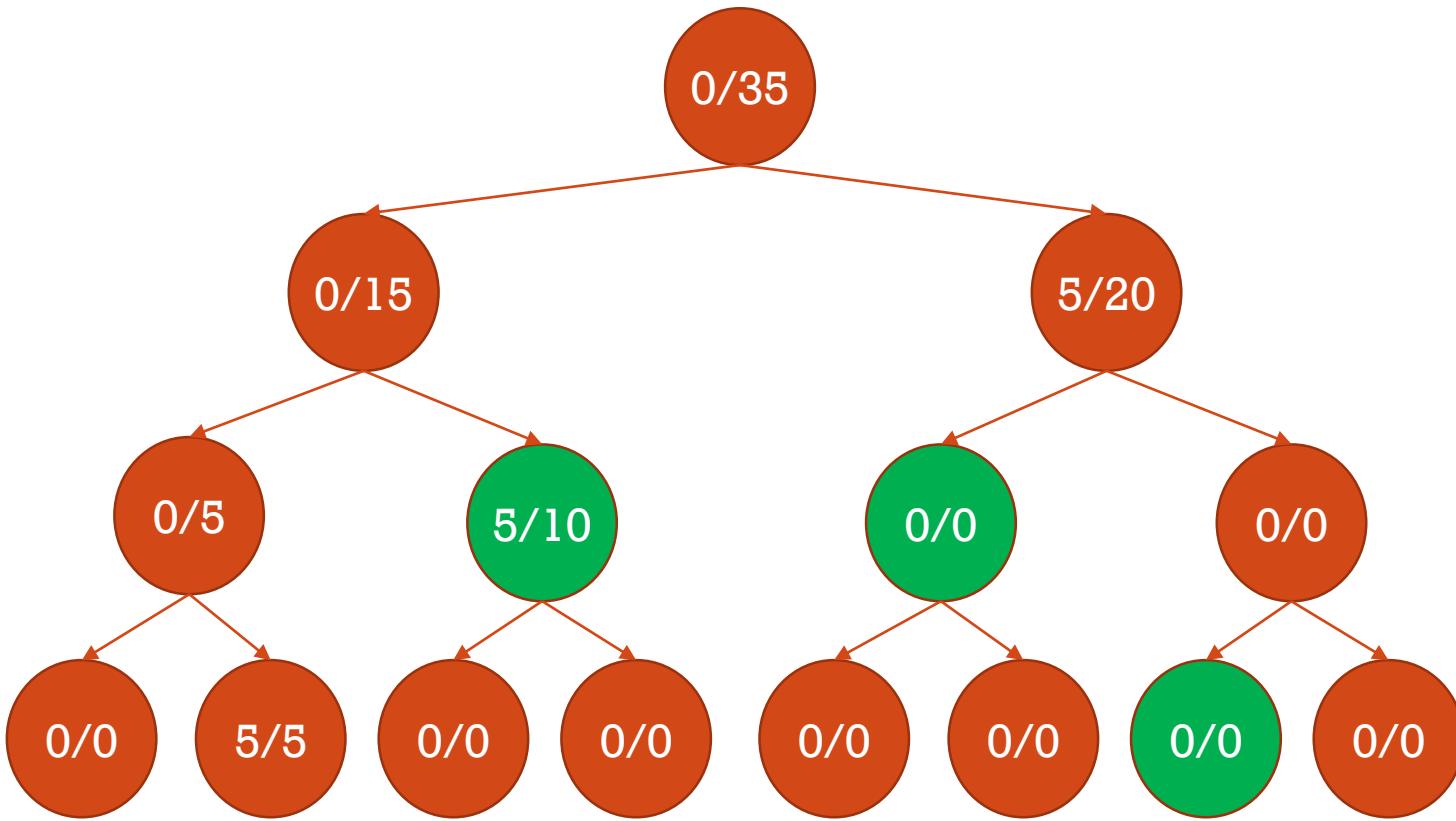
SUM
25



RANGE QUERY

- RANGE QUERY/RANGE UPDATE

- query(2,6) = 25



RANGE QUERY

- RANGE QUERY/RANGE UPDATE

```
int unitQuery(int idx,int len){  
    //조상의 unit을 len길이만큼 가져옴  
    int ret=0;  
    while(idx){  
        ret+=unit[idx]*len;  
        idx/=2;  
    }  
    return ret;  
}  
  
int query(int left,int right){  
    left+=k; right+=k;  
    int len=1;
```

```
    int ret=0;  
    while(left<=right){  
        //가장 상위 구간에서 sum과 조상의 unit  
        //에 대한 값을 가져온다  
        if(left%2==1)ret+=sum[left]+unitQuery(  
            left/2,len);  
        if(right%2==0)ret+=sum[right]+unitQu  
        ery(right/2,len);  
        left=(left+1)/2;  
        right=(right-1)/2;  
    }  
    return ret;  
}
```



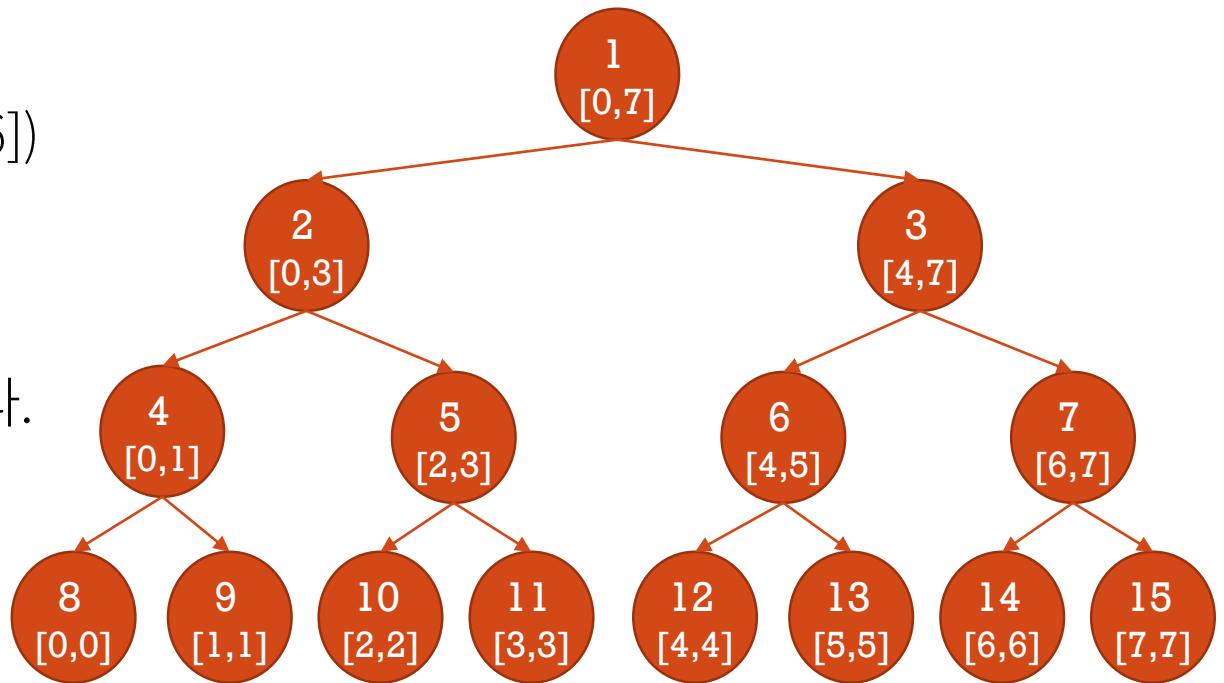
RANGE QUERY/RANGE UPDATE

- 위의 설명한 방식은 query나 update에 대하여 $O((\log n)^2)$ 의 시간 복잡도를 가진다.
- 하지만 해당 방법을 좀 더 최적화하면 $O(\log n)$ 의 시간 복잡도로 구현할 수 있다.
- 기존 방법은 가장 상위 구간을 구한 뒤, 각 구간별로 모든 조상에 대해 처리를 한다.
- 하지만 그 조상들을 잘 관찰하면 왼쪽에서 찾은 구간들과 오른쪽에서 찾은 구간들로 나누어 보면, 깊이가 깊은 구간에 얹은 구간의 조상이 모두 포함된다는 것을 알 수 있다.



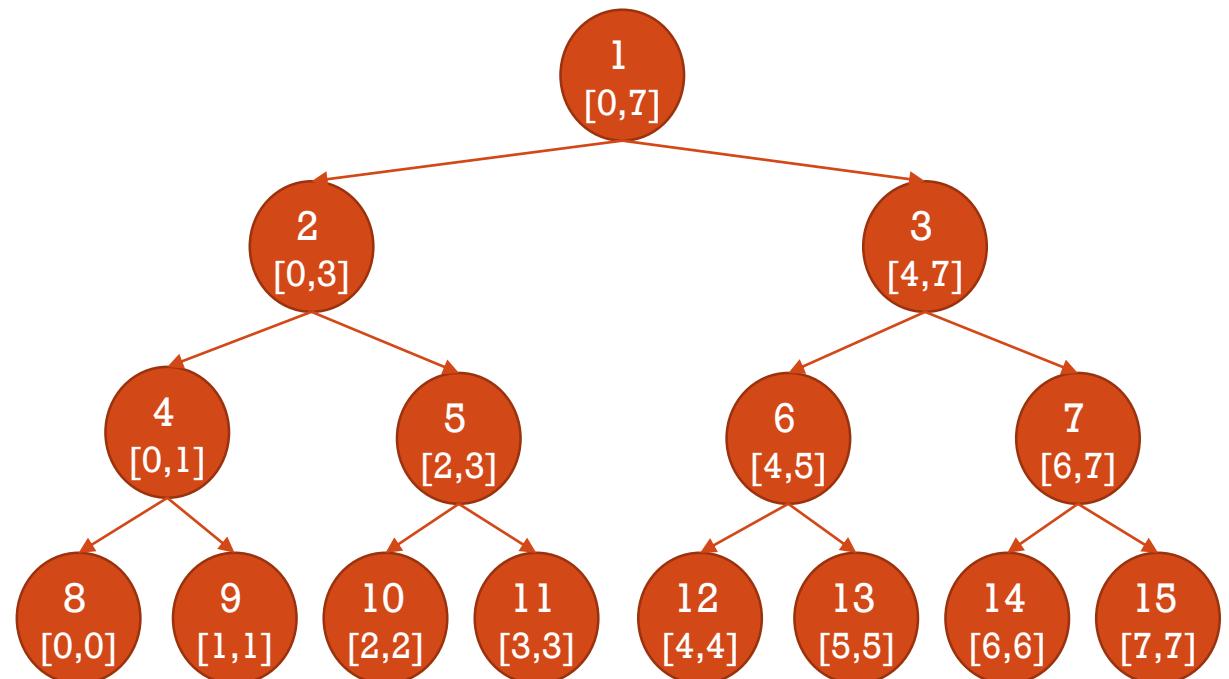
RANGE QUERY/RANGE UPDATE

- query(1,6)을 하면
- 왼쪽에서 찾은 구간은 5([2,3]), 9([1,1])
- 오른쪽에서 찾은 구간은 6([4,5]), 14([6,6])
- 5의 조상 {1,2}는 9의 조상 {1,2,4}에 포함되는 것을 확인 할 수 있다.
- 오른쪽에서 찾은 구간 또한 마찬가지이다.



RANGE QUERY/RANGE UPDATE

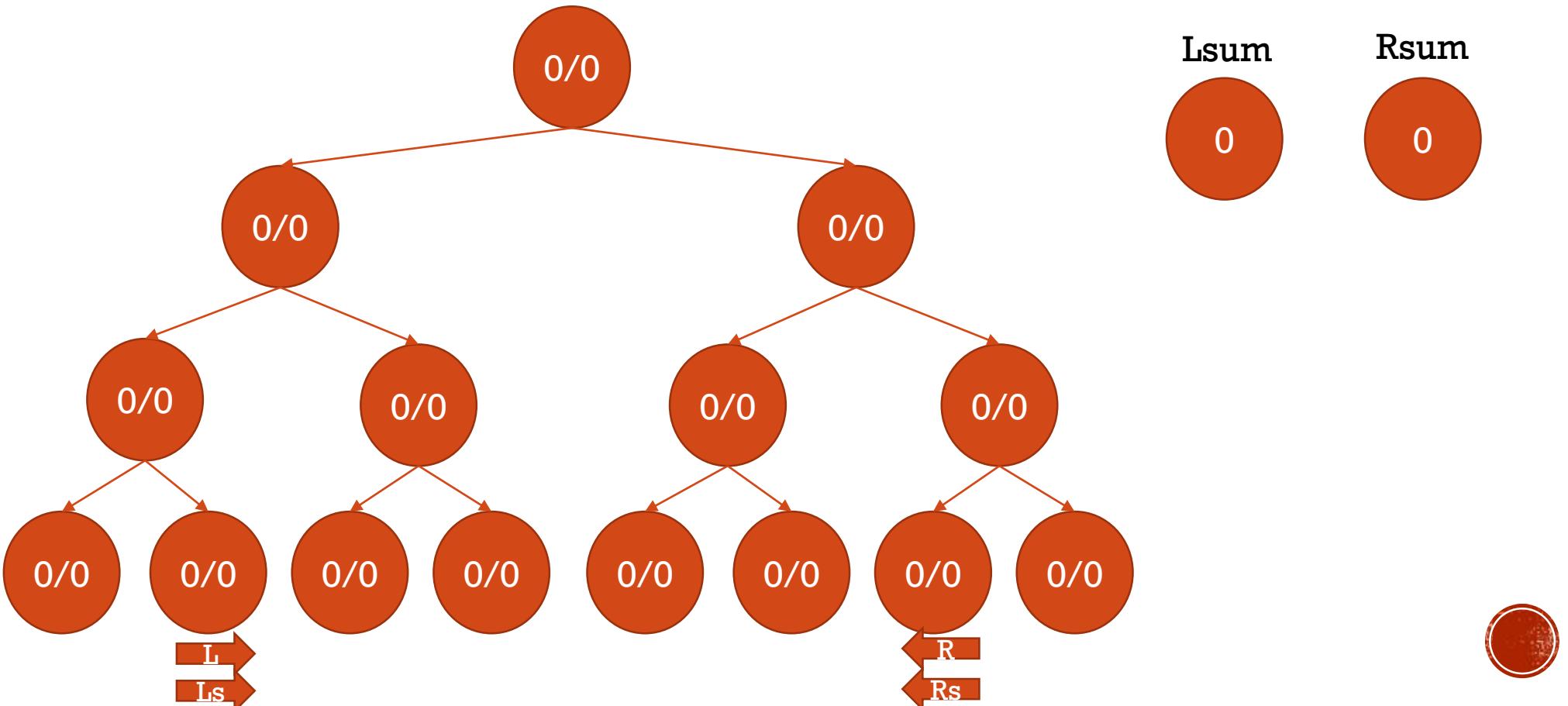
- 가장 상위 구간을 찾는 L,R과 달리 바로 조상으로만 올라가는 위치를 기록할 변수도 필요하다.
- update의 경우 왼쪽, 오른쪽 각각 조상들의 sum에 더해야 할 값(unit^*len)들을 Lsum ,Rsum이라는 곳에 따로 저장하고 올라가면서 sum에 더한다. 단, 자기 자신에 더할 것은 따로 처리해 준다.
- query의 경우 unit^* 구간의 길이이므로 unit에 곱해져야 할 길이를 Llen, Rlen이라는 곳에 따로 저장 해두어야 한다.



RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

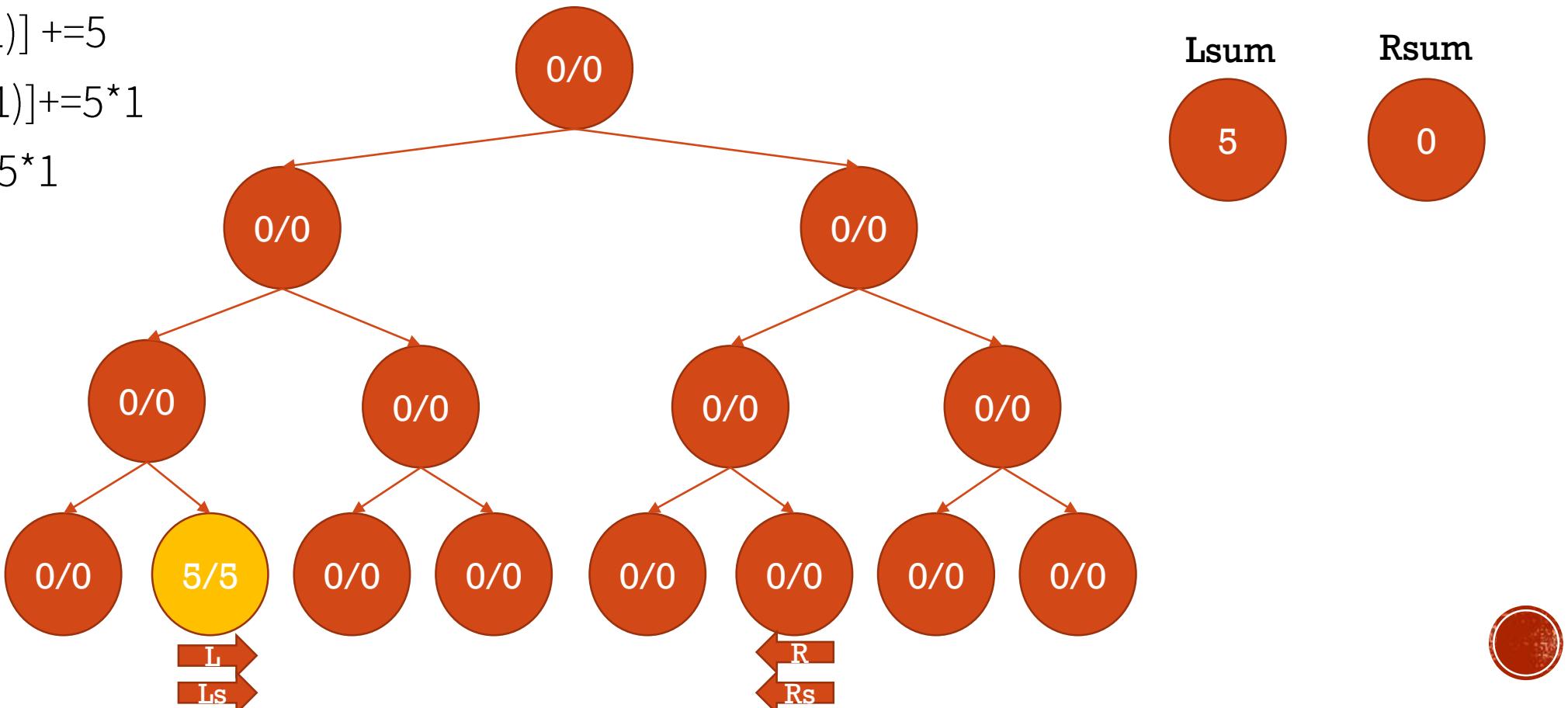
- update(range = [1,6],value = 5)



RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

- update(range = [1,5],value = 5)
 - unit[(1,1)] +=5
 - sum[(1,1)]+=5*1
 - Lsum+=5*1



RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

- update(range = [1,5], value = 5)

실제 배열

0	5	0	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---

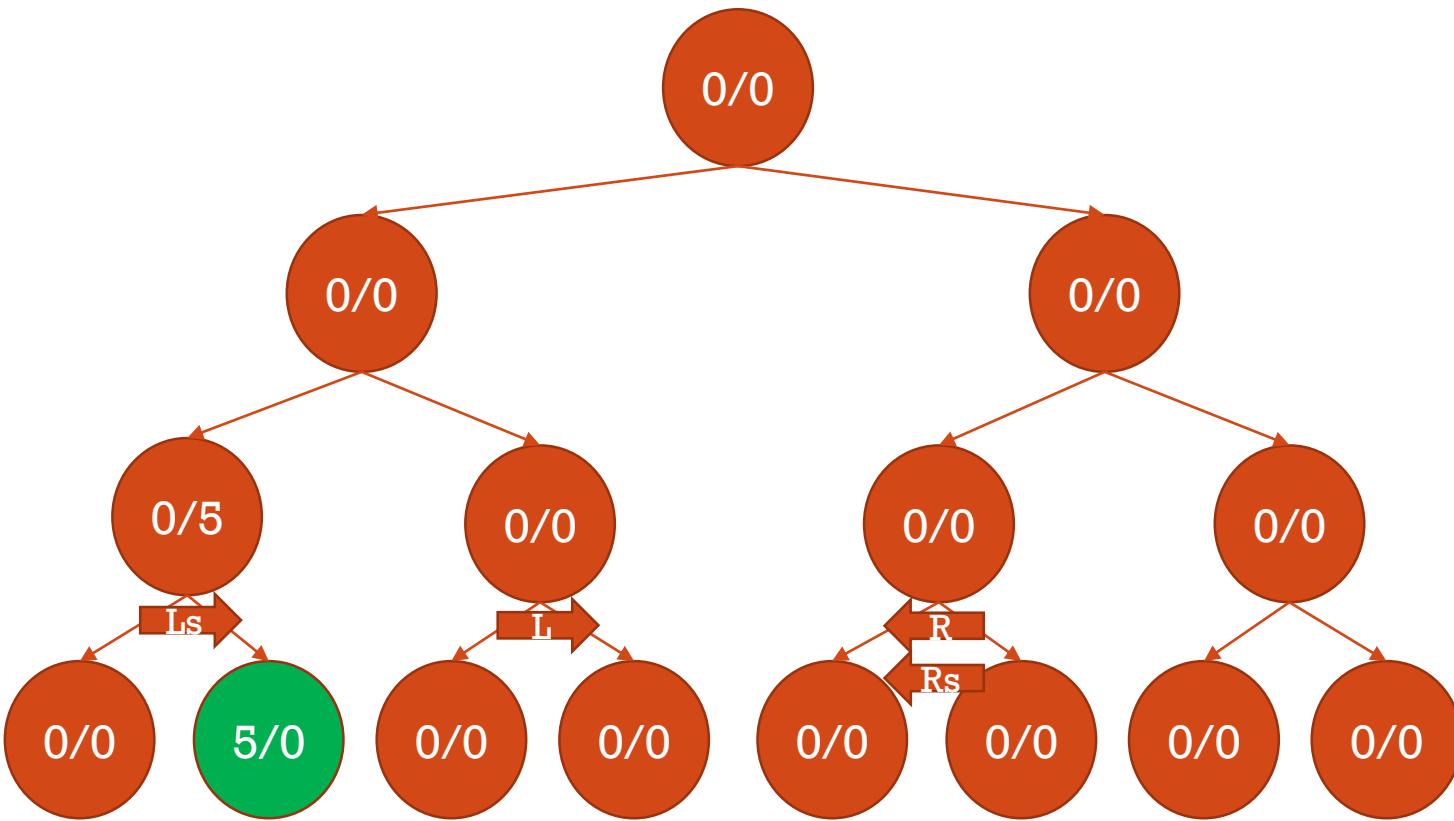
unit/
sum

Lsum

5

Rsum

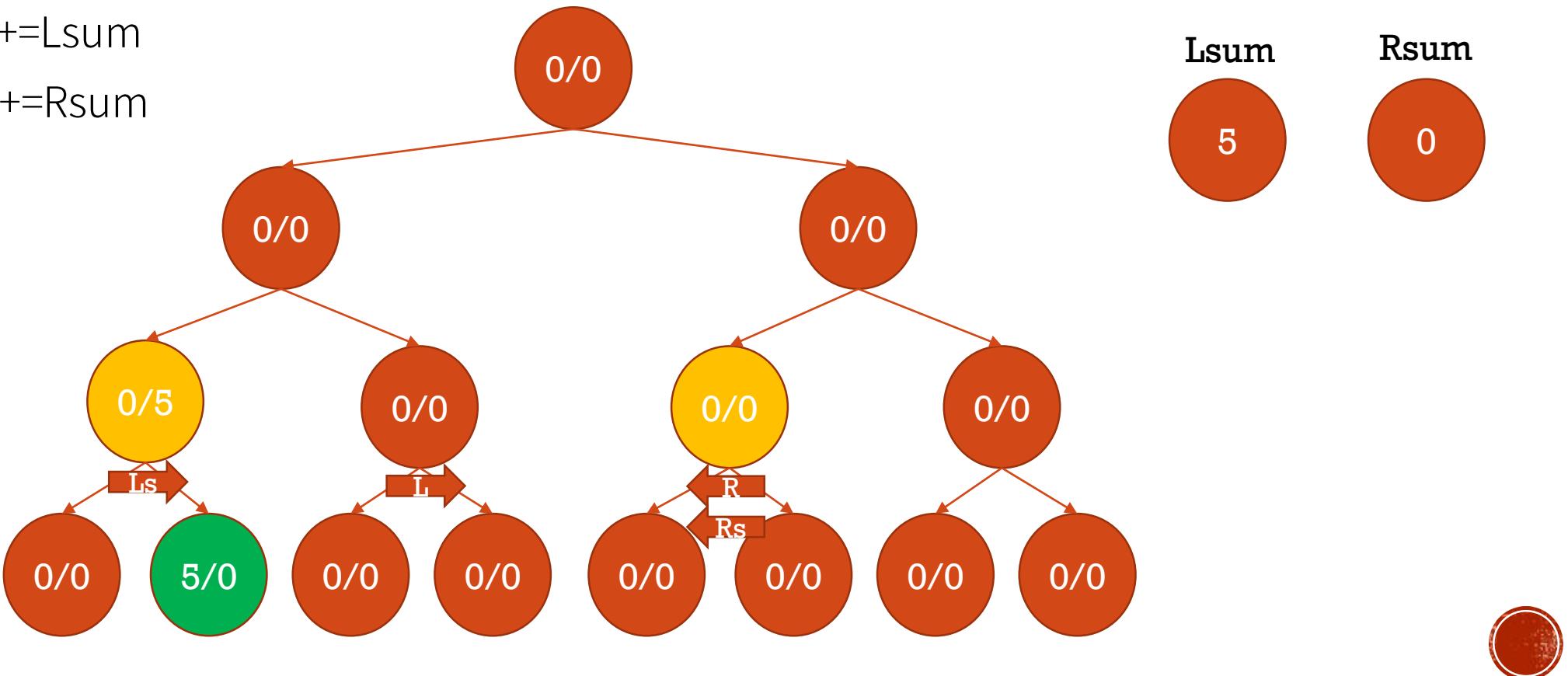
0



RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

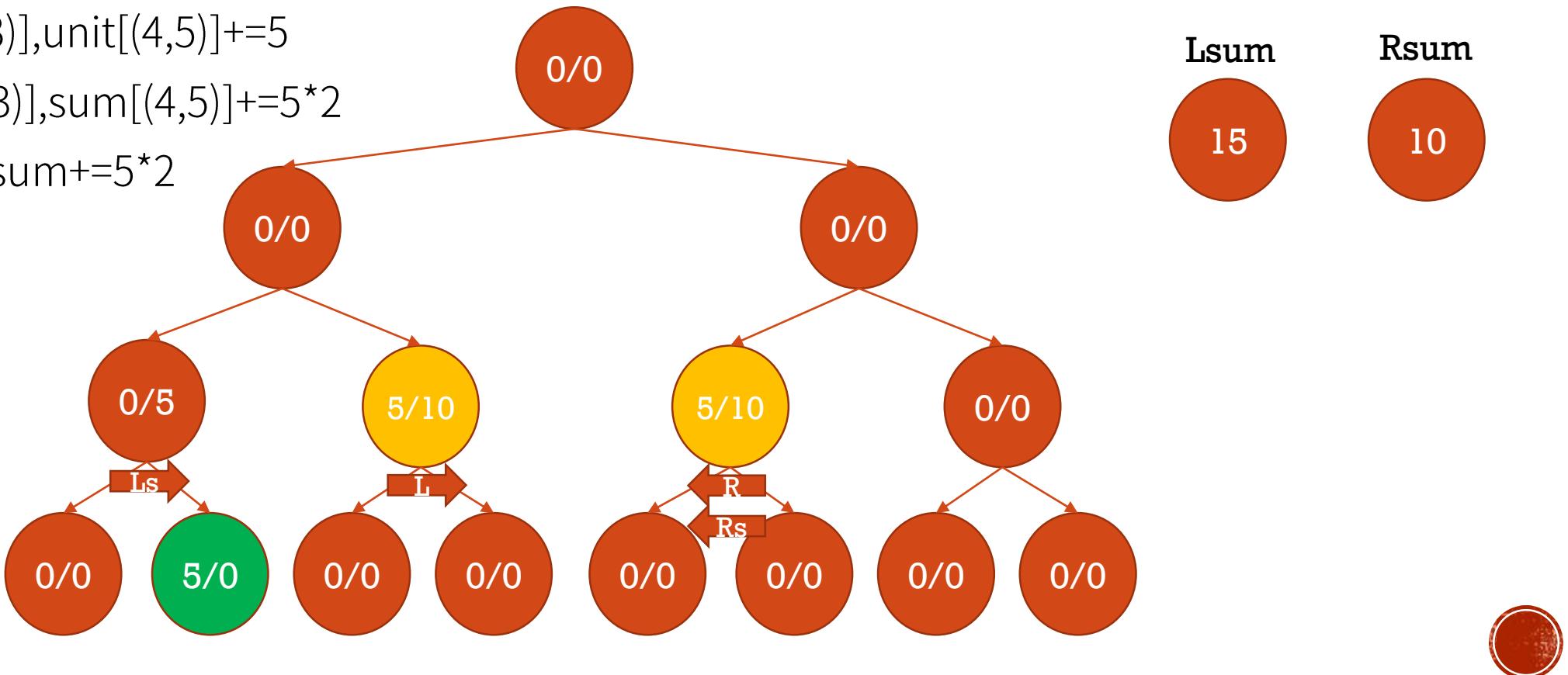
- `update(range = [1,5],value = 5)`
 - `sum[Ls]+=Lsum`
 - `sum[Rs]+=Rsum`



RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

- update(range = [1,5],value = 5)
 - unit[(2,3)],unit[(4,5)]+=5
 - sum[(2,3)],sum[(4,5)]+=5*2
 - Lsum,Rsum+=5*2



RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

- update(range = [1,5], value = 5)

실제 배열

0	5	5	5	5	5	5	0	0
---	---	---	---	---	---	---	---	---

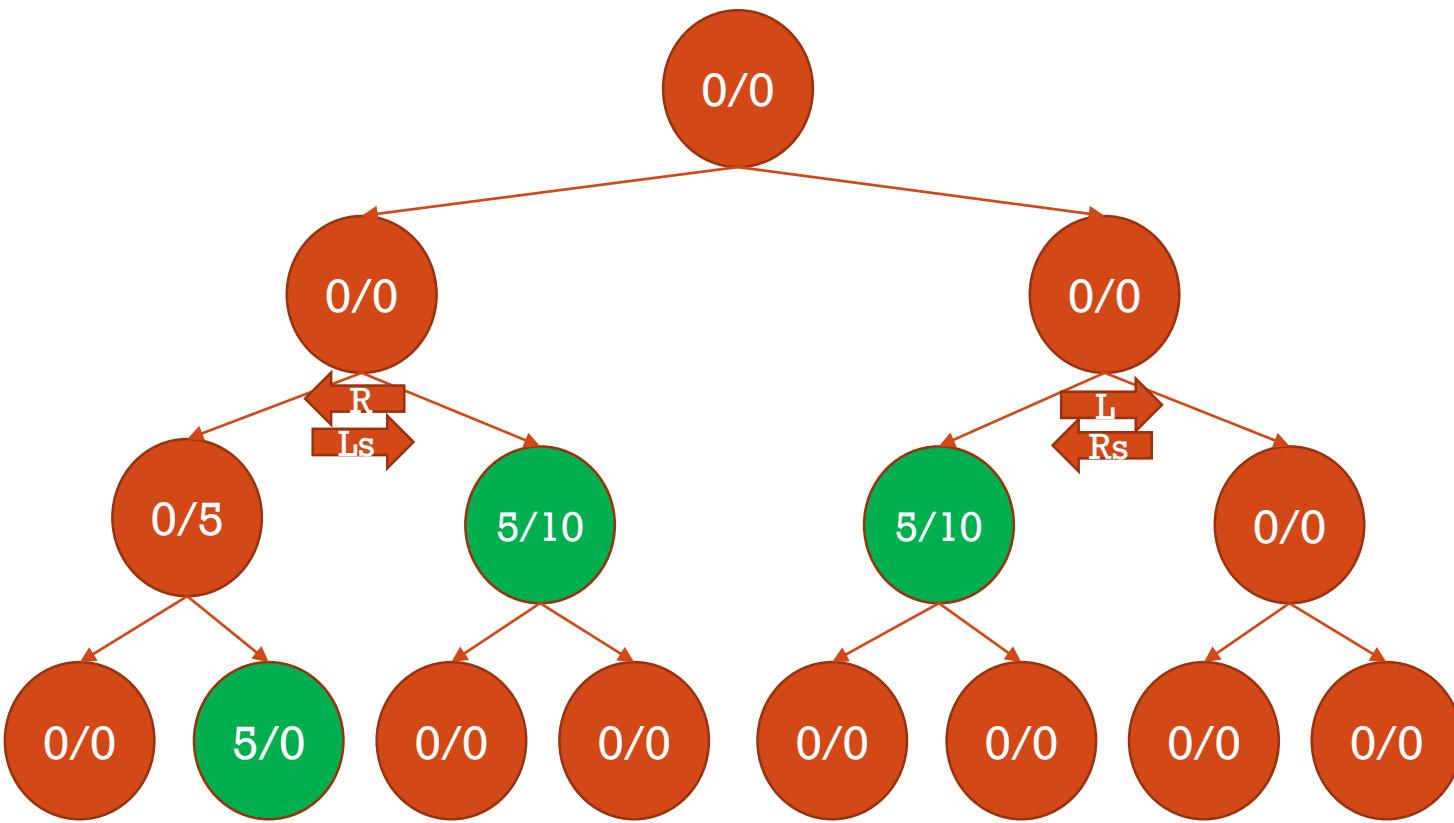
unit/
sum

Lsum

15

Rsum

10



RANGE UPDATE

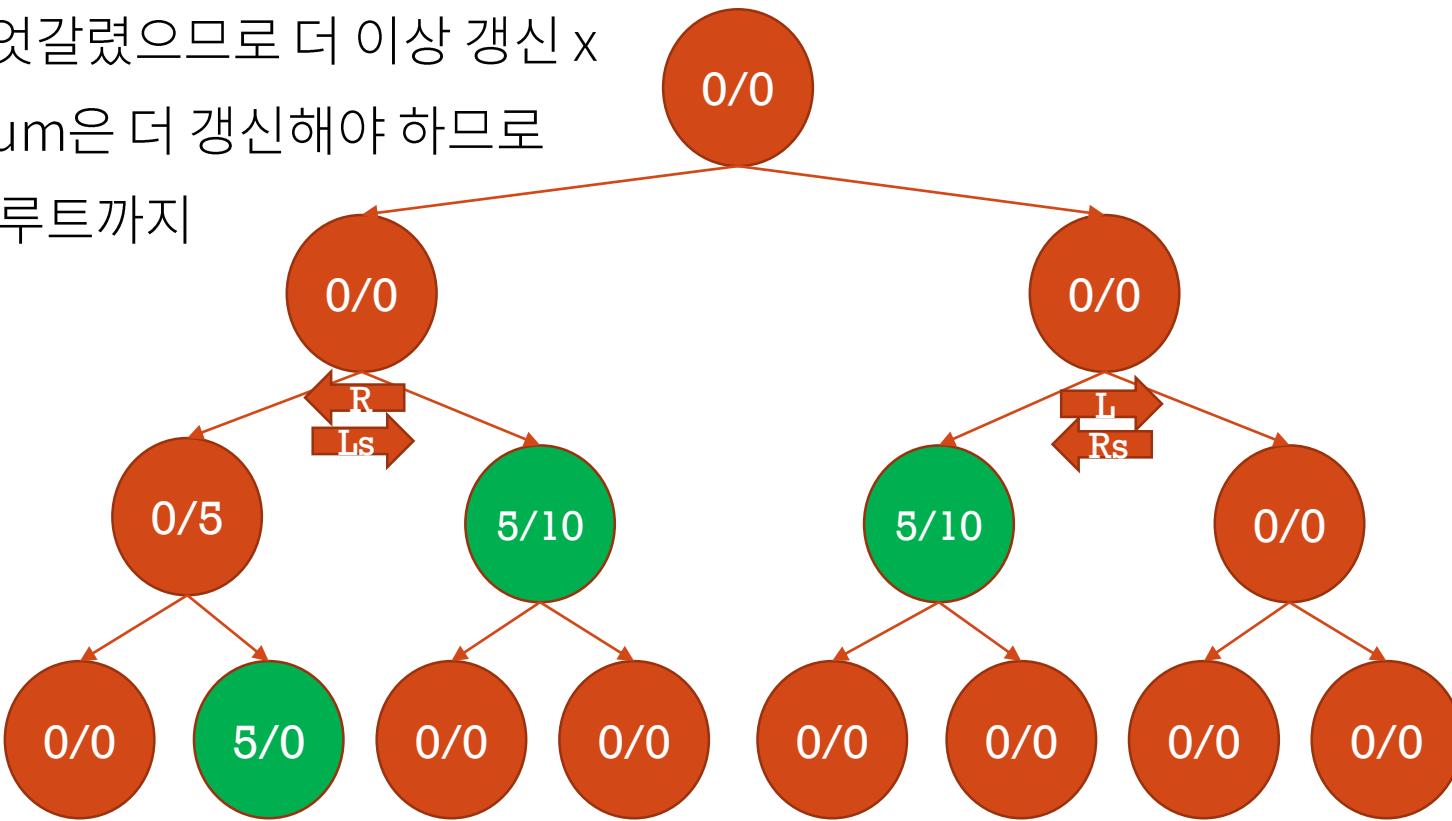
- RANGE QUERY/RANGE UPDATE

- update(range = [1,5], value = 5)
- L과 R은 엇갈렸으므로 더 이상 갱신 X
- 하지만 sum은 더 갱신해야 하므로
Ls와 Rs는 루트까지
올라간다.

실제 배열
0 5 5 5 5 5 0 0

unit/
sum

Lsum Rsum
15 10



RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

- update(range = [1,5],value = 5)
- sum[Ls]+=Lsum
- sum[Rs]+=Rsum

실제 배열

0	5	5	5	5	5	5	0	0
---	---	---	---	---	---	---	---	---

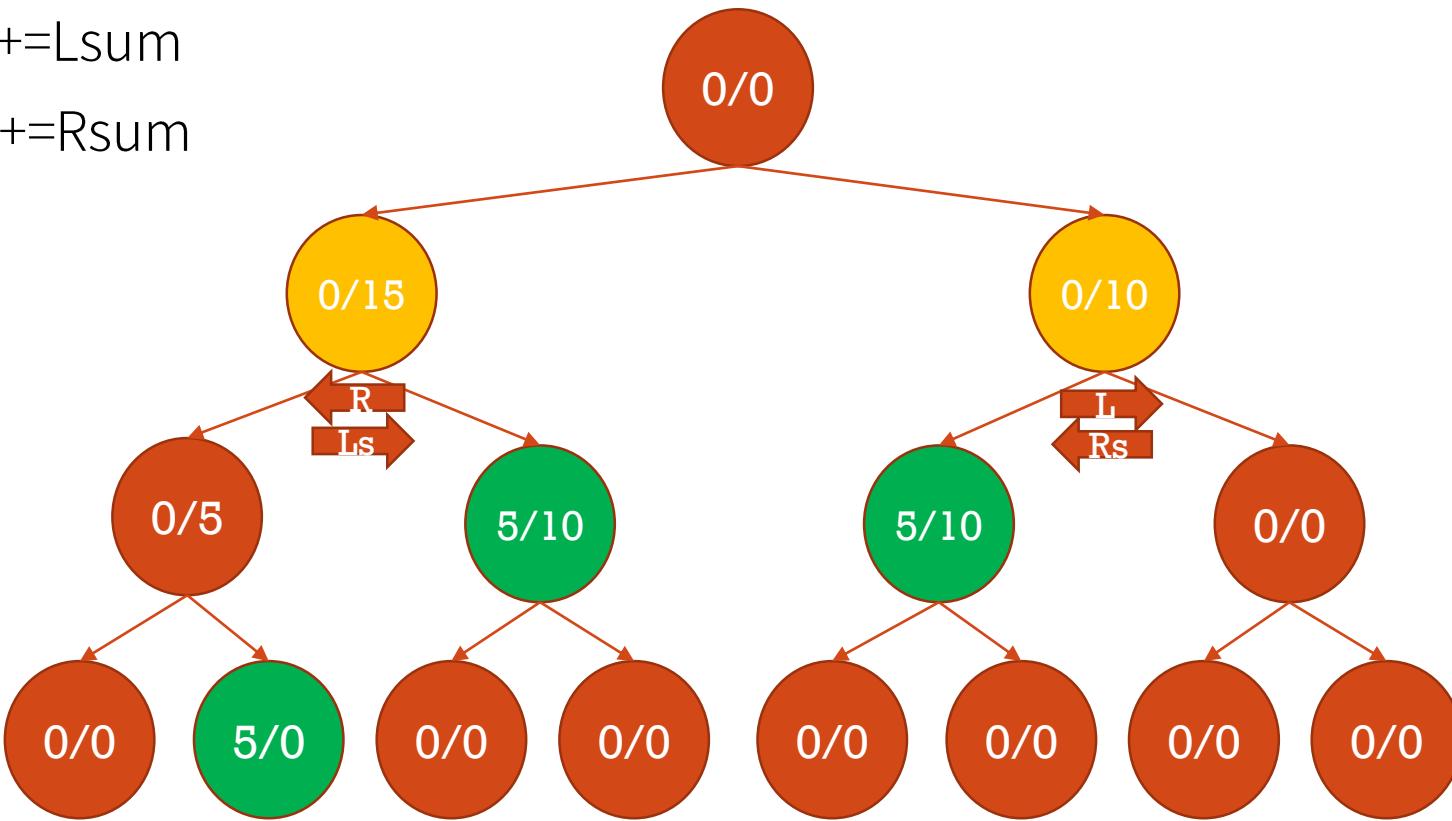
unit/
sum

Lsum

Rsum

15

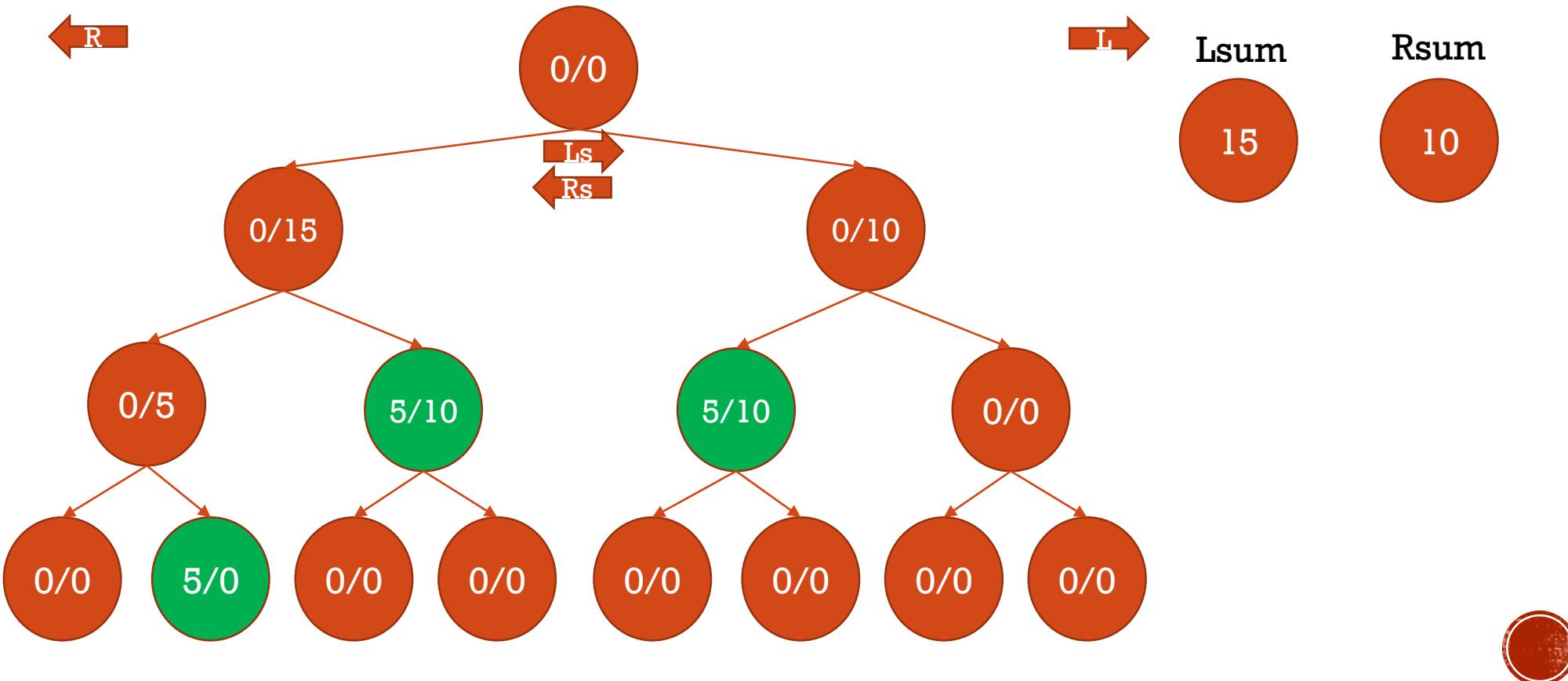
10



RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

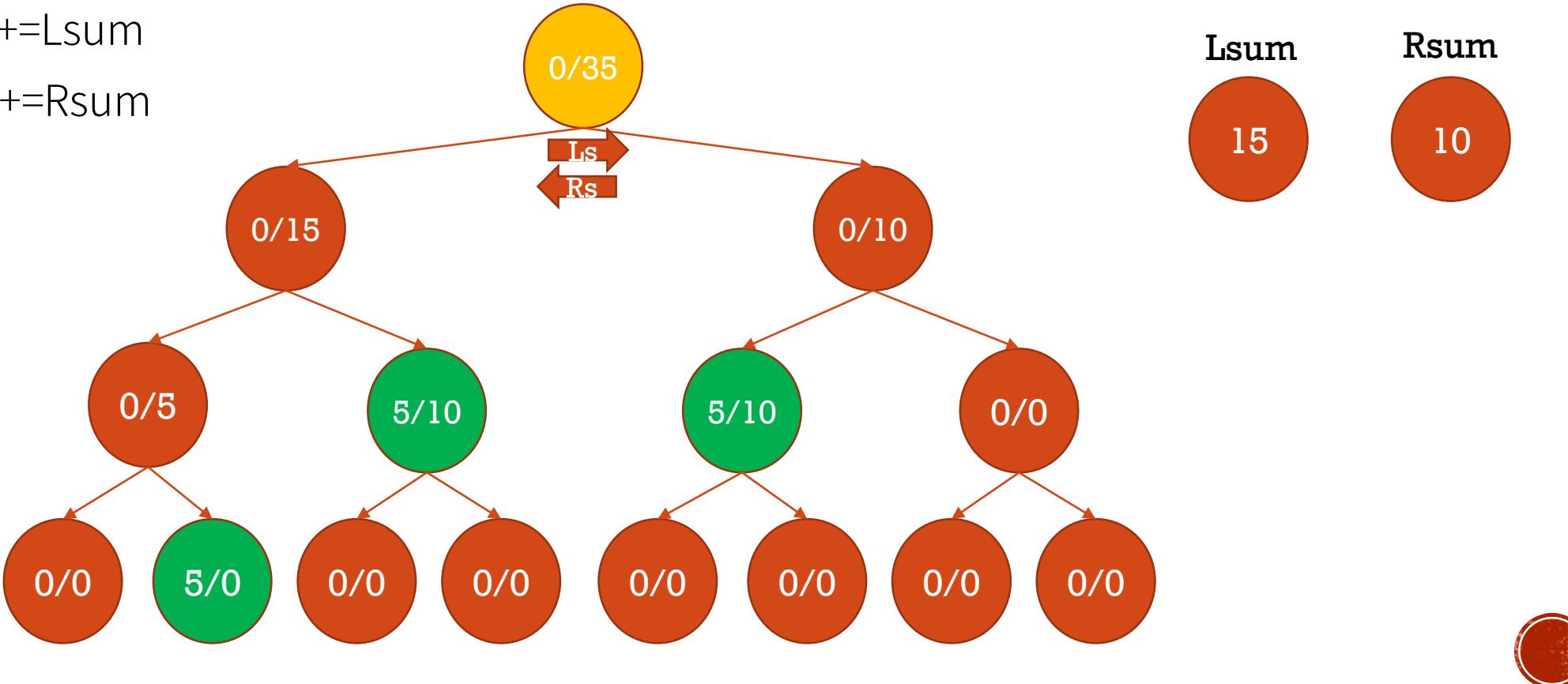
- update(range = [1,5],value = 5)



RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

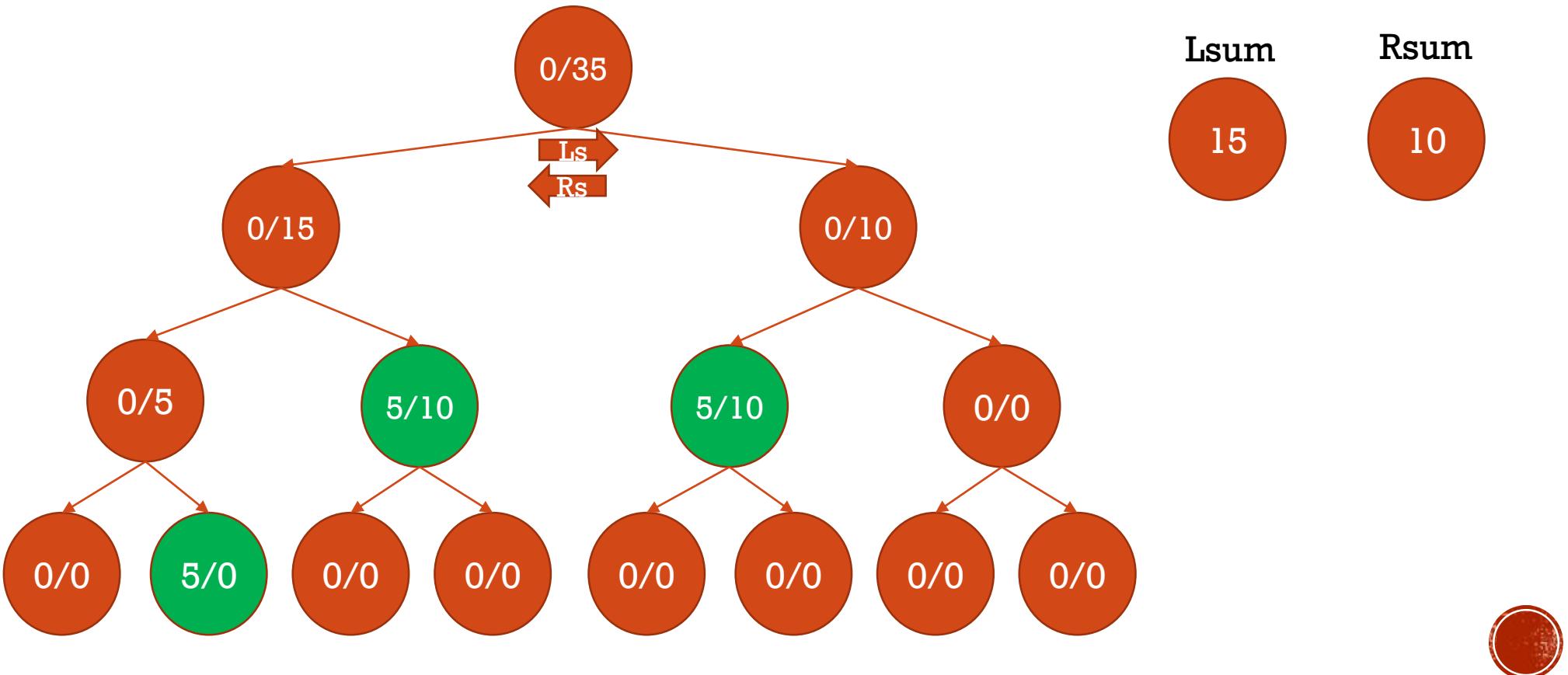
- update(range = [1,5],value = 5)
- sum[Ls]+=Lsum
- sum[Rs]+=Rsum



RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

- update(range = [1,5],value = 5)



RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

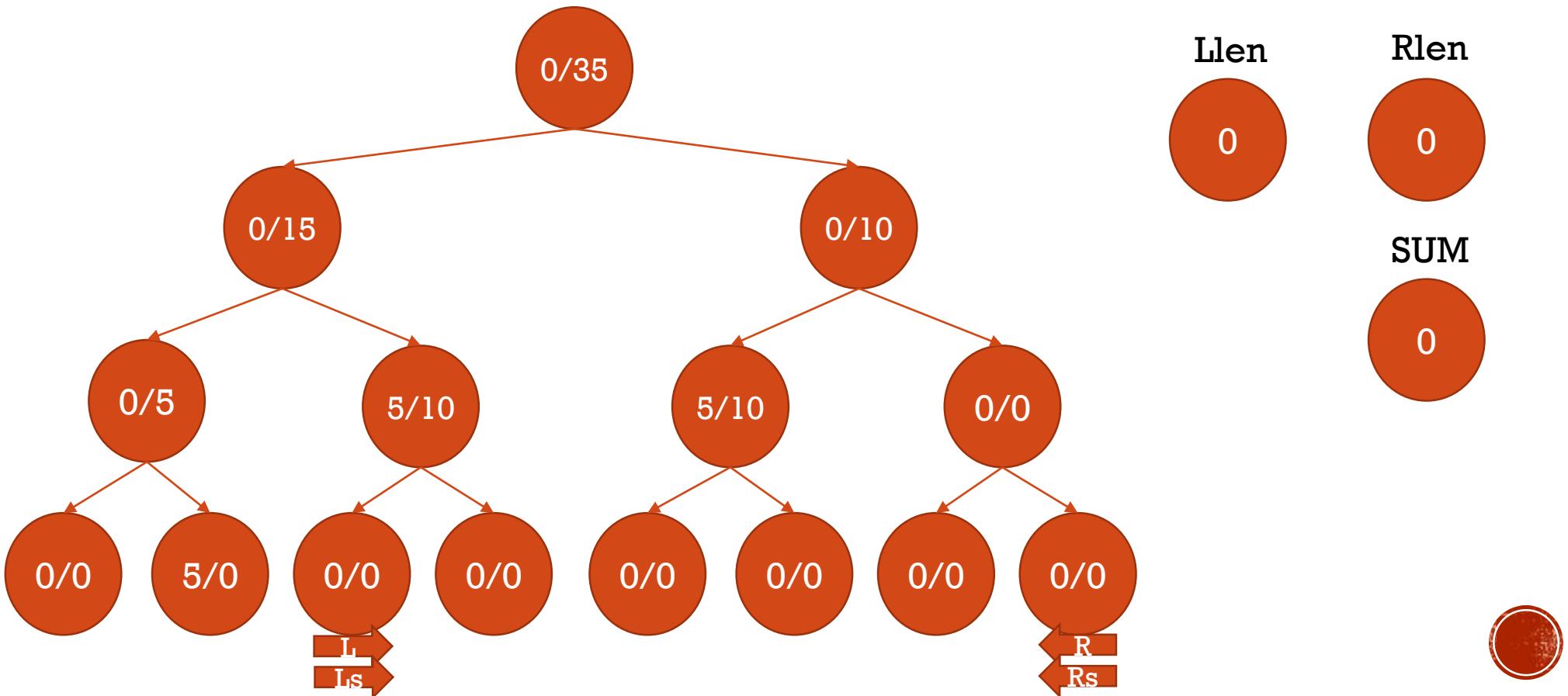
```
void update(int left, int right,int val) {  
    int L = left + k, R = right + k;  
    int Ls = L, Rs = R;  
    int Lsum = 0, Rsum = 0;  
    int len = 1;  
    while (Ls >= 1) {  
        sum[Ls] += Lsum;  
        sum[Rs] += Rsum;  
        if (L <= R) {  
            if (L % 2 == 1) {  
                unit[L] += val;  
                sum[L] += val*len;  
                Lsum += val*len;  
            }  
            if (R % 2 == 0) {  
                unit[R] += val;  
                sum[R] += val*len;  
                Rsum += val*len;  
            }  
        }  
        L = (L + 1) / 2, R = (R - 1) / 2;  
        Ls /= 2; Rs /= 2;  
        len *= 2;  
    }  
}
```



RANGE QUERY

- RANGE QUERY/RANGE UPDATE

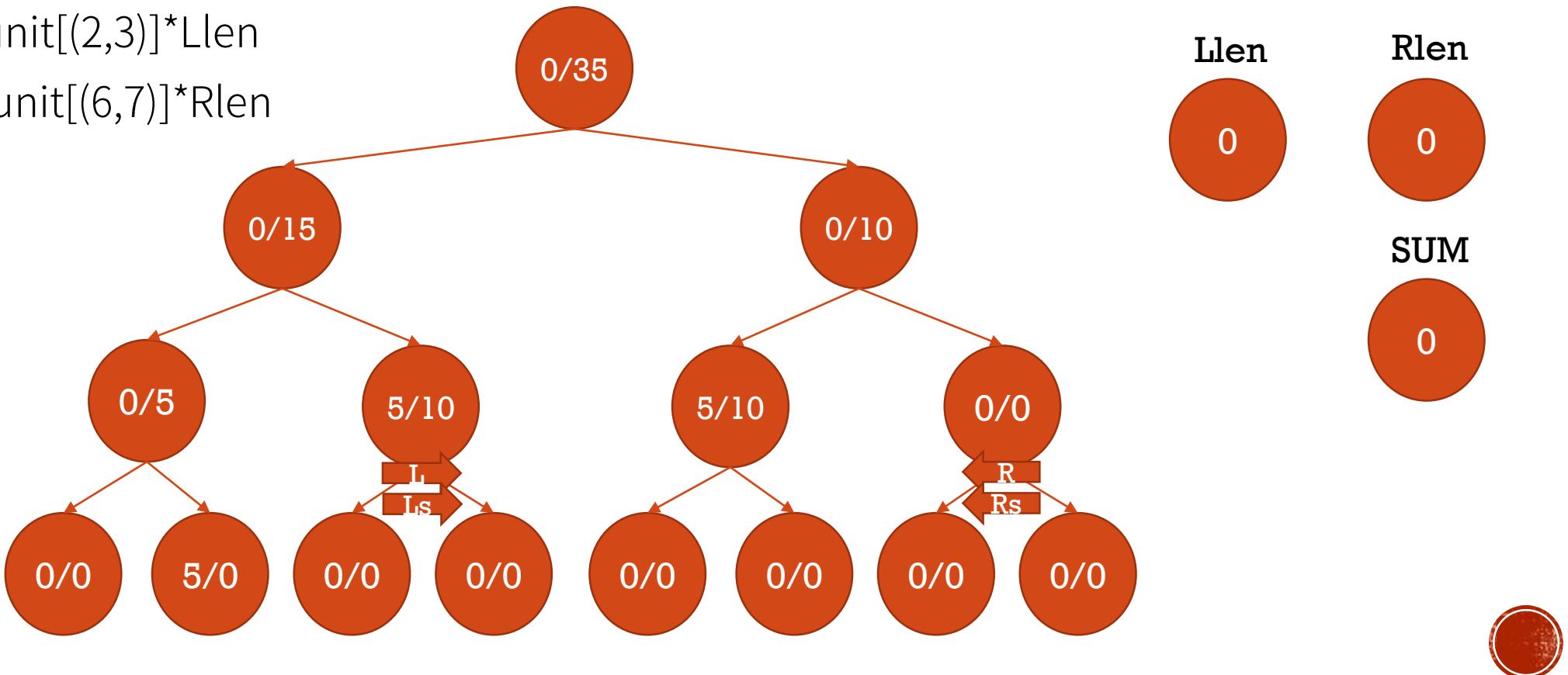
- query(2,7)



RANGE QUERY

- RANGE QUERY/RANGE UPDATE

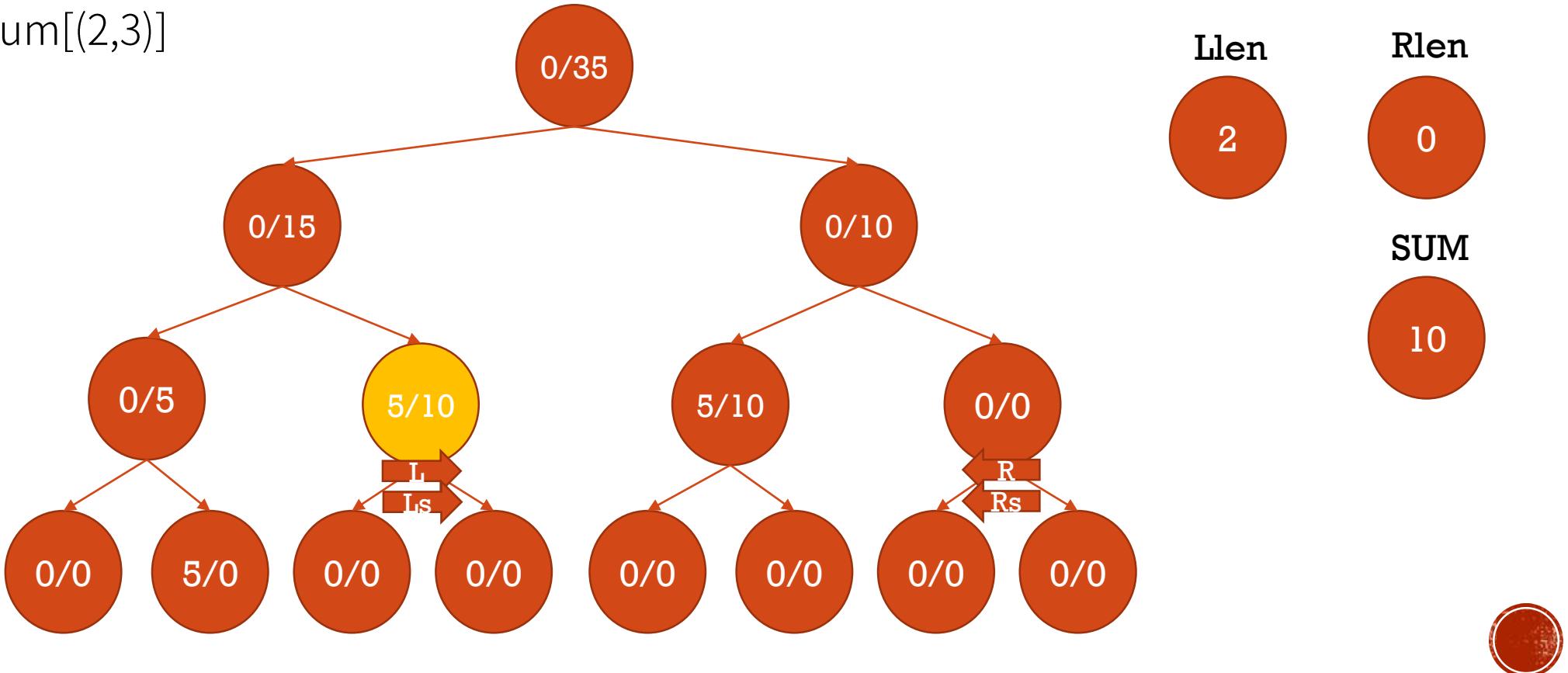
- query(2,7)
- $SUM += unit[(2,3)] * Llen$
- $SUM += unit[(6,7)] * Rlen$



RANGE QUERY

- RANGE QUERY/RANGE UPDATE

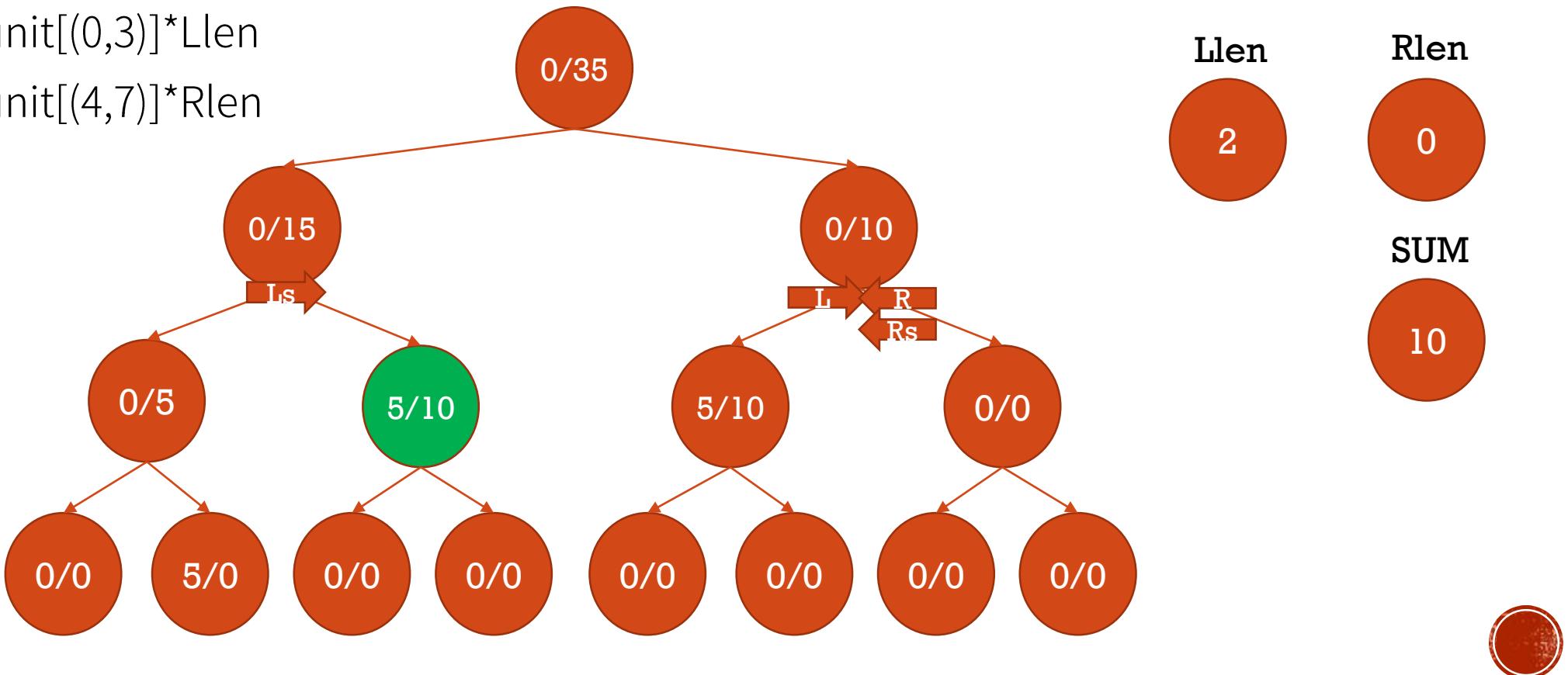
- query(2,7)
- SUM+=sum[(2,3)]
- Llen+=2



RANGE QUERY

- RANGE QUERY/RANGE UPDATE

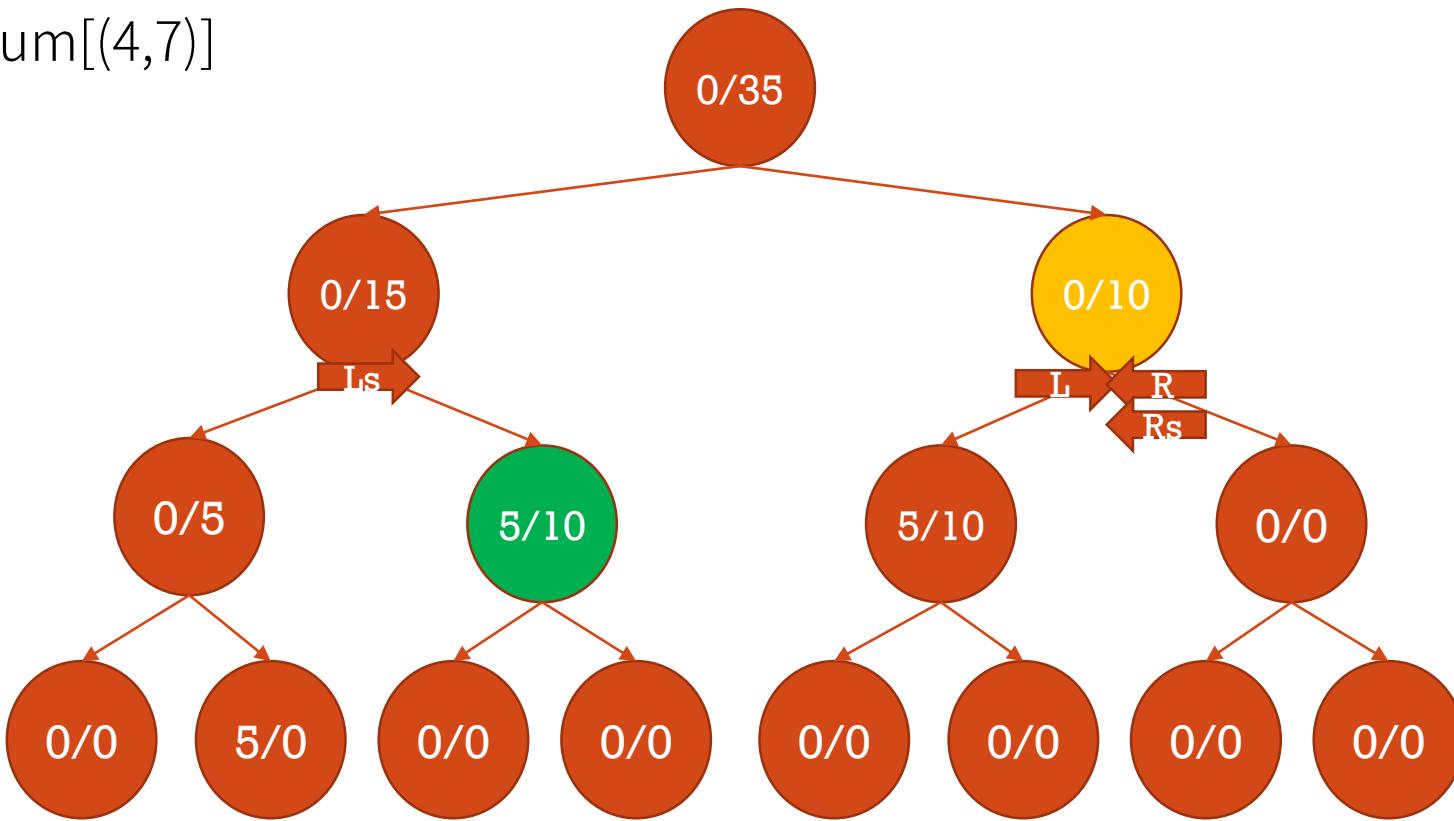
- query(2,7)
- $SUM += unit[(0,3)] * Llen$
- $SUM += unit[(4,7)] * Rlen$



RANGE QUERY

- RANGE QUERY/RANGE UPDATE

- query(2,7)
- SUM+=sum[(4,7)]
- Llen+=4



실제 배열

0	5	5	5	5	5	0	0
---	---	---	---	---	---	---	---

unit/
sum

Llen

6

Rlen

0

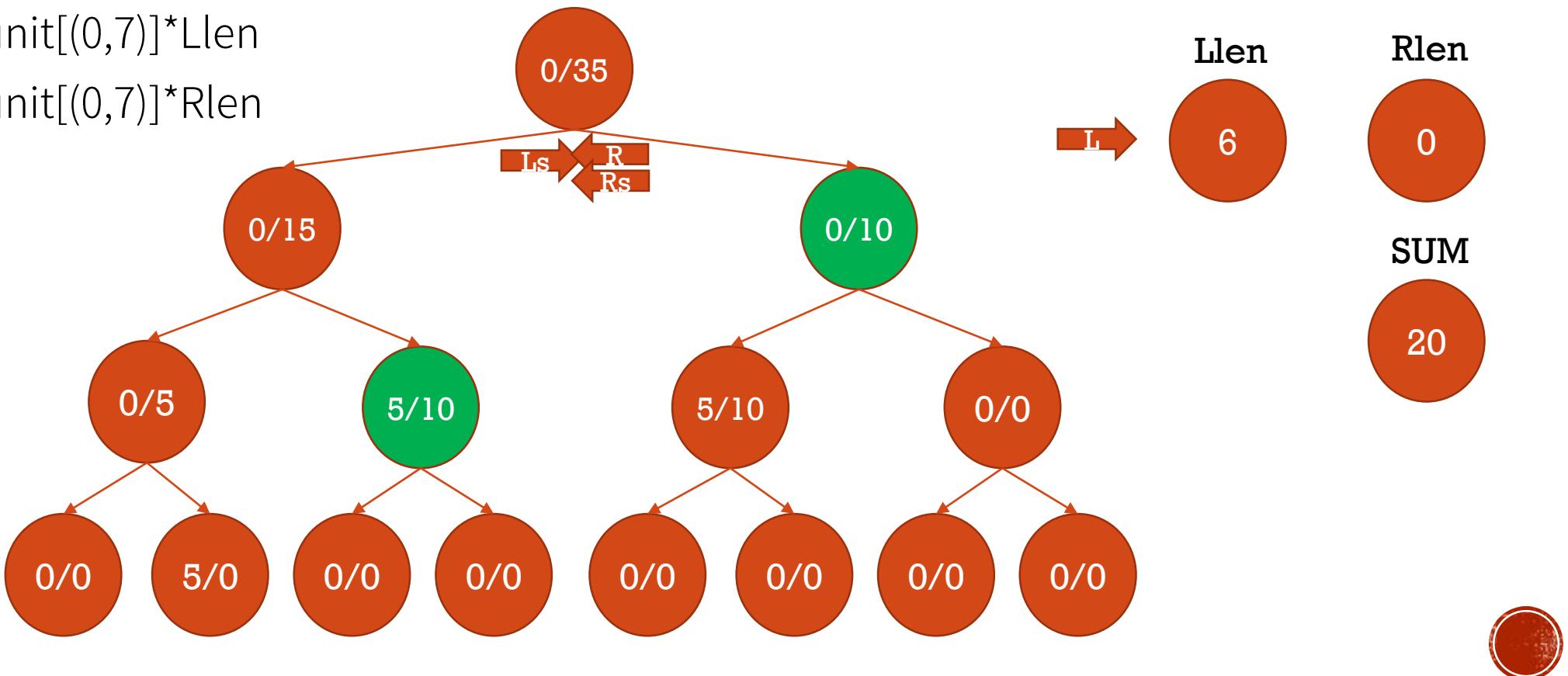
SUM

20

RANGE QUERY

- RANGE QUERY/RANGE UPDATE

- query(2,7)
 - SUM+=unit[(0,7)]*Llen
 - SUM+=unit[(0,7)]*Rlen



RANGE UPDATE

- RANGE QUERY/RANGE UPDATE

```
int query(int left, int right) {  
    int L = left + k, R = right + k;  
    int Ls = L, Rs = R;  
    int SUM = 0;  
    int Llen = 0, Rlen = 0;  
    int len = 1;  
    while (Ls >= 1) {  
        SUM += unit[Ls] * Llen + unit[Rs] * Rlen;  
        if (L <= R) {  
            if (L % 2 == 1) {  
                SUM += sum[L];  
                Llen += len;  
            }  
            if (R % 2 == 0) {  
                SUM += sum[R];  
                Rlen += len;  
            }  
        }  
        L = (L + 1) / 2, R = (R - 1) / 2;  
        Ls /= 2; Rs /= 2;  
        len *= 2;  
    }  
    return SUM;  
}
```



INDEXED TREE 응용

- 위의 예시에는 합으로 하였지만 그 외에도 다양한 것들이 대푯값으로 사용될 수 있다.
- 최대값, 최소값, bit연산 등등 (단, 최대값의 경우 update하는 원소가 증가하여야 하고, 최소값은 감소하여야 한다.)
- indexed tree를 사용하여 풀어 볼만한 문제들이다.
- <https://www.acmicpc.net/problem/5480>
- <https://www.acmicpc.net/problem/10999>
- <https://www.acmicpc.net/problem/12844>
- <https://www.acmicpc.net/problem/1395>
- <https://www.acmicpc.net/problem/10167>

