컴퓨터공학 All in One

C/C++ 문법, 자료구조 및 심화 프로젝트 (나동빈)
제 41강 - 세그먼트 트리

Fast campus

학습 목표

세그먼트 트리

1) 세그먼트 트리의 필요성과 구현 방법에 대해서 이해합니다.
2) 세그먼트 트리를 활용하여 구간 합을 구하는 방법을 알 수 있습니다.

세그먼트 트리

구간 합

1) 구간 합은 여러 개의 데이터가 연속적으로 존재할 때 특정한 범위의 데이터의 합을 구하는 것을 의미합니다.

Fast campus

세그먼트 트리

선형적으로 구간 합 구하기

- 배열 초기화하기

인덱스	0	1	2	3	4	5	6	7	8	9	10	11
값	7	1	9	5	6	4	1	8	3	7	2	7

세그먼트 트리

선형적으로 구간 합 구하기

- 구간 합 구하기: 인덱스 2부터 10까지 [합계: **45**]

인덱스	0	1	2	3	4	5	6	7	8	9	10	11
값	7	1	9	5	6	4	1	8	3	7	2	7

세그먼트 트리

선형적으로 구간 합 구하기

1) 선형적으로 구간 합을 구하는 과정은 $O(N)$의 시간 복잡도를 가집니다.

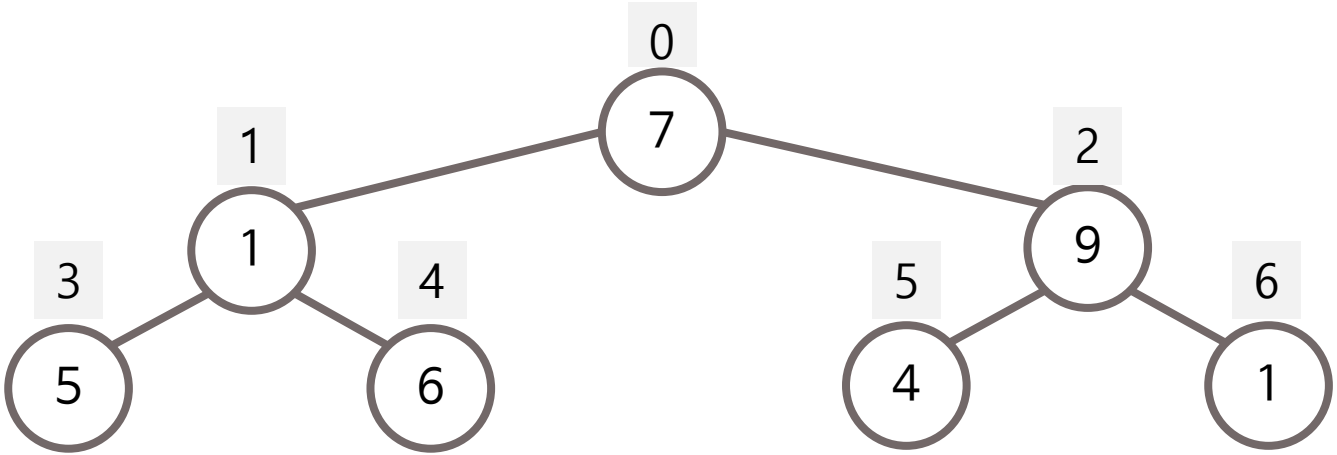
세그먼트 트리

트리 구조로 구간 합 구하기

1) 따라서 선형적으로 구간 합을 구하는 것은 비효율적입니다.
2) 트리 구조를 활용하여 구간 합을 구하는 과정은 $O(logN)$의 시간 복잡도를 가집니다.

세그먼트 트리

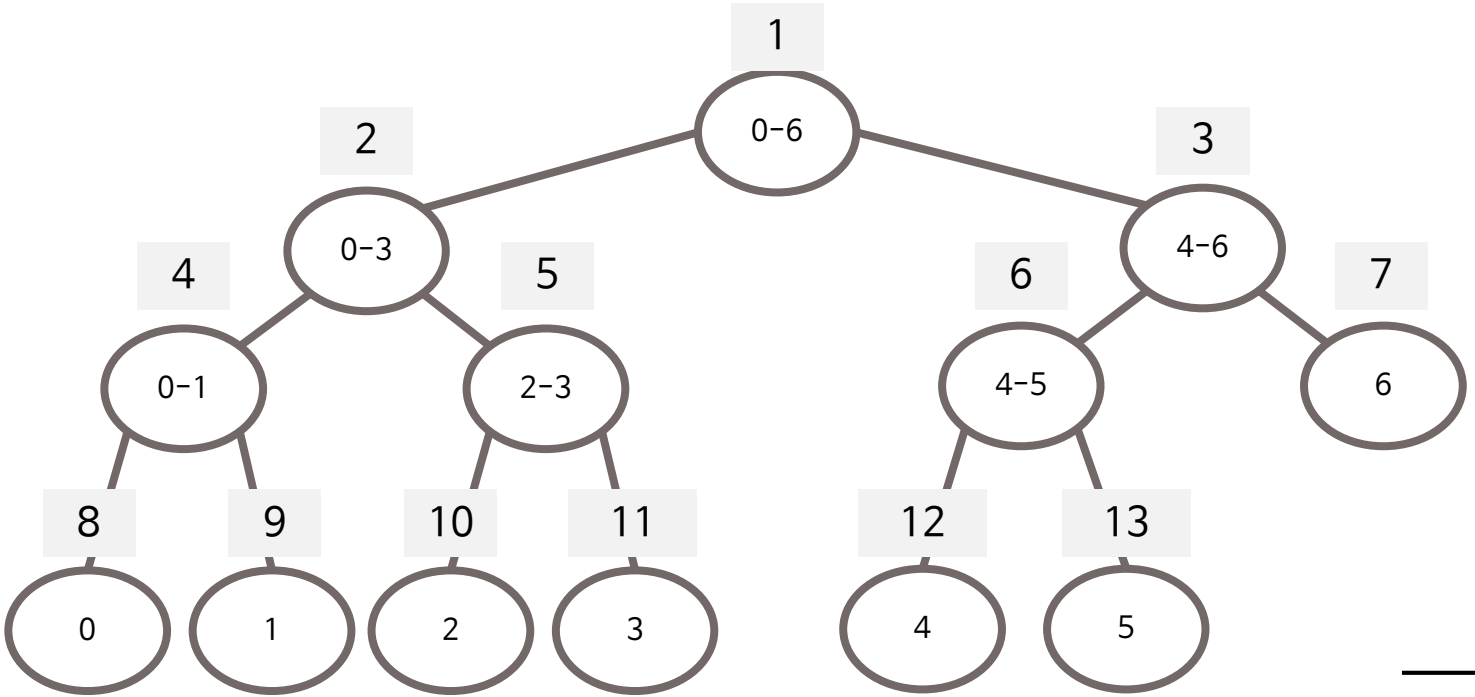
트리 구조로 구간 합 구하기 - 완전 이진 트리에 데이터 삽입하기



Fast campus

세그먼트 트리

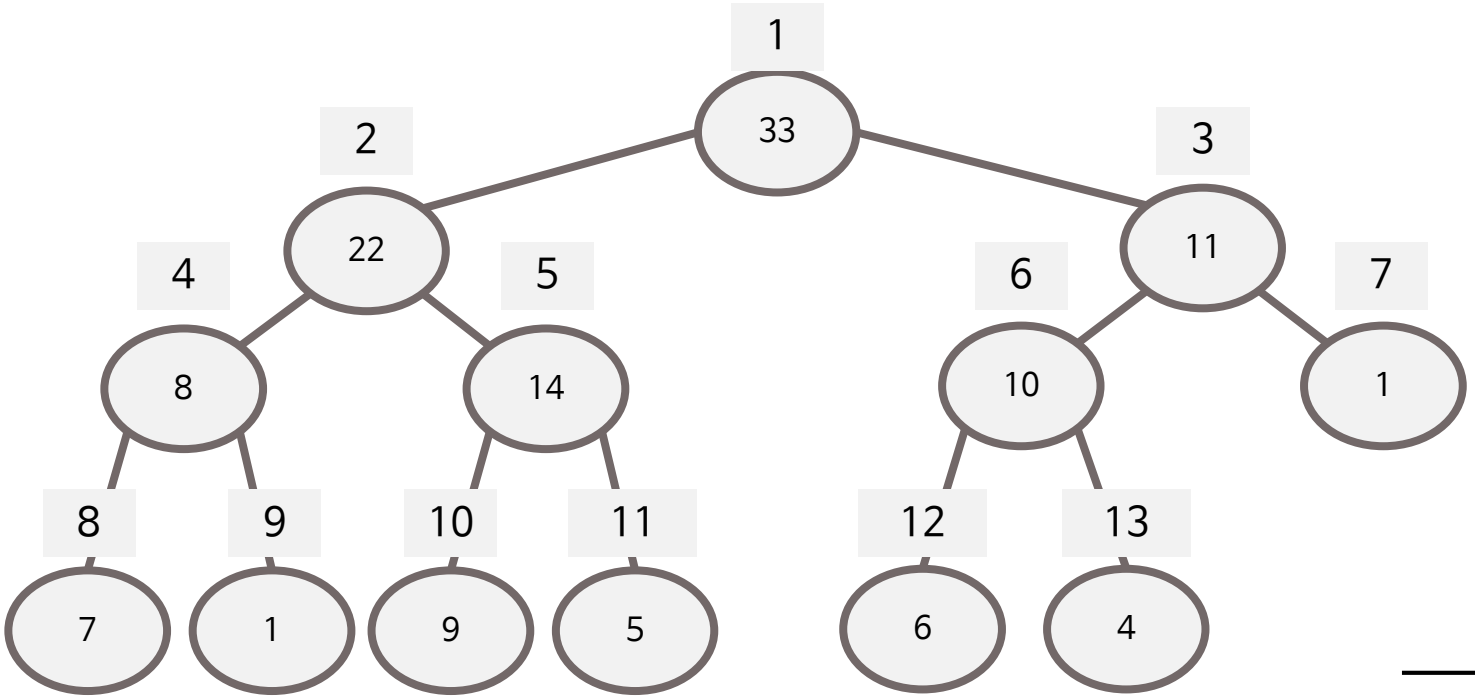
트리 구조로 구간 합 구하기 - 구간 합 트리 생성하기 [특정 범위 인덱스의 값을 저장]



Fast campus

세그먼트 트리

트리 구조로 구간 합 구하기 - 구간 합 트리 생성하기 [특정 범위 인덱스의 값을 저장]



세그먼트 트리

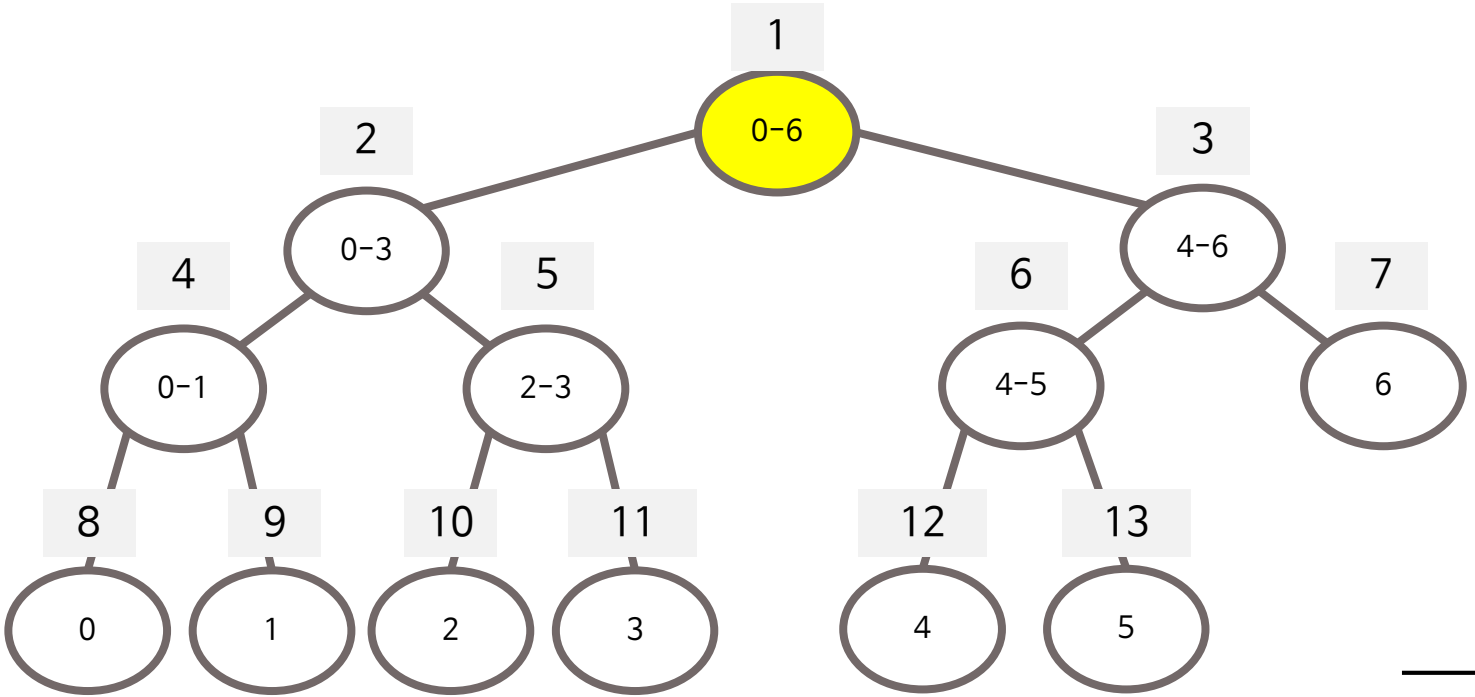
트리 구조로 구간 합 구하기 - 구간 합 트리 생성하기 [특정 범위 인덱스의 값을 저장]

```
#include <stdio.h>
#define NUMBER 7

int a[] = { 7, 1, 9, 5, 6, 4, 1 };
int tree[4 * NUMBER]; // 4를 곱하면 모든 범위를 커버할 수 있습니다.

// start: 시작 인덱스, end: 끝 인덱스
int init(int start, int end, int node) {
  if (start == end) return tree[node] = a[start];
  int mid = (start + end) / 2;
  // 재귀적으로 두 부분으로 나눈 뒤에 그 합을 자기 자신으로 합니다.
  return tree[node] = init(start, mid, node * 2) + init(mid + 1, end, node * 2 + 1);
}
```

Fast campus

세그먼트 트리

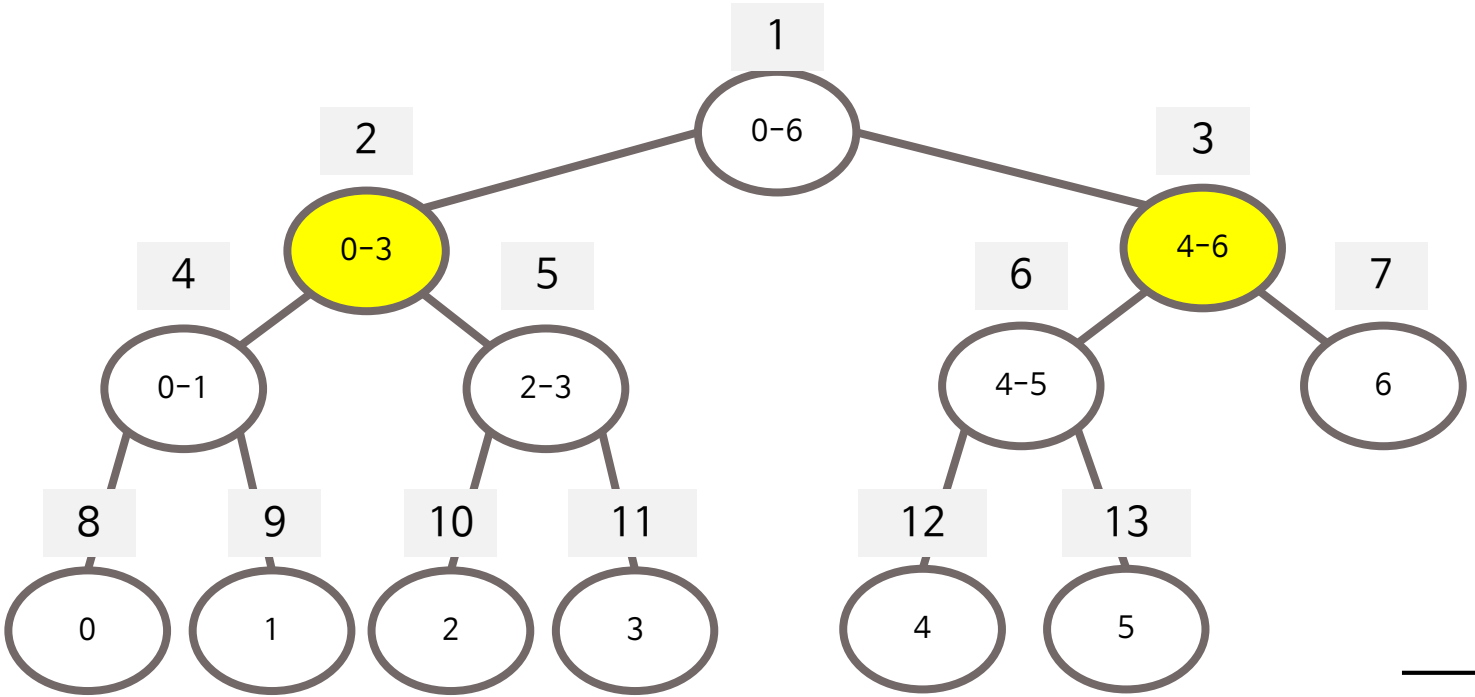
트리 구조로 구간 합 구하기 - 구간 합 계산하기 [인덱스 2부터 4까지]



Fast campus

세그먼트 트리

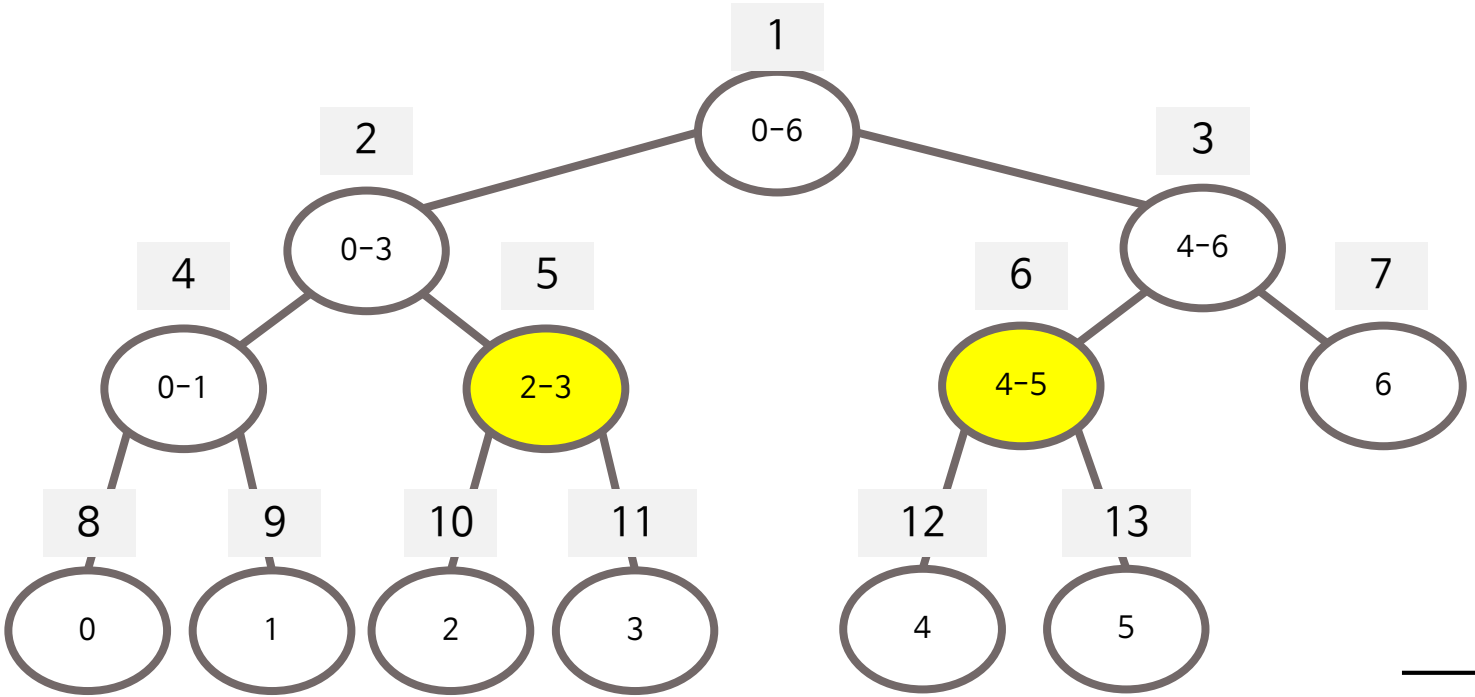
트리 구조로 구간 합 구하기 - 구간 합 계산하기 [인덱스 2부터 4까지]



Fast campus

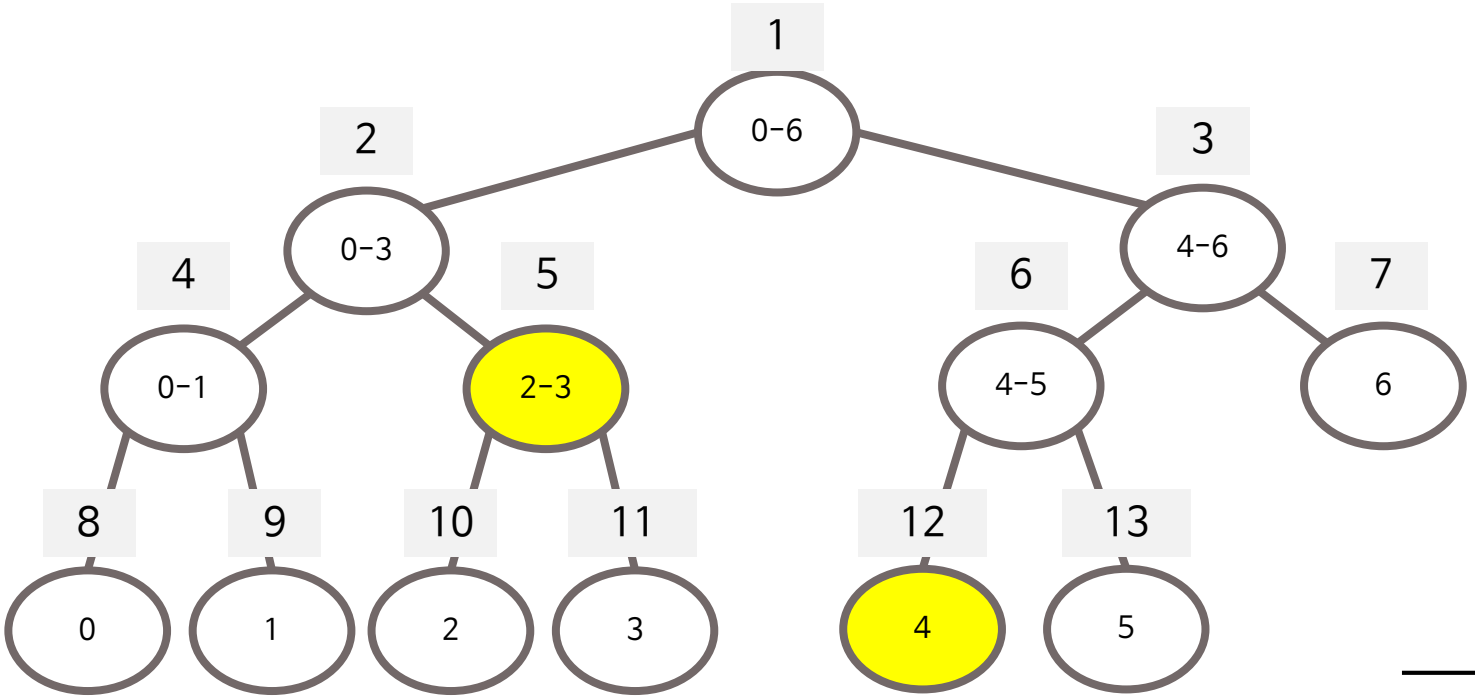
세그먼트 트리

트리 구조로 구간 합 구하기 - 구간 합 계산하기 [인덱스 2부터 4까지]



세그먼트 트리

트리 구조로 구간 합 구하기 - 구간 합 계산하기 [인덱스 2부터 4까지]



세그먼트 트리

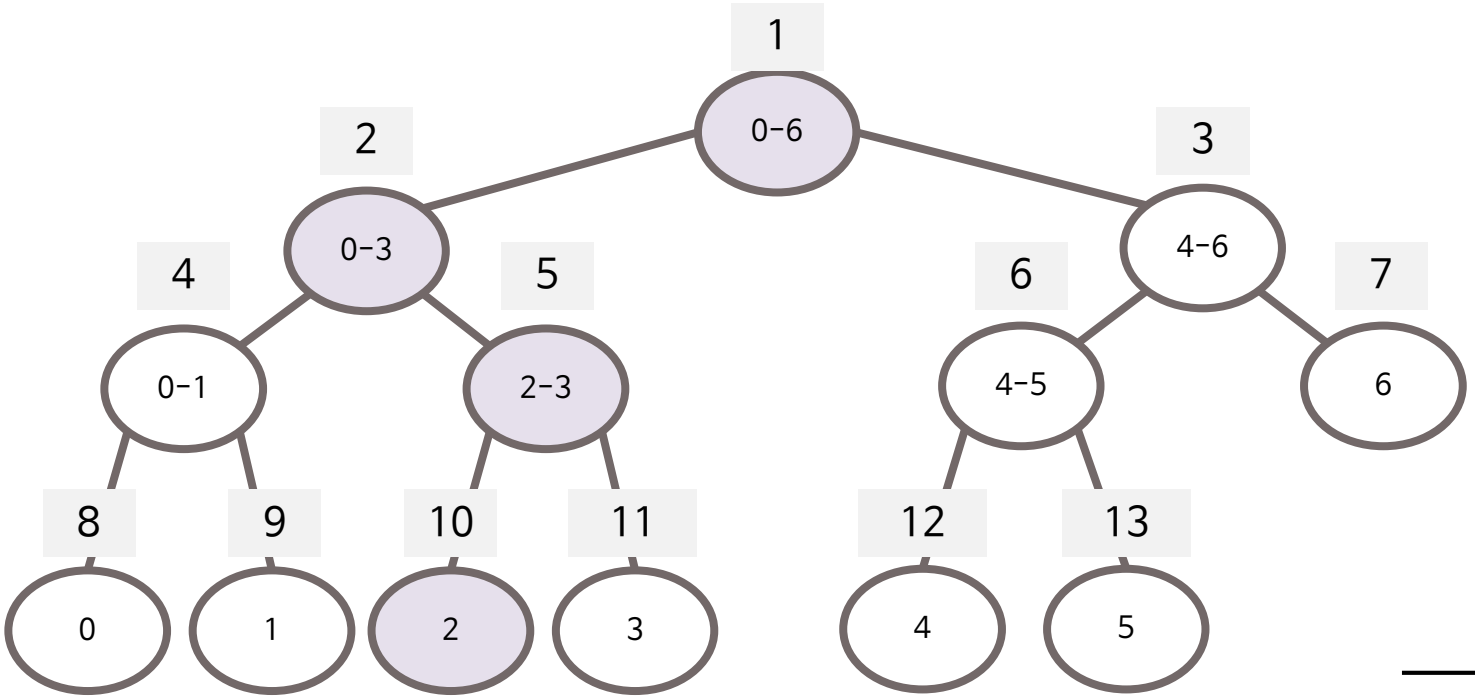
트리 구조로 구간 합 구하기 - 구간 합 계산하기

```
// start: 시작 인덱스, end: 끝 인덱스
// left, right: 구간 합을 구하고자 하는 범위
int sum(int start, int end, int node, int left, int right) {
  // 범위 밖에 있는 경우
  if (left > end || right < start) return 0;
  // 범위 안에 있는 경우
  if (left <= start && end <= right) return tree[node];
  // 그렇지 않다면 두 부분으로 나누어 합을 구하기
  int mid = (start + end) / 2;
  return sum(start, mid, node * 2, left, right) + sum(mid + 1, end, node * 2 + 1, left, right);
}
```

Fast campus

세그먼트 트리

트리 구조로 구간 합 구하기 - 구간 합 수정하기 [인덱스 2]



세그먼트 트리

트리 구조로 구간 합 구하기 - 구간 합 수정하기

```
// start: 시작 인덱스, end: 끝 인덱스
// index: 구간 합을 수정하고자 하는 노드
// dif: 수정할 값
void update(int start, int end, int node, int index, int dif) {
  // 범위 밖에 있는 경우
  if (index < start || index > end) return;
  // 범위 안에 있으면 내려가며 다른 원소도 갱신
  tree[node] += dif;
  if (start == end) return;
  int mid = (start + end) / 2;
  update(start, mid, node * 2, index, dif);
  update(mid + 1, end, node * 2 + 1, index, dif);
}
```

세그먼트 트리

트리 구조로 구간 합 구하기 - 세그먼트 트리 사용해보기

```
int main(void) {
  // 구간 합 트리의 인덱스를 제외하고는 모두 인덱스 0부터 시작합니다.
  // 구간 합 트리 생성하기
  init(0, NUMBER - 1, 1);
  // 구간 합 구하기
  printf("0부터 6까지의 구간 합: %d\n", sum(0, NUMBER - 1, 1, 0, 6));
  // 구간 합 갱신하기
  printf("인덱스 5의 원소를 +3만큼 수정\n");
  update(0, NUMBER - 1, 1, 5, 3);
  // 구간 합 다시 구하기
  printf("3부터 6까지의 구간 합: %d\n", sum(0, NUMBER - 1, 1, 3, 6));
  system("pause");
}
```

Fast campus

배운 내용 정리하기

세그먼트 트리

1) 세그먼트 트리에서의 구간 합 계산 및 원소 수정 과정은 $O(logN)$의 시간 복잡도를 가집니다.