# EX.1 BST & AVL

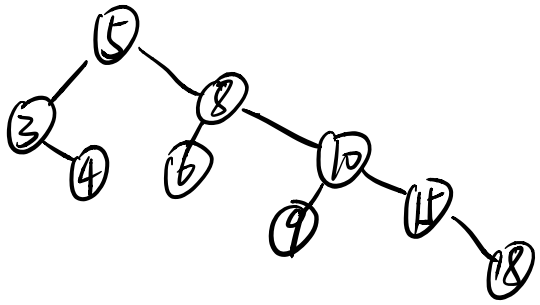
(i) what's common : $l(node.left) < l(node) < l(node.right)$

diff: BST → not necessarily balanced

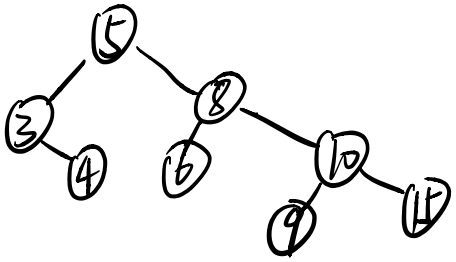
AVL → must be balanced

$$|h(node.left) - h(node.right)| \leq 1$$

BST example




AVL example

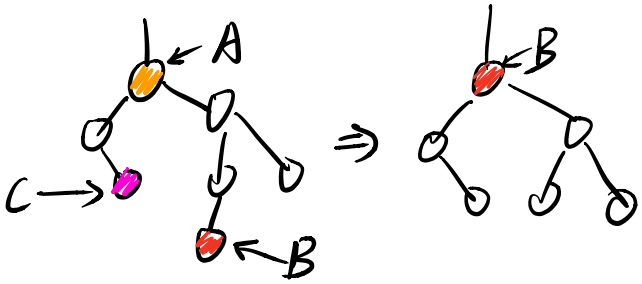



(ii)

find operation : find(node, a)

- a == l(node) → done
- a < l(node) → find(node.left, a)
- a > l(node) → find(node.right, a)

delete operation : BST:




delete (A):

- swap A and B
- remove A

(Can also use C instead of B)

delete operation in AVL is similar but more complicated. Need to update heights and rotate if necessary.

(iii) Min element → Go far left

Max element → Go far right

(iv) median → traverse the tree in-order

result must be sorted

∴ we can find the median in $O(n)$

(v) find elements e, s.t. $x \leq e \leq z$

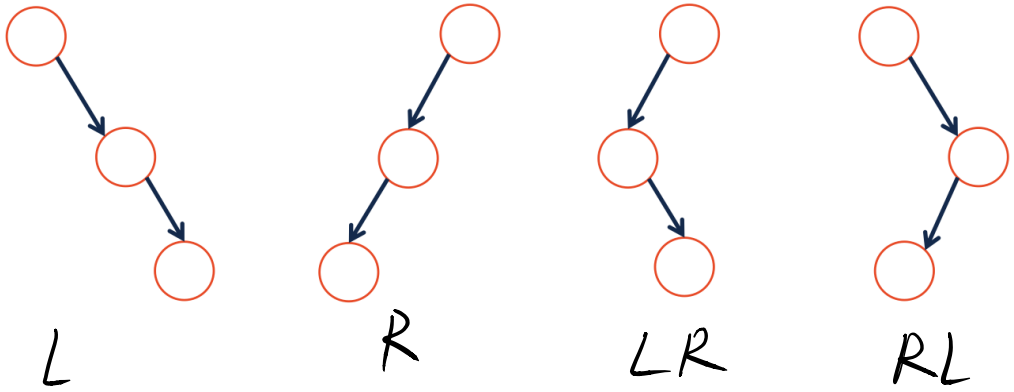
similar with (iv).

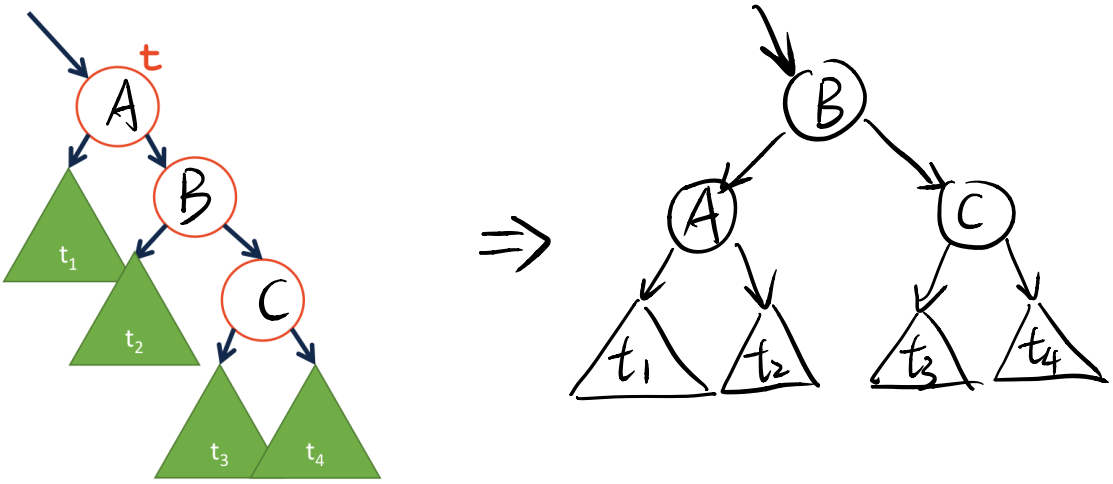
```
template <class K, class V>
void AVLTree<K, V>::insert(Node*& subtree, const K& key, const V& value)
{
    // your code here
    if(subtree == NULL){
        subtree = new Node (key, value);
    }
    else if(key < subtree->key){
        insert(subtree->left, key, value);
        rebalance(subtree);
    }
    else if(key > subtree->key){
        insert(subtree->right, key, value);
        rebalance(subtree);
    }
    subtree->height = max(heightOrNeg1(subtree->left), heightOrNeg1(subtree->right)) + 1;
}
```

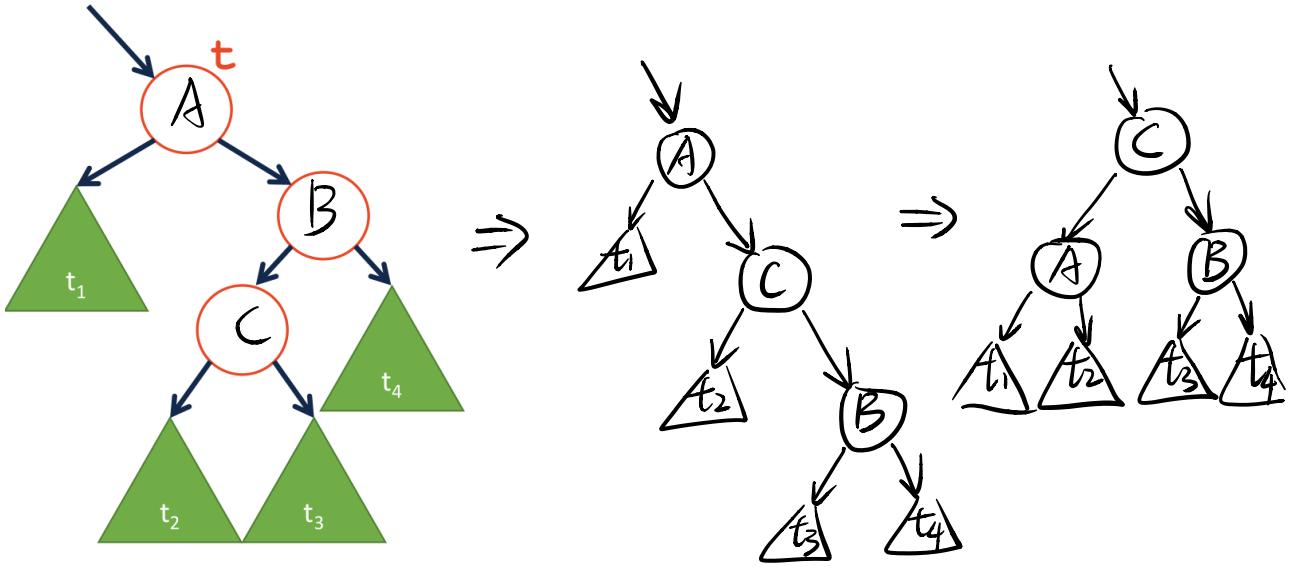
```
template <class K, class V>
void AVLTree<K, V>::rebalance(Node*& subtree)
{
    int difference = heightOrNeg1(subtree->left) - heightOrNeg1(subtree->right);
    if(difference <= 1 && difference >= -1) return;
    else if(difference >= 2){
        if(heightOrNeg1(subtree->left->left) > heightOrNeg1(subtree->left->right)) {
            rotateRight(subtree);
        }
        else rotateLeftRight(subtree);
    }
    else if(difference <= -2){
        if(heightOrNeg1(subtree->right->right) > heightOrNeg1(subtree->right->left)) {
            rotateLeft(subtree);
        }
        else rotateRightLeft(subtree);
    }
    subtree->height = max(heightOrNeg1(subtree->left), heightOrNeg1(subtree->right)) + 1;
}
```

```
template <class K, class V>
void AVLTree<K, V>::remove(Node*& subtree, const K& key)
{
    if (subtree == NULL) return;
    if (key < subtree->key) {
        remove(subtree->left, key);
        rebalance(subtree);
    }
    else if (key > subtree->key) {
        remove(subtree->right, key);
        rebalance(subtree);
    }
    else if(key == subtree->key) {
        if(subtree->left == NULL && subtree->right == NULL) {
        /* no-child remove */
            delete(subtree);
            subtree = NULL;
        }
        else if (subtree->left != NULL && subtree->right != NULL) {
                /* two-child remove */
            Node* temp = subtree->left;
            while(temp->right != NULL) temp = temp->right;
            subtree->key = temp->key;
            subtree->value = temp->value;
            remove(subtree->left, temp->key);
            rebalance(subtree);
        }
        else {
                /* one-child remove */
            Node* temp = subtree;
            if(subtree->left == NULL)   subtree = subtree->right;
            else  subtree = subtree->left;
            delete temp;
            temp = NULL;
        }
    }
    if(subtree == NULL) return;
    rebalance(subtree);
    return;
}
```

Four templates for rotations:



EX.2

(i) A tree rooted at "node" is BST

iff :
- l(node.left) < l(node), if exist
- l(node.right) > l(node), if exist
- node.left and node.right are roots of two BSTs.

LIFE IS PRECIOUS!

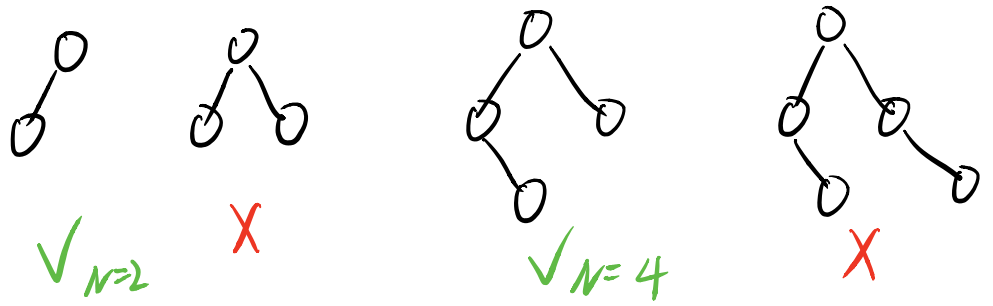
check recursively to save your time

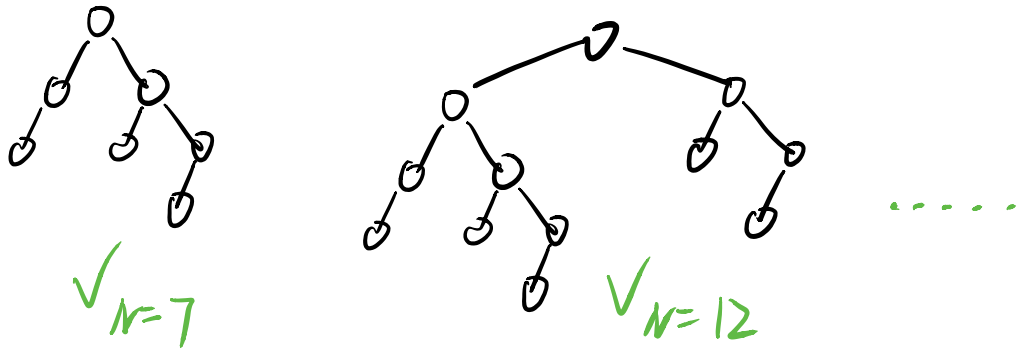
(ii) A tree rooted at "node" is AVL

iff
- l(node.left) < l(node), if exist
- l(node.right) > l(node), if exist
- $|h(\text{node.left}) - h(\text{node.right})| \leq 1$
- node.left and node.right are roots of two BSTs.

EX.3.

A Fibtree of height $h$ is an AVL tree with a min number of nodes among all AVL trees of height $h$.




(§§) A tree rooted at "node" is a fibtree

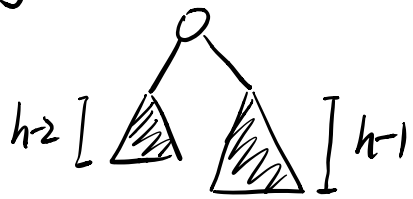
iff { either A ∈ some basic case fibtrees, e.g. h(node) ≤ 3

or { $|h(node.left) - h(node.right)| = 1$

node.left and node.right are roots of two fibtrees

(iii) Denote the number of nodes by $N(h)$

$N(h) = 1 + N(h-1) + N(h-2)$



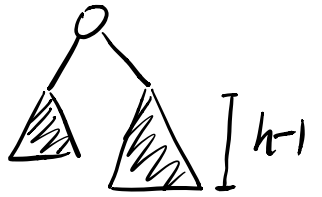

similar with Fibonacci Sequence

- Let's look at number of isomorphisms

  Denote the number of isomorphisms as $I(h)$

  Then we must have:

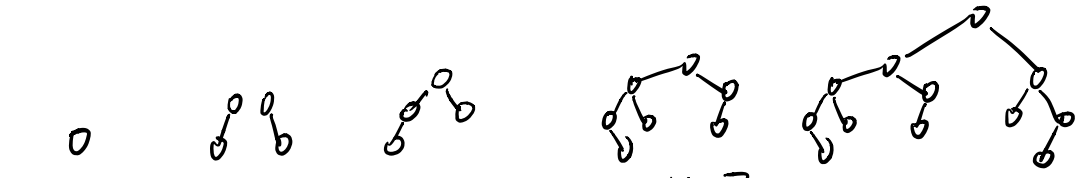
  $I(h) = 2 \times I(h-1) \times I(h-2)$




  Assume $I(h) = 2^{k_h}$

  $\Rightarrow k_h = 1 + k_{h-1} + k_{h-2}$, exactly what we get for $N(h)$ :)

  $\Rightarrow I(h) = 2^{N(h-1)}$



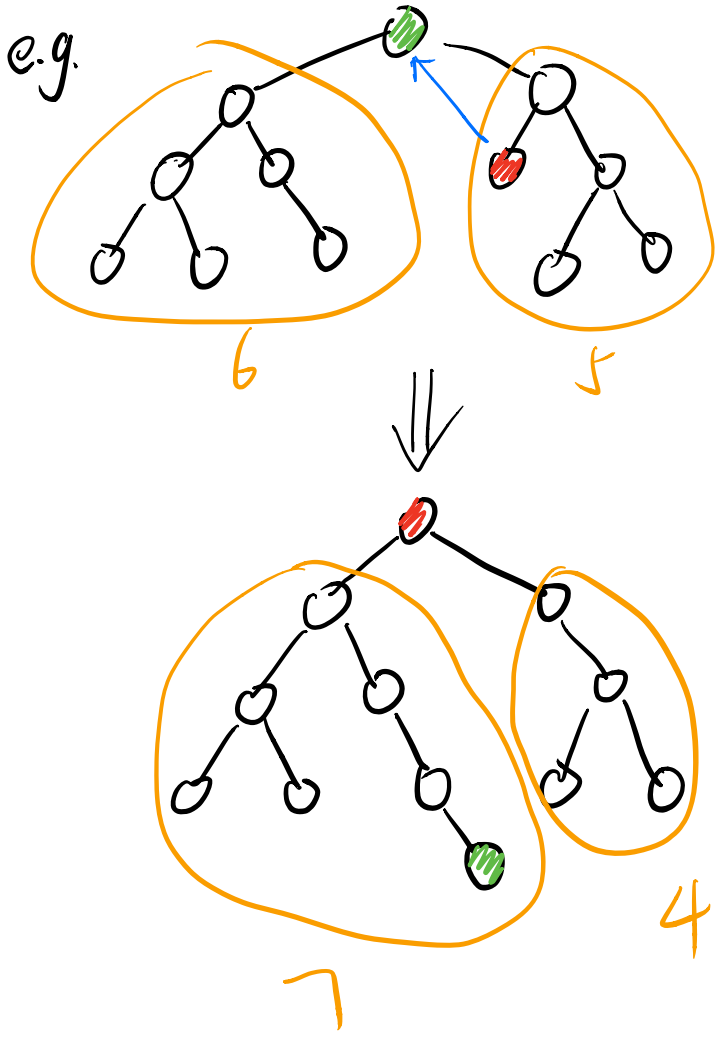
$N(h)=1$ $N(h)=2$ $N(h)=4$ $N(h)=7$ $N(h)=12$ ...

$I(h)=1=2^0$ $I(h)=2^1$ $I(h)=2^2$ $I(h)=2^4$ $I(h)=2^7$ ...

(iv) Easy to check since we know all possible $N(h)$.

(V) Firstly, the number of nodes should match the criterion.

Then: Adjust the number of nodes of two subtrees, with should be $N(h-1)$ and $N(h-2)$




Do this recursively to base cases.