6331 - Algorithms, Spring 2014, CSE, OSU
Lecture 5: Red-black trees

Instructor: Anastasios Sidiropoulos

January 17, 2014
Red-black trees

For every node $x$:

- $x.color$ : red or black
- $x.k$ : key
- $x.p$ : pointer to the parent of $x$
- $x.left$ : pointer to the left child of $x$
- $x.right$ : pointer to the right child of $x$
Properties of binary search trees

Let \( x \) be a node.

- For any node \( y \) in the left subtree of \( x \), we have \( y.key \leq x.key \).
- For any node \( y \) in the right subtree of \( x \), we have \( y.key \geq x.key \).
Properties of red-black trees

Properties:

1. Every node is either red or black.
2. The root is black.
3. Every leaf is black, and is represented by NIL.
4. If a node is red, then both its children are black.
5. For each node $x$, all paths from $x$ to a descendant leaf of $x$ contain the same number of black nodes.
For a node $x$, the *black-height* of $x$, denoted $bh(x)$ is the number of black nodes on any path from, but not including, $x$, to a descendant leaf of $x$. 
Lemma

A red-black tree with $n$ nodes has height $O(\log n)$. 
The height of a red-black tree

Lemma
A red-black tree with $n$ nodes has height $O(\log n)$.

Proof.
Any subtree rooted at a node $x$ contains at least $2^{bh(x)} - 1$ internal nodes.
The height of a red-black tree

Lemma
A red-black tree with $n$ nodes has height $O(\log n)$.

Proof.
Any subtree rooted at a node $x$ contains at least $2^{bh(x)} - 1$ internal nodes.
Proof by induction on $\text{height}(x)$. 

The height of a red-black tree

Lemma
A red-black tree with $n$ nodes has height $O(\log n)$.

Proof.
Any subtree rooted at a node $x$ contains at least $2^{bh(x)} - 1$ internal nodes.
Proof by induction on $\text{height}(x)$.
Each child of a node $x$ has black-height at least $bh(x) - 1$.

...
The height of a red-black tree

Lemma
A red-black tree with \( n \) nodes has height \( O(\log n) \).

Proof.
Any subtree rooted at a node \( x \) contains at least \( 2^{bh(x)} - 1 \) internal nodes.
Proof by induction on \( \text{height}(x) \).
Each child of a node \( x \) has black-height at least \( bh(x) - 1 \).

\[ \text{Let } h \text{ be the height of the tree. Then, } h \text{ is at most twice the black-height of the root.} \]
The height of a red-black tree

**Lemma**

A red-black tree with \( n \) nodes has height \( O(\log n) \).

**Proof.**

Any subtree rooted at a node \( x \) contains at least \( 2^{bh(x)} - 1 \) internal nodes.

Proof by induction on \( height(x) \).

Each child of a node \( x \) has black-height at least \( bh(x) - 1 \).

\[ \ldots \]

Let \( h \) be the height of the tree. Then, \( h \) is at most twice the black-height of the root.

\[ n \geq 2^{h/2} - 1. \]
The height of a red-black tree

Lemma
A red-black tree with n nodes has height $O(\log n)$.

Proof.
Any subtree rooted at a node $x$ contains at least $2^{bh(x)} - 1$ internal nodes.

Proof by induction on $height(x)$.
Each child of a node $x$ has black-height at least $bh(x) - 1$.

Let $h$ be the height of the tree. Then, $h$ is at most twice the black-height of the root.

\[ n \geq 2^{h/2} - 1. \]

\[ h = O(\log n). \]
Implications

The operations Search, Minimum, Maximum, Successor, and Predecessor take time $O(\log n)$. 
Implications

The operations Search, Minimum, Maximum, Successor, and Predecessor take time $O(\log n)$.

What about Insertion and Deletion?
Rotations

Left-Rotate\((T, x)\)
\[
\begin{align*}
y &= x.right \\
x.right &= y.left \\
\text{if } y.left &\neq NIL \\
\quad y.left.p &= x \\
y.p &= x.p \\
\text{if } x.p &= NIL \\
\quad T.root &= y \\
\text{elseif } x &= x.p.left \\
\quad x.p.left &= y \\
\text{else } x.p.right &= y \\
y.left &= x \\
x.p &= y
\end{align*}
\]
Rotations

Left-Rotate($T, x$)

Right-Rotate($T, y$)
Insertion

RB-Insert( \( T, z \) )
\[
\begin{align*}
y &= NIL \\
x &= T\.root
\end{align*}
\]
while \( x \neq NIL \)
\[
\begin{align*}
y &= x \\
    &\quad \text{if } z\.key < x\.key \\
    &\quad \quad x = x\.left \\
    &\quad \text{else } x = x\.right
\end{align*}
\]
\[
\begin{align*}
z\.p &= y \\
    &\quad \text{if } y = NIL \\
    &\quad \quad T\.root = z \\
    &\quad \text{elseif } z\.key < y\.key \\
    &\quad \quad y\.left = z \\
    &\quad \text{else } y\.right = z \\
    &\quad z\.left = NIL \\
    &\quad z\.right = NIL \\
    &\quad z\.color = RED
\end{align*}
\]
RB-Insert-Fixup( \( T, z \) )
Insertion

Does RB-Insert create a valid Red-black tree?
Insertion

Does RB-Insert create a valid Red-black tree?

What can go wrong?
Insertion

Does RB-Insert create a valid Red-black tree?

What can go wrong?

- If the parent of z is RED, then we have two consecutive RED nodes.
- If $T$ is empty, then after the insertion the root is RED.
Fixup

RB-Insert-Fixup(\(T, z\))
while \(z.p.color = RED\)
  if \(z.p = z.p.p.left\)
    \(y = z.p.p.right\)
    if \(y.color = RED\)
      \(z.p.color = BLACK\)
      \(y.color = BLACK\)
      \(z.p.p.color = RED\)
      \(z = z.p.p\)
  else
    if \(z = z.p.right\)
      \(z = z.p\)
      Left-Rotate(\(T, z\))
      \(z.p.color = BLACK\)
      \(z.p.p.color = RED\)
      Right-Rotate(\(T, z.p.p\))
    else (same with “left” and “right” exchanged)
    \(T.root.color = BLACK\)
Invariants of Fixup

(a) Node $z$ is red.
(b) If $z.p$ is the root, then $z.p$ is black.
(c) If the tree violates any of the properties, then it violates at most one of them, and the violation is either 2, or 4.
   - If it violates property 2, then $z$ is the root and is red.
   - If it violates property 4, then $z$ and $z.p$ are red, and no other node violates property 4.
Invariants of Fixup

Initialization?

(a) Node $z$ is red.
(b) If $z.p$ is the root, then $z.p$ is black.
(c) If the tree violates any of the properties, then it violates at most one of them, and the violation is either 2, or 4.
   - If it violates property 2, then $z$ is the root and is red.
   - If it violates property 4, then $z$ and $z.p$ are red, and no other node violates property 4.
Termination?

(a) Node $z$ is red.
(b) If $z.p$ is the root, then $z.p$ is black.
(c) If the tree violates any of the properties, then it violates at most one of them, and the violation is either 2, or 4.
   ▶ If it violates property 2, then $z$ is the root and is red.
   ▶ If it violates property 4, then $z$ and $z.p$ are red, and no other node violates property 4.
Assume w.l.o.g. that $z.p$ is a left child. The other case is symmetric.
Maintenance of invariants of Fixup

Case 1: z’s uncle y is red.
Maintenance of invariants of Fixup

Case 2: z’s uncle y is black and z is a right child.
Case 3: z’s uncle y is black and z is a left child.
Running time

The running time of RB-Insert-Fixup is $O(\log n)$. Therefore, the running time of RB-Insert is $O(\log n)$. 
Running time

The running time of RB-Insert-Fixup is $O(\log n)$. 
Running time

The running time of RB-Insert-Fixup is $O(\log n)$.

Therefore, the running time of RB-Insert is $O(\log n)$. 