Lecture 02/27/17

Lecturer: Xiaodi Wu

Reading: Chapter 13.4
Red-Black Trees: Deletion

Binary Tree Operations

- Tree-Search: $O(h) = O(\log(n))$.
- Tree-Insert, Tree-Delete: $O(h) = O(\log(n))$.
- Search: won't lead to any violation; Insert, Delete: potentially lead to violations of red-black trees.
Red-Black Trees: Deletion

Binary Tree Operations

- Tree-Search: $O(h) = O(\log(n))$.
- Tree-Insert, Tree-Delete: $O(h) = O(\log(n))$.
- Search: won’t lead to any violation; Insert, Delete: potentially lead to violations of red-black trees.

Tools to Deal with Violations

- Need to somehow change the tree structure.
- Fix violations without having new violations.
- **Rotations**: the first tool to make the chance, keeping the binary search tree property.
- **Deletion**: perform normal Tree-Delete and then RB-Delete-FIX-UP.
Red-Black Trees: Deletion

Naming the nodes: Tree-Delete

- First perform the normal Tree-Delete. (RB-Delete on page 324)
Red-Black Trees: Deletion

Naming the nodes: Tree-Delete

- First perform the normal Tree-Delete. (RB-Delete on page 324)
- Let $y$ point to the node that will be eventually removed.
- Let $x$ point to the node that will replace $y$’s place.
Red-Black Trees: Deletion

Naming the nodes: Tree-Delete

- First perform the normal Tree-Delete. (RB-Delete on page 324)
- Let $y$ point to the node that will be eventually removed.
- Let $x$ point to the node that will replace $y$’s place.
- What are $y$, $x$ in the Case 1, 2, 3 of Tree-Delete?
Red-Black Trees: Deletion

When $y$’s color is RED

- Claim: the red-black properties still hold when $y$ is removed.
Red-Black Trees: Deletion

When y’s color is RED

▶ Claim: the red-black properties still hold when y is removed.

1. No black-heights in the tree have changed.
2. No red nodes have been made adjacent. Because y takes z’s place in the tree, along with z’s color, we cannot have two adjacent red nodes at y’s new position in the tree. In addition, if y was not z’s right child, then y’s original right child x replaces y in the tree. If y is red, then x must be black, and so replacing y by x cannot cause two red nodes to become adjacent.
3. Since y could not have been the root if it was red, the root remains black.
What are possible violations?

Red-Black Tree Properties

- **Color Property**: every node is either red or black.
- **Root Property**: the root is black.
- **External Property**: every external node (leaf, NIL) is black.
- **Internal Property**: the children of a red node are black.
- **Depth Property**: For each node, all simple paths from the node to descendant leaves contain the same number of black nodes. (Equivalently, each node replaced by the root.)
What are possible violations?

Red-Black Tree Properties

- **Color Property**: every node is either red or black.
- **Root Property**: the root is black.
- **External Property**: every external node (leaf, NIL) is black.
- **Internal Property**: the children of a red node are black.
- **Depth Property**: For each node, all simple paths from the node to descendant leaves contain the same number of black nodes. (Equivalently, each node replaced by the root.)

**Violations when y is black**: Root Property, Internal Property, and Depth Property?
What are possible violations?

Red-Black Tree Properties

- **Color Property**: every node is either red or black.
- **Root Property**: the root is black.
- **External Property**: every external node (leaf, NIL) is black.
- **Internal Property**: the children of a red node are black.
- **Depth Property**: For each node, all simple paths from the node to descendant leaves contain the same number of black nodes. (Equivalently, each node replaced by the root.)

Violations when $y$ is black: Root Property, Internal Property, and Depth Property? Which can/must happen simultaneously?
Possible Violation

Combinations when $y$ is black

- Root Property violation: $y$ must be the root. ($x$ the new root). No violation of Internal Property, Depth Property.
Possible Violation

Combinations when \( y \) is black

- Root Property violation: \( y \) must be the root. (\( x \) the new root). No violation of Internal Property, Depth Property.
- Otherwise must violate the depth property (why?).

Potential violation of the internal property when \( x \) is red.

Easy fix when \( x \) is red
- Change \( x \)'s color to black. Why? (fix all three violations).
Possible Violation

Combinations when \( y \) is black

- Root Property violation: \( y \) must be the root. (\( x \) the new root). No violation of Internal Property, Depth Property.
- Otherwise must violate the depth property (why?).
- Potential violation of the internal property when \( x \) is red.
Possible Violation

Combinations when $y$ is black

- Root Property violation: $y$ must be the root. ($x$ the new root). No violation of Internal Property, Depth Property.
- Otherwise must violate the depth property (why?).
- Potential violation of the internal property when $x$ is red.

Easy fix when $x$ is red

- Change $x$’s color to black. Why? (fix all three violations)
Red-Black Trees: Tree-Deletion General Fix

RB-DELETE-FIXUP\((T, x)\)

1. while \(x \neq T\.root\) and \(x\.color == \text{BLACK}\)
2. if \(x == x\.p\.left\)
   3. \(w = x\.p\.right\)
   4. if \(w\.color == \text{RED}\)
      5. \(w\.color = \text{BLACK}\)  // case 1
      6. \(x\.p\.color = \text{RED}\)  // case 1
     7. LEFT-ROTATE\((T, x\.p)\)  // case 1
     8. \(w = x\.p\.right\)  // case 1
   9. if \(w\.left\.color == \text{BLACK}\) and \(w\.right\.color == \text{BLACK}\)
      10. \(w\.color = \text{RED}\)  // case 2
      11. \(x = x\.p\)  // case 2
     12. else if \(w\.right\.color == \text{BLACK}\)
        13. \(w\.left\.color = \text{BLACK}\)  // case 3
        14. \(w\.color = \text{RED}\)  // case 3
        15. RIGHT-ROTATE\((T, w)\)  // case 3
        16. \(w = x\.p\.right\)  // case 3
      17. \(w\.color = x\.p\.color\)  // case 4
      18. \(x\.p\.color = \text{BLACK}\)  // case 4
      19. \(w\.right\.color = \text{BLACK}\)  // case 4
    20. LEFT-ROTATE\((T, x\.p)\)  // case 4
    21. \(x = T\.root\)  // case 4
   22. else (same as then clause with “right” and “left” exchanged)
      23. \(x\.color = \text{BLACK}\)
Red-Black Trees: Tree-Deletion General Fix

General Fix

- Fix Root Property Violation: (Ex 13.4-1) Why?

The procedure RB-DELETE-FIXUP restores properties 1, 2, and 4. Exercises 13.4-1 and 13.4-2 ask you to show that the procedure restores properties 2 and 4, and so in the remainder of this section, we shall focus on property 1. The goal of the while loop in lines 1–22 is to move the extra black up the tree until

1. \(x\) points to a red-and-black node, in which case we color \(x\) (singly) black in line 23;

2. \(x\) points to the root, in which case we simply "remove" the extra black; or

3. having performed suitable rotations and recolorings, we exit the loop.
General Fix

- Fix Root Property Violation: (Ex 13.4-1) Why?
- Fix Internal Property Violation: (Ex 13.4-2) Why?
Red-Black Trees: Tree-Deletion General Fix

General Fix

- Fix Root Property Violation: (Ex 13.4-1) Why?
- Fix Internal Property Violation: (Ex 13.4-2) Why?

1. \( x \) points to a red-and-black node, in which case we color \( x \) (singly) black in line 23;
2. \( x \) points to the root, in which case we simply “remove” the extra black; or
3. having performed suitable rotations and recolorings, we exit the loop.
Red-Black Trees: Deletion Fix Hard Case

Hard-case

- Depth-Property: because of deleting one black node, one need to compensate the black depth somehow.
Red-Black Trees: Deletion Fix Hard Case

Hard-case

- Depth-Property: because of deleting one black node, one need to compensate the black depth somehow.
- Imagine, if we can add one black count to nodes, namely, black-red, black-black, nodes. Then we are done!
Red-Black Trees: Deletion Fix Hard Case

Hard-case

- Depth-Property: because of deleting one black node, one need to compensate the black depth somehow.
- Imagine, if we can add one black count to nodes, namely, black-red, black-black, nodes. Then we are done!
- Moreover, we want to make the change to the node $x$. What is the catch?
Red-Black Trees: Deletion Fix Hard Case

Hard-case

- Depth-Property: because of deleting one black node, one need to compensate the black depth somehow.
- Imagine, if we can add one black count to nodes, namely, black-red, black-black, nodes. Then we are done!
- Moreover, we want to make the change to the node $x$. What is the catch?
- Violation of the color property. What are the ways out?
Red-Black Trees: Deletion Fix Hard Case

Hard-case

- Depth-Property: because of deleting one black node, one need to compensate the black depth somehow.
- Imagine, if we can add one black count to nodes, namely, black-red, black-black, nodes. Then we are done!
- Moreover, we want to make the change to the node $x$. What is the catch?
- Violation of the color property. What are the ways out?
- Simple solution when $x$ is red or $x$ is the root. Why?
Hard-case when \( x \) is black

- Simple solution when \( x \) is red or \( x \) is the root. Thus, our goal is to gradually change \( x \) such that \( x \) is either closer to be the root or red.
Hard-case when $x$ is black

- Simple solution when $x$ is red or $x$ is the root. Thus, our goal is to gradually change $x$ such that $x$ is either closer to be the root or red.

- Enumerate 4 Cases: Case 1
Hard-case when $x$ is black

- Simple solution when $x$ is red or $x$ is the root. Thus, our goal is to gradually change $x$ such that $x$ is either closer to be the root or red.
- Enumerate 4 Cases: Case 2
Red-Black Trees: Deletion Fix Hard Case

Hard-case when $x$ is black

- Simple solution when $x$ is red or $x$ is the root. Thus, our goal is to gradually change $x$ such that $x$ is either closer to be the root or red.
- Enumerate 4 Cases: Case 3
Red-Black Trees: Deletion Fix Hard Case

Hard-case when $x$ is black

- Simple solution when $x$ is red or $x$ is the root. Thus, our goal is to gradually change $x$ such that $x$ is either closer to be the root or red.
- Enumerate 4 Cases: Case 4

![Diagram showing the transformation of a red-black tree with nodes A, B, C, D, and E, and pointers $\alpha, \beta, \gamma, \delta, \epsilon, \zeta$, with transformations for Case 4 and new $x = T.root$]
Red-Black Trees: Deletion Fix Hard Case

Hard-case when $x$ is black

- Simple solution when $x$ is red or $x$ is the root. Thus, **our goal** is to gradually change $x$ such that $x$ is either closer to be the root or red.
- Logic behind 4 cases:
  - Case 1 $\rightarrow$ Case 2, 3, 4.
  - Case 2 $\rightarrow$ $x$ moves up one level.
  - Case 3 $\rightarrow$ Case 4.
  - Case 4 $\rightarrow$ directly fix. $x$ is the root.

$O(h) = O(\log(n))$ iteration at most.
Red-Black Trees: Deletion Fix Hard Case

Hard-case when $x$ is black

- Simple solution when $x$ is red or $x$ is the root. Thus, **our goal** is to gradually change $x$ such that $x$ is either closer to be the root or red.
- Logic behind 4 cases:
  - Case 1 $\rightarrow$ Case 2, 3, 4.
  - Case 2 $\rightarrow$ $x$ moves up one level.
  - Case 3 $\rightarrow$ Case 4.
  - Case 4 $\rightarrow$ directly fix. $x$ is the root.
- $O(h) = O(\log(n))$ iteration at most.