# Balanced Binary Search Trees

EECS2011 N & Z:
Fundamentals of Data Structures
Winter 2022

CHEN-WEI WANG

# Learning Outcomes of this Lecture

This module is designed to help you understand:

- When the ***Worst-Case RT*** of a ***BST Search*** Occurs
- ***Height-Balance*** Property
- Performing ***Rotations*** to Restore Tree ***Balance***

# Balanced Binary Search Trees: Motivation

- After ***insertions*** into a BST, the ***worst-case RT*** of a ***search*** occurs when the ***height*** $h$ is at its ***maximum***: ***O(n)***:
  - e.g., Entries were inserted in an decreasing order of their keys

    $$\langle 100, 75, 68, 60, 50, 1 \rangle$$

    ⇒ ***One-path***, ***left-slanted*** BST
  - e.g., Entries were inserted in an increasing order of their keys

    $$\langle 1, 50, 60, 68, 75, 100 \rangle$$

    ⇒ ***One-path***, ***right-slanted*** BST
  - e.g., Last entry's key is in-between keys of the previous two entries

    $$\langle 1, 100, 50, 75, 60, 68 \rangle$$

    ⇒ ***One-path***, ***side-alternating*** BST
- To avoid the worst-case RT (∵ a ***ill-balanced tree***), we need to take actions *as soon as* the tree becomes ***unbalanced***.

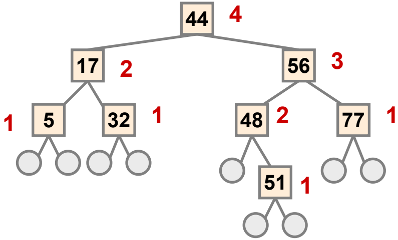
# Balanced Binary Search Trees: Definition

- Given a node *p*, the ***height*** of the subtree rooted at *p* is:

$$\textit{height}(p) = \begin{cases} 0 & \text{if } p \text{ is } \textit{external} \\ 1 + \textbf{MAX}\left(\{\ \textit{height}(c) \mid \textit{parent}\ (c) = p\ \}\right) & \text{if } p \text{ is } \textit{internal} \end{cases}$$

- A ***balanced*** BST *T* satisfies the ***height-balance property***:

  For every ***internal node*** *n*, ***heights*** of *n*'s child nodes differ $\leq$ ***1***.



**Q:** Is the above tree a ***balanced BST***? ✓

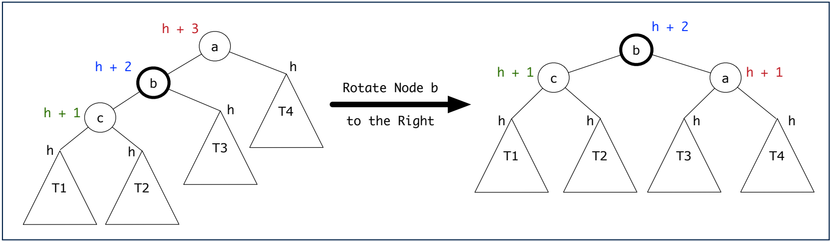
**Q:** Will the tree remain ***balanced*** after inserting 55? ×

**Q:** Will the tree remain ***balanced*** after inserting 63? ✓

# Fixing Unbalanced BST: Rotations

A tree ***rotation*** is performed:

- When the latest **insertion**/**deletion** creates ***unbalanced*** nodes, along the ***ancestor path*** of the node being inserted/deleted.
- To change the ***shape*** of tree, ***restoring*** the ***height-balance property***



**Q**. An ***in-order traversal*** on the resulting tree?

**A**. Still produces a sequence of ***sorted keys*** $\langle T_1,\ c,\ T_2,\ b,\ T_3,\ a,\ T_4\rangle$

- After ***rotating*** node *b* to the right:
  - Heights of ***descendants*** ($b$, $c$, $T_1$, $T_2$, $T_3$) and ***sibling*** ($T_4$) stay ***unchanged***.
  - Height of ***parent*** ($a$) is ***decreased by 1***.

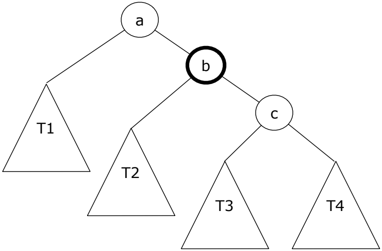
    ⇒ ***Balance*** of node *a* was ***restored*** by the ***rotation***.

# After Insertions:<br>Trinode Restructuring via Rotation(s)

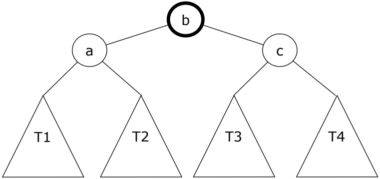
After ***inserting*** a new node *n*:

- **Case 1**: Nodes on *n*'s ***ancestor path*** remain ***balanced***.
  ⇒ No rotations needed
- **Case 2**: At least one of *n*'s ***ancestors*** becomes ***unbalanced***.
  1. Get the **first**/**lowest** ***unbalanced*** node *a* on *n*'s ***ancestor path***.
  2. Get *a*'s child node *b* in *n*'s ***ancestor path***.
  3. Get *b*'s child node *c* in *n*'s ***ancestor path***.
  4. Perform rotation(s) based on the ***alignment*** of *a*, *b*, and *c*:
     - Slanted the ***same*** way ⇒ ***single** rotation* on the **middle** node *b*
     - Slanted ***different*** ways ⇒ ***double** rotations* on the **lower** node *c*

# Trinode Restructuring: Single, Left Rotation



After a ***left rotation*** on the middle node *b*:



***BST property*** maintained? $\langle T_1, a, T_2, b, T_3, c, T_4 \rangle$

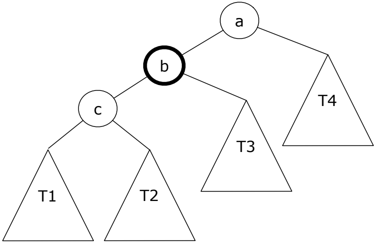
# Left Rotation

- ***Insert*** the following sequence of nodes into an empty BST:

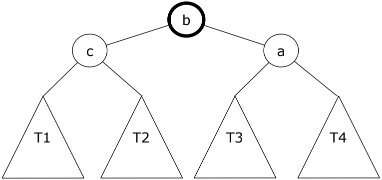
  $$\langle 44, 17, 78, 32, 50, 88, 95 \rangle$$

- Is the BST now ***balanced***?
- ***Insert*** 100 into the BST.
- Is the BST still ***balanced***?
- Perform a ***left rotation*** on the appropriate node.
- Is the BST again ***balanced***?

# Trinode Restructuring: Single, Right Rotation



After a ***right rotation*** on the middle node *b*:



***BST property*** maintained? $\langle T_1, a, T_2, b, T_3, c, T_4 \rangle$

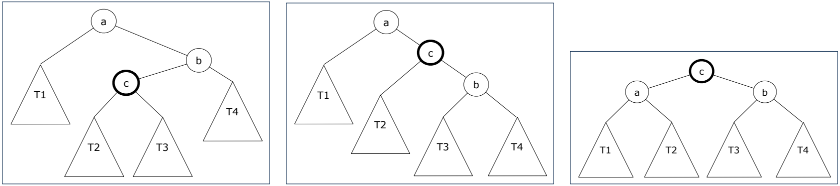
# Right Rotation

- ***Insert*** the following sequence of nodes into an empty BST:

$$\langle 44, 17, 78, 32, 50, 88, 48 \rangle$$

- Is the BST now ***balanced***?
- ***Insert*** $\underline{46}$ into the BST.
- Is the BST still ***balanced***?
- Perform a ***right rotation*** on the appropriate node.
- Is the BST again ***balanced***?

# Trinode Restructuring: Double, R-L Rotations



Perform a ***Right Rotation*** on Node *c* | Perform a ***Left Rotation*** on Node *c* | After Right-Left Rotations

***BST property*** maintained? $\langle T_1, a, T_2, c, T_3, b, T_4 \rangle$

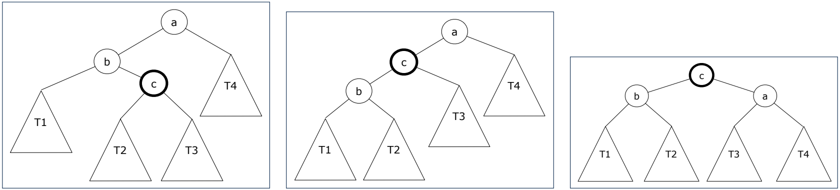
# R-L Rotations

- ***Insert*** the following sequence of nodes into an empty BST:

  $$\langle 44, 17, 78, 32, 50, 88, 82, 95 \rangle$$

- Is the BST now ***balanced***?
- ***Insert*** 85 into the BST.
- Is the BST still ***balanced***?
- Perform the ***R-L rotations*** on the appropriate node.
- Is the BST again ***balanced***?

# Trinode Restructuring: Double, L-R Rotations



Perform a ***Left Rotation*** on Node *c*

Perform a ***Right Rotation*** on Node *c*

After Left-Right Rotations

***BST property*** maintained?

$\langle T_1,\ b,\ T_2,\ c,\ T_3,\ a,\ T_4 \rangle$

# L-R Rotations

- ***Insert*** the following sequence of nodes into an empty BST:

$$\langle 44, 17, 78, 32, 50, 88, 48, 62 \rangle$$

- Is the BST now ***balanced***?
- ***Insert*** $\underline{54}$ into the BST.
- Is the BST still ***balanced***?
- Perform the ***L-R rotations*** on the appropriate node.
- Is the BST again ***balanced***?

# After Deletions: Continuous Trinode Restructuring

- Recall: ***Deletion*** from a BST results in removing a node with zero or one ***internal*** child node.
- After ***deleting*** an existing node, say its child is $n$:

  **Case 1**: Nodes on $n$'s ***ancestor path*** remain ***balanced***. ⇒ No rotations

  **Case 2**: At least one of $n$'s ***ancestors*** becomes ***unbalanced***.

  1. Get the **first**/**lowest** ***unbalanced*** node $a$ on $n$'s ***ancestor path***.
  2. Get $a$'s ***taller*** child node $b$. [ $b \notin$ $n$'s ***ancestor path*** ]
  3. Choose $b$'s child node $c$ as follows:
     - $b$'s two child nodes have ***different*** heights ⇒ $c$ is the ***taller*** child
     - $b$'s two child nodes have ***same*** height ⇒ $a$, $b$, $c$ slant the ***same*** way
  4. Perform rotation(s) based on the ***alignment*** of $a$, $b$, and $c$:
     - Slanted the ***same*** way ⇒ ***single** rotation* on the **middle** node $b$
     - Slanted ***different*** ways ⇒ ***double** rotations* on the **lower** node $c$
- As $n$'s ***unbalanced ancestors*** are found, keep applying **Case 2**, until **Case 1** is satisfied. [ $O(h) = O(\log n)$ ***rotations***]

# Single Trinode Restructuring Step

- ***Insert*** the following sequence of nodes into an empty BST:

$$\langle 44, 17, 62, 32, 50, 78, 48, 54, 88 \rangle$$

- Is the BST now ***balanced***?
- ***Delete*** 32 from the BST.
- Is the BST still ***balanced***?
- Perform a ***left rotation*** on the appropriate node.
- Is the BST again ***balanced***?

# Multiple Trinode Restructuring Steps

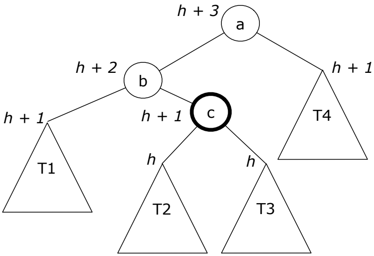
- ***Insert*** the following sequence of nodes into an empty BST:

$$\langle 50, 25, 10, 30, 5, 15, 27, 1, 75, 60, 80, 55 \rangle$$

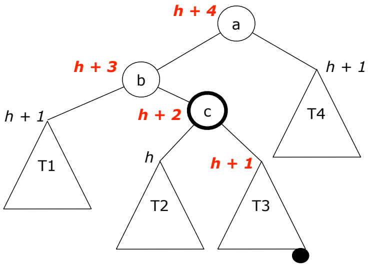
- Is the BST now ***balanced***?
- ***Delete*** 80 from the BST.
- Is the BST still ***balanced***?
- Perform a ***right rotation*** on the appropriate node.
- Is the BST now ***balanced***?
- Perform another ***right rotation*** on the appropriate node.
- Is the BST again ***balanced***?

# Restoring Balance from Insertions

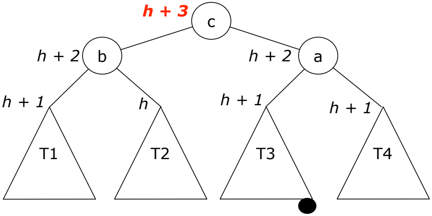
*Before Insertion into T3*



*After Insertion into T3*

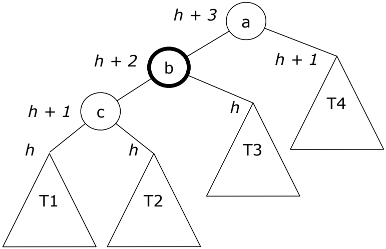


*After Performing L-R Rotations on Node c: Height of Subtree Being Fixed Remains* ***h + 3***

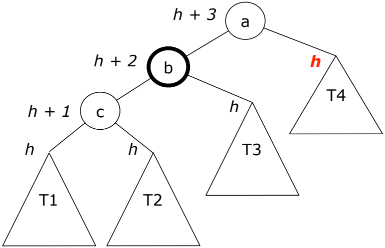


# Restoring Balance from Deletions

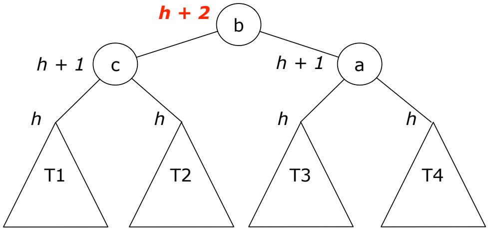
*Before Deletion from T4*



*After Deletion from T4*



*After Performing Right Rotation on Node b: Height of Subtree Being Fixed* ***Reduces its Height by 1****!*



# Restoring Balance: Insertions vs. Deletions

- Each ***rotation*** involves only ***POs*** of setting parent-child references.
  - ⇒ ***O(1)*** running time for each tree ***rotation***
- After each ***insertion***, a ***trinode restructuring*** step can ***restore the balance*** of the subtree rooted at the first ***unbalanced*** node.
  - ⇒ ***O(1)*** rotations suffices to restore the balance of tree
- After each ***deletion***, one or more ***trinode restructuring*** steps may ***restore the balance*** of the subtree rooted at the first ***unbalanced*** node.
  - ⇒ May take ***O(log n)*** rotations to restore the balance of tree

# Index (1)

# Index (2)