1. 기초 강의

해당 과정의 기초 강의는 없습니다.

2. 실전 강의

2.1. 개요

세그먼트 트리는 point update와 range query를 모두 O(logN)의 시간에 처리할 수 있는 자료구조입니다.

아직 Pro시험에 세그먼트 트리를 사용해야 풀리는 문제는 나오지 않았습니다. 하지만 세그먼트 트리를 사용하면 정해보다 더 효율적인 문제는 몇 개 존재합니다.

실행 시간을 계속 최적화해야 하는 Pro 시험이므로 고급 자료구조를 하나쯤 알아두면 도움이 될 것입니다. 또한 세그먼트 트리는 시간/공간 복잡도가 매우 효율적이고, 구현이 짧고, 정말 많은 변형이 가능한, 알고리즘 문제에서의 맥가이버 칼과 같은 자료구조이므로 Pro 시험 이외의 여러 문제 풀이에 도움이 될 것입니다.

여기서는 세그먼트 트리의 가장 기초적인 형태를 다룰 것이고, 추가적인 공부를 원하는 수강생들은 인터넷에서 자료를 찾아보시길 권합니다.

여기서 말하는 기초적인 형태란, 교환법칙이 성립하는 연산에 대한 range query를 처리하는 세그먼트 트리를 말합니다. 예를 들어, 구간 최소값이나 구간 합은 min(min(a, b), c) = min(a, min(b, c))이고, (a + b) + c = a + (b + c)이므로 min과 +연산은 둘 다 교환법칙이 성립합니다.

본문에서 다룰 세그먼트 트리는 이렇게 교환법칙이 성립하는 연산의 구간 쿼리를 처리할 수 있으며, 아래 수열에서의 구간 합을 예시로 들겠습니다.

a_0	a_1	a_2	a_3	a_4
5	-1	3	2	-8

2.2. 완전 탐색과 **DP**

다음은 구간 합을 구하는 완전 탐색 코드입니다.

```
int a[5] = {5, -1, 3, 2, -8};
```

```
// a[i]를 x로 바꿈
void set(size_t i, int x) {
	a[i] = x;
}

// [ l , r ) 구간의 합
int query(size_t l, size_t r) {
	int result = 0;
	for (size_t i = l; i < r; ++i) {
		result += a[i];
	}
	return result;
}
```

구간 합 쿼리가 있을 때마다 모든 값을 더합니다. 하지만 수열의 값 하나를 바꾸는 건 바로 할 수 있습니다.

point update O(1), range query O(N)입니다.

다이나믹 프로그래밍을 사용해 미리 O(N^2)개의 구간 합을 미리 구해 놓는다면, 쿼리를 O(1)에 구할 수 있습니다.

$dp_{l,r}$	$r=0$	$r=1$	$r=2$	$r=3$	$r=4$	$r=5$
$l=0$	0	5	4	7	9	1
$l=1$		0	-1	2	4	-4
$l=2$			0	3	5	-3
$l=3$				0	2	-6
$l=4$					0	-8

dp[l][r] = l이상 r미만 범위의 구간 합

하지만 수열의 값 하나가 바뀐다면 그 값을 포함하는 모든 구간 합도 바뀌어야 하므로 위의 DP 테이블에서 O(N^2)개의 값을 갱신해줘야 합니다.

a_0	a_1	a_2	a_3	a_4
5	-1	3	100	-8

$dp_{l,r}$	$r=0$	$r=1$	$r=2$	$r=3$	$r=4$	$r=5$
$l=0$	0	5	4	7	101	99
$l=1$		0	-1	2	102	94
$l=2$			0	3	103	95
$l=3$				0	100	92
$l=4$					0	-8

point update O(N^2), range query O(1)입니다.

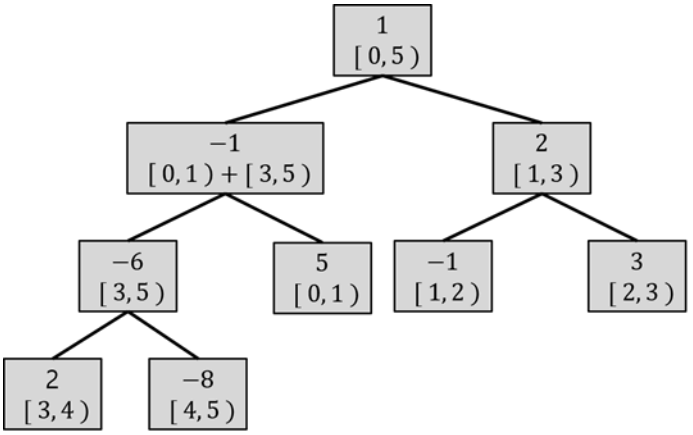
위의 두 방식에서 알 수 있듯이, 미리 여러 개의 구간 합을 구해 둘 경우 쿼리는 빨라지지만 업데이트는 느려집니다.

세그먼트 트리는 구간을 똑똑하게 나눠서 어떤 구간이 주어지든 O(logN)개의 합으로 나타낼 수 있으며, 어떤 값이 바뀌어도 같이 변해야 하는 구간은 O(logN)개입니다.

2.3. 세그먼트 트리

세그먼트 트리는 아래처럼 생긴 이진 트리입니다. 구간의 길이가 1인(원소의 개수가 하나인) 구간은 리프 노드가 되고, 나머지 노드는 자식 노드를 합친 구간 합을 들고 있습니다.

배열의 길이가 N이라면 세그먼트 트리의 노드 개수는 2N - 1개가 됩니다.



a_0	a_1	a_2	a_3	a_4
5	-1	3	2	-8

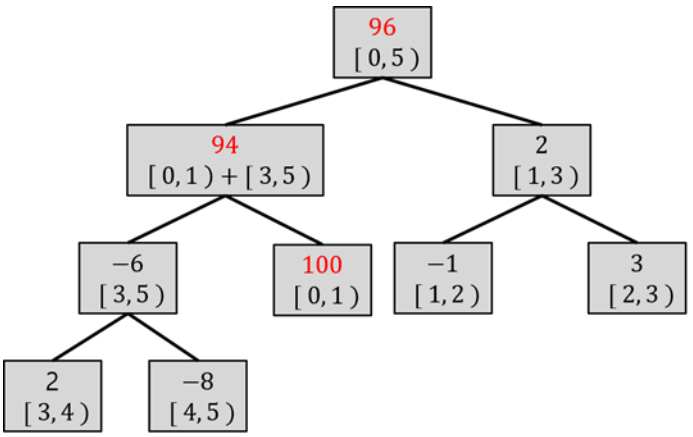
힙과 마찬가지로 세그먼트 트리는 배열로 구현할 수 있습니다. 0번 노드는 버리고, 1번 노드가 루트가 되는 방식으로 구현하면 완전 이진 트리의 모습을 하며, 필요한 배열의 길이는 2N이 되고, 리프 노드의 인덱스는 [N , 2N)입니다.

완전 이진 트리이기 때문에 높이는 O(logN)이 됩니다.

```
constexpr size_t n = 5;


int a[n] = {5, -1, 3, 2, -8};

int segtree[2 * n];


void init() {
	for (size_t i = 0; i < n; ++i) {
		segtree[i + n] = a[i];
	}
	for (size_t i = n - 1; i != 0; --i) {
		segtree[i] = segtree[i << 1] + segtree[i << 1 | 1];
	}
}
```

2.4. Point Update

원소 하나(리프 노드)가 바뀌면 그 원소를 포함하는 모든 구간(리프 노드의 모든 조상)의 값도 바뀌어야 합니다.



a_0	a_1	a_2	a_3	a_4
5	-1	3	2	-8

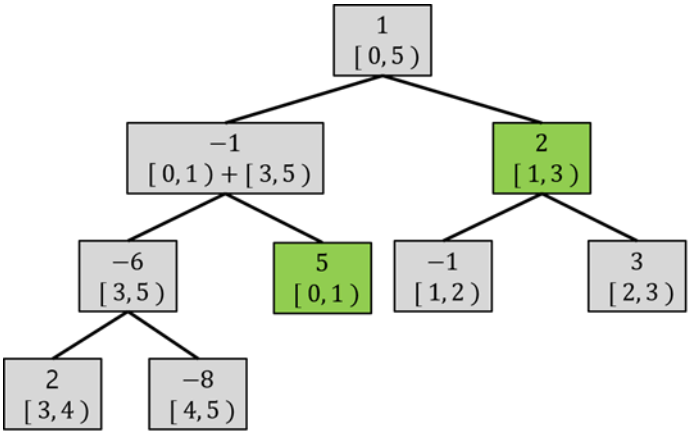
100

```
// i번째 값을 x로 바꿈
void update(size_t i, int x) {
	segtree[i += n] = x;
	while (i >>= 1) {
		segtree[i] = segtree[i << 1] + segtree[i << 1 | 1];
	}
}
```

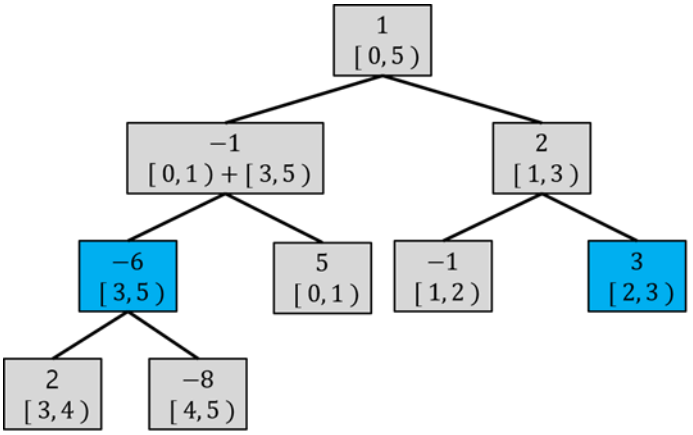
i번째 리프 노드의 인덱스는 i + N입니다. 그 값을 바꾸고자 하는 값으로 바꿔주고, 해당 리프 노드부터 루트 노드까지 부모 노드를 타고 올라가면서 조상 노드의 값을 갱신해줍니다.

2.5. Range Query

구간 쿼리가 들어오면, 몇 개의 노드를 골라서 해당 구간을 정확히 덮어야 합니다.

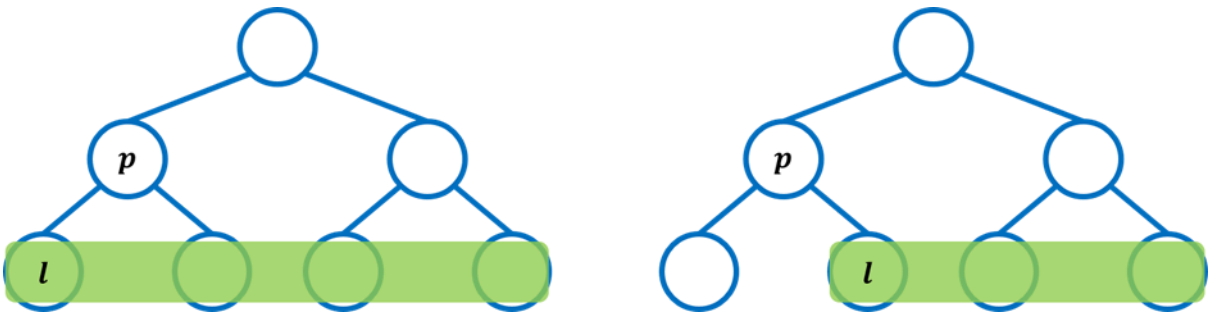


a_0	a_1	a_2	a_3	a_4
5	-1	3	2	-8



a_0	a_1	a_2	a_3	a_4
5	-1	3	2	-8

구간의 시작점 노드 l과 구간의 끝점의 다음 노드 r에서 부모 노드로 올라가면서, 해당 노드에 적힌 값을 결과값에 더해야 하면 더하고, 아니라면 건너뛰는 작업을 반복합니다.



예를 들어, l노드가 왼쪽 그림처럼 부모 노드의 왼쪽 자식일 경우(인덱스가 짝수일 경우), 굳이 저기 적힌 값을 더할 필요가 없습니다. 부모 노드 p에 적힌 값을 더하면 l 뿐만 아니라 오른쪽 자식의 값도 한 번에 더할 수 있으므로 건너뜁니다.

하지만 오른쪽 그림처럼 부모 노드의 오른쪽 자식일 경우(인덱스가 홀수일 경우), 지금 더하지 않고 부모 노드로 올라가게 되면 구간에 포함되지 않는 왼쪽 자식의 값이 섞이게 됩니다. 따라서 l에 적힌 값을 더하고 구간에서 l 노드를 제외하면 됩니다.

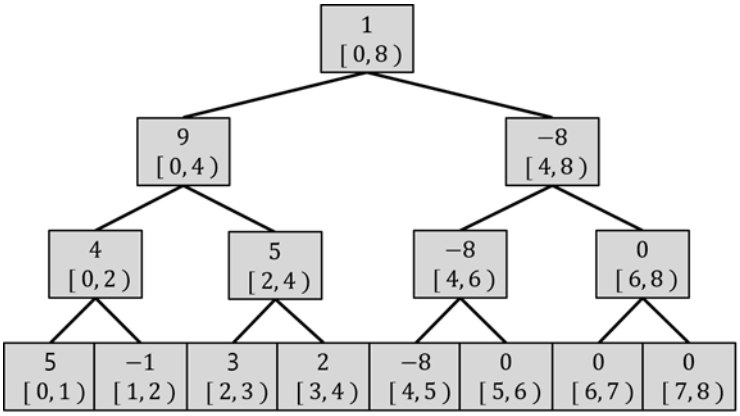
r 노드도 같은 논리로 부모 노드로 계속 올라가면서 필요할 때만 값을 더해주면 됩니다.

```
// [ l , r ) 구간 합
int query(size_t l, size_t r) {
	int result = 0;
	for (l += n, r += n; l != r; l >>= 1, r >>= 1) {
		if (l & 1) result += segtree[l++];
		if (r & 1) result += segtree[--r];
	}
	return result;
}
```

2.6. 기타

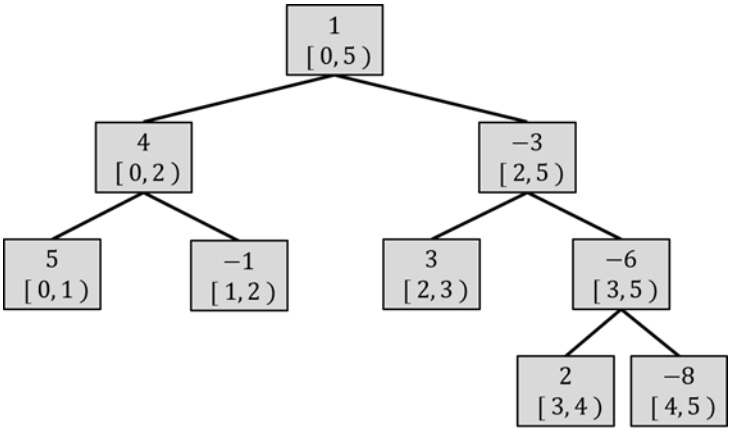
위에서 만든 세그먼트 트리는 노드 순서가 뒤죽박죽입니다. 왼쪽 자식이 오른쪽 구간을, 오른쪽 자식이 왼쪽 구간을 나타내는 경우도 있고 중간이 끊긴 구간을 나타내는 노드도 있습니다. 교환법칙이 성립하지 않는 일반적인 연산에 대한 쿼리를 처리하기 위해선 노드가 나타내는 구간이 정렬 상태를 유지해야 합니다.

이를 위해서 비재귀적으로 구현한 세그먼트 트리는 아래처럼 포화 이진 트리 모양입니다. 포화 이진 트리로 만들기 위해 리프 노드의 개수가 2의 거듭제곱이 되어야 하므로 더미 노드를 둡니다.



a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
5	-1	3	2	-8	0	0	0

또는 재귀적 방식으로 트리를 만들어서 노드 순서를 조정할 수도 있습니다.



3. 기본 문제

· Segment Tree 연습 - 1

· Segment Tree 연습 – 2

해당 문제는 입출력 양이 방대하므로 빠른 입출력 방식을 사용해야합니다.

C++에서 cin, cout을 사용할 경우, main 함수 첫 번째 줄에 std::cin.tie(nullptr)->sync_with_stdio(false); 넣으십시오. 단, 이 경우 테스트 케이스의 일부를 성공해도 출력은 제때 되지 않아서 부분 점수를 받지 못하며 scanf, printf를 같이 사용해선 안됩니다.

아니면 scanf, printf를 사용하십시오.

JAVA에서는 BufferedReader와 BufferedWriter를 사용하십시오.