**강의 내용 정리**

언제까지 휘낭시에 만들 것인가? 그러면 재료 얼마나 필요한가?

=>array면 쉽게 접근이 가능

하지만 bst는 분명 쉽게 하려고 한 건데 트리라서 몇 개 남았는지 알기가 힘들다

=>augmentation로 해결

본인을 포함한 서브트리 개수 쭉 적고, 특정 노드보다 같거나 작은 노드(지금까지 만든 수), 전체에서 뺀 수(앞으로 만들 수) == rank를 건다

bst는 루트에서 시작하고 원하는 값이 나올 때까지 쭉 내려간다.

만약 찾는 값보다 작거나 같은 게 있으면 1을 더하고, 왼쪽 서브트리 개수를 더하면 된다.

만약 오른쪽 노드의 휘낭시에 개수를 구하려면 왼쪽 다 더하고 본인보다 작은 거 있으면 더하고, 본인보다 크면 아무것도 안 한다.

힙의 연산 속도 Big(o) 노테이션 insert 기중 최대 O(lg n), delete 기준 최대 O(n)

힙의 단점은 왼쪽 오른쪽의 대소 비교가 안 된다.

바로 자기보다 크거나 작은 아이를 찾을 때 바로 위 아이가 그 다음 작은 건지는 모른다.

그래서 다른 쪽도 다 일일이 봐야 되는 게 힙의 단점이다.

만약 bst가 된다면 어떨까?

insert랑 delete는 h(높이)만큼 가는 연산이다.(형태가 정해지지 않아서 -> 어떤 것이든 루트가 될 수 있음)

input이 주어졌을 때, 트리를 잘 만들어서 h를 가능한 낮추면 연산이 빨라진다.

-> h를 낮추려면, 최대한 트리를 꽉꽉 채울수록 h가 낮아진다.

-> balanced

-> 조금 더 고급스럽게, 왼쪽과 오른쪽의 높이 차이가 최대 1 차이일 때

각 노드의 높이는 해당 노드에서 루트 노드까지의 가장 긴 거리이다.

**AVL tree**

1. AVL tree는 bst insert 그대로

2. balanced 안 맞을 수도 있다 -> 일단 넣고 고치면 된다

고치는 것은 어렵지 않다.

1. 일반적인 일자형일 경우

29

/

24

/

23

-> rotation(balanced를 위해서)

24

/ \

23 29

2. 지그재그인 경우

67

/

50

\

55

-> 1번처럼 돌리면 힙 성질을 위배한다.

-> 일단 50이랑 55를 바꾼 후에 rotation

rotation은 포인터만 바꿔주면 되기 때문에 노드 개수랑 전혀 무관하다 -> O(1)