

CSC 301: Analysis of Algorithms

Lecture 26: Red-Black Trees, Balance

William Rebelsky
[rebelskyw]

Grinnell College

Wednesday 3rd April, 2024

Reminders

- Homework 5 grades have been released. A 4 is required for an S
- Homework 6 has been released on gradescope, it is due next Wednesday
- The Essay is due next Wednesday
- Email me a revision of the midterm or set up time to meet with me in the next two weeks
- AA has asked me to remind students that the last day to withdraw from a class with a "W" is this Friday April 5. If you are thinking about doing so, I would prefer if you let me know.

Table of Contents

1 RBT

2 Rotation

3 Insertion

4 Deletion

Red Black Trees Basics

- RBT: BST with color (red or black)
- Keeps trees relatively balanced ($2\lg(n+1)=O(\lg n)$)
- Red Black Properties
 - 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, both its children are black
 - 5 For every node, all simple paths from the node to descendent leaves contain the same number of black nodes. This is the black-height ($bh(x)$) of the node x .
- For convenience, use a single sentinel ($T.NIL$) for all NIL values (see image)
- For convenience, we tend to omit drawing the NIL leaves

Practice 1

Do the following:

- ➊ Draw the complete BST of height 3 on the keys 1:15
- ➋ Add the NIL leaves
- ➌ Color the nodes in such a way that the black heights are 2, 3, and 4

Red Black Trees ADT

- Minimum: Return the minimum value in the tree
- Maximum: Return the maximum value in the tree
- Predecessor: Return the previous value in the tree when printed in sorted order
- Successor: Return the next value in the tree when printed in sorted order
- Insert: Add an element to the tree
- Delete: Remove an element from the tree
- All operations take time proportional to height ($O(\lg n)$)
- Key=Value/data (normally), Left Child, Right Child, Parent, **Color**

Practice 2

Which of the Operations can be copied directly from BST and which must be modified?

- Minimum: Return the minimum value in the tree
- Maximum: Return the maximum value in the tree
- Predecessor: Return the previous value in the tree when printed in sorted order
- Successor: Return the next value in the tree when printed in sorted order
- Insert: Add an element to the tree
- Delete: Remove an element from the tree

Practice 2

Which of the Operations can be copied directly from BST and which must be modified?

- Minimum: Return the minimum value in the tree
- Maximum: Return the maximum value in the tree
- Predecessor: Return the previous value in the tree when printed in sorted order
- Successor: Return the next value in the tree when printed in sorted order
- **Insert: Add an element to the tree**
- **Delete: Remove an element from the tree**

Table of Contents

- 1 RBT
- 2 Rotation
- 3 Insertion
- 4 Deletion

Rotation

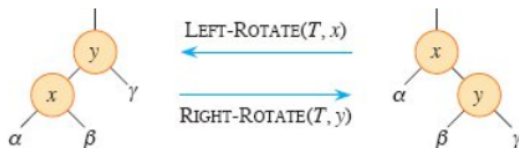


Figure 13.2 The rotation operations on a binary search tree. The operation $\text{LEFT-ROTATE}(T, x)$ transforms the configuration of the two nodes on the right into the configuration on the left by changing a constant number of pointers. The inverse operation $\text{RIGHT-ROTATE}(T, y)$ transforms the configuration on the left into the configuration on the right. The letters α , β , and γ represent arbitrary subtrees. A rotation operation preserves the binary-search-tree property: the keys in α precede $x.\text{key}$, which precedes the keys in β , which precede $y.\text{key}$, which precedes the keys in γ .

Practice 1

Assuming x , $y.p$, y , $y.\gamma$, $x.\alpha$, and $x.\beta$ exist, how many pointers need to change?

Table of Contents

- 1 RBT
- 2 Rotation
- 3 Insertion
- 4 Deletion

Insertion

- Insert z as in a normal BST
- Color z red
- Fix any issues with the coloring
- Note: of the 5 properties: 1,3,5 still hold. 2 is Trivial to fix, so we worry about property 4
- RBT Property 4: If a node is red, both it's children are black

Insertion 2

- Check if z's parent is a left or right child
- Determine the color of z's "uncle"
- Check which of the three cases:
 - ❶ Case 1: both z's parent and and uncle are red. Therefore the grandparent is black. Turn the grandparent red and both it's children black, repeat going up 2 levels.
 - ❷ Case 2: z's uncle is black and z is a right child. Perform a left rotation to get to Case 3.
 - ❸ Case 3: z's uncle is black and z is a left child. Perform a right rotation.

Rotating in cases 2 and 3 in insertion

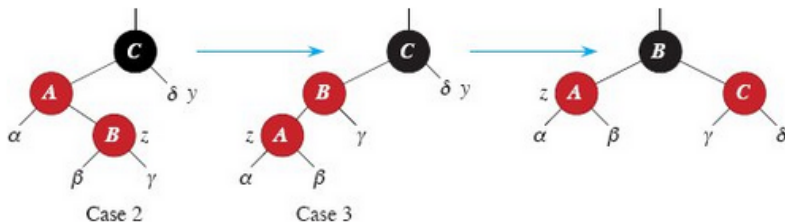


Figure 13.6 Cases 2 and 3 of the procedure RB-INSERT-FIXUP. As in case 1, property 4 is violat

Practice

- 13.3-1 Instead of setting the color of the new node z to red, we could set it to black. This would ensure we don't violate properties 2 or 4. Why don't we do that?
- 13.3-2 Show the red-black trees that result after successively inserting the keys 41,38,31,12,19,8 into an empty RB tree. Keep the final tree available

Table of Contents

- 1 RBT
- 2 Rotation
- 3 Insertion
- 4 Deletion

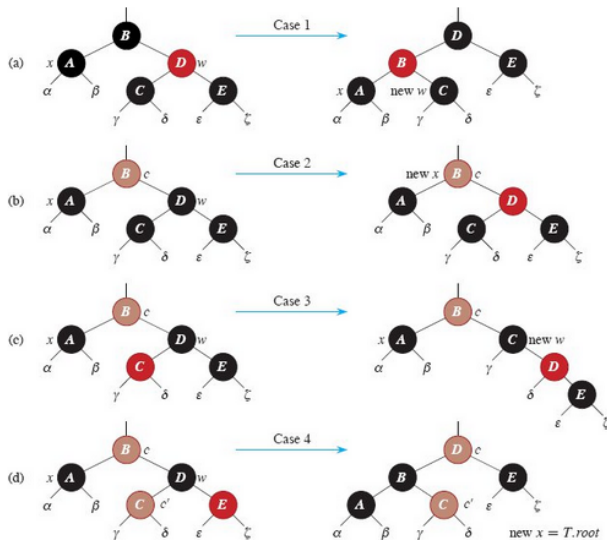
Deletion

- RB-Transplant changes slightly and becomes simpler
- Perform the same steps as in BST
- Clean up any red-black violations caused by moving y into z's position
 - If y was red, no black heights changed, no red nodes have been adjacent, and y could not have been the node, so no properties are violated
 - If y was black, a number of issues could exist. Property 2 could be violated if y was the root. If both x and its new parent are red then Property 4 would be violated. Moving y reduces the black nodes on one path violating Property 5. To fix property 5, we could move the black color of node y to node x that is moved to y's position, leaving x either black-black or red-black (which violates Attribute 1)

Deletion 2

- 4 cases:
 - ❶ Case 1: x's sibling w is red: swap colors and perform a left rotation to convert to another case
 - ❷ Case 2: x's sibling w is black, as are both of w's children: move the extra black up the tree
 - ❸ Case 3: x's sibling w is black, it's left child is red and right child is black: convert to Case 4 by exchanging colors and performing a right rotation
 - ❹ Case 4: x's sibling w is black and w's right child is red: Swap a couple colors and performing a left rotation.

Deletion Fix up Cases



Practice

- 13.4-4. Given the tree from Insertion, show the resulting RB trees that result from successive deletions in the order 8,12,19,31,38,41