

Segment Tree

쿼리

수열과 쿼리

수열과 쿼리	
문제	문제 제목
13545	수열과 쿼리 0
13537	수열과 쿼리 1
17410	수열과 쿼리 1.5
13543	수열과 쿼리 2
13544	수열과 쿼리 3
13546	수열과 쿼리 4
13547	수열과 쿼리 5
13548	수열과 쿼리 6
13550	수열과 쿼리 7

13553	수열과 쿼리 8
13554	수열과 쿼리 9
13557	수열과 쿼리 10
13704	수열과 쿼리 11
13887	수열과 쿼리 12
13925	수열과 쿼리 13
13927	수열과 쿼리 14
14427	수열과 쿼리 15
14428	수열과 쿼리 16
14438	수열과 쿼리 17
14504	수열과 쿼리 18
14899	수열과 쿼리 19
16903	수열과 쿼리 20

트리와 쿼리

트리와 쿼리	
문제	문제 제목
13510	트리와 쿼리 1
13511	트리와 쿼리 2
13512	트리와 쿼리 3
13513	트리와 쿼리 4
13514	트리와 쿼리 5
13515	트리와 쿼리 6
13516	트리와 쿼리 7
13517	트리와 쿼리 8
13518	트리와 쿼리 9

13519	트리와 쿼리 10
13539	트리와 쿼리 11
16912	트리와 쿼리 12
17936	트리와 쿼리 13
18372	트리와 쿼리 14
18794	트리와 쿼리 15
18932	트리와 쿼리 16
20030	트리와 쿼리 17
20148	트리와 쿼리 18
20564	트리와 쿼리 19
1921	트리와 쿼리 20
27974	트리와 쿼리 21
27975	트리와 쿼리 22
27976	트리와 쿼리 23

간단한 문제

- 길이 N 인 배열 `arr`이 주어진다
- l, r ($1 \leq l, r \leq N$)이 주어졌을 때,
- `arr[1] + arr[1+1] + ... + arr[r-1] + arr[r]` 의 값을 구하자



- Naïve (Bruteforce)
`for(int i = 1; i <= r; i++) ans += arr[i];`

조금 생각해볼 만한 문제

- 길이 N 인 배열 arr 이 주어진다
- l, r ($1 \leq l, r \leq N$)이 주어졌을 때,
- $arr[1] + arr[1+1] + \dots + arr[r-1] + arr[r]$ 의 값을 **Q 번** 구하자



- $N, Q \leq 100\,000$
- Naïve (Bruteforce) : $O(NQ)$

조금 생각해볼 만한 문제

1	4	2	6	7	9	2	6	8	4
1	5	7	13	20	29	31	37	45	49

- 누적 합 (Prefix Sum): $O(N + Q)$

```
for(int i = 1; i <= N; i++) p[i] = p[i-1] + a[i];  
for(int i = 1; i <= Q; i++) ans = p[r] - p[l-1];
```

어려운 문제 (BOJ 2042)

- 길이 N 인 배열 arr 이 주어진다
- l, r ($1 \leq l, r \leq N$)이 주어졌을 때, 다음 두 가지 연산 중 하나를 Q 번 수행한다.
- $arr[1] + arr[1+1] + \dots + arr[r-1] + arr[r]$ 의 값을 구하기
- $arr[i]$ 의 값을 k 로 변경하기



- Prefix sum with update? $O(NQ)$
- 최악의 경우는, $i = 1$ 인 변경 쿼리가 계속해서 들어올 때

시간 초과

구간 쿼리

- 어떤 배열이 있을 때, l 부터 r 까지의 대푯값을 찾는 함수
- $f(arr[i:i + p])$, $p = (r - l + 1)$ 을 효율적으로 계산하는 방법을 찾아야 한다
- 미리 $f(arr[x_1:y_1])$, $f(arr[x_2:y_2])$, \dots , $f(arr[x_k:y_k])$ 처럼 k 개의 구간에 대한 대푯값을 저장해 둔다.
- 미리 구한 대푯값을 통해서 원래 함수값을 구할 수 있다면, 즉
- $f(f(arr[x_{a_1}:y_{a_1}]), f(arr[x_{a_2}:y_{a_2}]), \dots, f(arr[x_{a_m}:y_{a_m}])) = f(arr[i:i + p])$ 이라면, p 개의 값 대신 **m 개의 값만으로 계산**할 수 있음

구간 쿼리

- $f(f(arr[x_{a_1}:y_{a_1}]), f(arr[x_{a_2}:y_{a_2}]), \dots, f(arr[x_{a_m}:y_{a_m}])) = f(arr[i:i+p])$
이라면, p 개의 값 대신 **m 개의 값만으로 계산**할 수 있음
- 길이가 1인 N 개의 구간 : 단순 배열
- 길이가 \sqrt{N} 인 \sqrt{N} 개의 구간 : Sqrt Decomposition
- 길이가 1, 2, 4, ..., N 인 N 개의 구간 : Fenwick Tree
- 길이가 1, 2, 4, ..., N 인 $N \log N$ 개의 구간 : Sparse Table
- 길이가 1, 2, 4, ..., N 인 $2 \times N$ 개의 구간 : **Segment Tree**

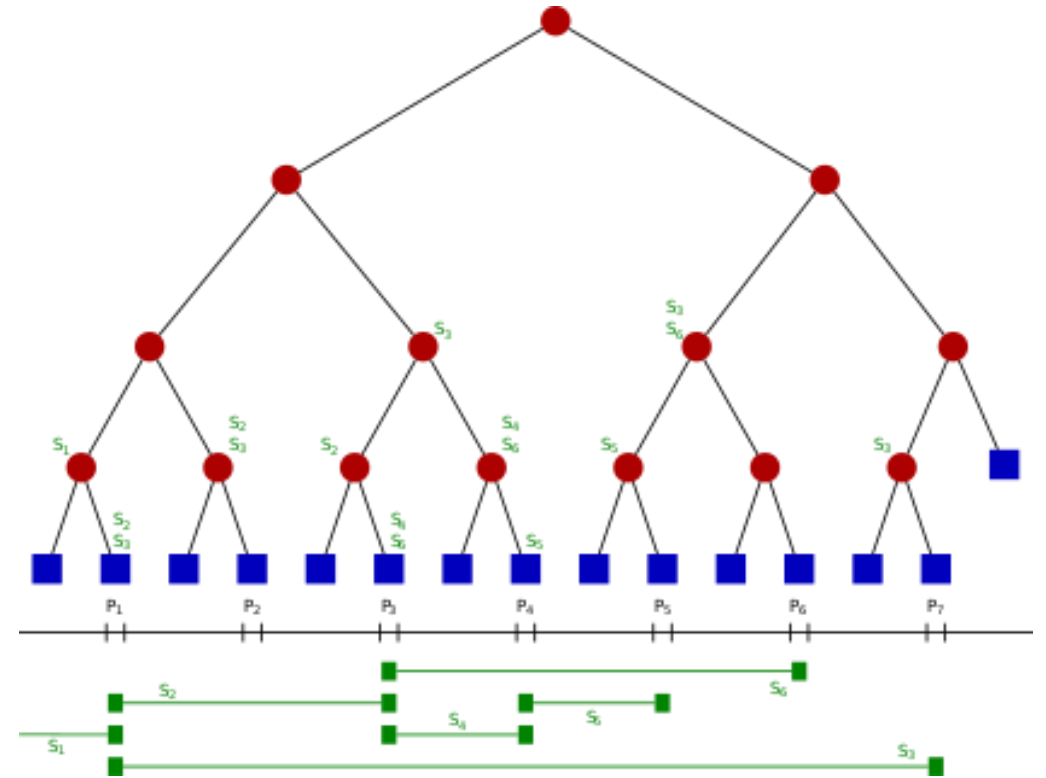
구간 쿼리

6 [1:3]			15 [4:6]			24 [7:9]		
1	2	3	4	5	6	7	8	9

Sqrt Decomposition

1	2	3	4	5	6	7	8
1		3		5		7	
3				11			
10							
36							

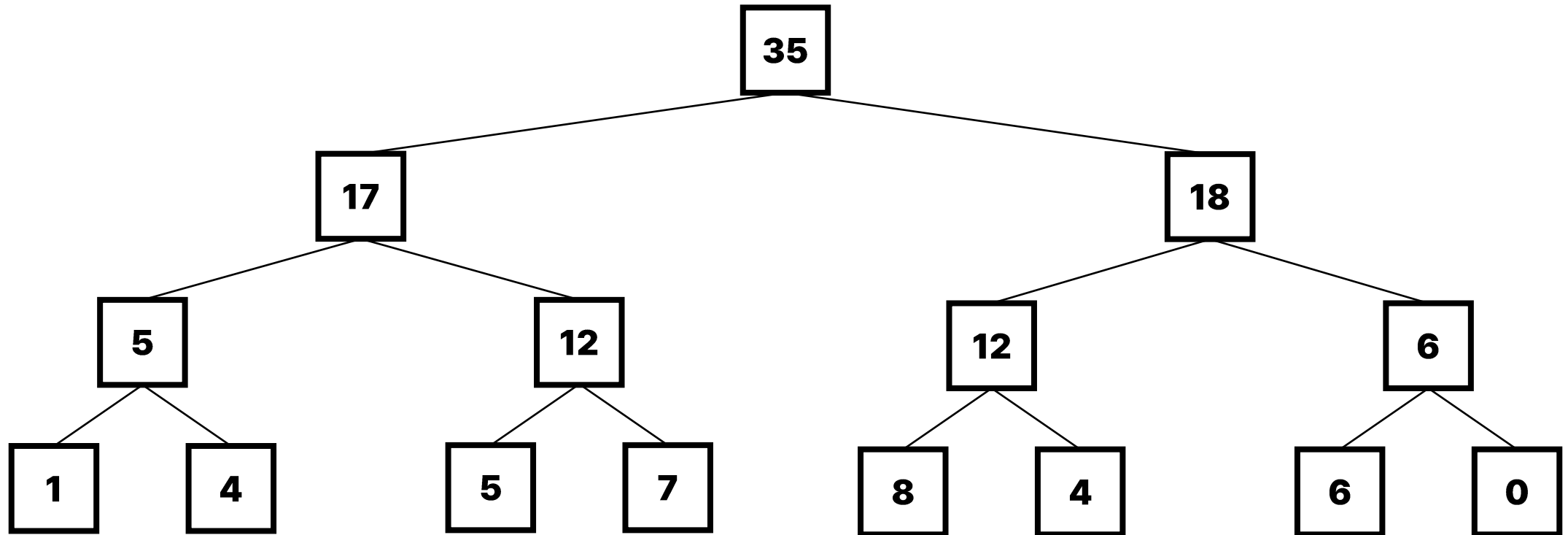
Fenwick Tree



Segment Tree

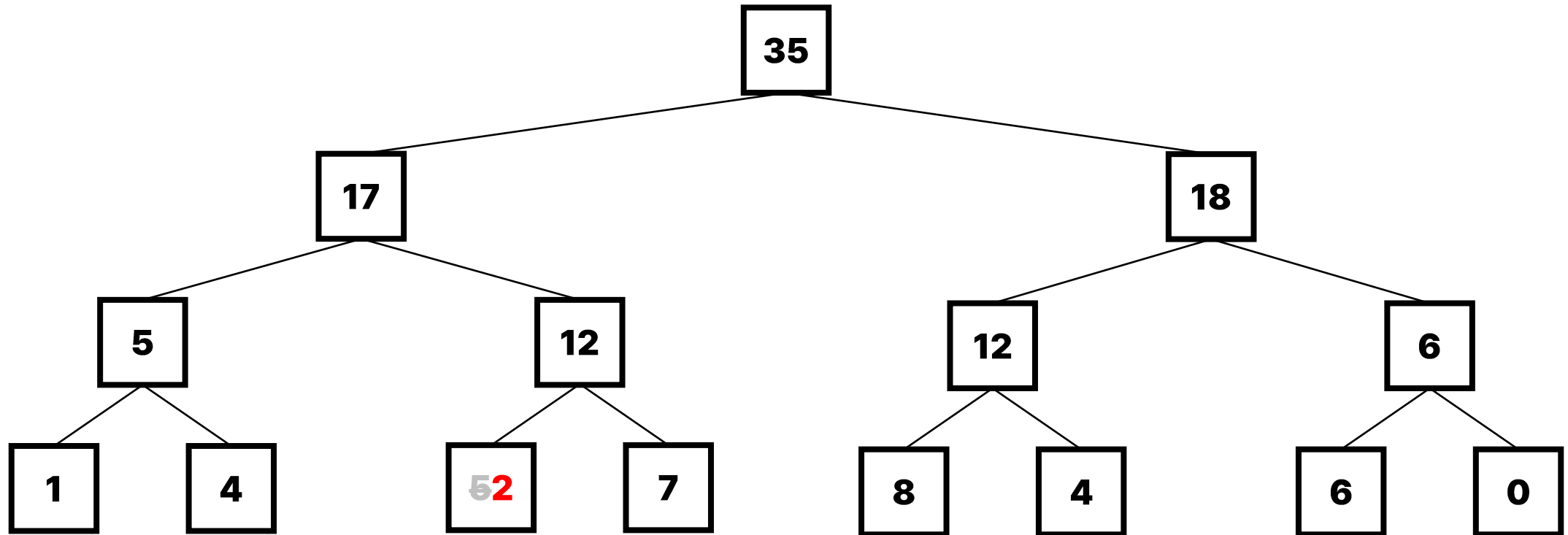
세그먼트 트리

- 트리의 형태를 배열에 저장할 수 있다. 루트를 1번 인덱스로 둔다.
- 왼쪽/오른쪽 자식을 각각 $i \times 2$, $i \times 2 + 1$ 번 인덱스로 둘 수 있다.



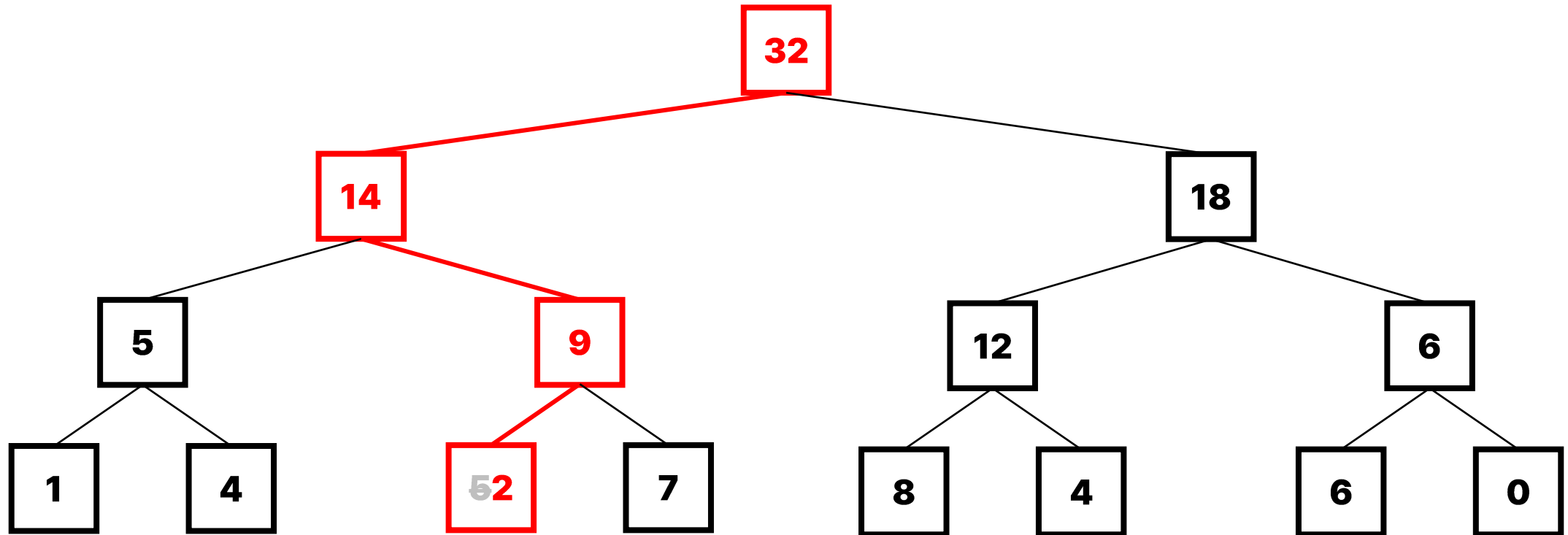
세그먼트 트리 update

- 리프 노드에 해당하는 값을 변경한다.



세그먼트 트리 update

- 부모로 올라가면서 구간 대푯값을 갱신한다. $O(\log N)$

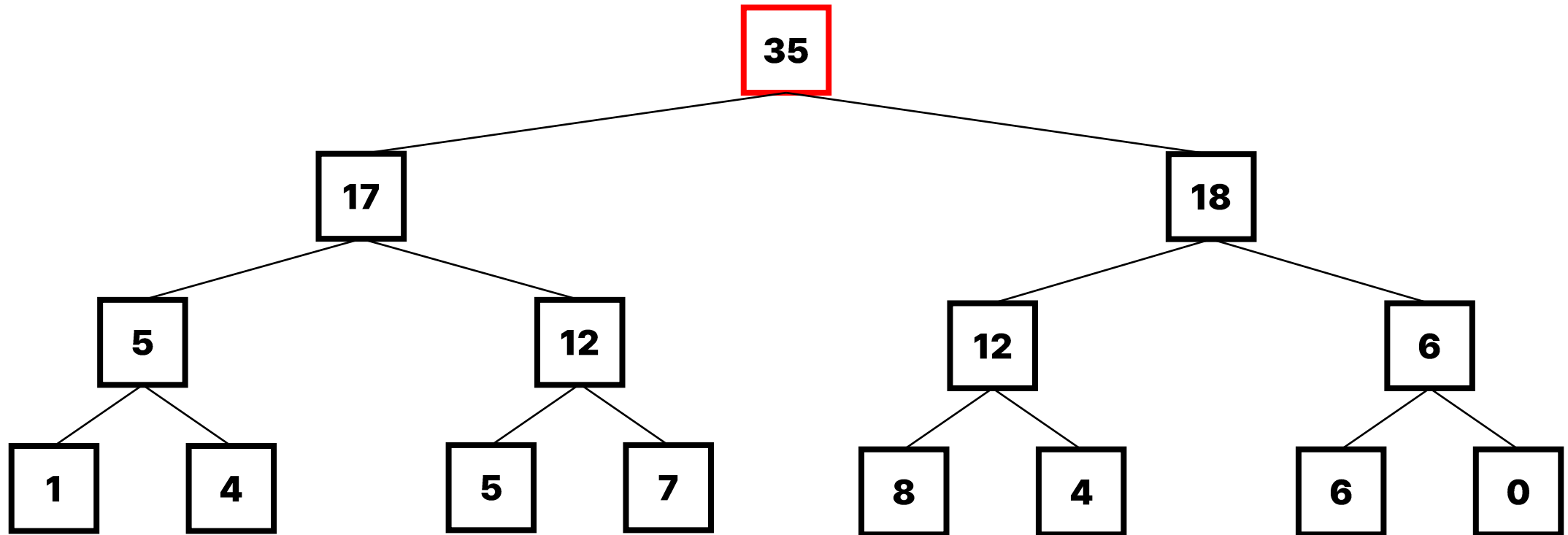


세그먼트 트리 query

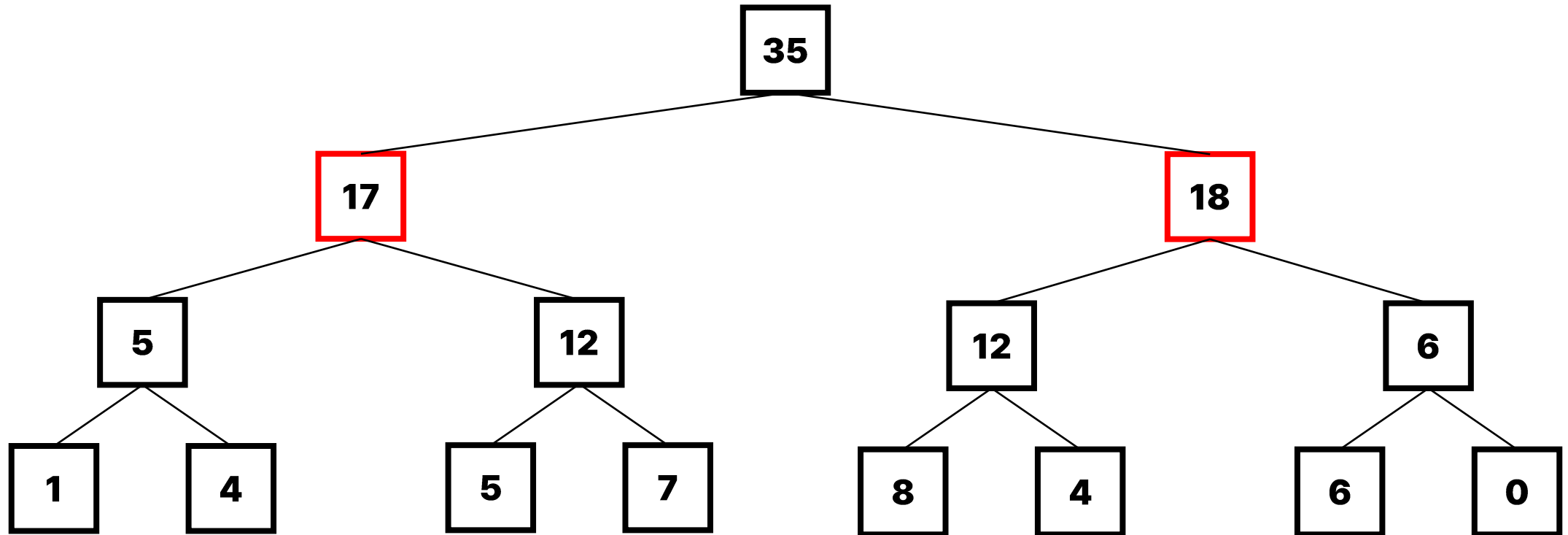
- 구하고자 하는 구간 $[l, r]$, 노드가 담당하는 구간 $[s, e]$ 라고 하자.
- $[l, r]$ 구간과 $[s, e]$ 구간의 교집합이 존재하지 않는 경우,
즉 $r < s$ 또는 $e < l$ 인 경우 볼 필요가 없다
- 구하고자 하는 구간 안에 노드가 담당하는 구간이 포함돼 있다면,
즉 $l \leq s$ 이고 $e \leq r$ 인 경우 해당 구간을 리턴한다.
- 두 경우가 모두 아니라면 두 자식 정점에 대해 재귀적으로 진행한다.
- 수학적 귀납법을 통해서 증명할 수 있지만, 간단하게 예시를 들어 보자

세그먼트 트리 query

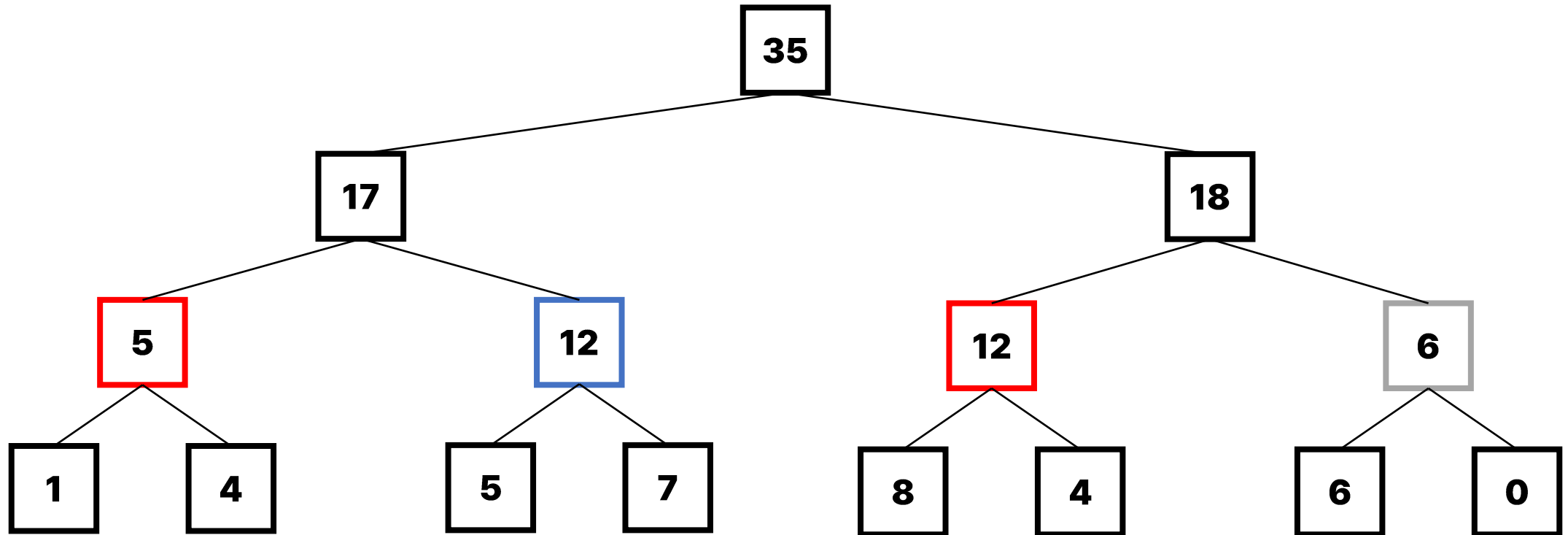
- query(2, 5)



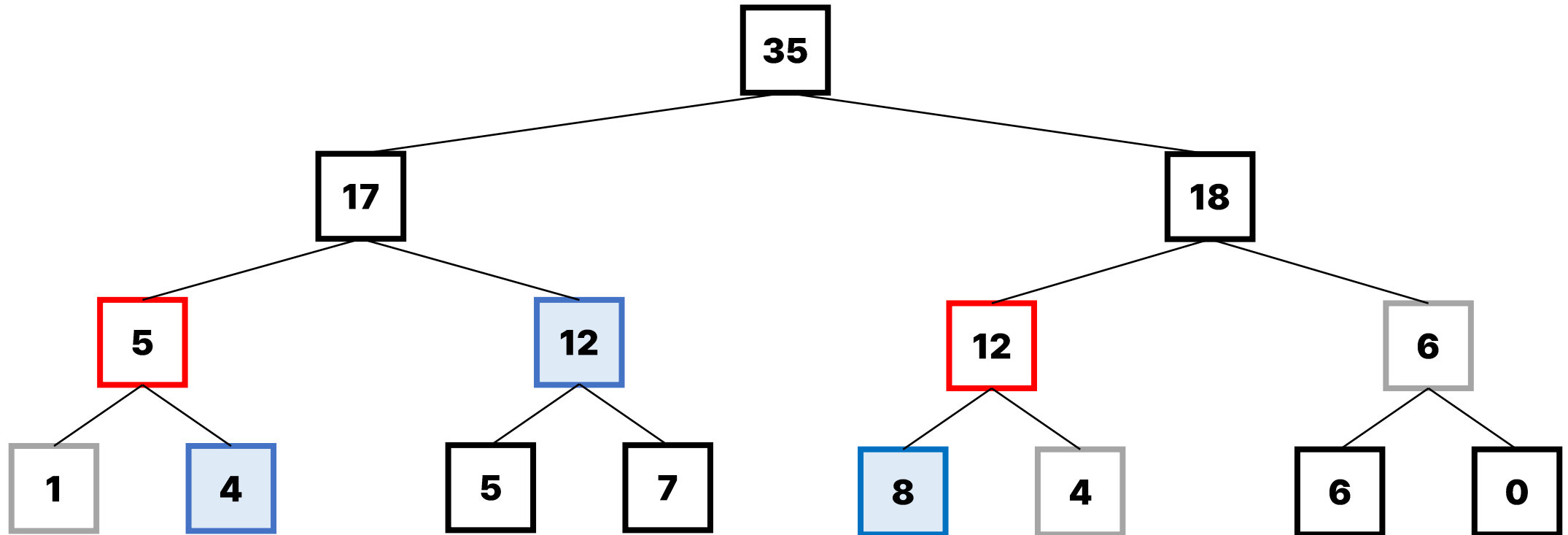
세그먼트 트리 query



세그먼트 트리 query

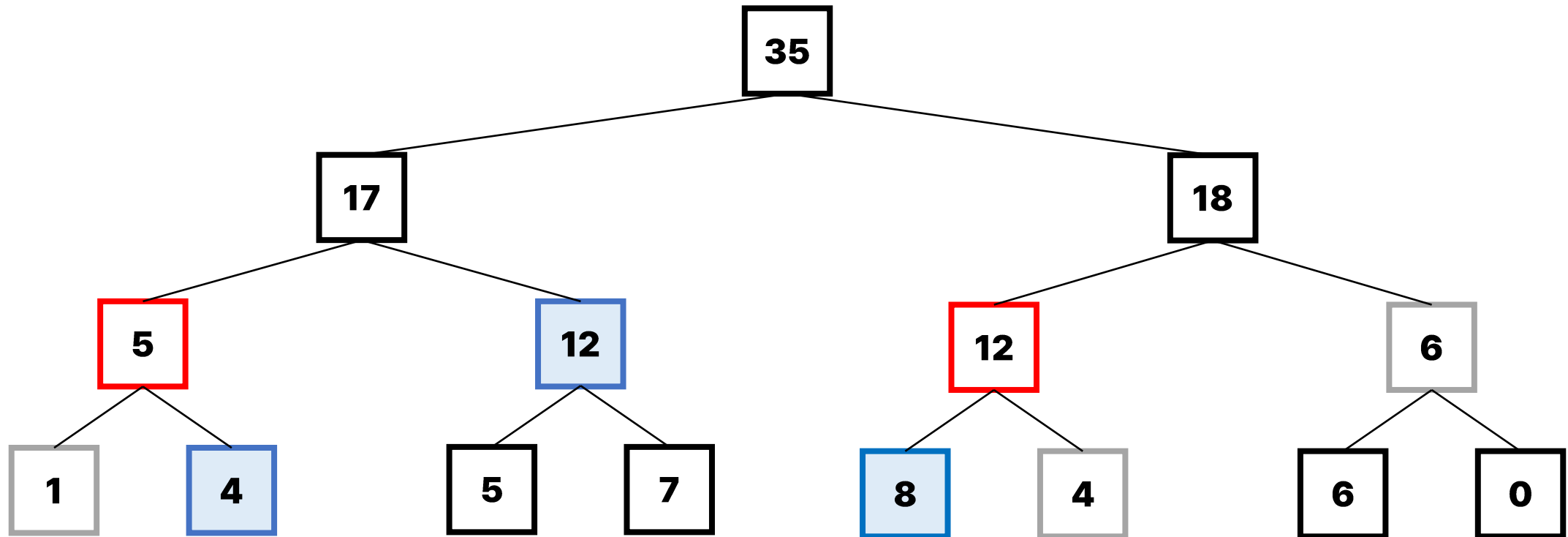


세그먼트 트리 query



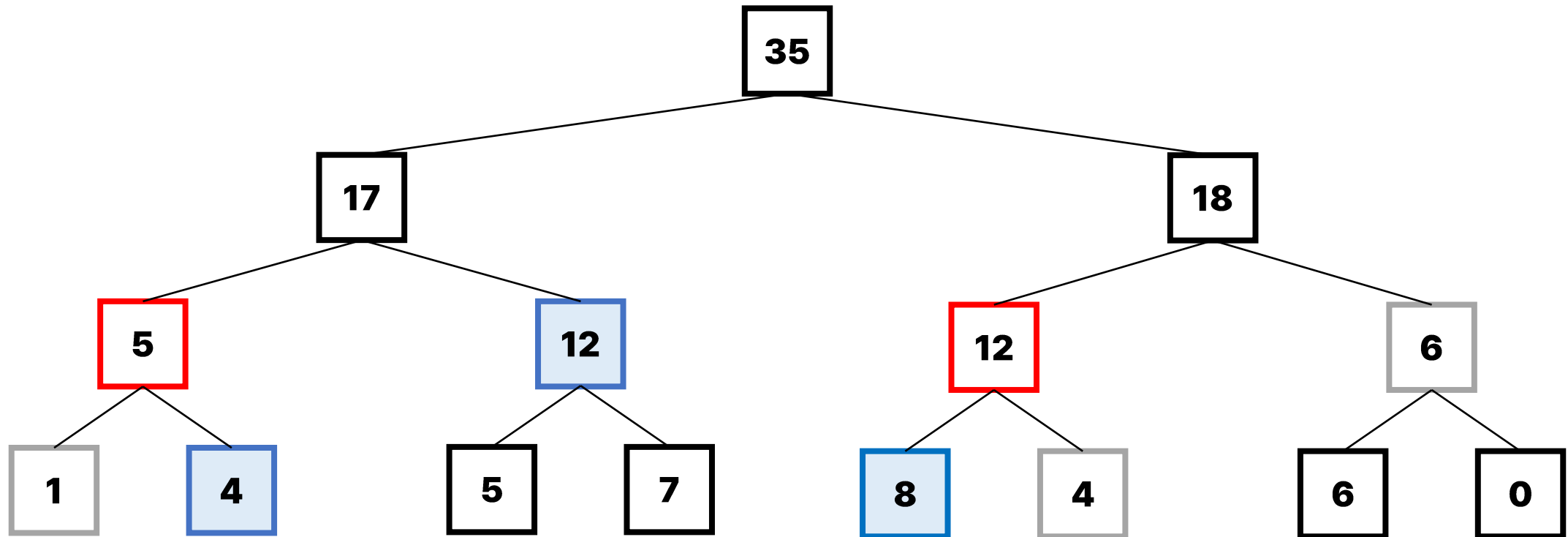
세그먼트 트리 Time Complexity

- 시간 복잡도는 어떻게 될까?
- 자식으로 내려가면서 최대 두 번의 함수 호출이 일어나는데 ?



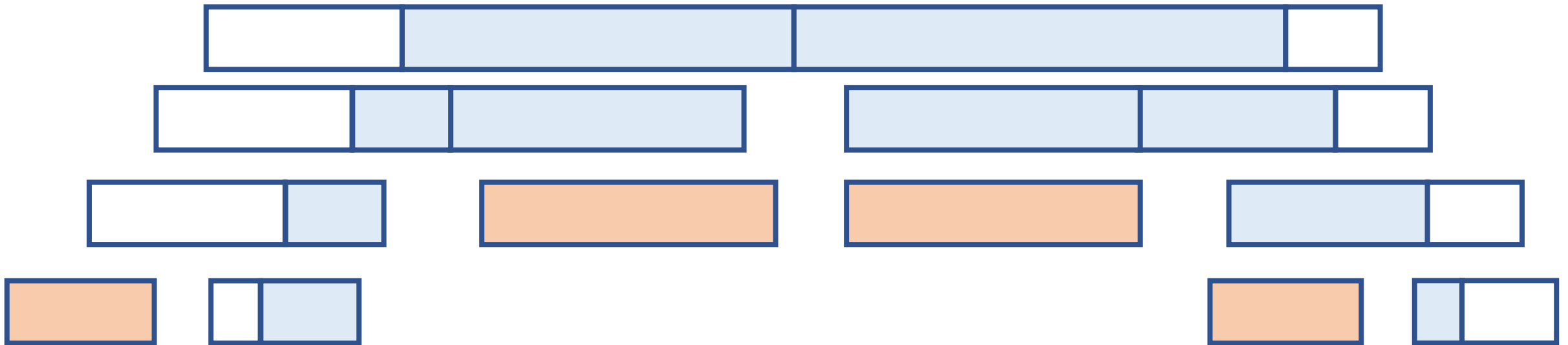
세그먼트 트리 Time Complexity

- 시간 복잡도는 어떻게 될까?
- 자식으로 내려가면서 최대 두 번의 함수 호출이 일어날 수 있다



세그먼트 트리 Time Complexity


- 최악의 경우는, 함수 호출이 두 번 일어나는 경우 (재귀적으로)
- 구하고자 하는 구간이 절반을 포함해야 한다.
- 이 경우, 처음 전체의 중간은 왼쪽 쿼리 노드의 끝이 되고, 오른쪽 쿼리 노드의 시작이 된다
- 둘 중 하나는 반드시 바로 리턴됨, $O(4 \times \log N) = O(\log N)$



세그먼트 트리를 짜 보자!

- 재귀와 수학적 귀납법을 잘 이해하고 있다면, 코드를 짜는 데 큰 무리가 없음
- tree는 보편적으로 $4 \times N$ 크기를 잡아 둔다

$$\begin{aligned} S(n) &\leq 2^{\lceil \log_2 n \rceil + 1} - 1 \\ &< 2 \cdot 2^{\lceil \log_2 n \rceil} \\ &= 4 \cdot 2^{\lceil \log_2 n \rceil - 1} \\ &\leq 4 \cdot 2^{\log_2 n} \\ &\leq 4n \end{aligned}$$



```
1 void init(int node, int s, int e){
2     if (s == e){ tree[node] = arr[s]; return; }
3     int mid = (s+e)/2;
4     init(node*2, s, mid);
5     init(node*2+1, mid+1, e);
6     tree[node] = tree[node*2] + tree[node*2+1];
7 }
```

세그먼트 트리를 짜 보자!

- query가 각각 왼쪽, 오른쪽 자식의 대푯값을 구해준다고 가정하자.
- 기계적으로 둘을 합해주면 (대푯값을 구하는 함수를 적용하면) 원하는 값을 얻을 수 있다
- Base case를 잘 생각해 주자.



```
1 ll query(int node, int s, int e, int l, int r){
2     if (e < l || r < s) return 0;
3     if (l <= s && e <= r) return tree[node];
4     int mid = (s+e)/2;
5     return query(node*2, s, mid, l, r) + query(node*2+1, mid+1, e, l, r);
6 }
```

세그먼트 트리를 짜 보자!

- update



```
1 void update(int node, int s, int e, int idx, ll value){
2     if (e < idx || idx < s) return;
3     if (s == e){ tree[node] = value; return;}
4     int mid = (s+e)/2;
5     update(node*2, s, mid, idx, value);
6     update(node*2+1, mid+1, e, idx, value);
7     tree[node] = tree[node*2] + tree[node*2+1];
8 }
```

사탕상자 2243

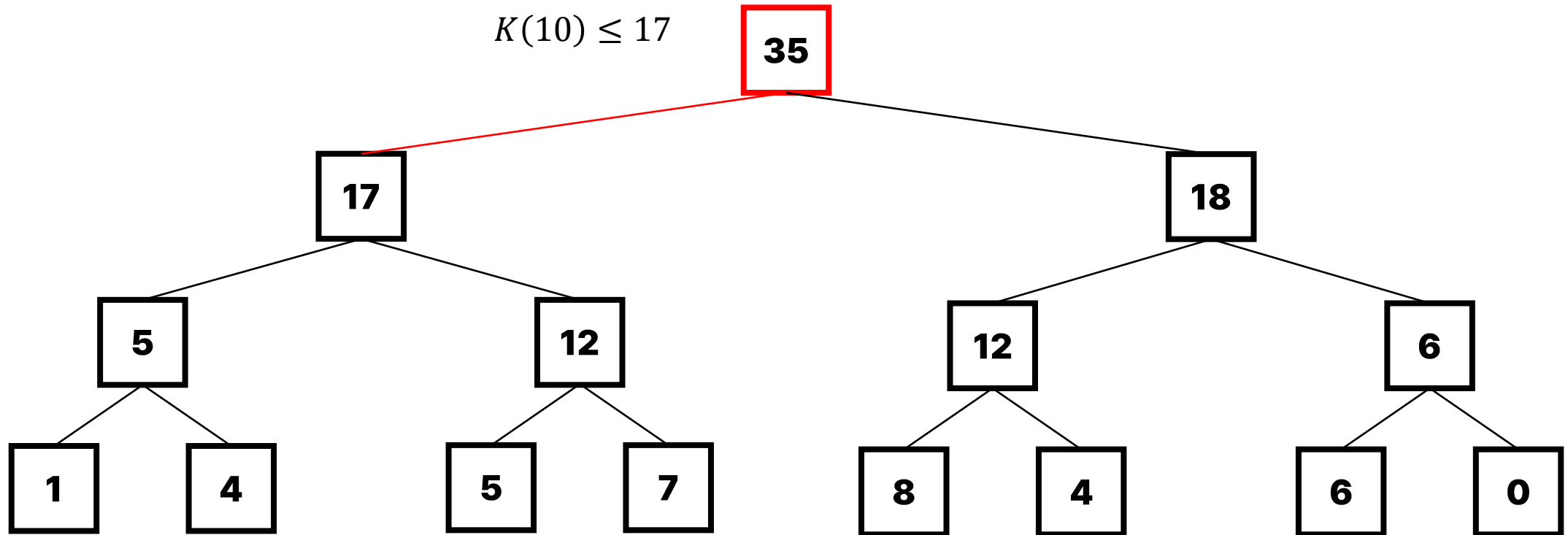
- 사탕의 맛은 1부터 1 000 000까지 존재한다 (작을 수록 맛있다)
- K번째로 맛있는 사탕을 꺼내거나, 같은 맛의 여러 사탕을 상자에 넣거나 뺄 수 있다
- 쿼리의 개수는 100 000개
- 세그먼트 트리를 쓴다고 하면, 같은 맛의 여러 사탕을 넣거나 빼는 것은 빠르게 할 수 있게 된다 ($Q \log N$)
- K번째로 맛있는 사탕은 어떻게 꺼낼 수 있을까?

사탕상자 2243

- 노드는 왼쪽 / 오른쪽 자식이 있고, 자식들의 대푯값을 합성한 것이 부모의 대푯값이 된다
- 사탕의 개수를 기록하는 세그먼트 트리라고 한다면, 부모 노드에 기록된 값은, 자식 노드가 바라보는 범위에서의 모든 사탕의 개수와 같다.
- 왼쪽 자식보다 K가 더 작거나 같으면 왼쪽으로 들어가면 된다
- 왼쪽 자식보다 K가 더 크다면, 왼쪽은 볼 필요가 없다. 단, 이 경우 오른쪽으로 탐색할 때 무시한 왼쪽의 개수만큼 제외해주어야 한다
- 예시를 보자

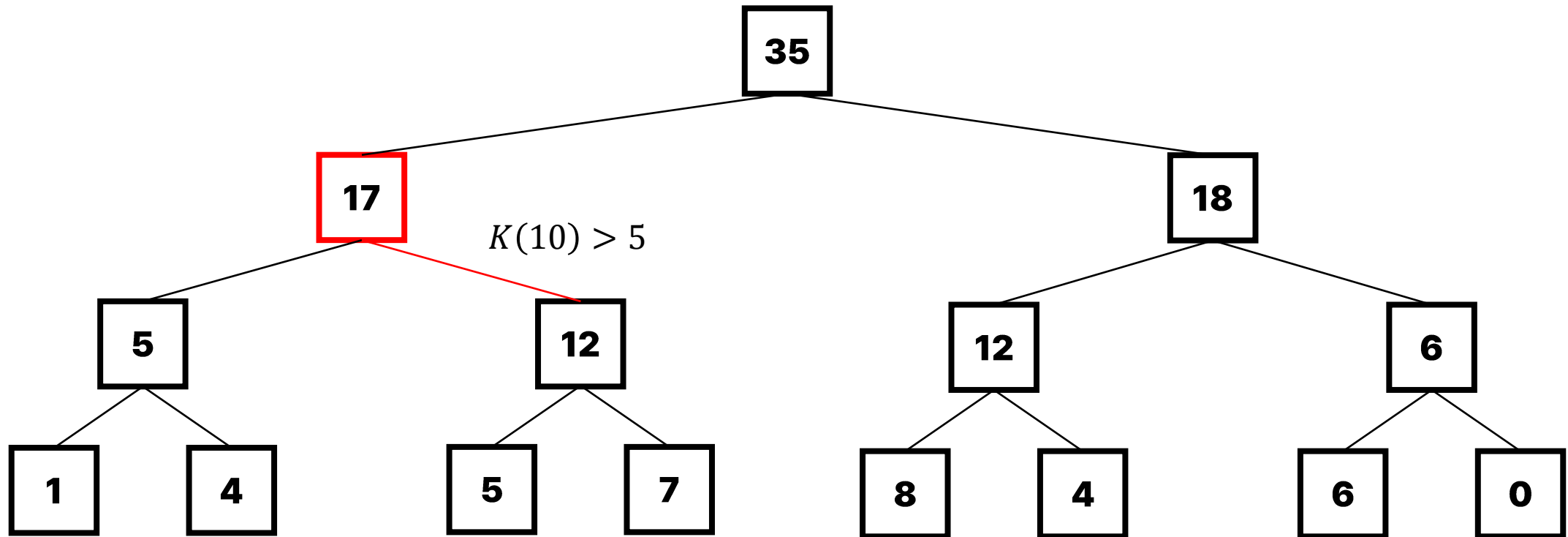
사탕상자 2243

- 아래와 같은 세그먼트 트리에서, 10번째로 맛있는 사탕 찾기 ($K = 10$)



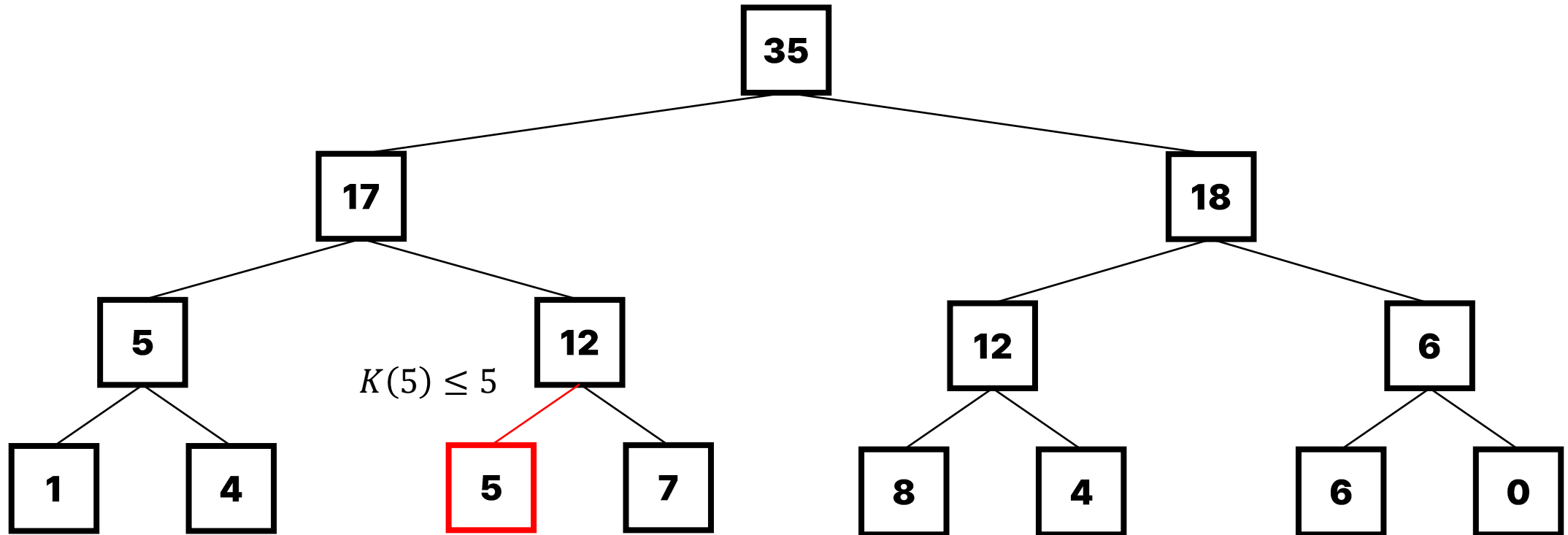
사탕상자 2243

- 아래와 같은 세그먼트 트리에서, 10번째로 맛있는 사탕 찾기 ($K = 10$)



사탕상자 2243

- 아래와 같은 세그먼트 트리에서, 10번째로 맛있는 사탕 찾기 ($K = 10$)
- 10번째로 맛있는 사탕은 맛이 3인 사탕이다



사탕상자 2243

- 사탕을 꺼내주는 데에도 최대 $O(\log N)$ 번, 일정 개수의 사탕을 넣거나 뺄 때에도 $O(\log N)$ 의 시간복잡도에 문제를 해결할 수 있다.
- 전체 사탕의 맛이 1 000 000이므로, 맛의 척도를 배열로 하는 세그먼트 트리를 만들면 된다. 노드에 저장되는 값은 구간에 존재하는 사탕의 개수이다.
- 문제에서 사탕상자에 손을 댄 횟수가 n 이므로, 전체 시간복잡도는 $O(n \log 1\,000\,000)$
- 세그먼트 트리의 노드의 범위를 나타내는 s, e 변수를 잘 활용해서 풀이해 보시다

달리기 2517

- 현재 달리기의 순위대로, 각 선수의 기량이 주어진다
 - 각 선수들은 앞선 선수보다 기량이 높을 경우, 해당 선수를 추월할 수 있다.
 - 각 선수별로 최선의 등수는 얼마일까?
 - 예를 들어, 각 선수의 기량이 앞에서부터 순서대로 (6, 2, 9, 20)의 경우, 각 선수별 최선의 등수는 1, 2, 1, 1위가 된다.
-
- $N \leq 500\,000, A_i \leq 1\,000\,000\,000$
 - 선수의 기량 범위가 매우 커서 배열로 세그먼트 트리를 만들기에는 메모리가 모자라다
 - N 개만으로는 세그먼트 트리를 구축할 수 있다

달리기 2517

- 문제의 제한에 초점을 두자. 참가한 선수들의 평소 실력은 모두 다르다.
 - 5, 2, 79, 67의 평소 실력은 2, 1, 4, 3의 평소 실력과 다를 게 없다.
 - 범위 때문에 문제를 해결하지 못할 때, 좌표를 압축해 문제를 해결할 수 있다.
 - 이제 새로 만들어진 A'_i 는 최대 500 000이다.
-
- 제한은 해결됐으니 문제를 해결해 보자

달리기 2517

- 각 선수마다 최선의 등수의 하한은 현재 순위입니다. 현재 순위를 유지하면서 결승선에 들어가면 된다.
- 이제 각 선수가 **앞선 선수들 중에서 얼마나 많은 사람들을 앞지를 수 있는지를** 관찰하자.
- 앞선 사람들 중에서, 자신보다 작은 사람을 앞지를 수 있다.
- 순위의 개념으로 생각하면, **자신보다 큰 사람의 수 + 1**이 곧 나의 최선의 등수가 된다.
- 세그먼트 트리에서, $[실력+1, N]$ 까지를 쿼리로 구한 뒤, 자신의 실력에 해당하는 구간을 1로 만들어주는 것을 반복하면 된다.
- 예시를 보자

달리기 2517

- (앞선 순서대로) 2, 8, 10, 7, 1, 9, 4, 15 -> 2, 5, 7, 4, 1, 6, 3, 8 (좌표 압축)
- 최초 세그먼트 트리의 리프 부분은 아래와 같다



- 1위의 평소 실력이 2이므로, [2, 8]의 구간 합 + 1이 곧 최선의 등수이다 (1)
- 이후 2를 1로 업데이트한다.



- 5, 7도 같은 방식으로 진행한다. 모두 최선의 순위는 1위이다.



달리기 2517

- (앞선 순서대로) 2, 8, 10, 7, 1, 9, 4, 15 -> 2, 5, 7, 4, 1, 6, 3, 8 (좌표 압축)

0	1	0	0	1	0	1	0
---	---	---	---	---	---	---	---

- 4위의 평소 실력은 4이다. 이 때 앞선 사람들 중 5, 7은 앞지를 수 없다. [5, 8]까지의 구간 합과 같다. 따라서 최선의 등수는 $2 + 1 = 3$ 위이다.
- 이런 식으로 달리는 순서대로 모든 선수들에 대해서 진행하면, 정답을 구할 수 있다.
- 전혀 세그먼트 트리처럼 보이지 않아도, 구간 합으로 문제를 변환해 세그먼트 트리로 멋있게 풀 수 있다!

수열과 쿼리 1 13537

- 쿼리는 단 한 개, $[i, j]$ 구간에서 k 보다 큰 원소의 개수 구하기
- 가장 보편적으로 알려져 있는 해답은 머지 소트 트리를 활용한 풀이
- 세그먼트 트리로도 풀 수 있다!

수열과 쿼리 1 13537

- 쿼리의 순서를 적당히 조작해서 문제를 해결해 보자 = 오프라인 쿼리
- 같은 구간에서, $K = 5$ 일 때 세어졌던 원소들은 $K = 4$ 일 때 한 번 더 세어진다
- K 가 가장 큰 값부터 업데이트한다면, 한 번 더 업데이트할 필요가 없지 않을까?
- 문제 쿼리에는 업데이트가 없었지만, 어떤 것을 원소로 하는 세그먼트 트리를 짜느냐에 따라 업데이트가 필요할 수 있음

수열과 쿼리 1 13537

- 초기 세그먼트 트리의 리프 노드 상태는 아래와 같다



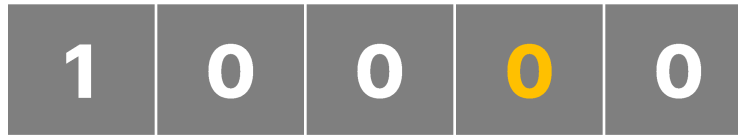
```
5
5 1 2 3 4
3
2 4 1
4 4 4
1 5 2
```

- 쿼리의 K를 기준으로 정렬해서, 큰 쿼리부터 실행하자
- (4, 4, 4)의 쿼리를 실행할 때, K보다 큰 5가 처리되어야 한다



수열과 쿼리 1 13537

- 이후 $[4, 4]$ 에 해당하는 구간 합을 구해주면 된다 $\rightarrow 0$



- $(1, 5, 2) : K = 4, 3$ 에 대해서 업데이트해야 한다



- $[1, 5]$ 구간에 대한 합은 3

```
5
5 1 2 3 4
3
2 4 1
4 4 4
1 5 2
```

수열과 쿼리 1 13537

- $(2, 4, 1) : K = 2$ 에 대해서 업데이트하면 된다.



- $[2, 4]$ 구간에 대한 합은 2



```
5
5 1 2 3 4
3
2 4 1
4 4 4
1 5 2
```

수열과 쿼리 1 13537

- 시간복잡도 분석
- 쿼리 정렬 $O(M \log M)$
- $1 \sim N$ 까지의 수가 어디에 있는지 알아야 하므로 해당 수를 정렬하는 데 $O(N \log N)$
- 쿼리 당 업데이트는 최대 N 번 이루어지므로 $O(M \log N)$
- $O(M \log M + N \log N + M \log N)$ 에 문제를 해결할 수 있다

주의할 점

- **결합법칙 및 교환법칙**이 성립해야 대푯값을 올바르게 갱신할 수 있다
- `+`, `-`, `*`, `min`, `max`등이 이에 해당한다
- query에서 교집합이 존재하지 않을 때 0을 리턴하면 안 되는 경우가 있다
(대표적으로 `min`)
- 파이썬의 경우, `sys.setrecursionlimit`을 통해 재귀 깊이를 설정할 필요가 있다
(기본적으로 1000이며, 이를 적절히 늘려야 함. Pypy에서는 사용하는 데 주의가 필요)
- 재귀가 아닌 방식으로도 세그먼트 트리를 구현할 수 있다. iterative segment tree에 대해서 찾아 보자!

나아가서

- 하나의 원소 업데이트, 구간 쿼리를 Point update, Range query라고 한다
- Range update, Range query는 어떻게 해야 할까?
[1, 5]의 구간에 모두 +5 연산하기
- Segment Tree with Lazy Propagation

문제

- 2042 : 구간 합 구하기
 - 2357: 최솟값과 최댓값
 - 2243 : 사탕상자
 - 2517 : 달리기
 - 10090 : Counting Inversions
-
- 1168 : 요세푸스 문제 2
 - 13537 : 수열과 쿼리 1
 - 13544: 수열과 쿼리 3

Reference

- <https://github.com/justiceHui/SSU-SCCC-Study>
- <https://book.acmicpc.net/ds/segment-tree>
- http://teferi.net/ps/구간_쿼리#세그먼트_트리
- https://www.quora.com/Why-does-4-*-N-space-have-to-be-allocated-for-a-segment-tree-where-N-is-the-size-of-the-original-array