COL 106 Lecture 20

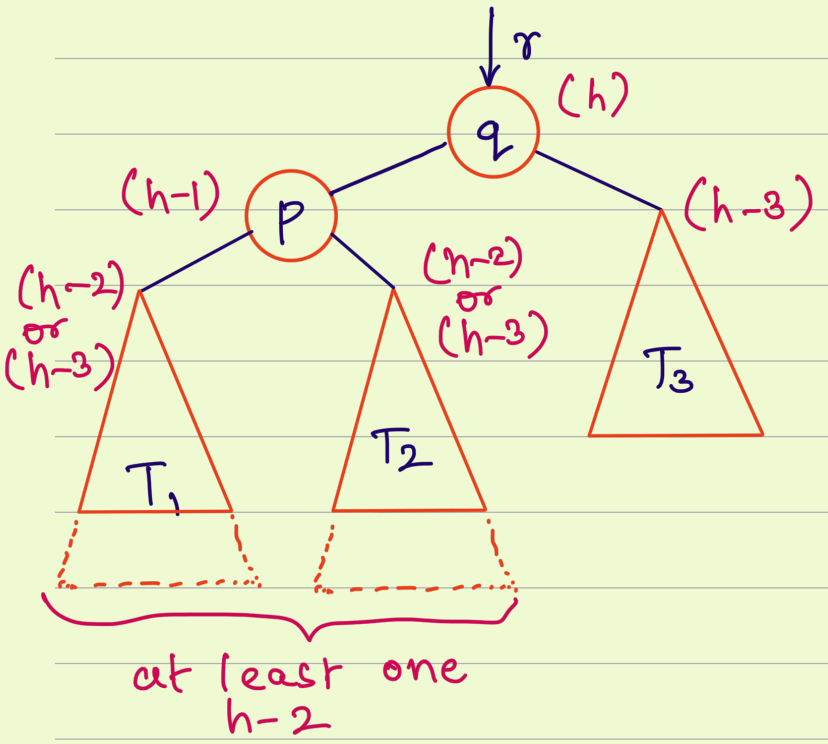
Topic : AVL Trees : Insertion

Modifier operations on AVL trees

- ☐ Add element (Insertion)
- ☐ Remove element (Deletion)

1. Insert / Delete like in a Binary Search tree
2. Fix imbalances
   ↳ How?

Recap: Rebalancing



Pre conditions:

① All nodes other than r are balanced.

② $\left|\begin{array}{l}\text{ht of r's} \\ \text{left subtree}\end{array} - \begin{array}{l}\text{ht of r's} \\ \text{right subtree}\end{array}\right| = 2$

Case 1

ht of $T_1 \geq$ ht of T_2

→ single rotation

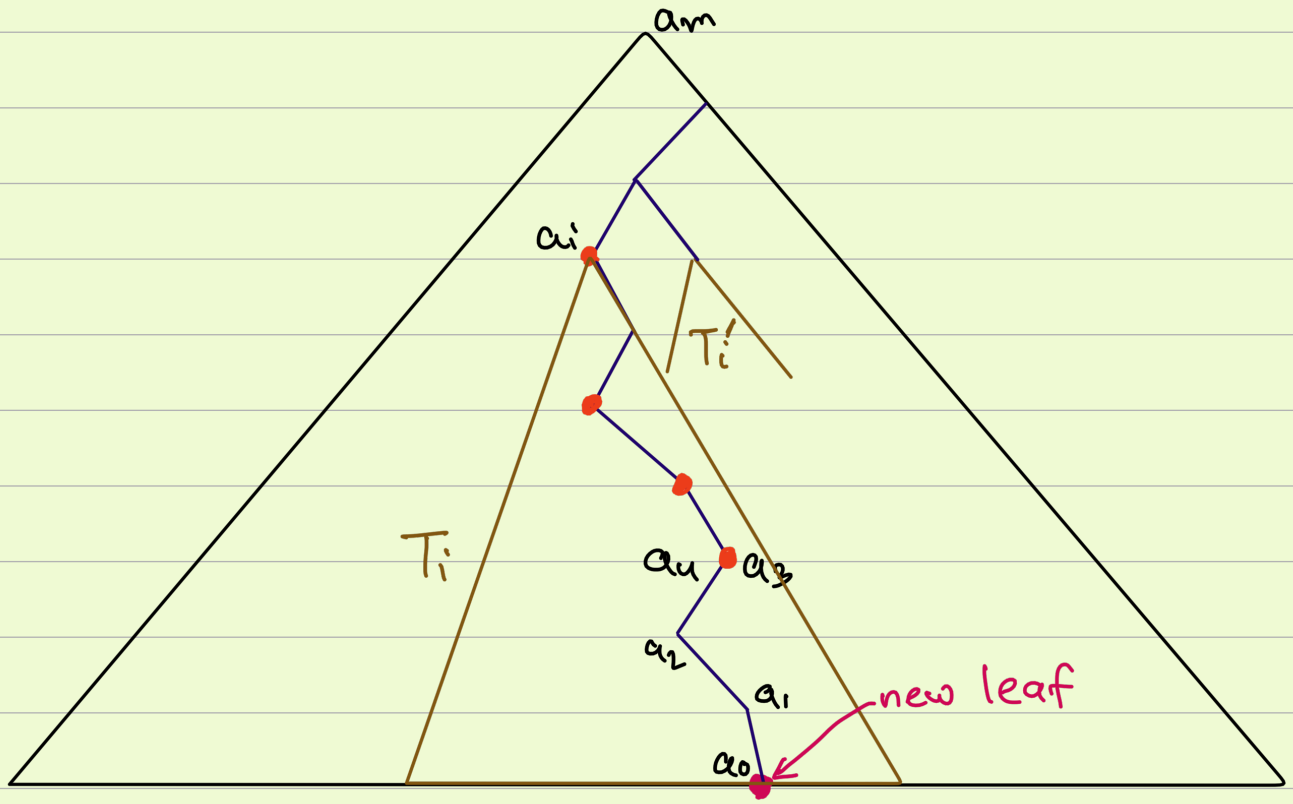
Case

ht of $T_1 <$ ht of T_2

→ double rotation

Obs (*) If ht of $T_1 \neq$ ht of T_2, then ht of tree ↓ on rebalancing

Rebalance after Insertion: the General Case



Obs: A node becomes unbalanced after insertion only if it is an ancestor of the newly created leaf.

AVL Tree Insertion.

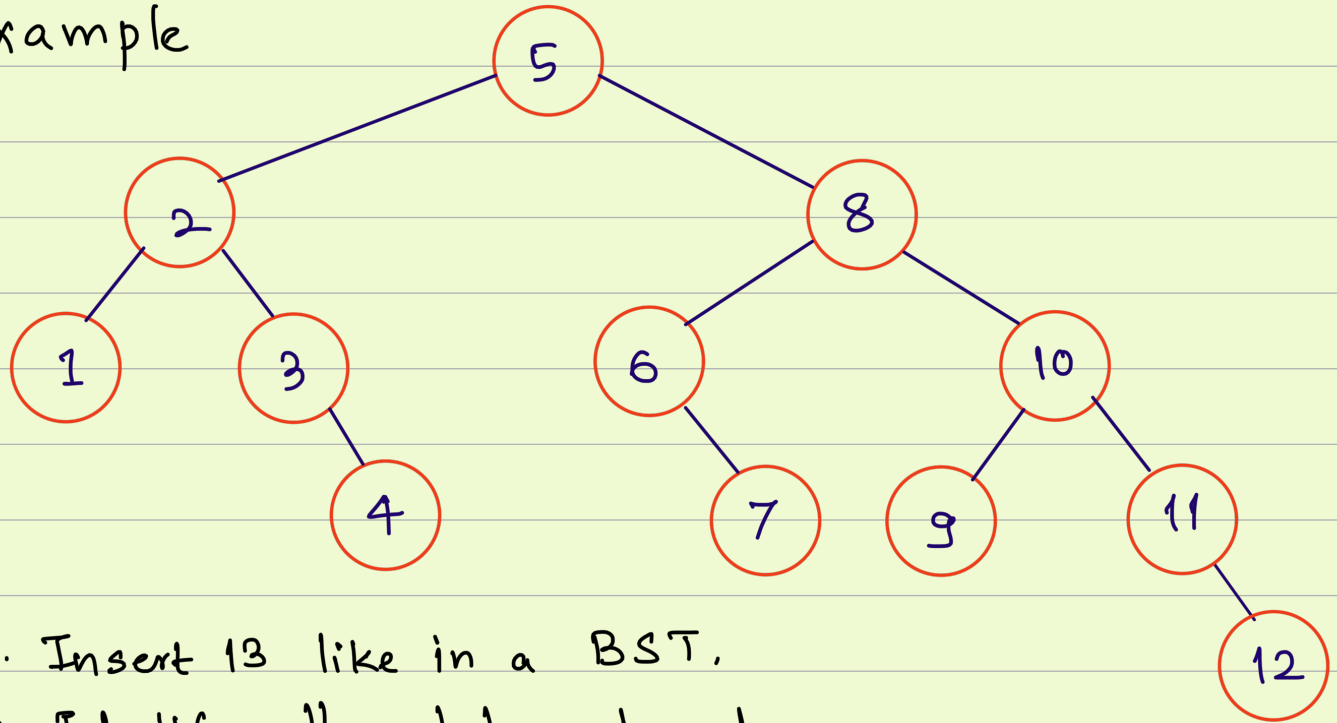
AVL Insert (x):

1. BST Insert (x) (Insert key like in a binary search tree.)
2. Rebalance the deepest unbalanced node.

That's it!

This maintains BST property because:
① Rotations maintain inorder
② Cut-and-paste property (ref. Lecture 18)

Example



1. Insert 13 like in a BST.
2. Identify all unbalanced nodes.
3. Balance the deepest unbalanced node.
4. Identify all unbalanced nodes again. What do you see?

Claim# After BST-insertion, rebalancing the deepest unbalanced node is enough to balance all unbalanced nodes.

Notation: $a_0, \ldots, a_m$: Ancestors of the new leaf
(a_0 ↑ new leaf; a_m ↑ root ; each a_i parent of a_{i-1}.

a_u: deepest unbalanced node after BST-Insert

T_i: subtree rooted at a_i

T_i': T_i's sibling.

$ht(T)$: height of T.

Sub-claim 1: $ht(T_u)$ increases in the BST-insert step.

Proof: We inserted a_0 in the taller subtree T_{u-1} of a_u, and this increased $ht(T_{u-1})$.

Sub-claim 2: In the rebalancing step, T_u is replaced by a tree whose height is 1 less than $ht(T_u)$

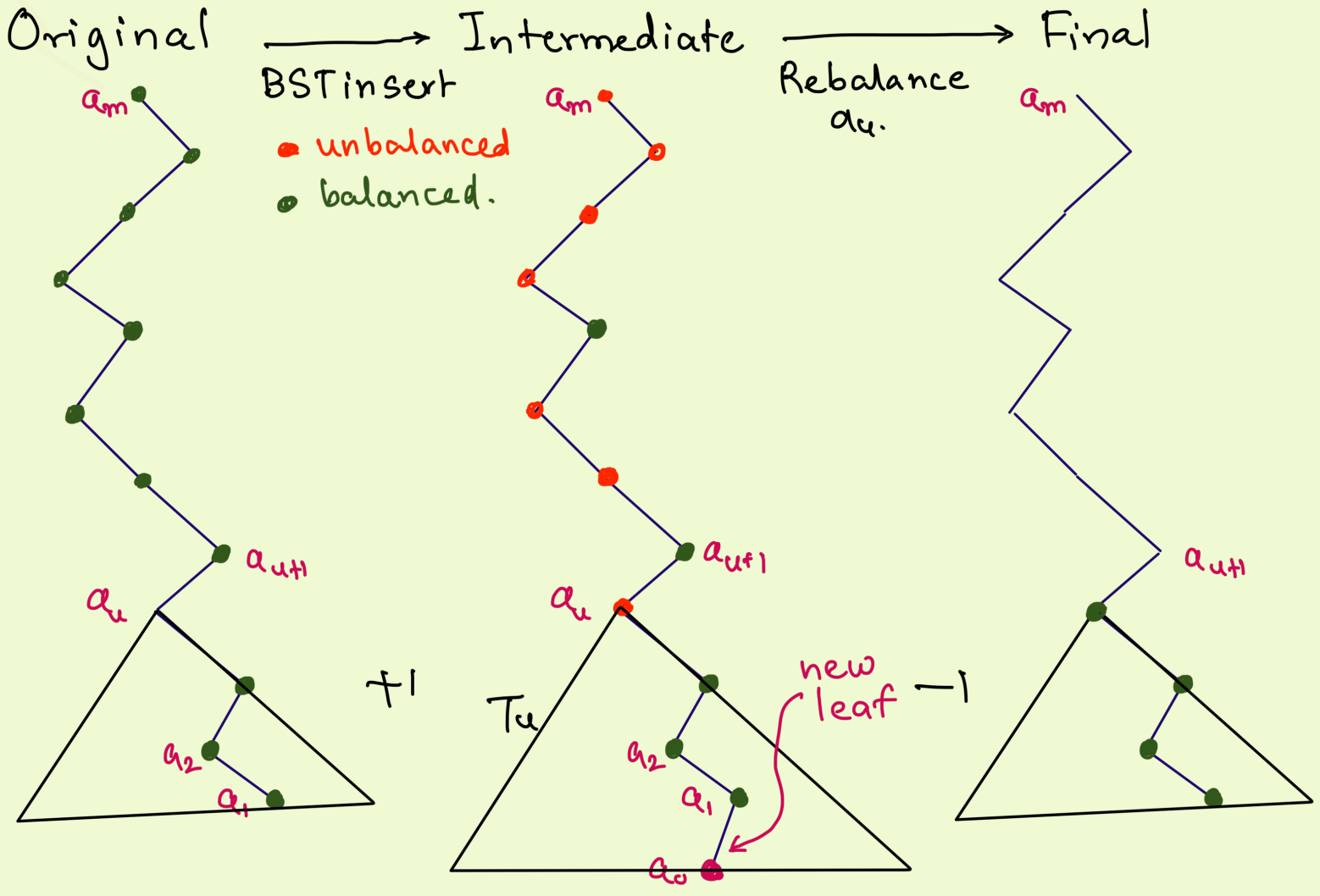
Proof: Later

We prove claim# assuming Sub-claims 1 and 2.

Inductive claim to prove claim #.

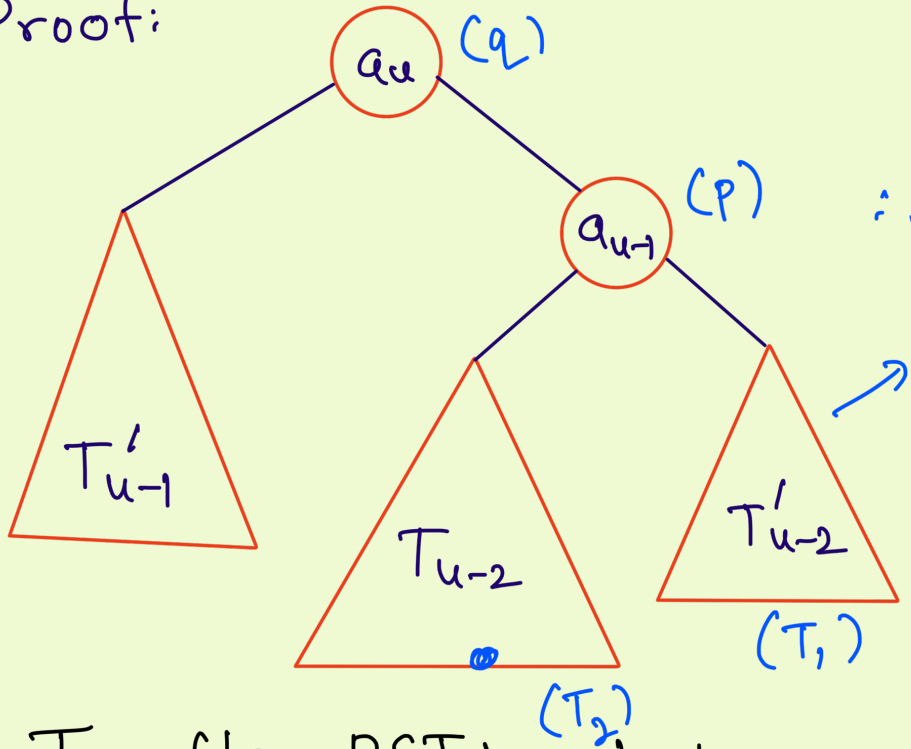
$\forall i \geq 0$ ① a_{u+i} is balanced after the rebalancing step

② $ht(T_{u+i})$ after rebalance $=$ $ht(T_{u+i})$ before BST insert



Sub-claim 2: In the rebalancing step, T_u is replaced by a tree whose height is 1 less than $ht(T_u)$

Proof:



$ht(T_{u-1})$ increased due to insertion

$\therefore ht(T_{u-2}) > ht(T'_{u-2})$ after insertion.

By Obs (*), the claim follows.

T_u after BST insert but before rebalancing.

Implementation Details:

AVLTreeNode attributes:

- key
- parent
- left
- right

} BinaryTreeNode attributes

- height → ht of the subtree rooted at this node

Yet another sorting algorithm ?

AVLsort (list)

1. Insert all items in list into an AVL tree — $O(n \log n)$
2. Perform inorder traversal. — $O(n)$