Lecture 10: Balanced Binary Search Trees
CSCI 700 - Algorithms I

Andrew Rosenberg

October 13, 2009
Last Time

- Balanced Binary Search Trees – AVL Trees
Today

- More Balanced Binary Search Trees
  - Red-Black Trees
  - 2-3 Trees
  - B-Trees
AVL Trees are **Binary Search Trees**

- Maintain **near perfect** balance on insert and delete
- Rotation operations
- Require storage at each node of balance factor $bf \in \{-2, -1, 0, 1, 2\}$ or height $h \in \mathbb{Z}$
Red-Black Trees are Binary Search Trees

In addition to BST Properties, they also satisfy Red-Black Tree Properties (or RBT Properties)

1. Every node is either red or black. (1-bit storage)
2. The root is black
3. Nulls (below leaves) are black.
4. If a node is red, all of its children are black.
5. For each node, all paths from the node to descendent leaves contain the same number of black nodes.

We’ll call this black-height of a node $bh(T)$. 
Example of a Red-Black Tree

\[ h = 4 \]
Example of a Red-Black Tree

- Node 7: bh = 2
- Node 18: bh = 2
- Node 10: bh = 1
- Node 3: bh = 0
- Node 8: bh = 1
- Node 11: bh = 1
- Node 22: bh = 1
- Node 26: bh = 1
We can track the balance of the whole tree using only local information about the color of a node and its parent and children.

Color information is stored in a single bit.

**Persistent data structures** In AVL trees deletion may require up to $O(\log n)$ rotations. In R-B Trees it will require $O(1)$. Therefore to store rollback information requires $\log n$ times the space when implemented using an AVL tree.
Height of a Red-Black Tree

**Theorem**

A red-black tree with $n$ internal nodes has height at most $2 \log (n + 1)$.

- Show that a subtree, $T$, has at least $2^{bh(T)} - 1$ internal nodes.
- Induction on height of $T$.
- Base case: If $T$.height == 0, then $T$ is a leaf. Therefore $T$ contains 0 internal nodes. $2^{bh(T)} - 1 = 2^0 - 1 = 0$. 
A red-black tree with $n$ internal nodes has height at most $2 \log (n + 1)$.

- Show that a subtree, $T$, has at least $2^{bh(T)} - 1$ internal nodes.
- Induction on height of $T$.
- Inductive step: $T$ has positive height, and is an internal node with 2 children. Each child has a black-height of $bh(T)$ (if the child is red) or $bh(T) - 1$ (if the child is black).
# Theorem

A red-black tree with $n$ internal nodes has height at most $2 \log (n + 1)$.

- Show that a subtree, $T$, has at least $2^{bh(T)} - 1$ internal nodes.
- Induction on height of $T$.
- Inductive step: $T$ has positive height, and is an internal node with 2 children. Each child has a black-height of $bh(T)$ (if the child is red) or $bh(T) - 1$ (if the child is black).
- By induction, each child has at least $2^{bh(T)-1} - 1$ internal nodes.
Theorem

A red-black tree with \( n \) internal nodes has height at most \( 2 \log (n + 1) \).

- Show that a subtree, \( T \), has at least \( 2^{bh(T)} - 1 \) internal nodes.
- Induction on height of \( T \).
- Inductive step: \( T \) has positive height, and is an internal node with 2 children. Each child has a black-height of \( bh(T) \) (if the child is red) or \( bh(T) - 1 \) (if the child is black).
- By induction, each child has at least \( 2^{bh(T) - 1} - 1 \) internal nodes.
- Therefore, \( T \) has at least \( (2^{bh(T) - 1} - 1) + (2^{bh(T) - 1} - 1) + 1 = 2^{bh(T)} - 1 \) internal nodes.
A red-black tree with $n$ internal nodes has height at most $2 \log (n + 1)$.

- Show that a subtree, $T$, has at least $2^{bh(T)} - 1$ internal nodes.
- Let $h$ be the height of $T$. 
A red-black tree with $n$ internal nodes has height at most $2 \log (n + 1)$.

- Show that a subtree, $T$, has at least $2^{bh(T)} - 1$ internal nodes.
- Let $h$ be the height of $T$.
- At least half of the items on any path from $T$ to a leaf must be black (Property 4).
Theorem

A red-black tree with \( n \) internal nodes has height at most \( 2 \log(n + 1) \).

- Show that a subtree, \( T \), has at least \( 2^{bh(T)} - 1 \) internal nodes.
- Let \( h \) be the height of \( T \).
- At least half of the items on any path from \( T \) to a leaf must be black (Property 4).
- Thus \( bh(T) \geq h/2 \)
Theorem

A red-black tree with \( n \) internal nodes has height at most

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

- Show that a subtree, \( T \), has at least \( 2^{bh(T)} - 1 \) internal nodes.
- Let \( h \) be the height of \( T \).
- At least half of the items on any path from \( T \) to a leaf must be black (Property 4).
- Thus \( bh(T) \geq h/2 \)
- So....\( n \geq 2^{h/2} - 1 \)
- \( n + 1 \geq 2^{h/2}. \)
- \( \log (n + 1) \geq h/2 \)
- \( 2 \log (n + 1) \geq h \)
Theorem

A red-black tree with $n$ internal nodes has height at most $2 \log (n + 1)$.

- Show that a subtree, $T$, has at least $2^{bh(T)} - 1$ internal nodes.
- Let $h$ be the height of $T$.
- At least half of the items on any path from $T$ to a leaf must be black (Property 4).
- Thus $bh(T) \geq h/2$
- So....$n \geq 2^{h/2} - 1$
- $n + 1 \geq 2^{h/2}$.
- $\log (n + 1) \geq h/2$
- $2 \log (n + 1) \geq h$
- Runtime in $O(h) = O(\log n)$
Search, Min, Max, Successor, and Predecessor all run in $O(h) = O(\log n)$ time, on a red-black tree with $n$ nodes.
- The operation itself is unchanged.
- May require color changes.
- May require rotations to maintain Property 4 (If a node is red, it’s children are black).
Rotation Review

- Rotation maintains the **BST property**.
- Rotations take $O(1)$. 
Red-Black Tree Insertion

- Idea: Insert $x$ into the tree $T$.
- Color $x$ red. – Thus $bh(T)$ is maintained for all subtrees that $x$ is a member of.
- However, Property 4 – If a node is red, all of its children are black – may not hold
Red-Black Tree Insertion

- Idea: Insert $x$ into the tree $T$.
- Color $x$ red. – Thus $bh(T)$ is maintained for all subtrees that $x$ is a member of.
- However, Property 4 – If a node is red, all of its children are black – may not hold
- Recolor and rotate until the RBT Property is restored.
Red-Black Insert Example

Example:
Example:
- Insert $x = 15$.
- Recolor, moving the violation up the tree.
Example:
• Insert $x = 15$.
• Recolor, moving the violation up the tree.
• Right-Rotate(18).
Example:

- Insert $x = 15$.
- Recolor, moving the violation up the tree.
- **RIGHT-ROTATE**(18).
- **LEFT-ROTATE**(7) and recolor.
Example:
• Insert $x = 15$.
• Recolor, moving the violation up the tree.
• `RIGHT-ROTATE(18)`.
• `LEFT-ROTATE(7)` and recolor.
Red-Black Insert Pseudocode

\[
\text{RB-Insert}(T, x) \\
\begin{align*}
\text{Insert}(T, x) \\
x.\text{Color} &\leftarrow \text{RED} \\
\text{while } x \neq T \text{ and } x.\text{parent}.\text{color} = \text{RED} \text{ do} \\
&\quad \text{if IsLeftChild}(x.\text{parent}) \text{ then} \\
&\quad \quad T.\text{color} \leftarrow \text{BLACK} \\
&\quad \quad y \leftarrow x.\text{parent}.\text{parent}.\text{right} \\
&\quad \quad \text{if } y.\text{color} = \text{RED} \text{ then} \\
&\quad \quad \quad \text{Case 1} \\
&\quad \quad \text{else} \\
&\quad \quad \quad \text{if IsRightChild}(x) \text{ then} \\
&\quad \quad \quad \quad \text{Case 2} \\
&\quad \quad \quad \text{end if} \\
&\quad \quad \quad \text{Case 3} \\
&\quad \text{end if} \\
&\quad \text{else} \\
&\quad \quad \text{swap left and right} \\
&\quad \text{end if} \\
\text{end while}
\end{align*}
\]
Red-Black Case 1

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

Recolor

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

\textbf{LEFT-ROTATE}(A)

Transform to Case 3.
Red-Black Case 3

\[ \text{RIGHT-ROTATE}(C) \]

Done! No more violations of RB property 3 are possible.
Red-Black Insert Analysis

- First traverse up the tree recoloring.
- If Case 2 or 3 occurs, rotate once or twice.
- First traverse up the tree recoloring.
- If Case 2 or 3 occurs, rotate once or twice.
- Runtime: $O(h) = O(\log n)$ and $O(1)$ rotations.

Red-Black Delete has the same running time and number of rotations as insert. Refer to the text for this.
2-3 Trees are Search Trees where each node can have 1 or 2 keys and 2 or 3 children.
Example 2-3 Trees
Each leaf has the same depth and contains 1 or 2 keys.

Each interior node:
- contains 1 key and has 2 children (2-node) or
- contains 2 keys and has 3 children (3-node)

In a 2-node $T$ with key $a$
- each key in its left subtree has key $\leq a$
- each key in its right subtree has key $> a$

In a 3-node $T$ with keys $a$ and $b$
- each key in its left subtree has key $\leq a$
- each key in its middle subtree has $a < key \leq b$
- each key in its right subtree has key $> b$
What is the height, \( h \) of a tree containing \( n \) values?
What is the height, $h$ of a tree containing $n$ values?
Each internal node can have up to 3 children.
There are $3^{h-1}$ leaves.
What is the height, \( h \) of a tree containing \( n \) values?

- Each internal node can have up to 3 children.
- There are \( 3^{h-1} \) leaves.
- Each leaf can have up to 2 values

\[
n \leq 2 \times 3^{h-1}
\]

\[
\log_6 n - 1 \leq h
\]
What is the height, $h$ of a tree containing $n$ values?
Each internal node has at least to 2 children.
There are at least $2^{h-1}$ leaves.
What is the height, $h$ of a tree containing $n$ values?

- Each internal node has at least 2 children.
- There are at least $2^{h-1}$ leaves.
- Each leaf has at least 1 value
- $n \geq 1 \times 2^{h-1}$
- $\log_2 n - 1 \geq h$
What is the height, \( h \) of a tree containing \( n \) values?

- Each internal node has at least 2 children.
- There are at least \( 2^{h-1} \) leaves.
- Each leaf has at least 1 value

\[
n \geq 1 \times 2^{h-1} \\
\log_2 n - 1 \geq h \\
\log_2 n - 1 \geq h \geq \log_6 n - 1 \\
h = \Theta(\log n)
\]
2-3 Tree Insert

- Find the leaf $l$ to insert $x$ as in BST Insert.
- If $l$ has 3 keys, move the middle key of $l$ up to its parent, $p$, and split $l$ into 2 leaves.
Find the leaf $l$ to insert $x$ as in BST Insert.

- If $l$ has 3 keys, move the middle key of $l$ up to its parent, $p$, and split $l$ into 2 leaves.
- While $p$ has 3 keys (and 4 children)
  - Split $p$ into $p_1$ and $p_2$.
  - Register $p_1$ and $p_2$ instead of just $p$ as children of the parent of $p$.
  - $p \leftarrow p.parent$

- If the root was split, insert a new root to hold $p_1$ and $p_2$.
2-3 Insert example
2-3 Insert example

Insert(26)
2-3 Insert example
What is the runtime of 2-3 Insert?
What is the runtime of 2-3 Insert?

\( \Theta(\log n) \)
2-3 Delete

- Find the node $T$ containing $x$.
- If $T$ is a 1-leaf, remove it.
- Else, replace $x$ by $\text{Successor}(x)$
- This might make a node $w$ have no keys (illegal).
- While $w$ is illegal:
  - If $w$ has a sibling $w'$ with 2 keys, steal one of the keys.
  - If $w$ has a sibling $w'$ with 1 key, merge them.

- If $w$ is the root, delete $w$, and let $\text{root} \leftarrow w.\text{child}$.
Find the node $T$ containing $x$.

If $T$ is a 1-leaf, remove it.

Else, replace $x$ by $\text{Successor}(x)$

This might make a node $w$ have no keys (illegal).

While $w$ is illegal:

- If $w$ has a sibling $w'$ with 2 keys, steal one of the keys.
  - Let $s$ be the key in the parent $u$ of $w$ and $w'$ separating them.
    move $s$ from $u$ to $w$, replace $s$ in $u$ by nearest key in $w'$.
- If $w$ has a sibling $w'$ with 1 key, merge them.

if $w$ is the root, delete $w$, and let $\text{root} \leftarrow w.\text{child}$. 
Find the node $T$ containing $x$.

If $T$ is a 1-leaf, remove it.

Else, replace $x$ by $\text{Successor}(x)$

This might make a node $w$ have no keys (illegal).

While $w$ is illegal:

- If $w$ has a sibling $w'$ with 2 keys, steal one of the keys.
  - Let $s$ be the key in the parent $u$ of $w$ and $w'$ separating them.
    - move $s$ from $u$ to $w$, replace $s$ in $u$ by nearest key in $w'$.
  - If $w$ has a sibling $w'$ with 1 key, merge them.
    - Merge $w$ and $w'$ to a new 3-node $w''$ with keys $s$ and that of $w'$.
    - $w \leftarrow parent(w)$ – may have become illegal.

- if $w$ is the root, delete $w$, and let $root \leftarrow w.child$. 
2-3 Delete examples

The left most key has just been deleted.
- **B-trees** are a generalization of 2-3 trees where each node has between $B$ and $2B-1$ children.
- Essentially an $(a,b)$-tree, where $b = 2a-1$. 
B-Trees

- **B-trees** are a generalization of 2-3 trees where each node has between B and 2B-1 children.
- Essentially an (a,b)-tree, where \( b = 2a - 1 \).
- Disk based storage. Databases.
- If each page can hold 2B records, this is an efficient use of disk reads.
Bye

Next time (10/15)
- Greedy Algorithms

For Next Class
- Read 16.1, 16.2, 16.3