Design and Analysis of Algorithms

CSE 5311

Lecture 10 Balanced Search Trees

Song Jiang, Ph.D.
Department of Computer Science and Engineering
Balanced search trees

**Balanced search tree:** A search-tree data structure for which a height of $O(\lg n)$ is guaranteed when implementing a dynamic set of $n$ items.

**Examples:**
- AVL trees
- 2-3 trees
- 2-3-4 trees
- B-trees
- Red-black trees
Red-black trees

This data structure requires an extra one-bit color field in each node.

**Red-black properties:**

1. Every node is either red or black.
2. The root and leaves (NIL’s) are black.
3. If a node is red, then its parent is black.
4. All simple paths from any node \( x \) to a descendant leaf have the same number of black nodes \( = \) black-height(\( x \)).
Example of a red-black tree

```
    7
   / \
  3   18
 / \  /  \
NIL NIL 10 22
     /   /  \
    8   11 26
   / \  /   /  \
NIL NIL NIL NIL
```

$h = 4$
Example of a red-black tree

1. Every node is either red or black.
Example of a red-black tree

2. The root and leaves (NIL’s) are black.
Example of a red-black tree

3. If a node is red, then its parent is black.
4. All simple paths from any node $x$ to a descendant leaf have the same number of black nodes = \textit{black-height}(x).
Height of a red-black tree

**Theorem.** A red-black tree with $n$ keys has height
\[ h \leq 2 \lg(n + 1). \]

**Proof.** (The book uses induction. Read carefully.)

**Intuition:**
- Merge red nodes into their black parents.
Height of a red-black tree

**Theorem.** A red-black tree with \( n \) keys has height

\[
h \leq 2 \lg(n + 1).
\]

**Proof.** (The book uses induction. Read carefully.)

**Intuition:**
- Merge red nodes into their black parents.
Height of a red-black tree

**Theorem.** A red-black tree with \( n \) keys has height

\[ h \leq 2 \log(n + 1). \]

**Proof.** (The book uses induction. Read carefully.)

**Intuition:**
- Merge red nodes into their black parents.
Height of a red-black tree

Theorem. A red-black tree with $n$ keys has height

$$h \leq 2 \lg(n + 1).$$

Proof. (The book uses induction. Read carefully.)

Intuition:
- Merge red nodes into their black parents.
Height of a red-black tree

**Theorem.** A red-black tree with n keys has height

\[ h \leq 2 \log(n + 1). \]

**Proof.** (The book uses induction. Read carefully.)

**Intuition:**
- Merge red nodes into their black parents.
Theorem. A red-black tree with \( n \) keys has height
\[
h \leq 2 \log(n + 1).
\]

Proof. (The book uses induction. Read carefully.)

Intuition:
- Merge red nodes into their black parents.
- This process produces a tree in which each node has 2, 3, or 4 children.
- The 2-3-4 tree has uniform depth \( h' \) of leaves.
Proof (continued)

- We have $h' \geq h/2$, since at most half the leaves on any path are red.

- The number of leaves in each tree is $n + 1$
  \[ \Rightarrow n + 1 \geq 2^{h'} \]
  \[ \Rightarrow \lg(n + 1) \geq h' \geq h/2 \]
  \[ \Rightarrow h \leq 2 \lg(n + 1). \]
Query operations

**Corollary.** The queries `SEARCH`, `MIN`, `MAX`, `SUCCESSOR`, and `PREDECESSOR` all run in $O(lg n)$ time on a red-black tree with $n$ nodes.
Modifying operations

The operations \texttt{INSERT} and \texttt{DELETE} cause modifications to the red-black tree:

- the operation itself,
- color changes,
- restructuring the links of the tree via \texttt{“rotations”}.
Rotations

Rotations maintain the inorder ordering of keys:
- $a \in \alpha$, $b \in \beta$, $c \in \gamma \implies a \leq A \leq b \leq B \leq c$.

A rotation can be performed in $O(1)$ time.
Insertion into a red-black tree

**Idea:** Insert $x$ in tree. Color $x$ red. Only red-black property 3 might be violated. Move the violation up the tree by recoloring until it can be fixed with rotations and recoloring.

**Example:**

```
    7
   / \
  3   18
 /     / \
8 10   11 22
   /    /   \
  26   21   20
```
Insertion into a red-black tree

**IDEA:** Insert $x$ in tree. Color $x$ red. Only red-black property 3 might be violated. Move the violation up the tree by recoloring until it can be fixed with rotations and recoloring.

**Example:**
- Insert $x = 15$.
- Recolor, moving the violation up the tree.
Insertion into a red-black tree

**IDEA:** Insert $x$ in tree. Color $x$ red. Only red-black property 3 might be violated. Move the violation up the tree by recoloring until it can be fixed with rotations and recoloring.

**Example:**
- Insert $x = 15$.
- Recolor, moving the violation up the tree.
- RIGHT-ROTATE(18).
**Insertion into a red-black tree**

**Idea:** Insert $x$ in tree. Color $x$ red. Only red-black property 3 might be violated. Move the violation up the tree by recoloring until it can be fixed with rotations and recoloring.

**Example:**
- Insert $x = 15$.
- Recolor, moving the violation up the tree.
- **Right-Rotate(18).**
- **Left-Rotate(7) and recolor.**
Insertion into a red-black tree

**Idea:** Insert $x$ in tree. Color $x$ red. Only red-black property 3 might be violated. Move the violation up the tree by recoloring until it can be fixed with rotations and recoloring.

**Example:**
- Insert $x = 15$.
- Recolor, moving the violation up the tree.
- **Right-Rotate(18).**
- **Left-Rotate(7) and recolor.**
Pseudocode

RB-INSERT($T, x$)
   TREE-INSERT($T, x$)
   $color[x] \leftarrow $ RED \quad \triangleright \text{only RB property 3 can be violated}$

while $x \neq \text{root}[T]$ and $color[p[x]] = \text{RED}$
   do if $p[x] = \text{left}[p[p[x]]]$
      then $y \leftarrow \text{right}[p[p[x]]]$ \quad \triangleright y = \text{aunt/uncle of } x$
      if $color[y] = \text{RED}$
         then ⟨Case 1⟩
      else if $x = \text{right}[p[x]]$
         then ⟨Case 2⟩ \quad \triangleright \text{Case 2 falls into Case 3}
      ⟨Case 3⟩
   else ⟨“then” clause with “left” and “right” swapped⟩

$color[\text{root}[T]] \leftarrow $ BLACK
Graphical notation

Let \( \text{\textcircled{\textbullet}} \) denote a subtree with a black root.

All \( \text{\textcircled{\textbullet}} \)'s have the same black-height.
Case 1

(Or, children of $A$ are swapped.)

Push $C$’s black onto $A$ and $D$, and recurse, since $C$’s parent may be red.
Case 2

\textbf{LEFT-ROTATE}(A)

Transform to Case 3.
Case 3

**RIGHT-ROTATE**(C)

Done! No more violations of RB property 3 are possible.
Analysis

• Go up the tree performing Case 1, which only recolors nodes.

• If Case 2 or Case 3 occurs, perform 1 or 2 rotations, and terminate.

**Running time:** \( O(\log n) \) with \( O(1) \) rotations.

**RB-DELETE** — same asymptotic running time and number of rotations as **RB-INSERT** (see textbook).