Algorithm and Data Structure

AVL-Tr

Overview

AVL-Trees:

- Find, insert, remove

Runtimes for Binary Search Tree

Find, insert, remove:

Worst case: $\Theta(n)$

Best case: $\Theta(\log n)$

Average case: $\Theta(\log n)$

Aim: Time O(log n) in the worst case

AVL-Tree

Observation:

- Binary search trees can get imbalanced when applying ins operations.

Idea:

- Whenever a subtree rooted at a node v gets imbalanced, apply operations that balance it out in time O(log n).

AVL Tree

Let h(T) be the height of a tree T.

Let v be a node in T and T_l and T_r be the left and right subtree

We denote by ance degree of v.

Definition: A binary search tree T is called an AVL-tree if for each $v \in T, b(v) \in \{-1, 0, 1\}$ holds.

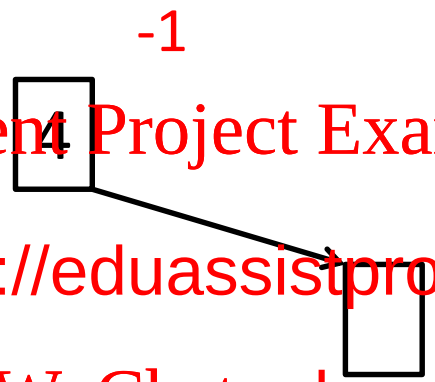
Height of an AVL-tree

Theorem(without proof) Let T be an AVL-tree consisting of n nodes. Then $h(T) \leq 1.44 \log n$

We have to consider the op ind, insert, and delete for AVL-trees.

- Find is as for Binary Search Trees.
- For insert and remove we might have to rebalance the tree.

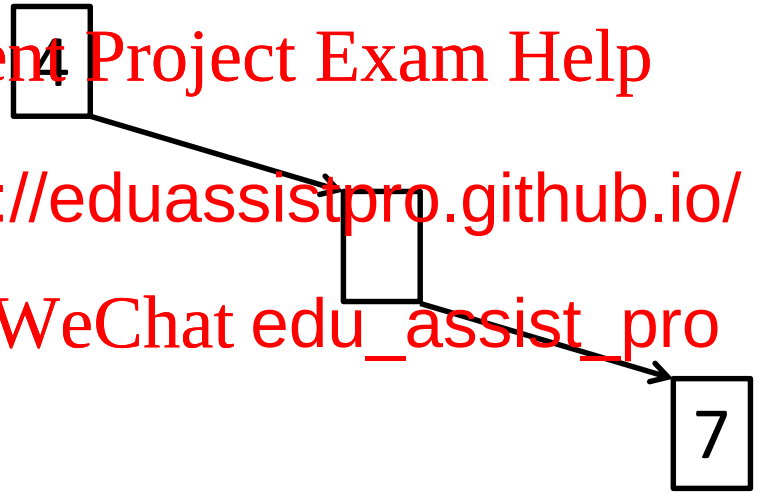
Example Insert



Balance values

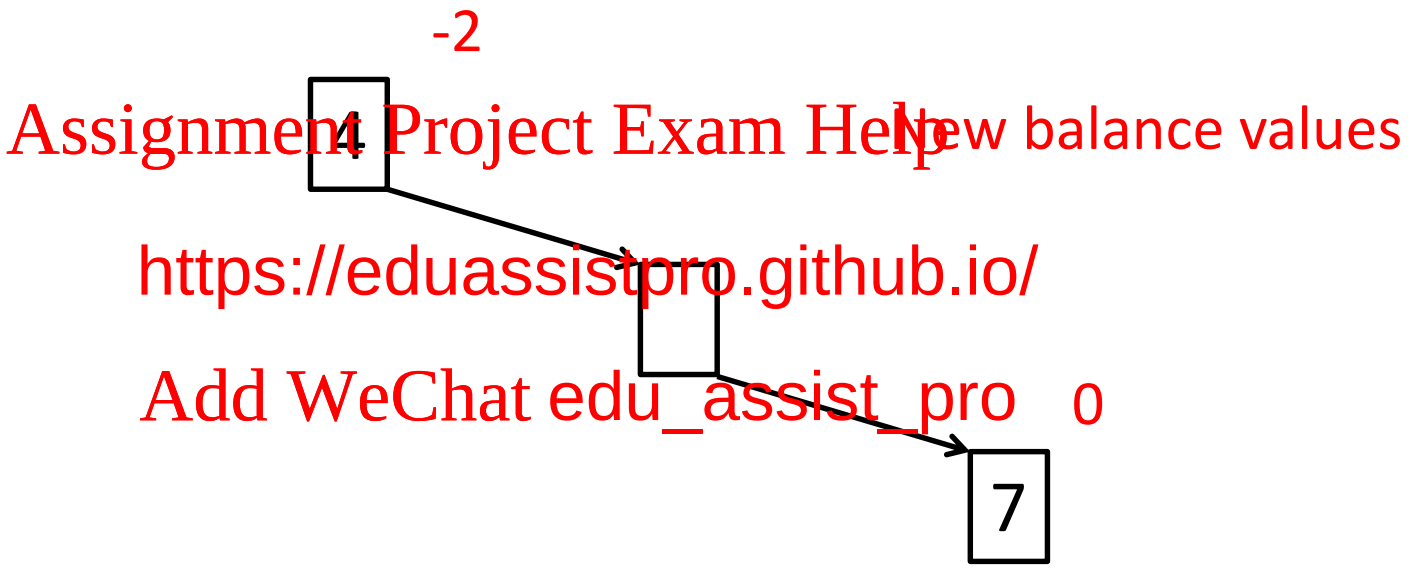
Insert 7

Example Insert

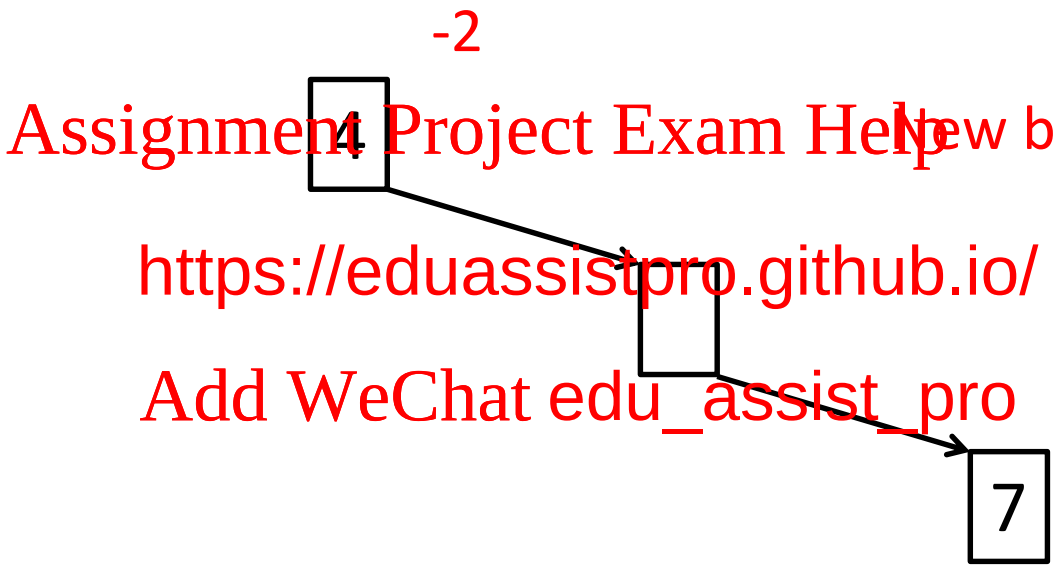


Consider path from new leaf
to the root and check balance values

Example Insert



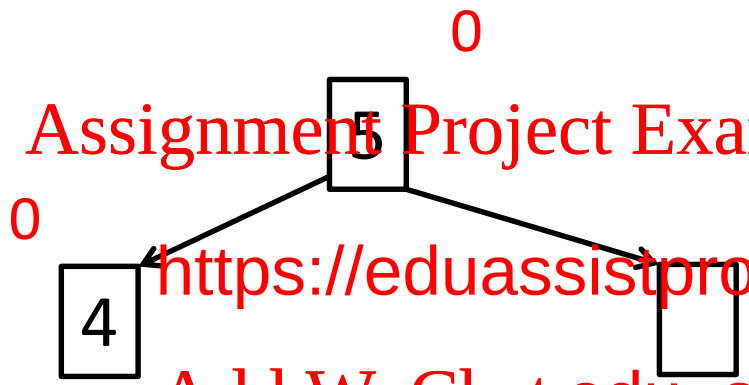
Example Insert



New balance values

AVL-property at node 4 violated

Example Insert



1 Rotation establishes AVL-property again

Insertion

Inserting a new element z can violate the AVL-property.

Consider path erted leaf z to the root and repair AVL-pr

Rebalancing

Let z be the newly inserted leaf.

Consider the path from z to the root (reverse the insertion path

Update the ba

Repair AVL-property (if ne

Insert

- we insert new node z as for Binary Search Trees.
- bal(z)=0 hol
- bal(v) might ode v on the path from z to the root.
- If $b(v) \notin \{-1, -0, 1\}$ rebalance

Rebalancing

Start examining for v, where v is the parent of z, and continue with the parent of v (if necessary).

Assume that the right child x of node v is on the path from z to the root.

Before insertion -> After insertion

- bal(v) = 1 -> bal(v)=0 (height of tree rooted at v has not changed, stop rebalancing)
- bal(v)=0 -> bal(v) = -1 (height of tree rooted at v has increased by 1, stop rebalancing only if v is root, otherwise examine parent of v)
- bal(v)=-1 -> bal(v) = -2 (AVL-property violated, carry out rotation)

Left Rotation

Assume node v and right child x on the path.

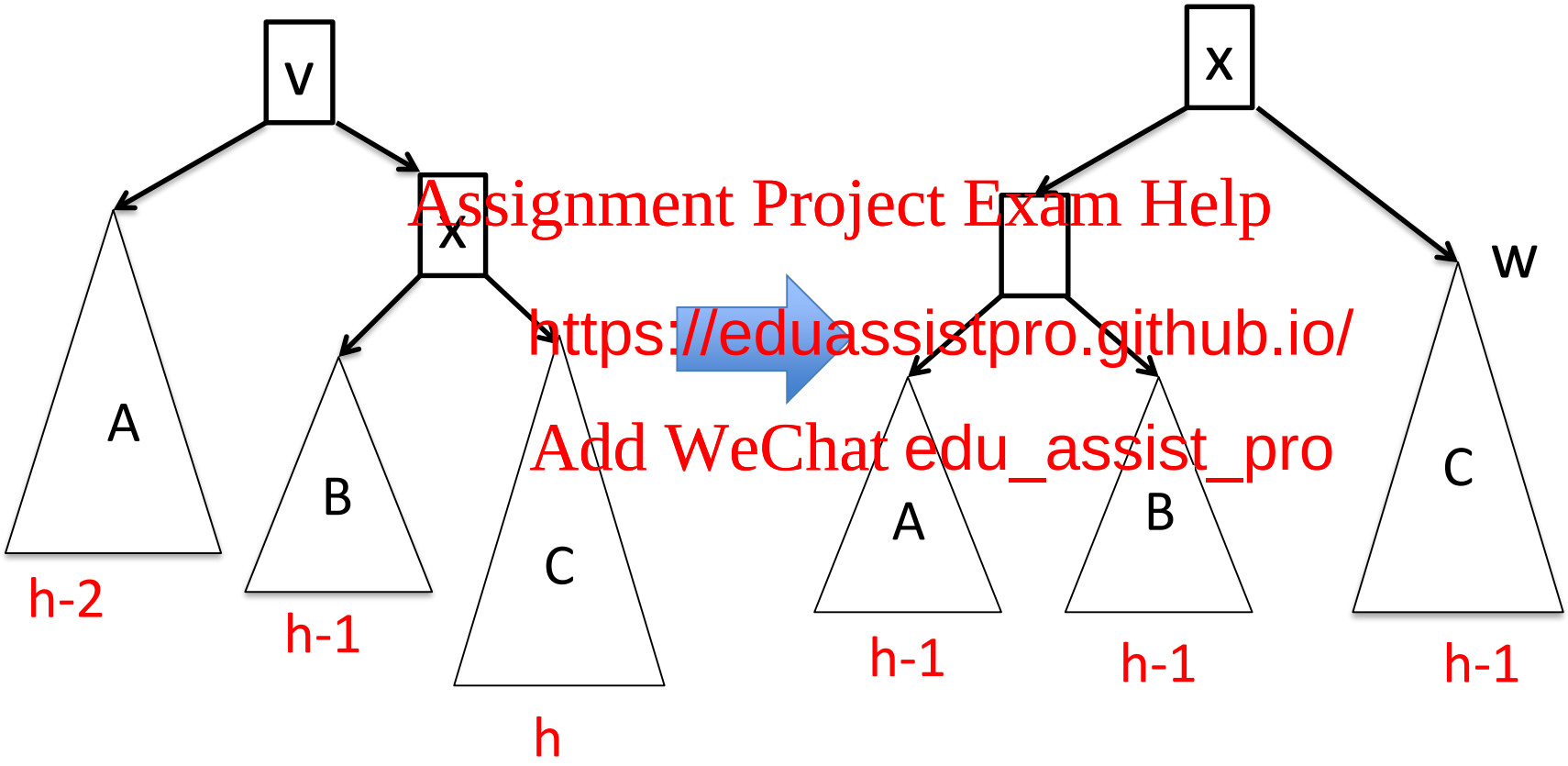
- w is right child of x on the path

  -> Left rotati

  New balance (v)=0

Analogous: Right Rotation

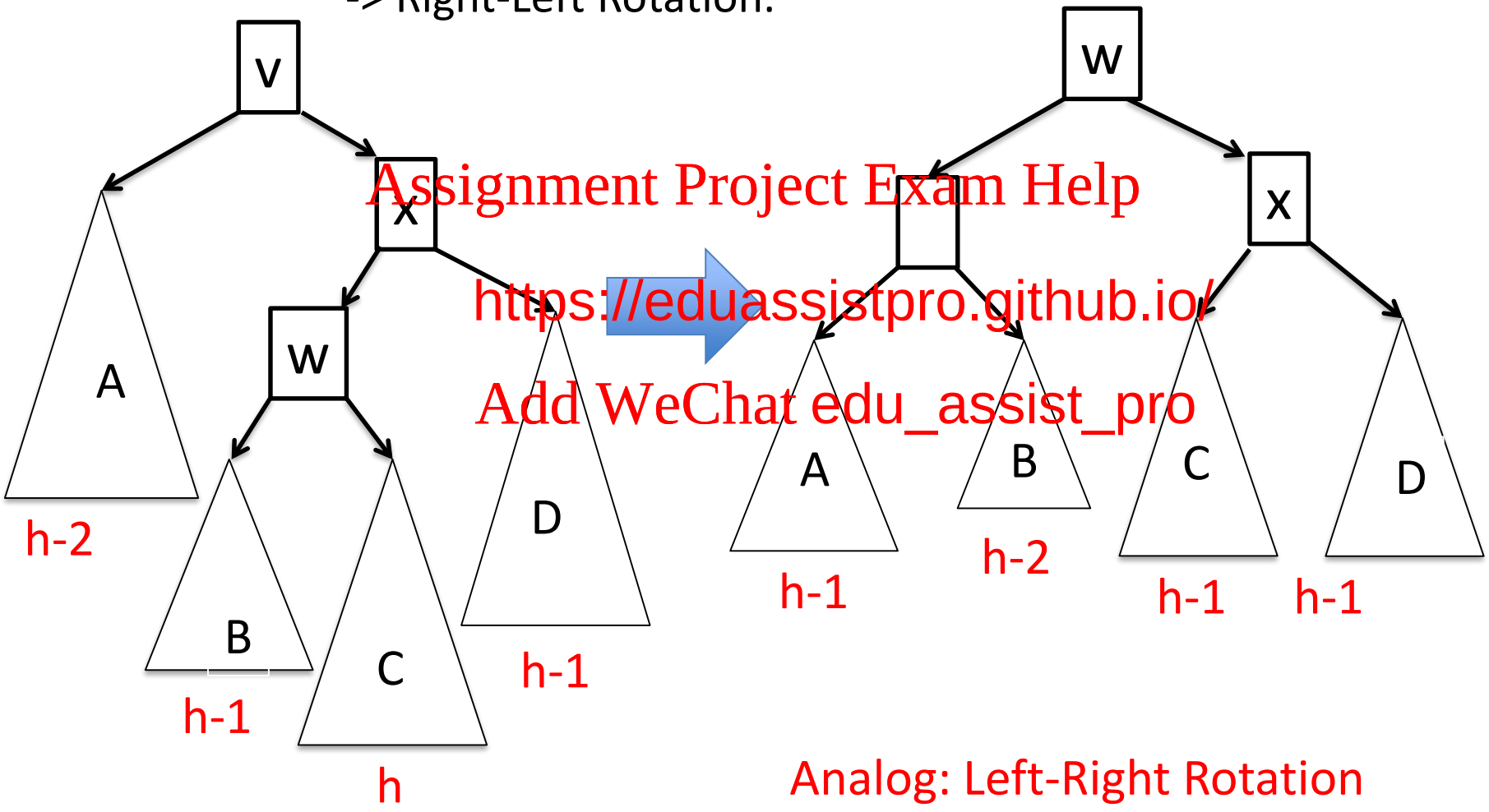
Left Rotation



Analog: Right Rotation

Right-Left Rotation

w is left child of x on the path
-> Right-Left Rotation.



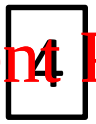
Analog: Left-Right Rotation

Example Insert

Create AVL-Tree for sequence 4, 5, 7, 2, 1, 3, 6

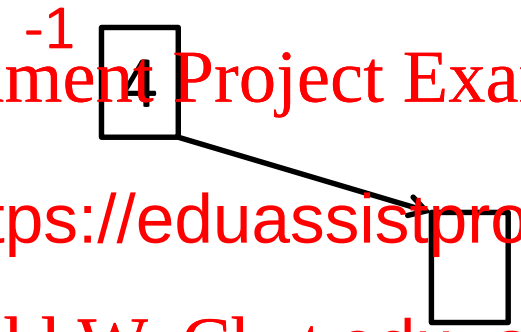
Example Insert

Create AVL-Tree for sequence 4, 5, 7, 2, 1, 3, 6



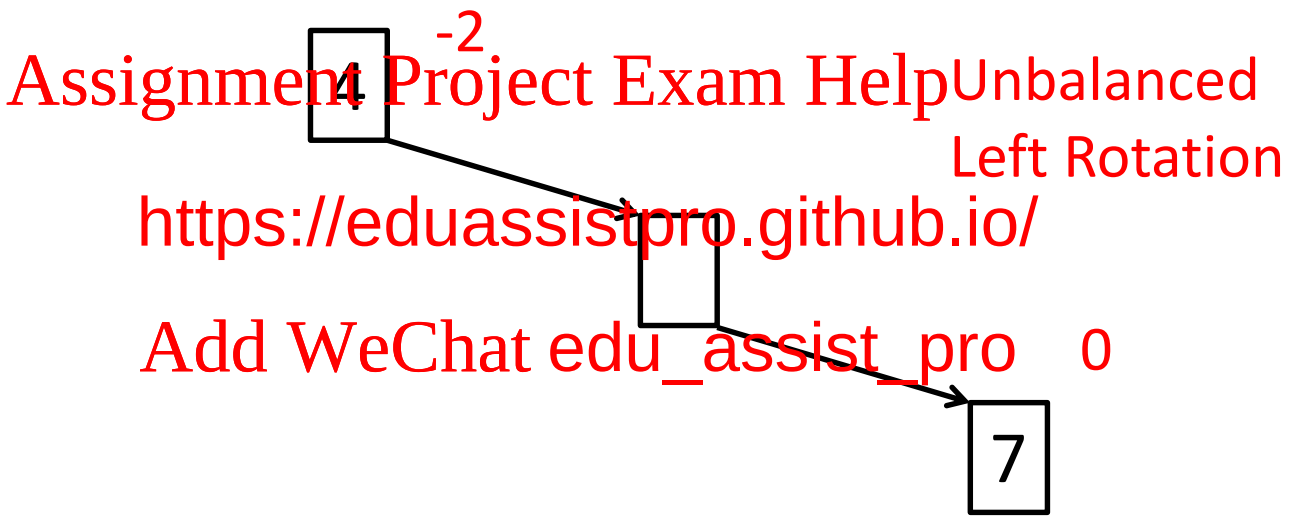
Example Insert

Create AVL-Tree for sequence 4, 5, 7, 2, 1, 3, 6



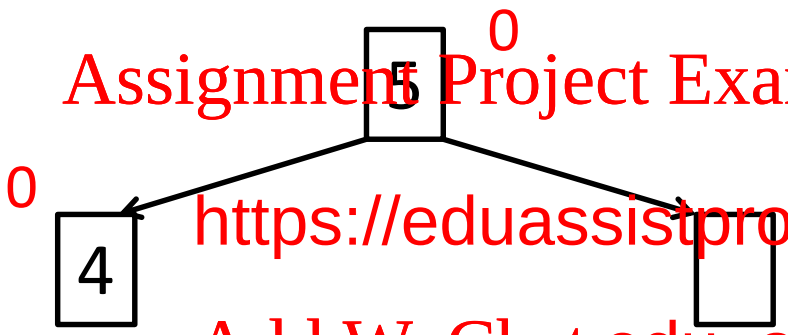
Example Insert

Create AVL-Tree for sequence 4, 5, 7, 2, 1, 3, 6



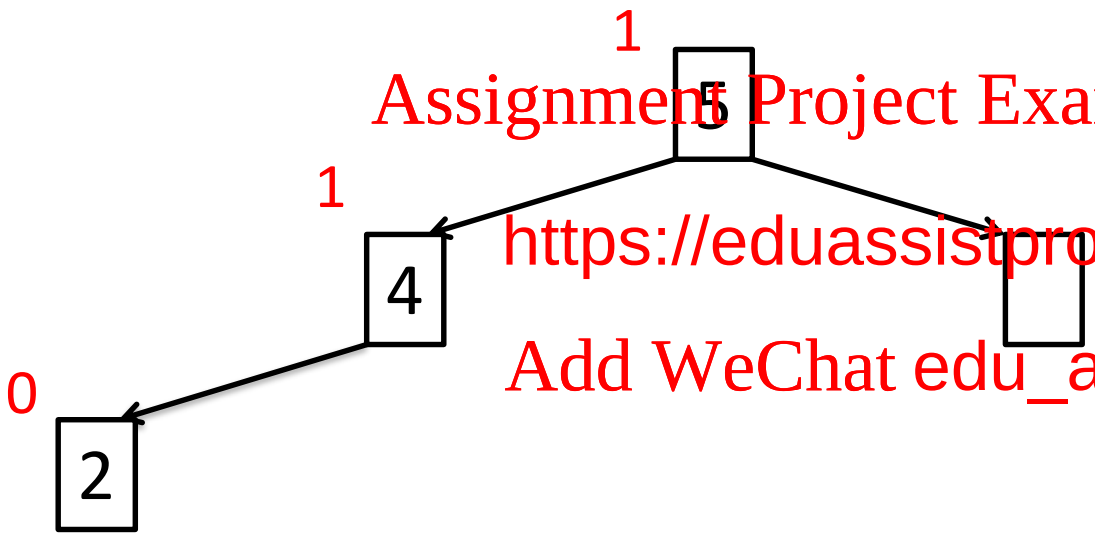
Example Insert

Create AVL-Tree for sequence 4, 5, 7, 2, 1, 3, 6



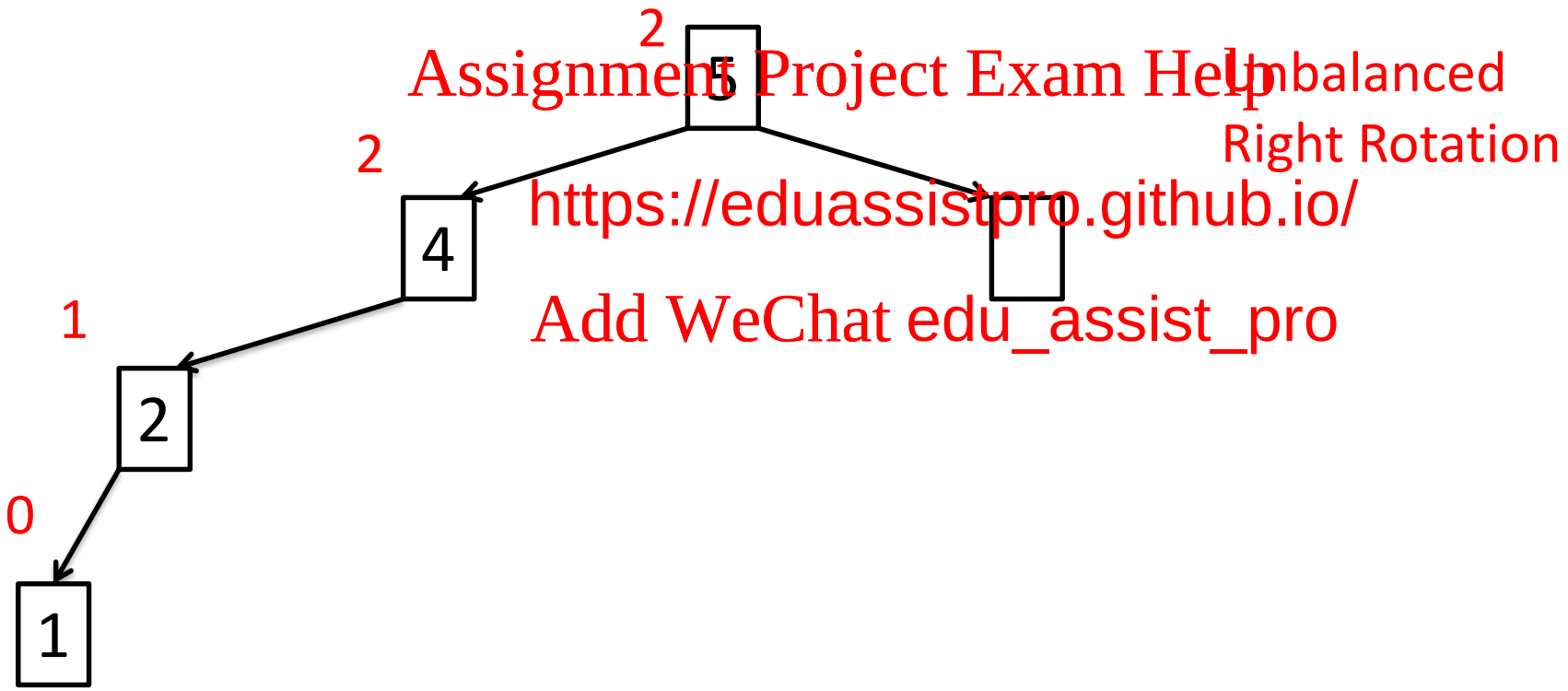
Example Insert

Create AVL-Tree for sequence 4, 5, 7, 2, 1, 3, 6



Example Insert

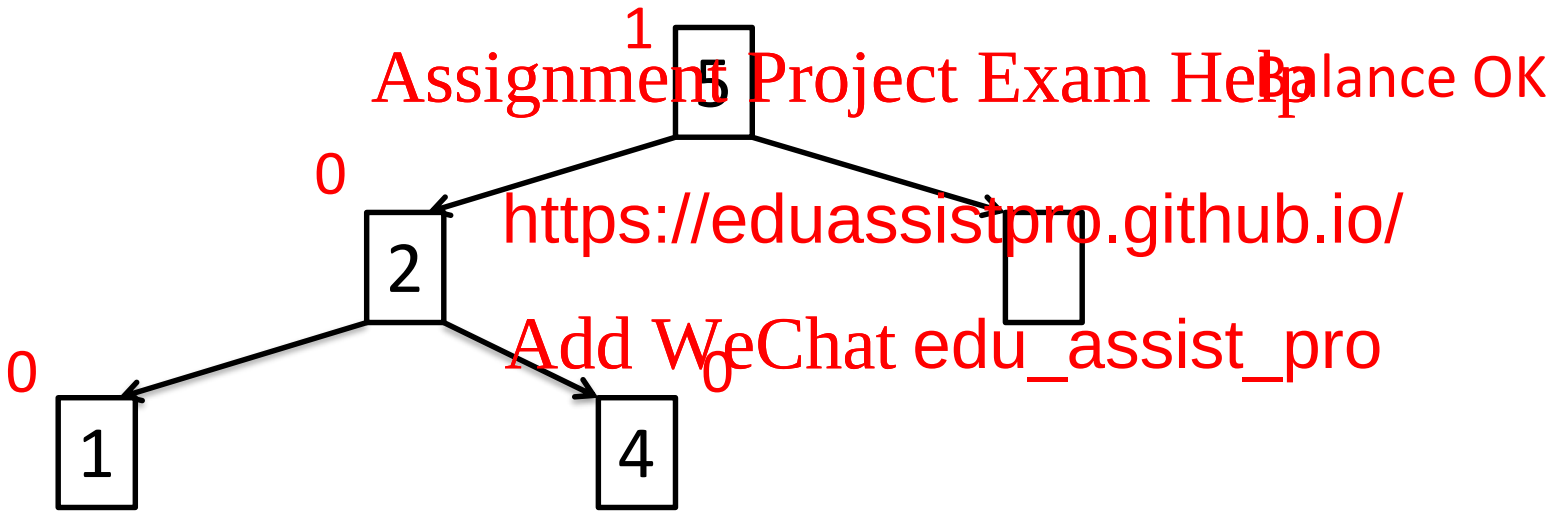
Create AVL-Tree for sequence 4, 5, 7, 2, 1, 3, 6



Unbalanced
Right Rotation

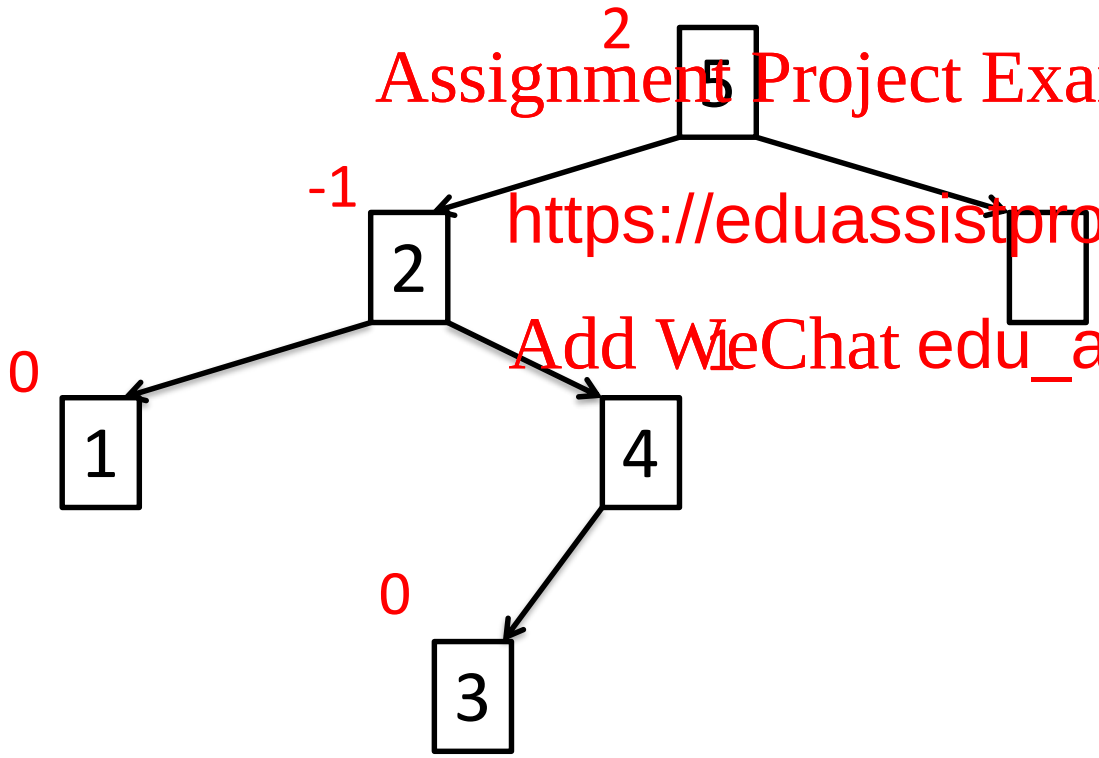
Example Insert

Create AVL-Tree for sequence 4, 5, 7, 2, 1, 3, 6



Example Insert

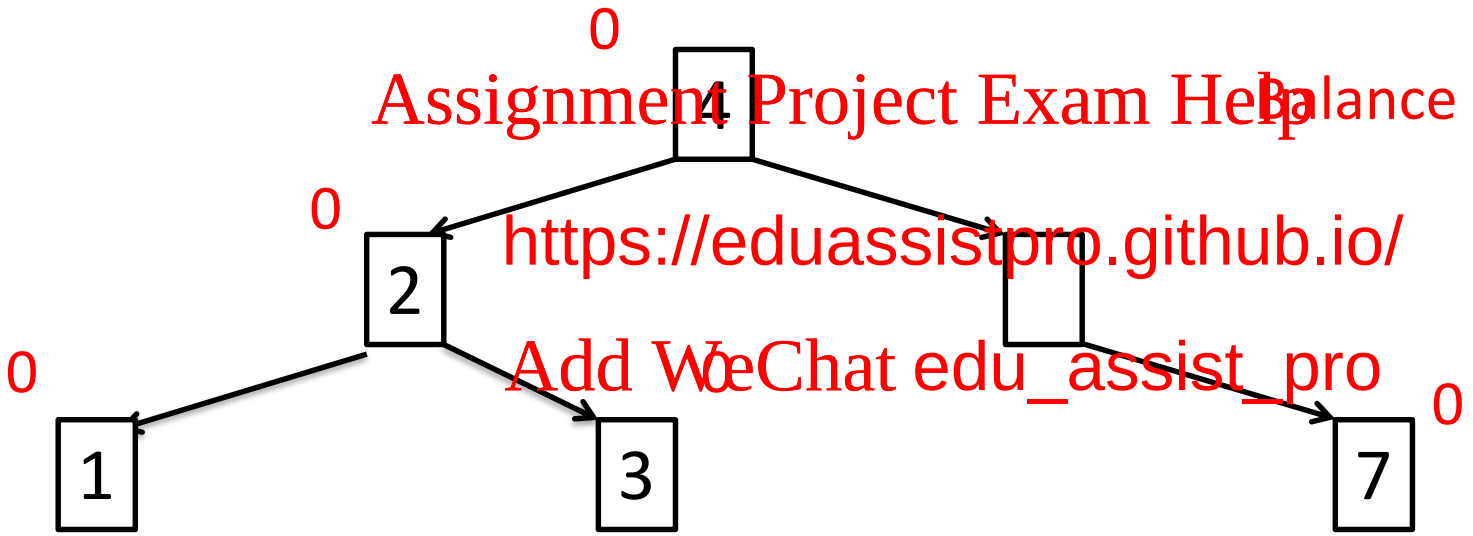
Create AVL-Tree for sequence 4, 5, 7, 2, 1, 3, 6



Unbalanced
Left-right Rotation

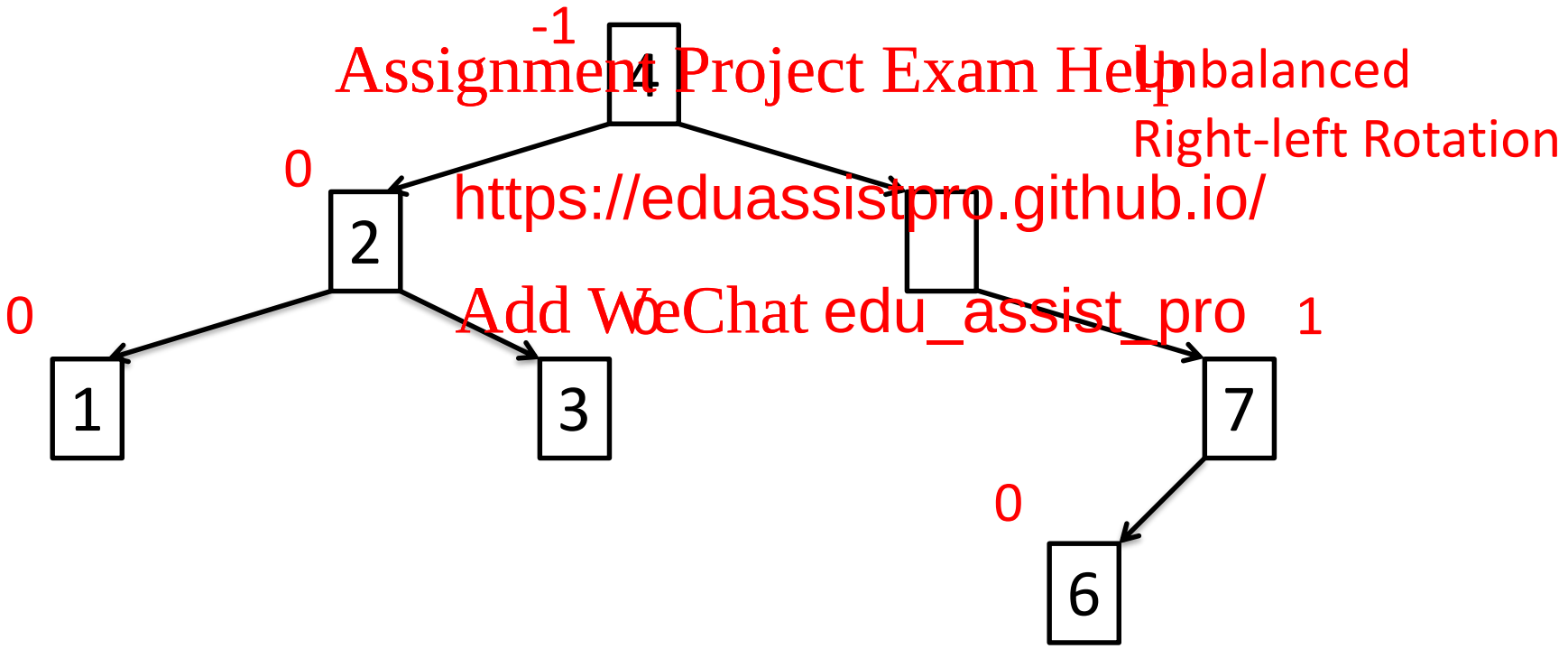
Example Insert

Create AVL-Tree for sequence 4, 5, 7, 2, 1, 3, 6



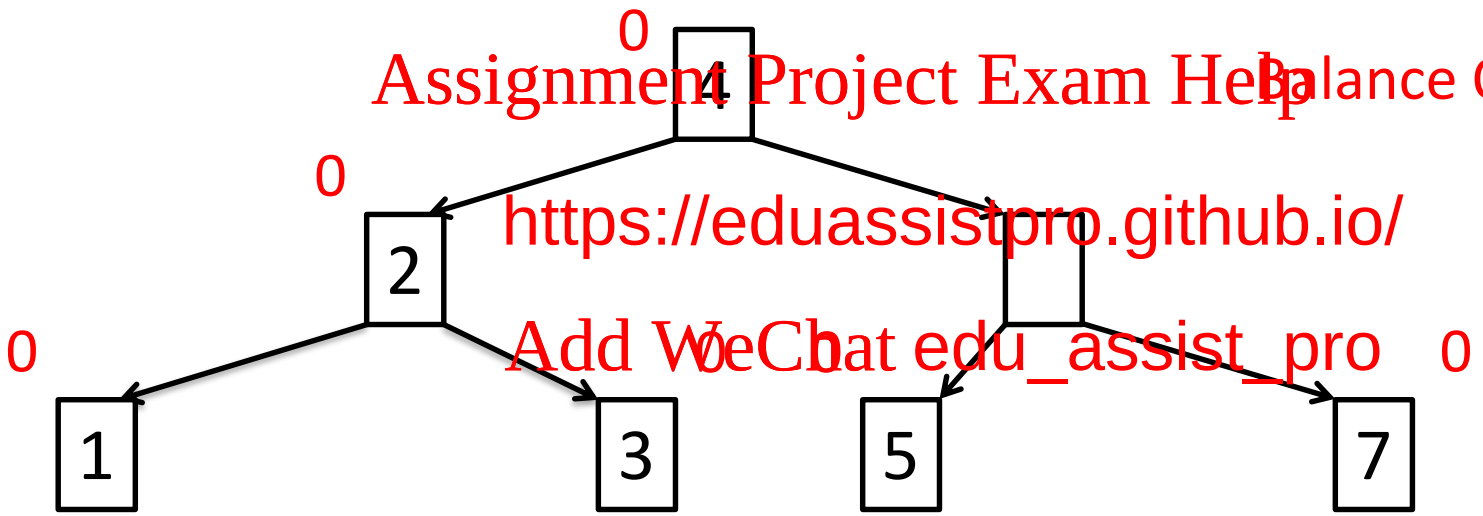
Example Insert

Create AVL-Tree for sequence 4, 5, 7, 2, 1, 3, 6



Example Insert

Create AVL-Tree for sequence 4, 5, 7, 2, 1, 3, 6



Time complexity of for insertion

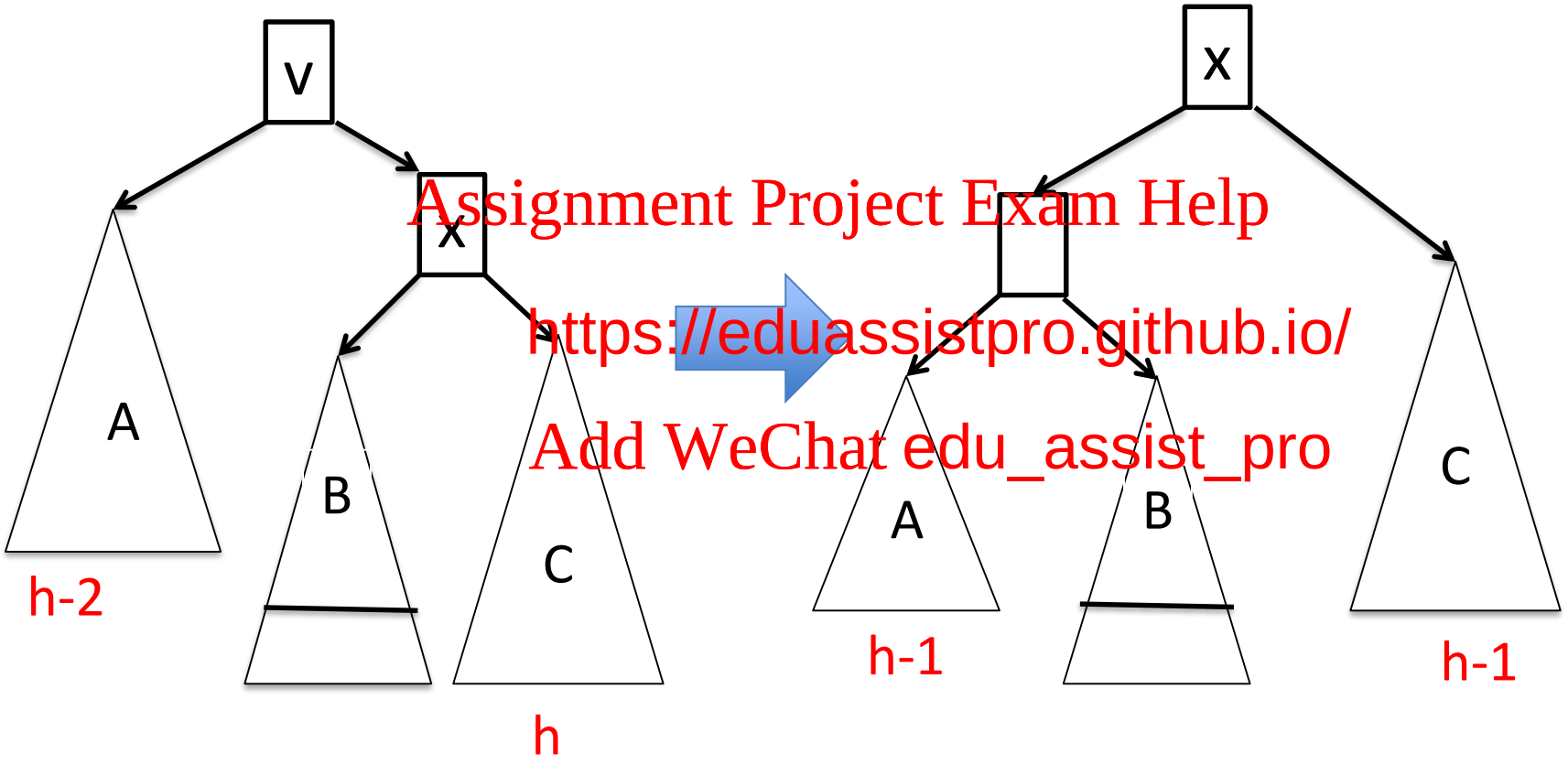
Inserting a node involves:

- Finding the location and adding the node to the tree
- Moving up t ether AVL property is violated and rotation as needed

O(log n) + O (log n) + O(1)

Remove – Left Rotation

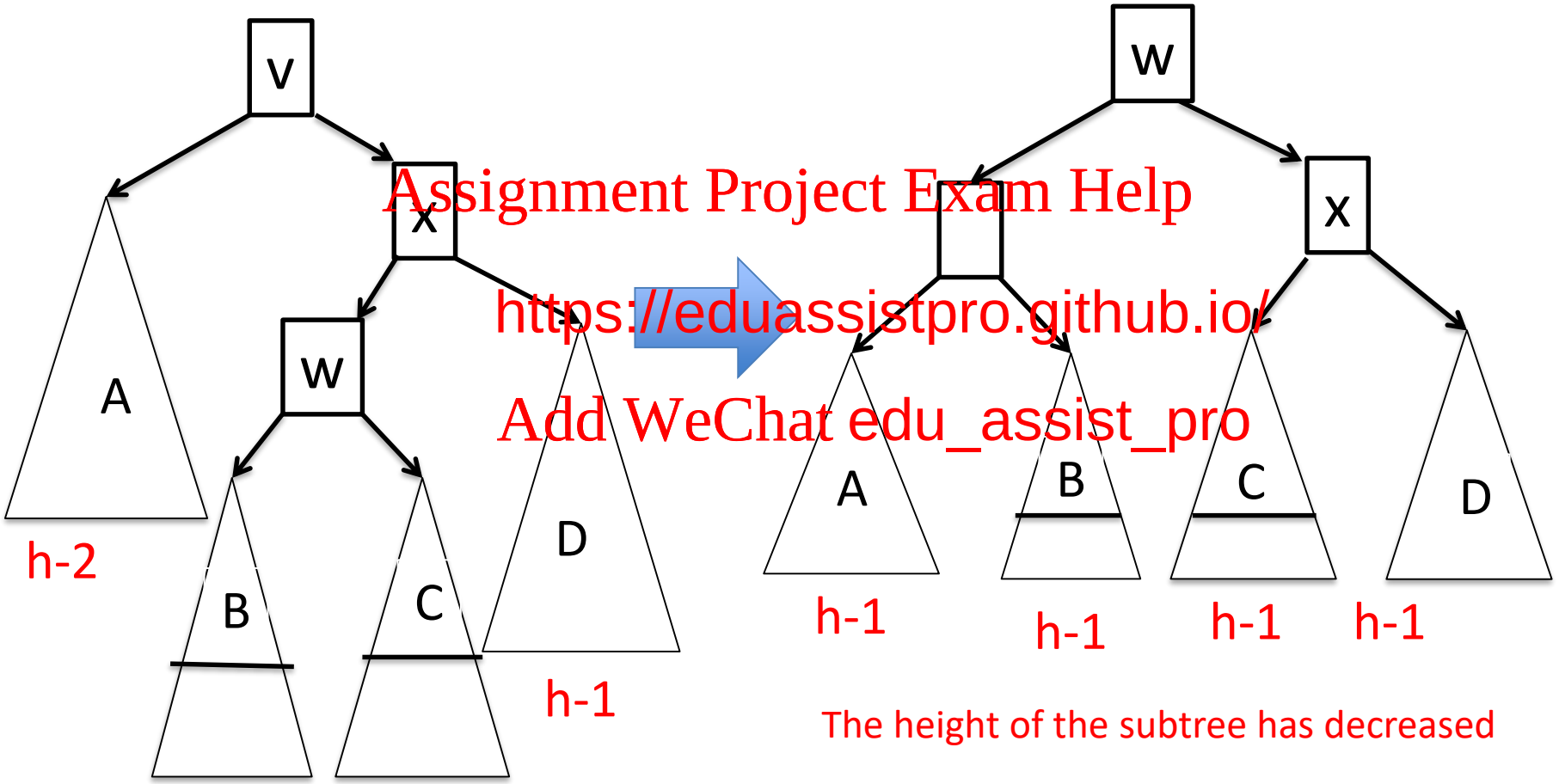
W.l.o.g. assume that the deleted node was in the left subtree of v and height of this tree has decrease by 1.



If B had height h-1 before deletion,
the height of the subtree has decreased

Analog: Right Rotation

Right-Left Rotation



The height of the subtree has decreased

Either B or C might have height h-1

Analog: Left-Right Rotation

Rebalancing after Deletion

- After having rebalanced for node v the height of the tree previously rooted at v might have decreased a balancing.
- If this is the might be imbalanced.
- We might have to continue rebalancing until the root has been reached.

Runtime AVL-trees

Theorem: The operations find, insert, and delete can be implemented for AVL-trees in worst-case time O(log n).