Lecture 7 - Sep 24

Self-Balancing Binary Search Trees

After Insertion: Left Rotation
After Insertion: Right-Left Rotations
BST Deletion: Cases 1 — 3

Announcements/Reminders

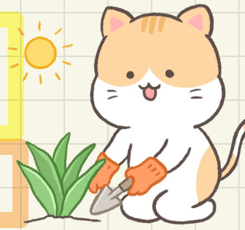
- First Class (Syllabus) recording & notes posted
- Today's class: notes template posted
- Exercises:
  - Tutorial Week 1 (2D arrays)
  - Tutorial Week 2 (2D arrays, Proving Big-O)
  - Tutorial Week 3 (avg case analysis on doubling strategy)

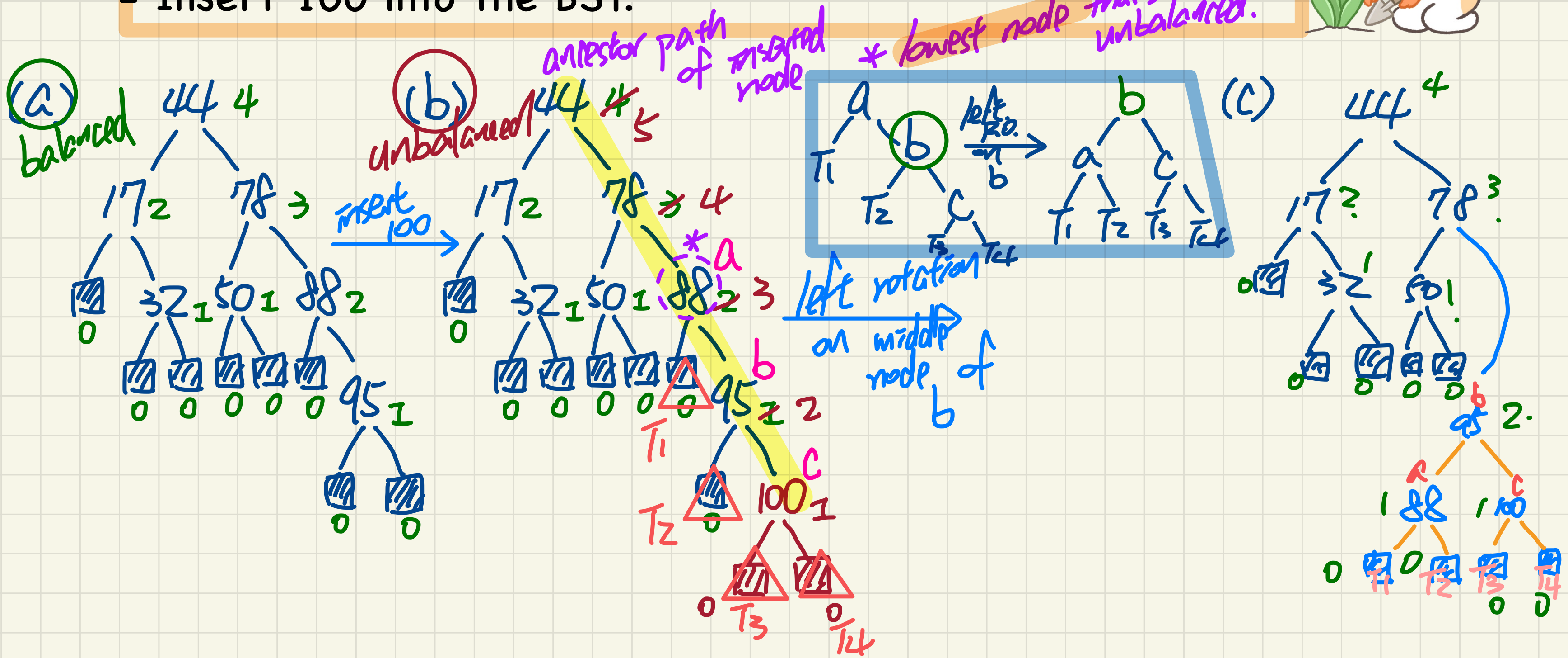
Trinode Restructuring after Insertion: Left Rotation

- Insert the following sequence of keys into an empty BST:

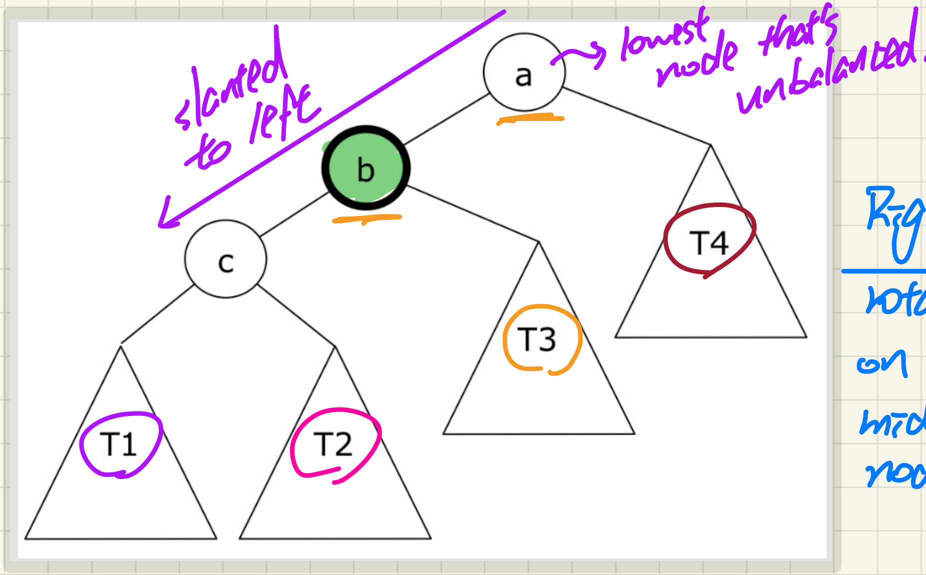
  <44, 17, 78, 32, 50, 88, 95>

- Insert 100 into the BST.

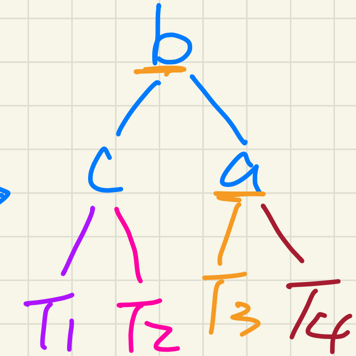




Trinode Restructuring: Single, Right Rotation



Right rotation on the middle node b



I.O.T. : $T_1, c, T_2, b, T_3, a, T_4$

Trinode Restructuring after Insertion: Right Rotation

- Insert the following sequence of keys into an empty BST:

  <44, 17, 78, 32, 50, 88, 48>

- Insert 46 into the BST.

Trinode restructuring step   a, b, c

middle node

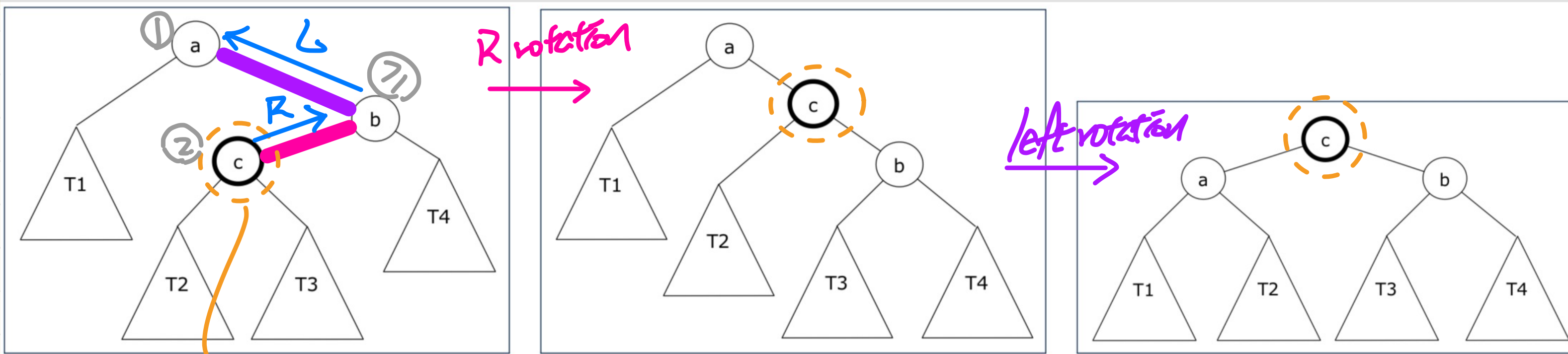
↳ one rotation (L, R)

↳ a, b, c slanted the same way.

↳ two rotations (R-L, L-R)

↳ a, b, c slanted differently.

Trinode Restructuring: Double, Right-Left Rotations



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

to be promoted up 2 levels.

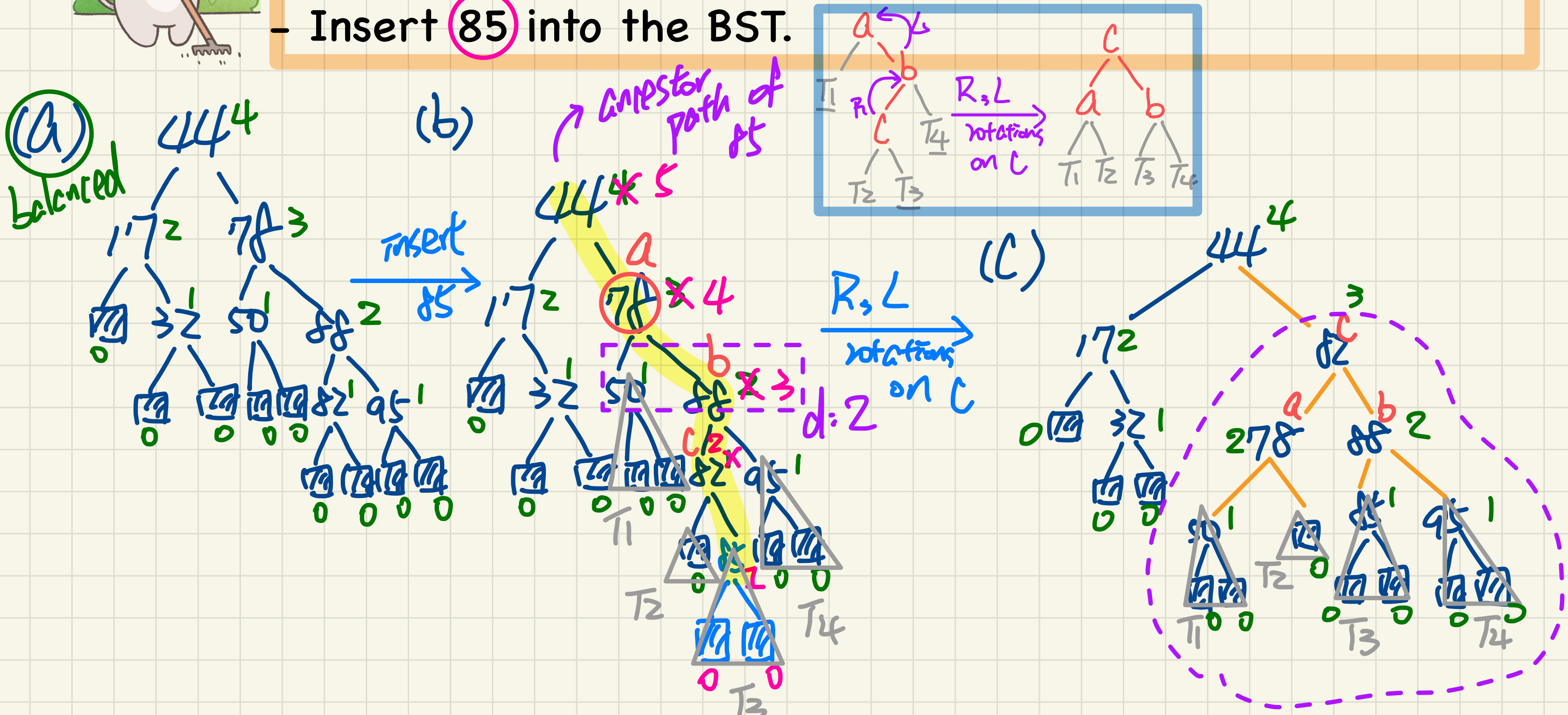
I.O.T. : T_1 a T_2 c T_3 b T_4

Trinode Restructuring after Insertion: R-L Rotations

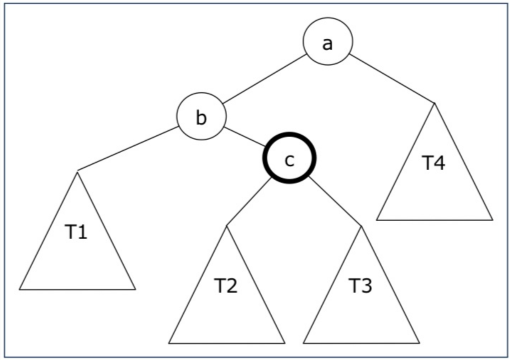
- Insert the following sequence of **keys** into an empty BST:

  <44, 17, 78, 32, 50, 88, 82, 95>

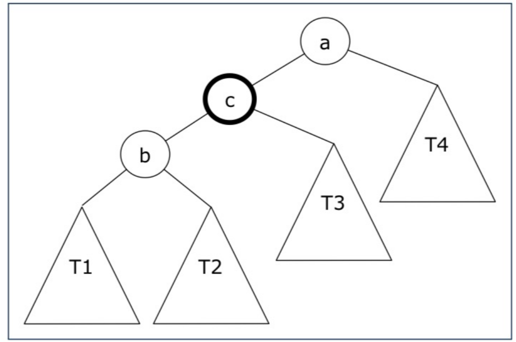
- Insert 85 into the BST.



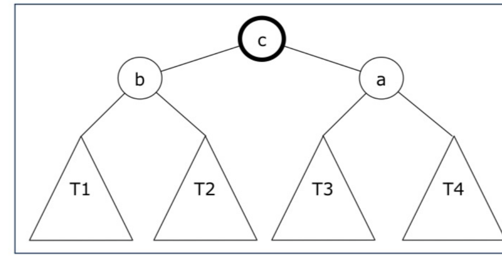
Trinode Restructuring: Double, Left-Right Rotations



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



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



After Left-Right Rotations

Trinode Restructuring after Insertion: L-R Rotations

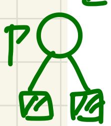
- Insert the following sequence of keys into an empty BST:
  <44, 17, 78, 32, 50, 88, 48, 62>
- Insert 54 into the BST.

BST Operation: Cases of Deletion

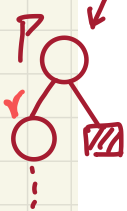
To *delete* an ***entry*** (with **key** k) from a BST rooted at ***node*** n:

Let node $\boldsymbol{p}$ be the return value from $\boxed{\texttt{search}}$(n (root), k (key of entry to be deleted.)) .

- **Case 1**: Node $\boldsymbol{p}$ is ***external***.
  k is not an existing key ⇒ Nothing to remove
- **Case 2**: Both of node $\boldsymbol{p}$'s child nodes (internal) are $\boxed{\textbf{\textit{external}}}$.
  No "orphan" subtrees to be handled ⇒ Remove $\boldsymbol{p}$ [Still BST?]



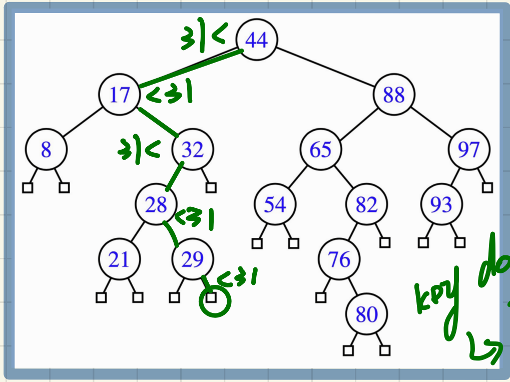
- **Case 3**: One of the node $\boldsymbol{p}$'s children, say r, is ***internal***.
  - r's sibling is ***external*** ⇒ Replace node $\boldsymbol{p}$ by node r [Still BST?]



- **Case 4**: Both of node $\boldsymbol{p}$'s children are ***internal***.
  - Let r be the ***right-most internal node p***'s ***LST***.
    ⇒ r contains the ***largest key s.t. key(r)*** < ***key(p)***.
    Exercise: Can r contain the ***smallest key s.t. key(r)*** > ***key(p)***?
  - Overwrite node $\boldsymbol{p}$'s entry by node r's entry. [Still BST?]
  - r being the ***right-most internal node*** may have:
    - ◇ Two ***external child nodes*** ⇒ Remove r as in **Case 2**.
    - ◇ An ***external, RC*** & an ***internal LC*** ⇒ Remove r as in **Case 3**.

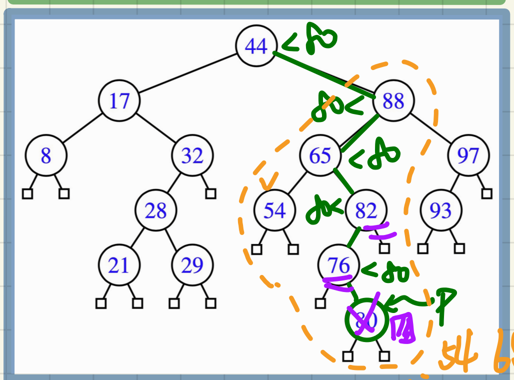
Visualizing BST Operation: Deletion

Case 1: Delete Entry with Key 31



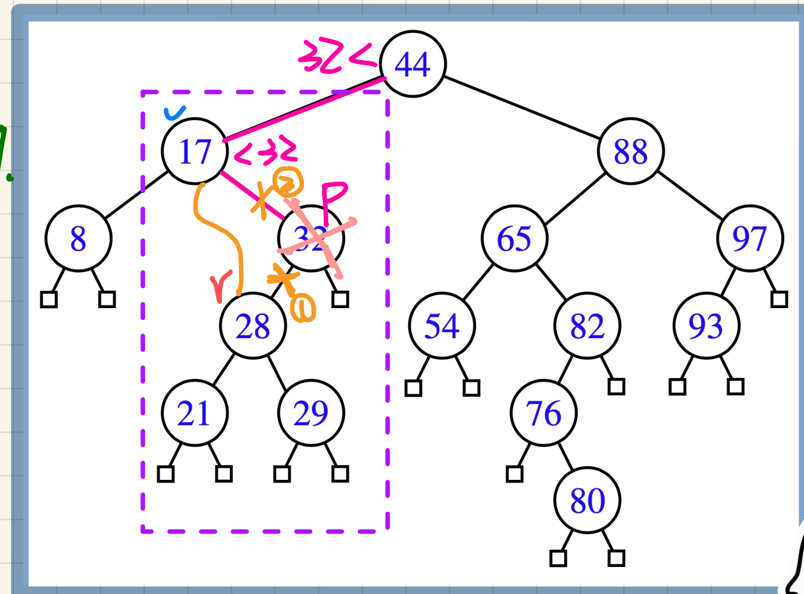
key doesn't exist
↳ do nothing.

Case 2: Delete Entry with Key 80



54 65 76 ~~80~~ 82 88 ...

Case 3: Delete Entry with Key 32



I.O.T. 17, 21, 28, 29, ~~32~~