EECS3101 Algorithms (Summer'26)

Digest of Tutorial 4, June 1

Trinode Restructuring Steps After Insertions and Deletions

Jackie Wang

How to use this Digest

This digest **is** meant to help you identify the main ideas, important examples, common pitfalls, and practice targets from the lecture. It is **not** a replacement for attending class, watching the lecture recording, reviewing the slides and iPad notes, or doing your own practice and reflection. The digest intentionally summarizes only the roadmap and highlights; **you are expected to fill in the details** by revisiting the lecture materials, tracing examples, writing code, solving problems, and checking your own understanding. A good way to use this digest is as a checklist: after reviewing the lecture, you should be able to explain and apply the items listed under "*What you should be able to do*".

This digest is an AI-assisted study roadmap reviewed by Jackie. It was produced from the lecture transcript, a carefully designed tool workflow, and Jackie's actual hand-written iPad notes. Although the digest is reviewed before being posted, it **might** still contain typos, inconsistencies, omissions, or errors, just like any manually prepared course material. Therefore, use it as a study guide, **not** as an unquestionable authority. If something seems unclear or inconsistent, you are responsible for asking for clarification and verifying your understanding. Possible errors in the digest should **not** be treated as permission to submit incorrect work, or as a substitute for understanding the course materials.

Table of Contents

Main Ideas

- Tutorial 4 focused on tree rotations as the practical mechanism for restoring the height-balance property in self-balancing BSTs.
- Jackie connected the paper rotation exercises directly to Assignment 1, which is already released, due Wednesday, June 17, and intended as strong preparation for Programming Test 1.
- The tutorial reviewed how to compute node heights bottom-up and use height differences to decide whether a subtree violates the height-balance property.
- The abstract A, B, C trinode restructuring pattern was used to explain why rotations preserve the BST sorting property through the same in-order traversal.
- A left-left imbalance was repaired by a single right rotation on the middle node B; the same-slant versus different-slant distinction determines single versus double rotations.
- A right-left insertion example showed how to identify the lowest unbalanced node A, choose B and C along the insertion ancestor path, and promote the lower node C through a double rotation.
- Deletion rebalancing was treated as more subtle than insertion rebalancing: after deletion, B and C are chosen using subtree heights, and multiple restructuring steps may be required.
- Jackie emphasized implementation discipline for Assignment 1: do not modify the provided `BSTNode` class, use the designated model package, and treat the JUnit tests as the specification.
- Programming Test 1 will have a similar format and submission workflow to Assignment 1, and Jackie stated that it will not cover graphs.

Roadmap

Assignment 1 and Programming Test 1 logistics

Jackie began by showing Assignment 1, which had been released earlier than scheduled to give students more preparation time. The assignment is due on Wednesday, June 17, is purely programming-based, and asks students to implement core self-balancing BST functionality such as insertion, height calculation, ancestor paths, and rotations. Jackie emphasized that the assignment format is similar to Programming Test 1, that Programming Test 1 will not cover graphs, and that students should not wait for solutions before making a serious attempt.

The starter project should be imported into Eclipse as an existing project from an archive file. The provided `BSTNode` class will also be given in the programming test and should not be modified. The empty `model` package is where students should put their implementation

work. The starter JUnit test class is the complete specification for the assignment, and the tests are organized to support gradual development from easier pieces to more advanced ones.

Tutorial 4 plan: rotations on paper

The tutorial then moved to paper exercises on rotations. The notes list three major pieces: abstracting the trinode restructuring step, doing a right-left restructuring after insertion, and understanding restructuring after deletion. Jackie explained that some exercises would be completed together and that other rotation exercises would be left for students to attempt with posted solutions or walkthroughs later.

Restoring balance in the abstract left-left case

The first technical exercise used an abstract tree with nodes A, B, and C, and subtrees T_1, T_2, T_3, T_4 all of height h. In the starting shape, A has left child B, and B has left child C. Students computed heights bottom-up: C has height $h+1$, B has height $h+2$, and A has height $h+3$. Since the two child heights at A differ by 2, node A violates the height-balance property.

This setup also illustrates why the repair is a single right rotation on the middle node B. Before rotation, the in-order traversal has the order

$$T_1, C, T_2, B, T_3, A, T_4.$$

After the right rotation, B becomes the local root, C becomes its left child, A becomes its right child, and the same in-order sequence is preserved. Jackie used this to show both parts of the repair: the sorting property remains valid and the height-balance property is restored.

Single versus double trinode restructuring

Jackie summarized the general trinode restructuring distinction. If A, B, and C slant the same way, such as left-left or right-right, the repair is a single rotation on the middle node B. If they slant differently, such as left-right or right-left, the repair is a double rotation on the lower node C.

For paper exercises, Jackie recommended thinking of a double rotation as a single reconstruction guided by the in-order traversal, rather than drawing every intermediate step. For implementation, however, students may either call two single-rotation helper methods in sequence or implement a direct restructuring method, provided the references and resulting tree are correct.

Right-left rotation after inserting 85

The main insertion example began from the sequence of keys

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

inserted into an empty BST. After drawing the balanced tree and annotating heights, students inserted key 85 as the right child of 82. Jackie highlighted the ancestor path of the inserted node and checked balance bottom-up. Node 78 is the lowest unbalanced node, so $A = 78$; the child on the insertion path is $B = 88$; and the next node on the path is $C = 82$.

Because the path goes right from A to B and left from B to C, this is a right-left case. The double rotation promotes $C = 82$ to become the local root. The left side of the rebuilt local subtree contains the part before 82 in the in-order traversal, and the right side contains the part after 82. Jackie emphasized that the Java implementation should not create new nodes for this; it should rearrange parent and child references, which is a constant-time local operation.

Restructuring after deletion

The final technical part shifted from insertion to deletion. Jackie pointed students to the deletion slides and highlighted that after deletion the rebalancing logic is different. If no ancestor becomes unbalanced, no rotation is needed. If an ancestor becomes unbalanced, choose A as the lowest unbalanced node. Then choose B as A's taller child. To choose C, inspect B's children: if their heights are different, choose the taller child; if their heights are the same, choose C so that A, B, C slant the same way.

The major difference is that deletion may require more than one trinode restructuring step. After one local rotation, students must continue checking upward along the ancestor path. Since the tree is balanced, the number of possible restructuring steps is $O(h) = O(\log n)$.

Deletion example: removing 80

The deletion example used the key sequence

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

inserted into an empty BST. After deleting 80, the first lowest unbalanced node is $A = 75$. Its taller child is $B = 60$, and B's taller child is $C = 55$. These nodes slant left-left, so the first repair is a single right rotation on the middle node $B = 60$. Jackie left the rest as an exercise: after that first rotation, continue upward to check whether another unbalanced node remains.

Key Terms, Notation, and Structures

- **Height-balance property**: At every internal node, the heights of the left and right children differ by at most 1.
- **Node height**: Computed bottom-up as $1 + \max(\text{height of left child}, \text{height of right child})$ for an internal node.
- **External node**: A placeholder child in the BST model; in the height calculations used here, external nodes have height 0.

- **Ancestor path**: The path from an inserted or affected node upward toward the root; balance checking after an update is driven by this path.
- **Lowest unbalanced node**: The first unbalanced node encountered when checking bottom-up from the affected node; this node is labeled A.
- **Trinode restructuring**: The rotation-based repair using nodes A, B, and C and the four surrounding subtrees T_1, T_2, T_3, T_4.
- **Single rotation**: A left or right rotation used when A, B, and C slant the same way.
- **Double rotation**: A left-right or right-left rotation used when A, B, and C form a zig-zag shape.
- **Middle node**: In a single rotation, the node B is promoted and becomes the local root after restructuring.
- **Lower node**: In a double rotation, the node C is promoted two levels and becomes the local root after restructuring.
- **In-order traversal invariant**: Rotations must preserve the same in-order traversal sequence, which is what preserves the BST sorting property.
- **Right rotation**: The single rotation used for a left-left shape; in the abstract example, it promotes B above A and C.
- **Right-left rotation**: A double rotation for a right-left shape; in the tutorial example, it promotes $C = 82$ after inserting 85.
- **Deletion rebalancing rule for B**: After deletion, choose B as the taller child of A, not merely the next node on the deletion path.
- **Deletion rebalancing rule for C**: Choose C as B's taller child; if the two child heights tie, choose C so that A, B, C slant the same way.
- $O(1)$ **local rotation**: A single local restructuring changes only a constant number of references in Java.
- $O(h) = O(\log n)$ **deletion restructuring**: Deletion can require continuing upward through the height of a balanced tree.
- `BSTNode`: The provided node class for Assignment 1; Jackie stated that students should not modify it.
- `model` **package**: The package where students should implement the required assignment logic.
- **JUnit tests**: The assignment tests act as the complete specification and should be followed in order as a development guide.
- **Java generics**: The assignment uses generic node/tree structures; students who are rusty should review the generics section and optional pre-study materials.

Key Examples and Demonstrations

Assignment 1 starter project and JUnit specification

Setup: Jackie imported the Assignment 1 starter project into Eclipse, showed the provided `BSTNode` class, the empty `model` package, and the JUnit test class.

Main point: The assignment is not just extra practice; it is a structured rehearsal for Programming Test 1. The provided tests describe what the code must do, and the intended workflow is to implement gradually until the tests pass.

What to practise: Import the starter project, run the JUnit tests before writing code, read the tests carefully, and then implement the required methods in the order suggested by the test structure.

Abstract left-left imbalance and right rotation

Setup: The notes use a tree with A above B above C, all slanting left, and subtrees T_1, T_2, T_3, T_4 all of height h.

Main point: Computing heights bottom-up shows that A is unbalanced. A right rotation on the middle node B restores the height-balance property while preserving the in-order traversal $T_1, C, T_2, B, T_3, A, T_4$.

What to practise: Recompute the heights without looking at the annotations, then redraw the rotated tree and verify both the in-order traversal and the child-height differences.

Right-left double rotation after inserting 85

Setup: Starting from the tree built from $\langle 44, 17, 78, 32, 50, 88, 82, 95\rangle$, insert 85 and update heights along the new node's ancestor path.

Main point: The lowest unbalanced node is $A = 78$, with $B = 88$ and $C = 82$. Since the shape is right-left, the repair is a double rotation on the lower node $C = 82$.

What to practise: Identify T_1, T_2, T_3, T_4 in the concrete tree, reconstruct the subtree with 82 as the local root, and then verify the final heights.

Deletion example: first restructuring after deleting 80

Setup: The notes build a larger BST from $\langle 50, 25, 10, 30, 5, 15, 27, 1, 75, 60, 80, 55\rangle$ and then delete 80.

Main point: After deletion, choose A as the lowest unbalanced node, then choose B and C by height. Here $A = 75$, $B = 60$, and $C = 55$, leading to a single right rotation as the first restructuring step.

What to practise: Finish the example from the notes: perform the right rotation, then continue checking upward to find whether another unbalanced node remains.

Common Pitfalls and Misconceptions

Modifying the provided `BSTNode` class

Pitfall: Changing `BSTNode` to make the assignment feel easier.

Why it causes problems: Jackie stated that this class is provided and should not be modified. Changing it can break the expected test environment and will not prepare you for Programming Test 1, where the same kind of provided class will be fixed.

How to avoid it: Put your logic in the required model class or helper methods indicated by the JUnit tests, and treat `BSTNode` as part of the fixed interface.

Waiting for solutions instead of practising

Pitfall: Looking briefly at the assignment, waiting for solutions, and hoping the solution will be enough preparation.

Why it causes problems: Assignment 1 is meant to expose where you struggle before Programming Test 1. Reading a solution after the fact does not build the tracing and coding habits needed under test conditions.

How to avoid it: Attempt the assignment early, record which methods or tests block you, and use any later solution only to repair specific misunderstandings.

Rotating when no rotation is needed

Pitfall: Assuming every insertion or deletion must trigger a rotation.

Why it causes problems: Rotations are only needed when the height-balance property is violated. If the ancestor path remains balanced, changing the tree structure can create errors instead of fixing anything.

How to avoid it: First compute or update heights and check balance along the relevant ancestor path; rotate only after identifying a lowest unbalanced node.

Forgetting that rotations must preserve in-order traversal

Pitfall: Treating rotations as arbitrary shape changes that only aim to make the tree look balanced.

Why it causes problems: A balanced-looking tree is still wrong if it violates the BST sorting property. The same in-order traversal before and after rotation is the key check.

How to avoid it: For every paper rotation, write or mentally check the in-order sequence. For code, think about how each child and parent reference preserves the same left-root-right order.

Confusing insertion and deletion rules for choosing B and C

Pitfall: Using the insertion-path rule for a deletion case.

Why it causes problems: After insertion, B and C come from the ancestor path of the inserted node. After deletion, the height loss means the repair is driven by taller subtrees, so B and C are chosen by height.

How to avoid it: Say the update type out loud before labeling A, B, C: after insertion, follow the insertion path; after deletion, choose the taller child, and then the taller child again unless there is a tie.

Assuming deletion always needs only one restructuring step

Pitfall: Stopping immediately after the first deletion rotation.

Why it causes problems: Deletion can reduce subtree height in a way that causes another imbalance higher up. Jackie emphasized that multiple trinode restructuring steps may be needed.

How to avoid it: After a deletion rotation, continue checking upward until the remaining ancestor path is balanced.

What You Should Be Able to Do

After this lecture, you should be able to:

- explain the height-balance property and apply it by comparing child-node heights;
- compute heights bottom-up for abstract trees and concrete BST diagrams;
- identify when a node violates the height-balance property and label the lowest unbalanced node as A;
- explain why the in-order traversal sequence must be preserved by any BST rotation;
- perform a single right rotation for a left-left trinode shape and verify that the result is balanced;
- distinguish single rotations from double rotations using the slant pattern of A, B, C;
- perform a right-left double rotation on the insertion example where key 85 is inserted;
- identify the four surrounding subtrees T_1, T_2, T_3, T_4 and reattach them correctly after restructuring;
- state the different rules for choosing B and C after insertion versus after deletion;
- explain why insertion requires at most one trinode restructuring step, while deletion can require multiple steps;
- justify why deletion restructuring may take $O(h) = O(\log n)$ steps in a balanced tree;
- import Assignment 1, run the JUnit tests, and use the test order as a development plan;
- explain why `BSTNode` should not be modified and why the `model` package is where your implementation belongs;
- connect the paper rotation diagrams to Java reference updates involving parent, left, and right links.

Suggested Practice

- Start Assignment 1 early: import the starter project, run the tests, read the instructions, and begin with the first failing test cases.
- Review the generics section of the assignment instructions if `BSTNode<E>` and generic helper classes feel unfamiliar.
- Redo the abstract left-left example from page 2 of the notes: compute heights, identify the imbalance, perform the right rotation, and verify in-order traversal.
- Redo the right-left insertion example from page 5 of the notes without looking at the final diagram; focus on labeling $A = 78$, $B = 88$, and $C = 82$ correctly.
- Complete at least one other insertion rotation exercise from the Tutorial 4 instructions, especially a left-right or single-right case not fully worked in class.
- Finish the deletion example from page 9 of the notes after the first right rotation, then continue searching upward for any remaining imbalance.
- Practise deletion cases from the self-balancing BST slides even though deletion is not required in Assignment 1, because Jackie connected it to Programming Test 1 preparation.
- For each rotation you practise, write down the in-order sequence before and after the rotation to check that the BST property is preserved.
- When coding, map every paper-level child/parent change to the actual Java reference updates needed in your implementation.

Where to Review

- **Tutorial 4 notes, page 1**: overview of the tutorial focus: tree rotations, abstract trinode restructuring, right-left insertion restructuring, and deletion restructuring.
- **Tutorial 4 notes, page 2**: abstract left-left imbalance, right rotation on B, and the in-order traversal check $T_1, C, T_2, B, T_3, A, T_4$.
- **Tutorial 4 notes, page 3**: summary of single versus double rotations based on whether A, B, C slant the same way or differently.
- **Tutorial 4 notes, page 4**: right-left double rotation, including the two approaches: sequential single rotations or simultaneous reconstruction using in-order traversal.
- **Tutorial 4 notes, page 5**: the insertion of 85 and the right-left restructuring example.
- **Tutorial 4 notes, pages 6–7**: deletion rebalancing rules, including how to choose A, B, and C and why multiple restructuring steps may be needed.

- **Tutorial 4 notes, pages 8–9**: deletion restructuring exercises, including the partially worked multiple-rotation example after deleting 80.
- **Assignment 1 instructions and starter project**: review the generics explanation, requirements, submission instructions, provided `BSTNode`, empty `model` package, and JUnit tests.
- **Self-balancing BST slides**: revisit the rotation and deletion slides Jackie pointed to, especially the deletion and post-deletion rotation rules.
- **Posted tutorial solutions or walkthrough videos**: use them after attempting the paper exercises yourself, not as a substitute for first-pass practice.

Closing Reminder

This tutorial digest is only a roadmap for review. To learn the material, you still need to redraw the rotation diagrams, recompute heights, trace ancestor paths, finish the deletion example, and connect each paper restructuring step to Java reference updates in Assignment 1. Reading the digest or waiting for solutions will not be enough preparation for Programming Test 1; use the official notes, recording, starter code, JUnit tests, and your own practice to check that you can actually perform and implement the rotations.