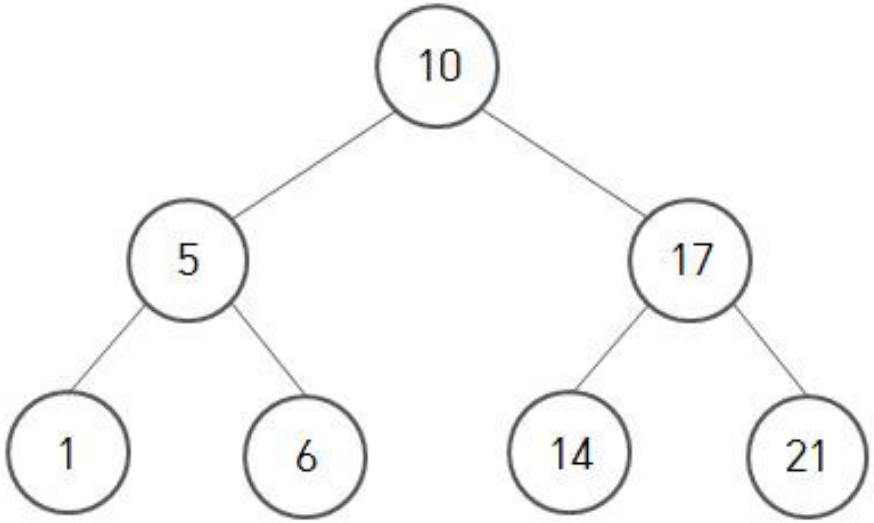
# 이진 검색 트리

NEXTERS

# 이진 검색 트리

이진 탐색에서 아이디어를 가져와서 만듬

매번 후보의 수를 절반으로 줄여 O(lgN) 시간에 값을 찾음



# 이진 검색 트리

각 노드에 값이 있다.

각 노드의 키 값은 모두 달라야 한다.

값들은 전순서가 있다.

노드의 왼쪽 서브트리에는 그 노드의 값보다 작은 값들을 지닌 노드들로 이루어져 있다.

노드의 오른쪽 서브트리에는 그 노드의 값보다 큰 값들로 이루어져 있다.

# 순회

이진 검색 트리를 중위 순회화면 크기 순서로 정렬된 원소의 목록을 얻을 수 있다.

왼쪽 연결로 끝까지 내려가면 최소 원소를 얻을 수 있고 오른쪽 연결로 끝까지 내려가면 최대 원소를 얻을 수 있다.

# 조작

삽입 : 새 원소가 들어갈 위치를 찾고 노드를 추가하기만 하면 된다.

삭제 : 트리 전체의 구조에 영향을 준다. 합치기 연산을 이용하여 구현 하는 방법이 있음

# AVL 트리

Adelson-Velskii와 Landis에 의해 제안 되었다.

각 노드의 왼쪽 서브트리의 높이와 오른쪽 서브트리의 높이 차이가 1 이하인 이진 탐색 트리이다.

모든 노드들이 AVL 성질을 만족한다.

# 트립

AVL 트리나 레드 블랙 트리의 구현이 복잡하기 때문에 간단하게 구현하고자 할 때 주로 사용 됨.

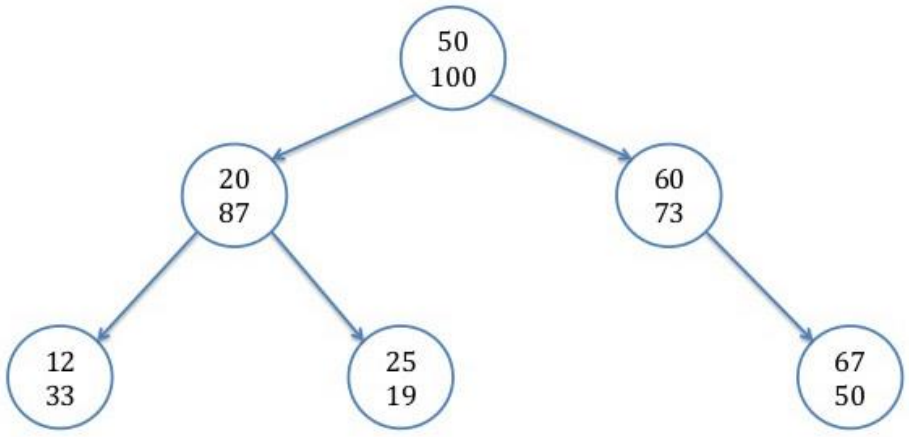
트리의 형태가 원소들의 추가 순서에 따라 결정되지 않고 난수에 따라 임의로 결정 됨

원소들이 순서대로 추가 되거나 삭제 되어도 높이의 기대치가 일정 함

# 트립의 조건

모든 노드에 대해 왼쪽 서브트리에 있는 노드들의 원소는 해당 노드의 원소보다 작고 오른쪽 서브 트리에 있는 노드들의 원소는 해당 노드의 원소보다 큽니다.

모든 노드의 우선순위는 각자의 자식 노드보다 크거나 같음.



# 트립의 구현 (1)

```
struct Node {
        KeyType key;
        int priority;
        int size;
        Node * left;
        Node * right;
        Node(const KeyType& _key) :
        key(_key),priority(rand()), size(1), left(NULL),
        right(NULL) {}
};
```

# 트립의 구현 (2)

```
void setLeft(Node* newLeft) {
            left = newLeft;
            calcSize();
}
void setRight(Node* newRight) {
            right = newRight;
            calcSize();
}
void calcSize() {
            size = 1;
            if (left) size += left->size;
            if (right) size += right->size;
}
```

# 문제 너드인가, 너드가 아닌가?

문제의 수 p, 라면의 수 q 가 주어졌을 때 사람이 참가 할 때마다 너드의 수를 계산 하는 프로그램

주어진 p와 q를 이차원 평면에 표로 나타내어 추가 될 때마다 이전의 점에 지배당하는지 검사한다.

✓문제

https://algospot.com/judge/problem/read/NERD2

✓풀이코드

https://github.com/Nexters/algorithmStudy/blob/master/seokjoong/Chapter22/Nerd2.cpp

# 문제 삽입 정렬 뒤집기

삽입 정렬이 이루어진 과정을 거꾸로 보여주고 원래의 수열을 찾아내는 문제

뒤 에서부터 문제를 풀어 트립을 이용해 k번째 원소를 찾아내는 과정을 반복한다.

✓문제

https://algospot.com/judge/problem/read/INSERTION

✓풀이코드

https://github.com/Nexters/algorithmStudy/blob/master/seokjoong/Chapter22/Insertion.cpp