CS350: Data Structures

Red-Black Trees

James Moscola
Department of Engineering & Computer Science
York College of Pennsylvania
Red-Black Tree

• An alternative to AVL trees

• Insertion can be done in a bottom-up or a top-down fashion
  - An AVL tree uses a pass down the tree for an insertion and a second pass back up the tree to update node heights and potentially rebalance the tree
  - A top-down insertion into a red-black tree requires only a single pass down the tree
    • We’ll focus on bottom-up insertion
Red-Black Tree (Cont.)

• A red-black tree is a binary search tree that has the following properties:

  (1) Every node is colored either red or black

  (2) The root node is black

  (3) If a node is red, its children must be black

  (4) Every path from a node to a null node must contain the same number of black nodes

• These properties must be maintained after each insertion or deletion operation

• A null node is considered black
Example Red-Black Tree

1. Every node is colored either red or black
2. The root node is black
3. If a node is red, its children must be black
4. Every path from a node to a null link must contain the same number of black nodes
Red-Black Tree Insertion

• When inserting a new leaf node, what color should it be?

\[ \text{insert}(24) \]

(1) Every node is colored either red or black
(2) The root node is black
(3) If a node is red, its children must be black
(4) Every path from a node to a null link must contain the same number of black nodes
Red-Black Tree Insertion

- Inserting as a **black** node violates property #4

(1) Every node is colored either red or black
(2) The root node is black
(3) If a node is red, its children must be black

**X** (4) Every path from a node to a null link must contain the same number of black nodes
Red-Black Tree Insertion

- Inserting as a red node satisfies all four properties (in this case)

1. Every node is colored either red or black
2. The root node is black
3. If a node is red, its children must be black
4. Every path from a node to a null link must contain the same number of black nodes
Red-Black Tree Insertion

- Inserting as a red may not always satisfy the four properties

```python
insert(99)
```

(1) Every node is colored either red or black
(2) The root node is black
(3) If a node is red, its children must be black
(4) Every path from a node to a null link must contain the same number of black nodes
Red-Black Tree Insertion

• Easier to fix a violation of property #3 than it is to fix a violation of property #4
  - So, insert all nodes as red nodes

• Must repair violations of property #3 and any new violations that occur as a result of the tree modification
  - Operations for repairing the tree include:
    • Single and Double Rotations (similar to AVL trees)
    • Color changes
Red-Black Tree Insertion (Bottom-Up)

- Insert nodes into a red-black tree using the standard binary search tree insertion
  - Make the newly inserted node red
  - If the parent of the newly inserted node is black, then no violations have occurred and the insertion is complete
  - If the parent of the newly inserted node is red, then property #3 has been violated and must be fixed through rotations and recoloring
    - Four different cases must be considered
Red-Black Tree Insertion -- Violations

- The four cases to consider when property #3 is violated (i.e. when a red node is inserted as the child of another red node)
  
  1. Parent’s sibling is red and new node is inserted as an outside grandchild
  2. Parent’s sibling is red and new node is inserted as an inside grandchild
  3. Parent’s sibling is black and new node is inserted as an outside grandchild
  4. Parent’s sibling is black and new node is inserted as an inside grandchild

- There is a different approach to fix each of these cases
Insert Violation -- **Case #1**

1. Parent’s sibling is **red** and new node is inserted as an outside grandchild.
Fixing the Insertion Violation -- **Case #1**

No rotation necessary

Make Parent and Sibling black, Grandparent becomes red

Since G becomes red, more work might be needed further up the tree
Insert Violation -- **Case #2**

(2) Parent’s sibling is **red** and new node is inserted as an inside grandchild
Fixing the Insertion Violation -- **Case #2**

No rotation necessary

Make Parent and Sibling black, Grandparent becomes red

Since G becomes red, more work might be needed further up the tree
Insert Violation -- **Case #3**

(3) Parent’s sibling is **black** and new node is inserted as an **outside grandchild**
Fixing the Insertion Violation -- **Case #3**

Perform a **single** rotation

- P becomes new root of subtree
- Nodes P and G get a color change
Insert Violation -- **Case #4**

(4) Parent’s sibling is **black** and new node is inserted as an **inside grandchild**
Fixing the Insertion Violation -- **Case #4**

Perform a **double rotation**

- **X becomes new root of subtree**
- **Nodes X and G get a color change**
Red-Black Tree Insertion Example

\textbf{insert(99)}

(1) Every node is colored either red or black
(2) The root node is black
(3) If a node is red, its children must be black
(4) Every path from a node to a null link must contain the same number of black nodes
Red-Black Tree Insertion Example

Case #1: Parent’s sibling is red and new node is inserted as an outside grandchild

insert(99)
Red-Black Tree Insertion Example

Case #1: Parent’s sibling is red and new node is inserted as an outside grandchild.

Push black down from the grandparent.

insert(99)
Red-Black Tree Insertion Example

```
insert(99)
```

Push black down from the grandparent.
Red-Black Tree Insertion Example

```
insert(99)
```

Balanced
Red-Black Tree Insertion Example

(1) Every node is colored either red or black
(2) The root node is black
(3) If a node is red, its children must be black
(4) Every path from a node to a null link must contain the same number of black nodes
Red-Black Tree Insertion Example

Case #4: Parent’s sibling is **black** and new node is inserted as an inside grandchild
Red-Black Tree Insertion Example

Perform a double rotation:
First, rotate between nodes 23 & 24,
then, rotate between nodes 23 & 20
and change their colors
Red-Black Tree Insertion Example

Hey, look who showed up after the first rotation!! It's case #3: Parent's sibling is black and new node is inserted as an outside grandchild.

Perform a double rotation:
First, rotate between nodes 23 & 24, then, rotate between nodes 23 & 20 and change their colors.
Red-Black Tree Insertion Example

perform a double rotation:
first, rotate between nodes 23 & 24,
then, rotate between nodes 23 & 20
and change their colors
Red-Black Tree Insertion Example

insert(23)

Finally, recolor nodes 23 and 20
Red-Black Tree Insertion Example

insert(23)

Balanced
Red-Black Tree Deletion

• Start with standard BST deletion
  - In BST deletion, when deleting a node with two children, the node was not actually deleted -- its contents were replaced
    1. Contents replaced with contents from successor or predecessor
    2. Then the successor/predecessor node was deleted
      - Successor/predecessor can have 0 or 1 child (if the node had two children, then one of its children would be the successor/predecessor)
    - Replacing the contents of a node do not affect the coloring of the node, therefore the properties of the red-black tree are not altered
    - Removal of the node with 0 or 1 child needs consideration as it can potentially cause violations in the red-black tree
Red-Black Tree Deletion

• If the node removed is red, then there are no problems and the 4 properties of the red-black tree will still be true -- there is nothing to do in this case

• If the node removed is black, then there is a new violation in the red-black tree -- violation of property #4

(4) Every path from a node to a null node must contain the same number of black nodes

- It is necessary to work back up the tree and account for the missing black node