Computer Science 331

Red Black Trees: Rotations and Insertions

Mike Jacobson

Department of Computer Science
University of Calgary

Lecture #17

Outline

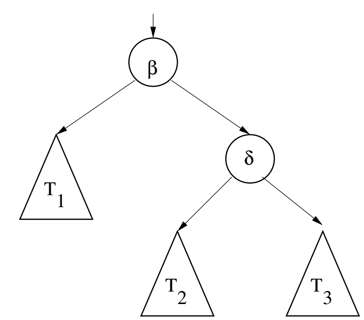
What is a Rotation?

Rotation:

- a local operation on a binary search tree
- preserves the binary search tree property
- used to implement operations on red-black trees (and other height-balanced trees)
- two types:
  - *Left Rotations*
  - *Right Rotations*

Left Rotation: Tree Before Rotation

Tree Before Performing Left Rotation at β:



Assumption: β has a right child, δ

Useful Consequences of Binary Search Tree Property

Lemma 1

For all $\alpha \in T_1$, $\gamma \in T_2$, *and* $\zeta \in T_3$,

$$\alpha < \beta < \gamma < \delta < \zeta$$

Proof.

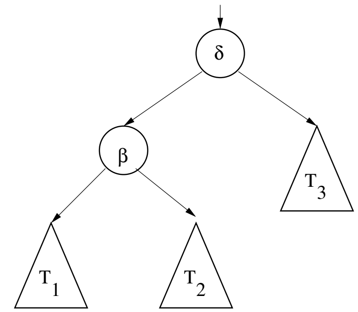
T_1: is the left subtree of β (so $\alpha < \beta$)

T_2: is contained in the right subtree of β (so $\beta < \gamma$)
is the left subtree of δ (so $\gamma < \delta$)

T_3: is the right subtree of δ (so $\delta < \zeta$)

Thus, T is a BST. □

Left Rotation: Tree After Rotation

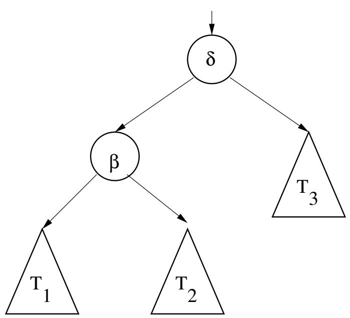


Notice that this is still a BST (inequalities on previous slide still hold)

Pseudocode: *Introduction to Algorithms*, page 278

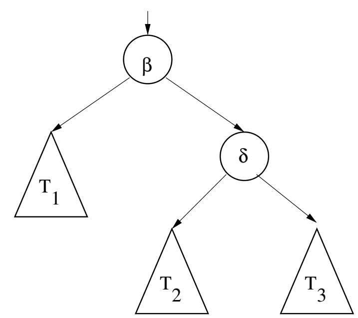
Right Rotation: Tree Before Rotation

Tree Before Performing Right Rotation at δ:



Assumption: δ has a left child, β

Right Rotation: Tree After Rotation



Note: This is both the mirror-image, and the *reversal*, of a left-rotation.

Effects of a Rotation

Exercises:

1. Confirm that a tree is a BST after a rotation if it was one before.
2. Confirm that a (single left or right) rotation can be performed using

$$\Theta(1) \text{ operations}$$

including comparisons and assignments of pointers or references

Red-Black Properties

Recall that the following properties must be maintained:

1. Every node is either red or black.
2. The root is black.
3. Every leaf (NIL) is black.
4. If a node is red, then both its children are black.
5. For each node, all paths from the node to descendant leaves contain the same number of black nodes.

Beginning an Insertion

Suppose we wish to insert an object x into a red black tree T.

if T includes an object with the same key as x **then**

- throw `FoundException` (and terminate)

else

- Insert a new node storing the object x in the usual way.
  Both of the children of this node should be (black) leaves.
- Color the new node *red*.
- Let z be a pointer to this new node.
- Proceed as described next...

How To Continue

Strategy for Finishing the Operation:

- At this point, T is not necessarily a red-black tree, but there is only a problem at one *problem area* in the tree.
  - newly-inserted node (color red) may violate red-black tree properties #2 or #4
- Rotations and recoloring of nodes will be used to move the "problem area" closer to the root.
- Once the "problem area" has been moved to the root, at most one correction turns T back into a red-black tree.

Structure of Rest of Insertion Algorithm

Recall our assumption from the last lecture: parent of root is a dummy node with color black

Note:

- During the execution of this algorithm, z *always* points to a red node; this is the only place where there might be a problem
- z initially points to the newly-inserted node (color red)

while the parent of z is red **do**
  Make an adjustment (to be described shortly)
end while
if z is the root **then**
  Change the color of z to black
end if

Loop Invariant

z is red and **exactly one** of the following is true:

1. The parent of z is also red.
   All other red-black properties are satisfied.
2. z is the root.
   All other red-black properties are satisfied.
3. All red-black properties are satisfied.
   Thus T is a red-black tree.

Note: Loop invariant + failure of loop test $\Rightarrow$ 2 or 3.

Loop Variant

Loop Variant: depth of z

Consequence:

- number of executions of loop body is linear in the height of T.

Note:

- We will need to check that this is a loop variant!
- This is the case if z is moved closer to the root after every iteration.

Subcases of Case 1

Note: Since the parent of z is red it is not the root; the *grandparent* of z must be black.

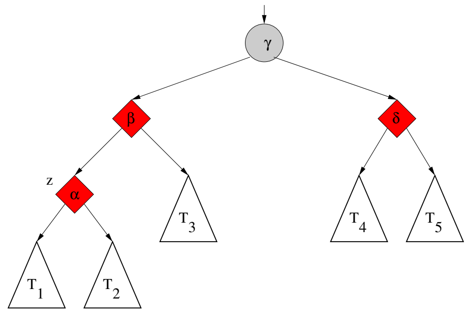
1. Parent of z is a left child; sibling y of parent of z is red.
   1. z is a left child.
   2. z is a right child.
2. Parent of z is a left child; sibling y of parent of z is black.
   z is a right child.
3. Parent of z is a left child; sibling y of parent of z is black.
   z is a left child.

Subcases 4–6: Mirror images of subcases 1–3:

- Exchange "left" and "right;" parent(z) is now a right child

Subcase 1a: Tree Before Adjustment

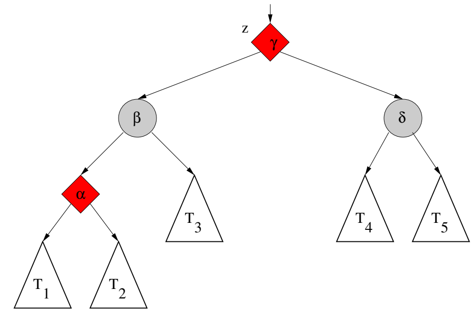
z is left child, parent of z is a left child; sibling y of parent of z is red



Adjustment:

-

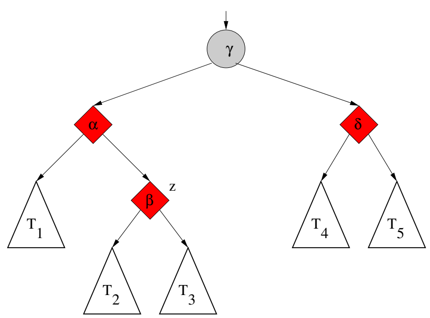
Subcase 1a: Tree After Adjustment



Node z may still cause violations of red-black tree properties #2 or #4, but z has moved closer to the root.

Subcase 1b: Tree Before Adjustment

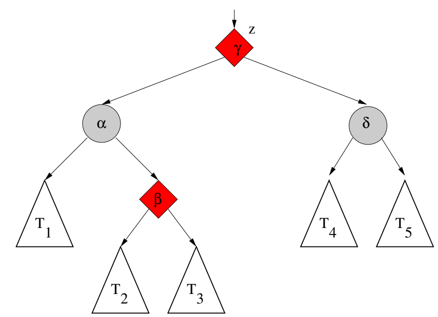
z is right child; parent of z is a left child; sibling y of parent of z is red;



Adjustment:

-

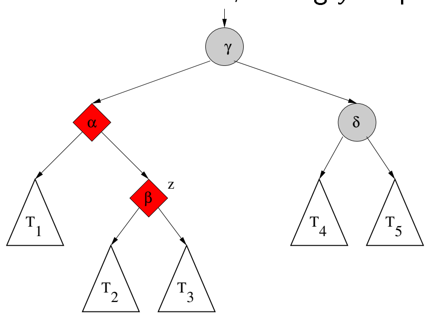
Subcase 1b: Tree After Adjustment



Node z may still cause violations of red-black tree properties #2 or #4, but z has moved closer to the root.

Case 2: Tree Before Adjustment

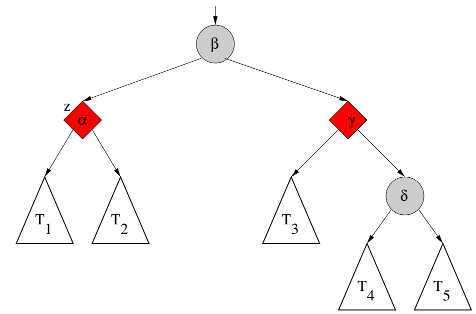
z is right child; parent of *z* is left child; sibling *y* of parent of *z* is black;



Adjustment:

-
-

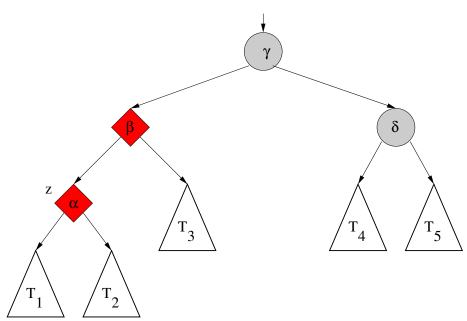
Case 2: Tree After Adjustment



Parent of *z* is now black, so the while loop terminates and we are finished.

Case 3: Tree Before Adjustment

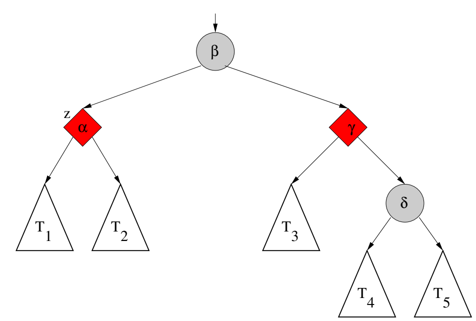
z is left child; parent of *z* is left child; sibling *y* of parent of *z* is black;



Adjustment:

-

Case 3: Tree After Adjustment



Parent of *z* is now black, so the while loop terminates and we are finished.

Exercises

3. Describe cases 4–6 and draw the corresponding trees.
4. Confirm that the "loop invariant" holds after each adjustment.
5. Confirm that the distance of z from the root decreases after each adjustment — so the claimed "loop variant" satisfies the properties it should.

Note: These cases are described in the text (Section 11.3), although the numbering of the cases is slightly different from our's.

Handling Cases B and C

Case B: z is the root (so, the root is red)

- *All other red-black properties are satisfied.*
- *Adjustment:* change the color of the root to black.

Case C: T is a red-black tree.

- *Adjustment:* We're finished!

Pseudocode for adjustments: *Introduction to Algorithms*, page 281

Exercises:

6. Show that the "insertion" algorithm as a whole is correct.
7. Confirm that the total number of steps used by the insertion algorithm is at most linear in the depth of the given tree.