Lecture 02/22/17

Lecturer: Xiaodi Wu

Reading: Chapter 13.2
Binary Tree Operations

- Tree-Search: $O(h) = O(\log(n))$.
- Tree-Insert, Tree-Delete: $O(h) = O(\log(n))$.
- Search: won’t lead to any violation; Insert, Delete: potentially lead to violations of red-black trees.
Red-Black Trees: Rotation

Binary Tree Operations

- Tree-Search: $O(h) = O(\log(n))$.
- Tree-Insert, Tree-Delete: $O(h) = O(\log(n))$.
- Search: won’t lead to any violation; Insert, Delete: potentially lead to violations of red-black trees.

Tools to Deal with Violations

- Need to somehow change the tree structure.
- Fix violations without having new violations.
- **Rotations:** the first tool to make the chance, keeping the **binary search tree** property.
Red-Black Trees: Rotations

Figure 13.2 The rotation operations on a binary search tree. The operation \textsc{Left-Rotate}(T, x) transforms the configuration of the two nodes on the right into the configuration on the left by changing a constant number of pointers. The inverse operation \textsc{Right-Rotate}(T, y) transforms the configuration on the left into the configuration on the right. The letters $\alpha$, $\beta$, and $\gamma$ represent arbitrary subtrees. A rotation operation preserves the binary-search-tree property: the keys in $\alpha$ precede $x$, which precedes the keys in $\gamma$, which precede the key in $\beta$.

LEFT-ROTATE(T, x)

\begin{itemize}
\item $\alpha$
\item $\beta$
\item $x$
\item $y$
\item $\gamma$
\end{itemize}

RIGHT-ROTATE(T, y)

\begin{itemize}
\item $\alpha$
\item $\beta$
\item $y$
\item $x$
\item $\gamma$
\end{itemize}

Inorder: $\alpha, x, \beta, y, \gamma$

Exercises

13.2-1 Write pseudocode for \textsc{Right-Rotate}.

13.2-2 Argue that in every $n$-node binary search tree, there are exactly $n/2$ possible rotations.
Red-Black Trees: Left-Rotation

**LEFT-ROTATE**(*T*, *x*)
1.  \( y = x.right \)  
   \[\text{\}// \text{set } y\]
2.  \( x.right = y.left \)  
   \[\text{\}// \text{turn } y\text{'}s \text{left} \text{subtree} \text{into} \text{ } x\text{'}s \text{right} \text{subtree}\]
3.  \textbf{if} \( y.left \neq T.nil \)
4.     \( y.left.p = x \)
5.  \( y.p = x.p \)  
   \[\text{\}// \text{link } x\text{'}s \text{parent} \text{to} \text{ } y\]
6.  \textbf{if} \( x.p == T.nil \)
7.     \( T.root = y \)
8.  \textbf{elseif} \( x == x.p.left \)
9.     \( x.p.left = y \)
10.  \textbf{else} \( x.p.right = y \)
11.  \( y.left = x \)  
   \[\text{\}// \text{put } x \text{on} \text{ } y\text{'}s \text{left}\]
12.  \( x.p = y \)
Red-Black Trees: Left-Rotation

**LEFT-ROTATE** \((T, x)\)

1. \(y = x.right\) \quad // set \(y\)
2. \(x.right = y.left\) \quad // turn \(y\)'s left subtree into \(x\)'s right subtree
3. **if** \(y.left \neq T.nil\)
   4. \(y.left.p = x\)
5. \(y.p = x.p\) \quad // link \(x\)'s parent to \(y\)
6. **if** \(x.p == T.nil\)
   7. \(T.root = y\)
8. **elseif** \(x == x.p.left\)
   9. \(x.p.left = y\)
10. **else** \(x.p.right = y\)
11. \(y.left = x\) \quad // put \(x\) on \(y\)'s left
12. \(x.p = y\)

**Time complexity:** \(O(1)\). Right Rotation?
Red-Black Trees: Left-Rotation Example

Figure 13.3

An example of how the procedure $\text{LEFT-ROTATE}(T, x)$ modifies a binary search tree. Inorder tree walks of the input tree and the modified tree produce the same listing of key values.

13.2-3
Let $a$, $b$, and $c$ be arbitrary nodes in subtrees $\alpha$, $\beta$, and $\gamma$ respectively, in the left tree of Figure 13.2. How do the depths of $a$, $b$, and $c$ change when a left rotation is performed on node $x$ in the figure?

13.2-4
Show that any arbitrary $n$-node binary search tree can be transformed into any other arbitrary $n$-node binary search tree using $O(n)$ rotations. (Hint: First show that at most $n/\ln 2$ right rotations suffice to transform the tree into a right-going chain.)

13.2-5
We say that a binary search tree $T_1$ can be right-converted to binary search tree $T_2$ if it is possible to obtain $T_2$ from $T_1$ via a series of calls to $\text{RIGHT-ROTATE}$. Give an example of two trees $T_1$ and $T_2$ such that $T_1$ cannot be right-converted to $T_2$.

Then, show that if a tree $T_1$ can be right-converted to $T_2$, it can be right-converted using $O(n^2)$ calls to $\text{RIGHT-ROTATE}$. 

"
Ex 13.2-3
Let $a, b, c$ be arbitrary nodes in subtrees $\alpha, \beta, \gamma$. How do the depths of $a, b, c$ change when a left/right rotation happens?