알고리즘

세그먼트 트리
Segment Tree

이진 트리 기반으로 만들어진 자료구조로
부분에 대한 값들을 구하는데 좋은 자료구조

세그먼트 트리

- 구역을 분할하여 트리를 만드는 것
- 특정 구간 내의 연산(쿼리)를 빠르게 구할 수 있는 자료구조
- 특정 구간의 구간 합, 최대값, 최솟값, 평균값 등을 구하는데 용이함

세그먼트 트리

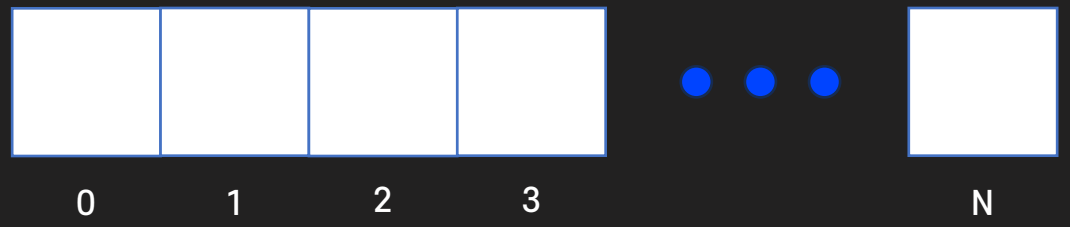
대부분 코딩테스트에서는 누적합(DP), 분할정복 에 대해서 다뤄짐
코딩테스트가 목적이라면 위 개념들을 더 다루는 것이 좋음

하지만 그럼에도 불구하고?

얘 왜 쓰는데 그럼?

0 ~ N 까지 최대값을 구한다고 하면…

❖ BRUTE FORCE



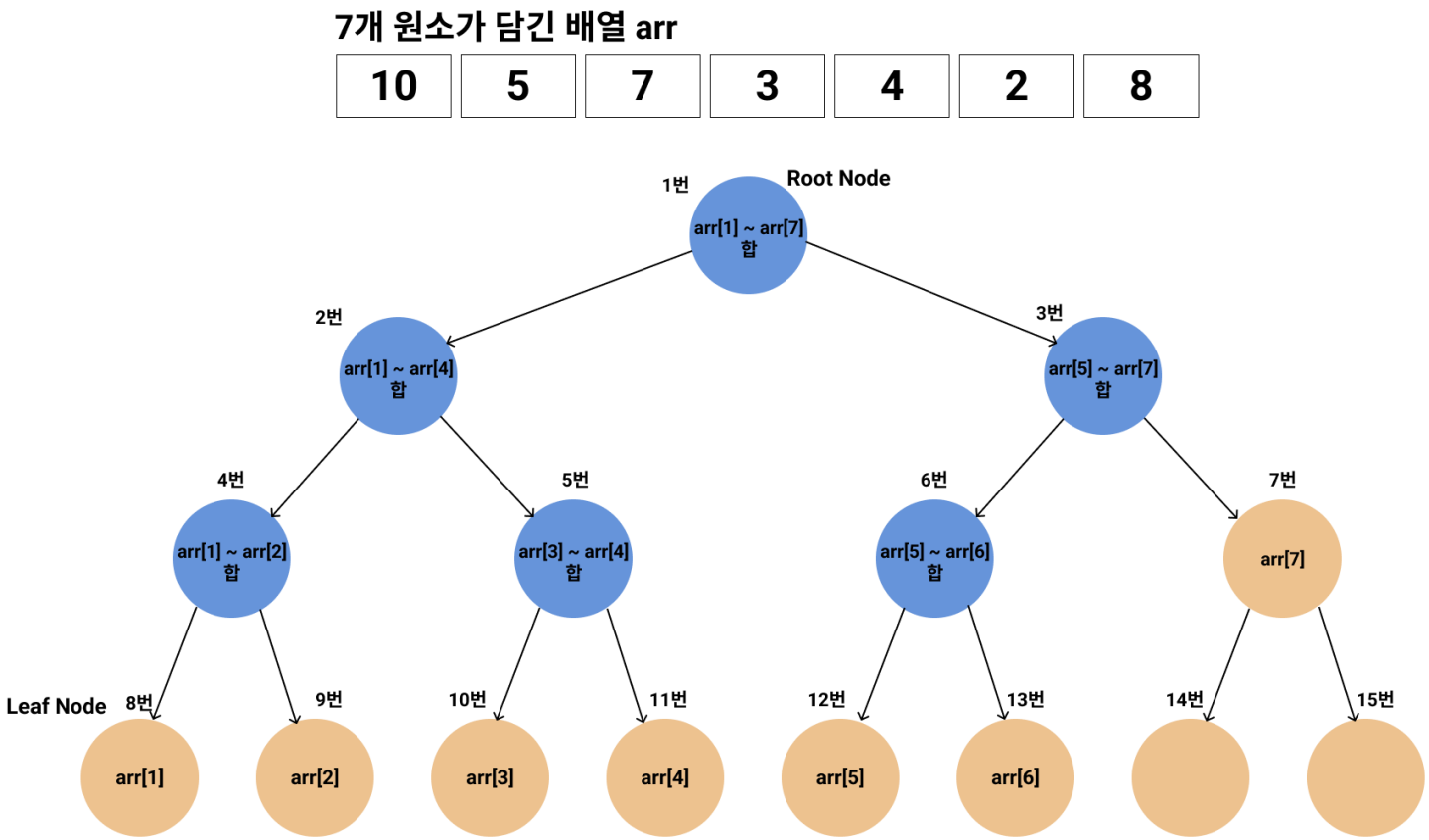
쿼리가 늘어나거나 구하고자 하는 구간이 넓으면…

❖ DP

3	2	2	1	1
	2	2	1	1
		7	1	1
			1	1
				9

속도는 빠르다. 하지만 속도를 위해 메모리를 포기…

세그먼트 트리

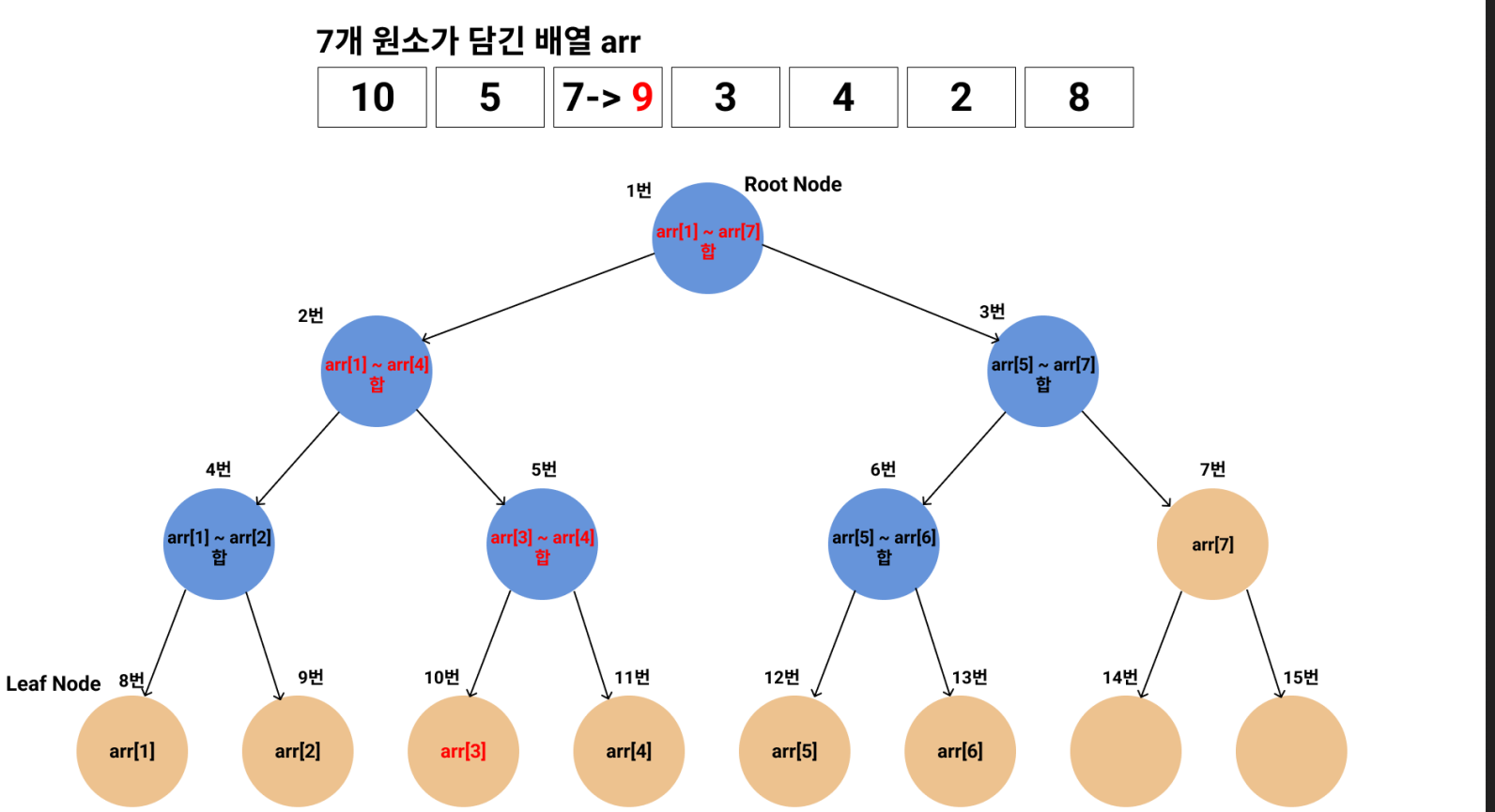


- 이진 트리 특징에 따라 부모, 왼쪽, 오른쪽 자식 인덱스 연산 가능
- 트리의 크기 h 는 3
- 단말 노드에 arr 의 원소들을 저장
- 배열의 길이, 노드의 개수, 경로를 통해 길이가 8인 배열을 위한 트리의 크기는 2^(3 + 1) - 1임을 알 수 있음
- 트리의 인덱스 연산을 위해 0은 비워두므로 2^(h+1) 이면 충분하다.

세그먼트 트리


```java
public class SegmentTree {
    long[] tree;        // 각 원소가 담길 트리
    int treeSize;       // 트리의 크기

    public SegmentTree(int arrSize) {
        // 트리 높이 구하기 - 2의 제곱꼴이 아닐때 +1 효과를 내기위해 올림 처리
        int h = (int) Math.ceil(Math.log(arrSize) / Math.log(2));

        // 높이를 이용한 배열 사이즈 구하기
        this.treeSize = (int) Math.pow(2, h+1);

        // 배열 생성
        tree = new long[treeSize];
    }

    /**
     * 1. 생성 및 구성
     *
     * @param arr : 원소 배열
     * @param node : 현재 노드
     * @param start : 현재구간 배열 시작
     * @param end : 현재구간 배열 끝
     *
     * @return : 원소 배열 값 or 자식노드의 합
     */
    public long init(long[] arr, int node, int start, int end) {
        // 배열의 시작과 끝이 같다면 단말 노드이므로 원소 배열 값 그대로 담는다
        if (start == end) {
            return tree[node] = arr[start];
        }

        // 단말 노드가 아니면 자식노드 합 담기
        return tree[node] = init(arr, node: node * 2, start, end: (start + end) / 2)
                + init(arr, node: node * 2 + 1, start: (start + end) / 2 + 1, end);
    }
```

세그먼트 트리

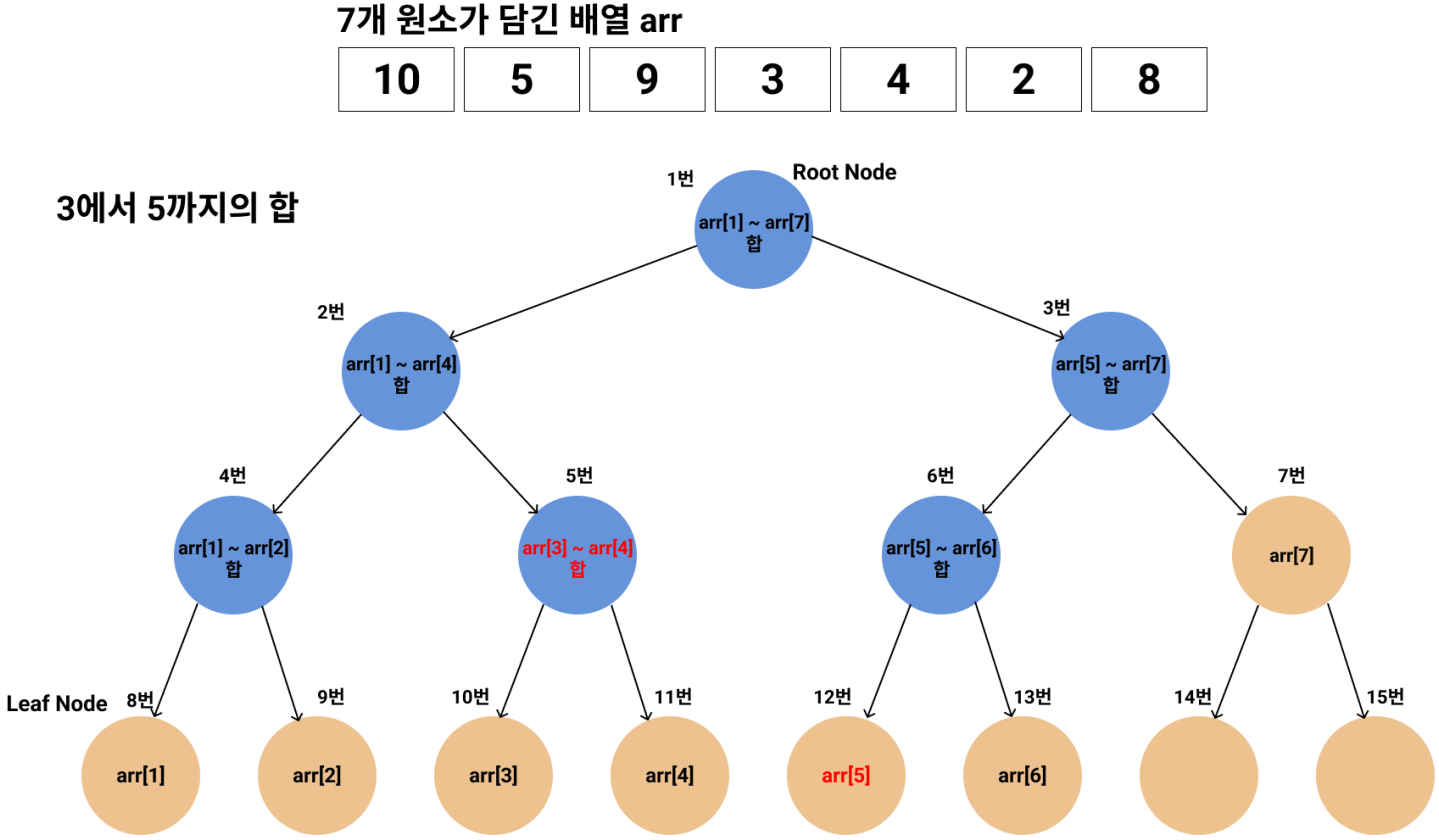


- 배열 원소 값이 변경되면 관련된 모든 부분의 변경이 필요
- 변경할 값과 원래 값의 차이를 구하고 관련된 범위에 차이를 더해서 변경

세그먼트 트리


```java
/**
 * 2. 데이터 변경
 *
 * @param node : 현재 노드 idx
 * @param start : 배열의 시작
 * @param end : 배열의 끝
 * @param idx : 변경된 데이터의 idx
 * @param diff : 원래 데이터 값과 변경 데이터값의 차이
 */
public void update(int node, int start, int end, int idx, long diff) {
    // 만약 변경할 index 값이 범위 바깥이면 확인 불필요
    if (idx < start || end < idx) return;

    // 변경된 값과 원래 값의 차이를 저장
    tree[node] += diff;

    // 단말 노드가 아니면 아래 자식들도 확인
    if (start != end) {
        update( node: node * 2, start,  end: (start + end) / 2, idx, diff);
        update( node: node * 2 + 1,  start: (start + end) / 2 + 1, end, idx, diff);
    }
}
```

세그먼트 트리



- 3 ~ 5번까지 합을 구한다면 5번, 12번 노드 합만 구하면 됨
- 자식 노드로 찾아 내려가다 범위를 벗어나면 0 return
- 찾고자 하는 범위면 그 값을 그대로 return

세그먼트 트리


```java
/**
 * 3. 구간 합 구하기
 *
 * @param node : 현재 노드
 * @param start : 배열의 시작
 * @param end : 배열의 끝
 * @param left : 원하는 누적합의 시작
 * @param right : 원하는 누적합의 끝
 * @return : 누적합
 */
public long sum(int node, int start, int end, int left, int right) {
    // 범위를 벗어나는 경우 더할 필요 없다.
    if (left > end || right < start) {
        return 0;
    }

    // 범위 내 완전히 포함시에는 더 내려가지 않고 바로 리턴
    if (left <= start && end <= right) {
        return tree[node];
    }

    // 그 외의 경우 좌, 우로 지속 탐색 진행
    return sum( node: node * 2, start,  end: (start + end) / 2, left, right) +
            sum( node: node * 2 + 1,  start: (start + end) / 2 + 1, end, left, right);
}
```

BOJ 2042 구간 합 구하기

2042번 | 제출 | 맞힌 사람 | 숏코딩 | 재채점 결과 | 채점 현황 | 내 제출 | 난이도 기여 | 강의 | 질문 게시판

구간 합 구하기 성공

골드 I

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	256 MB	92993	22727	11684	25.180%

문제

어떤 N개의 수가 주어져 있다. 그런데 중간에 수의 변경이 빈번히 일어나고 그 중간에 어떤 부분의 합을 구하려 한다. 만약에 1,2,3,4,5 라는 수가 있고, 3번째 수를 6으로 바꾸고 2번째부터 5번째까지 합을 구하라고 한다면 17을 출력하면 되는 것이다. 그리고 그 상태에서 다섯 번째 수를 2로 바꾸고 3번째부터 5번째까지 합을 구하라고 한다면 12가 될 것이다.

입력

첫째 줄에 수의 개수 N(1 ≤ N ≤ 1,000,000)과 M(1 ≤ M ≤ 10,000), K(1 ≤ K ≤ 10,000) 가 주어진다. M은 수의 변경이 일어나는 횟수이고, K는 구간의 합을 구하는 횟수이다. 그리고 둘째 줄부터 N+1번째 줄까지 N개의 수가 주어진다. 그리고 N+2번째 줄부터 N+M+K+1번째 줄까지 세 개의 정수 a, b, c가 주어지는데, a가 1인 경우 b(1 ≤ b ≤ N)번째 수를 c로 바꾸고 a가 2인 경우에는 b(1 ≤ b ≤ N)번째 수부터 c(b ≤ c ≤ N)번째 수까지의 합을 구하여 출력하면 된다.

입력으로 주어지는 모든 수는 -2^{63}보다 크거나 같고, $2^{63}-1$보다 작거나 같은 정수이다.

출력

첫째 줄부터 K줄에 걸쳐 구한 구간의 합을 출력한다. 단, 정답은 -2^{63}보다 크거나 같고, $2^{63}-1$보다 작거나 같은 정수이다.

BOJ 2042 구간 합 구하기


```java
public class Main {

    static int N, M, K;
    static long[] arr, tree;

    public static void main(String[] args) throws Exception {
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        StringTokenizer st = new StringTokenizer(br.readLine());

        N = Integer.parseInt(st.nextToken());
        M = Integer.parseInt(st.nextToken());
        K = Integer.parseInt(st.nextToken());

        arr = new long[N + 1];
        for (int i = 1; i <= N; i++) {
            arr[i] = Long.parseLong(br.readLine());
        }

        int k = (int) Math.ceil(Math.log(N) / Math.log(2)) + 1;
        int size = (int) Math.pow(2, k);

        tree = new long[size];

        init( start: 1, N,  node: 1);
```

BOJ 2042 구간 합 구하기


```
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < M + K; i++) {
            st = new StringTokenizer(br.readLine());

            int a = Integer.parseInt(st.nextToken());
            int b = Integer.parseInt(st.nextToken());
            long c = Long.parseLong(st.nextToken());

            switch (a) {
                case 1:
                    long diff = c - arr[b];
                    arr[b] = c;
                    update( start: 1, N,  node: 1, b, diff);
                    break;
                case 2:
                    sb.append(sum( start: 1, N,  node: 1, b, (int) c)).append("\n");
                    break;
            }
        }

        System.out.println(sb);
    }
```

BOJ 2042 구간 합 구하기


```
static long init(int start, int end, int node) {
    if (start == end) return tree[node] = arr[start];

    int mid = (start + end) / 2;
    return tree[node] = init(start, mid, node: node * 2) + init( start: mid + 1, end, node: node * 2 + 1);
}

static long sum(int start, int end, int node, int left, int right) {
    if (left > end || start > right) return 0;
    if (left <= start && end <= right) return tree[node];

    int mid = (start + end) / 2;
    return sum(start, mid, node: node * 2, left, right) + sum( start: mid + 1, end, node: node * 2 + 1, left, right);
}

static void update(int start, int end, int node, int idx, long diff) {
    if (idx < start || end < idx) return;

    tree[node] += diff;

    if (start != end) {
        int mid = (start + end) / 2;

        update(start, mid, node: node * 2, idx, diff);
        update( start: mid + 1, end, node: node * 2 + 1, idx, diff);
    }
}
```