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

Andrew Rosenberg

October 12, 2010
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**
  \[ bf \in \{-1, 0, 1\} \text{ or height } h \in \mathbb{Z} \]
Red-Black Trees

- Red-Black Trees are **Binary Search Trees**
- In addition to **BST Properties**, the 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
  - Child 3
    - bh = 1
    - NIL
    - NIL
  - Child 18
    - bh = 2
    - Child 10
      - bh = 1
      - NIL
      - NIL
    - Child 22
      - NIL
      - NIL
  - Child 8
    - bh = 0
    - NIL
    - NIL
  - Child 11
    - NIL
    - NIL
    - NIL
    - NIL
  - Child 26
    - NIL
    - NIL
    - NIL
Who cares about Red-Black Trees

- 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.
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. \( 2^{bh(T)} - 1 = 2^0 - 1 = 0 \).
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).
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.
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.
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 \( T \).
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).
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... \(n \geq 2^{h/2} - 1\)
- \(n + 1 \geq 2^{h/2}\).
- \(\log (n + 1) \geq h/2\)
- \(2 \log (n + 1) \geq h\)
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.
- 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 $O(1)$.
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.
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.
- $\text{RIGHT-ROTATE}(18)$.
- $\text{LEFT-ROTATE}(7)$ 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

```
RB-Insert(T,x)

Insert(T,x)
x.Color ← RED
while x ≠ T and x.parent.color = RED do
    if IsLeftChild(x.parent) then
        T.color ← BLACK
        y ← x.parent.parent.right
        if y.color = RED then
            Case 1
        else
            if IsRightChild(x) then
                Case 2
            end if
            Case 3
        end if
    else
        swap left and right
    end if
end while
```
Red-Black Case 1

Recolor

(Or, children of A are swapped.)

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

**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 two 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 to 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
2-3 Insert example
What is the runtime of 2-3 Insert?
2-3 Insert Runtime

- What is the runtime of 2-3 Insert?
  - $\Theta(\log n)$
2-3 Delete

- Find the node $T$ containing $x$.
- If $T$ 2-leaf, delete $x$ from the leaf.
- 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 $\text{root} \leftarrow w\.child$. 
2-3 Delete

- Find the node $T$ containing $x$.
- If $T$ 2-leaf, delete $x$ from the leaf.
- 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 $\text{root} \leftarrow w.\text{child}$. 
2-3 Delete

- Find the node $T$ containing $x$.
- If $T$ 2-leaf, delete $x$ from the leaf.
- 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 $w'$.
    - $w \leftarrow \text{parent}(w)$ – may have become illegal.
  - if $w$ is the root, delete $w$, and let $\text{root} \leftarrow w.\text{child}$. 
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

- 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.
Next time (10/15)
  - Greedy Algorithms
For Next Class
  - Read 16.1, 16.2, 16.3