**세그먼트 트리(구간 트리)란?**

A 배열의 부분 합을 구할 때 A 배열이 계속해서 바뀔 수 있다고 할 때, 부분 합을 트리구조에 저장함으로서 O(logN)의 속도로 A 배열의 부분 합을 빠르게 구할 수 있게 한다.

① : 구간 I, r(I <= r)이 주어졌을 때, A[i] + A[i+1] + … + A[r]을 구해서 출력하기

② : i번째 수를 v로 바꾸기 A[i] = v

수행해야하는 연산은 최대 M번이라 할 때,

1. 구간합 알고리즘

* S[0] = A[0];

for (int i=1; i<n; i++) {

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

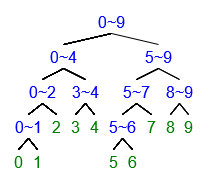
* ①의 경우 O(N), ②의 경우 O(NM) + O(M) = O(NM)이다.

1. 세그먼트 트리

* 세그먼트 트리를 이용하면 ①을 O(logN), ②도 O(logN)만에 수행할 수 있다.
* 세그먼트 트리의 리프 노드와 리프 노드가 아닌 다른 노드는 다음과 같은 의미를 가진다.
  + 리프 노드 : 배열의 그 수 자체
  + 다른 노드 : 왼쪽 자식과 오른쪽 자식의 합을 저장함

어떤 노드의 번호가 x일 때, 왼쪽 자식의 번호는 2\*x, 오른쪽 자식의 번호는 2\*x+1이 된다.

Ex) int arr[] = {3, 4, 5, 6, 7, 8, 9, 10, 11, 12}일 때 구간 합 세그먼트 트리



위의 그림은 N=10 일 때이다.

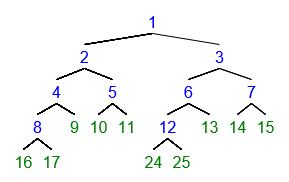
제일 아래 external 노드들이 각각 arr[0], arr[1], … , arr[9]를 의미한다.

x~y의 의미는 x부터 y까지의 합의 범위를 의미한다.

Ex) 0~1 : arr[0] + arr[1] = 7

0~2 : arr[0] + arr[1] + arr[2] = 12

여기에 각 노드의 번호를 그림으로 나타내면 다음과 같다.



**세그먼트 트리의 전체 크기 구하기**

N=10일 때의 세그먼트 트리의 전체 크기(배열 사이즈)를 구하기 위해서는 2^k > N을 만족하는 k의 최솟값을 찾아야 한다. 이 경우 k=4이다.

그리고 난 뒤 2^k를 하면 16이 되고 16에 \*2를 하면 우리가 원하는 세그먼트 트리의 크기가 된다.

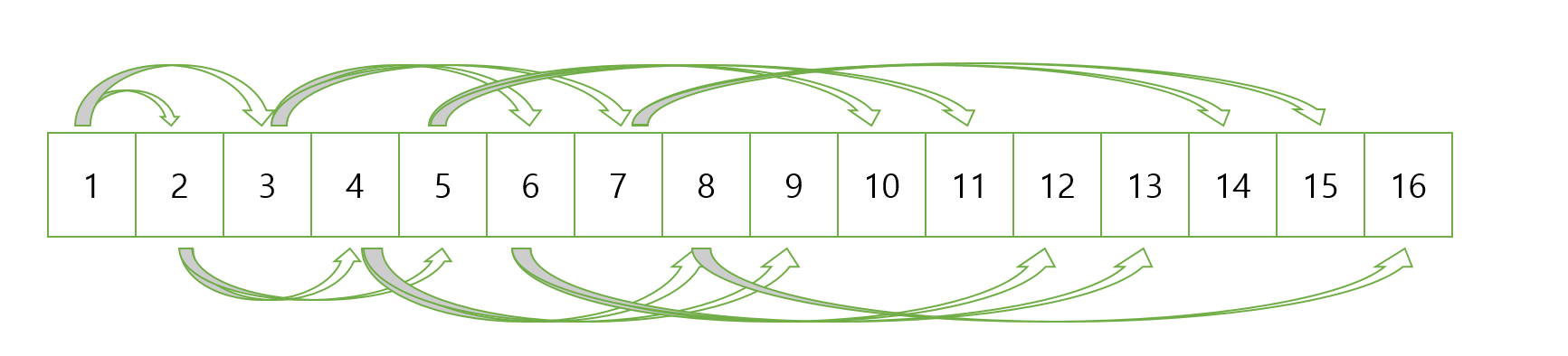
int treeSize = (1 << (int)Math.ceil(log2(N)) + 1);

1 << (4+1) = 1 << 5 = 32가 세그먼트 트리의 크기가 된다.

세그먼트 트리를 형성할 때, 루트 노드를 1로 생각한다.

세그먼트 트리는 완전이진트리에 가깝기 때문에 배열에 모든 값들이 꽉꽉차서 올 가능성이 매우 높아서 배열을 이용해서 만든다.

따라서 왼쪽 노드는 부모 노드의 인덱스 \* 2, 오른쪽 노드는 부모 노드의 인덱스 \* 2 + 1이다.



**세그먼트 트리를 만드는 방법**

1. **초기화(init)**

위와 같은 트리를 생성하는 과정이다.

**public** **static** long init(int[] arr, long[] tree, int node, int start, int end){

**if**(start == end)

**return** tree[node] = arr[start];

int mid = (start + end) / 2;

**return** tree[node] = init(arr, tree, node\*2,

start, mid) + init(arr, tree, node\*2+1, mid+1, end);

}

arr : 입력 배열

tree : 세그먼트 트리

node : 노드번호

start : 노드의 시작번호

end : 노드의 끝번호

각 노드의 왼쪽 자식과 오른쪽 자식으로 분리되는 과정을 보여주는 것이다.

왼쪽 자식에는 start~mid를 보내고, 오른쪽 자식에는 mid+1~end를 보낸다.

arr의 값을 반으로 나누어 재귀적으로 실행한다.(값의 크기가 홀수일 경우는 왼쪽에 하나 더 많이 보낸다).

arr[10] = {3, 4, 5, 6, 7, 8, 9, 10, 11, 12}일 경우

init(arr, tree, 2, 0, 4)와 init(arr, tree, 3, 5, 9)로 나누어 진다.

1번 루트의 왼쪽 자식에는 arr[0]~arr[4]가 들어가고, 오른쪽 자식에는 arr[5]~arr[9]가 들어간다.

Base case는 start와 end가 같아지는 경우이다.

이 의미는 노드의 범위가 1인 리프 노드를 의미한다. 따라서 가장 아래까지 내려왔으므로 node번째인tree[node]에 arr[start]값을 대입해준다는 의미이다.

1. **합(sum)**

합을 구하는 과정은 4가지 과정으로 나뉜다.

i. [left, right]와 [start, end]가 겹치지 않는 경우

* 구간 합을 구하고자 하는 범위와 상관이 없는 경우, 겹치기 않기 때문에 0을 리턴한다.

**if**(left > end || right < start)

ii. [left, right]가 [start, end]를 완전히 포함하는 경우

* 구하고자 하는 합 구간에 포함되는 경우, 구해야 하는 범위인 [left, right]가 [start, end]를 포함하고, 그 노드의 자식도 모두 포함되기 때문에 더 이상 호출을 하는 것은 비효율적이다. 따라서 tree[node]를 리턴해 탐색을 종료한다.

**if**(left <= start && end <= right)

iii. [start, end]가 [left, right]을 완전히 포함하는 경우

* 구하고자 하는 구간 합 범위보다는 크게 있지만 그 내부에 구하고자 하는 구간 합 범위가 있는 경우

**return** sum(tree, node\*2, start, mid, left, right) + sum(tree, node\*2+1, mid+1, end, left, right)

iv. [left, right]와 [start, end]가 겹쳐져 있는 경우(1, 2, 3을 제외한 나머지 경우)

* left <= start <= right <= end

**return** sum(tree, node\*2, start, mid, left, right) + sum(tree, node\*2+1, mid+1, end, left, right)

iii.과 iv.는 왼쪽 자식과 오른쪽 자식을 루트로 하는 트리에서 다시 탐색을 해야 한다.

**private** **static** long sum(long[] tree, int node, int start, int end, int left, int right){

**if**(left > end || right < start)

**return** 0;

**if**(left <= start && end <= right)

**return** tree[node];

int mid = (start + end) / 2;

**return** sum(tree, node\*2, start, mid, left, right) + sum(tree, node\*2+1, mid+1, end, left, right);

}

1. **수 변경하기(update)**

추가수정

[[1]](#endnote-1)참고

1. <https://www.acmicpc.net/blog/view/9>

   <https://www.crocus.co.kr/648>

   <https://wondy1128.tistory.com/150> [↑](#endnote-ref-1)