COMP251: Red-black trees

Jérôme Waldispühl
School of Computer Science
McGill University

Based on (Cormen et al., 2002)

Based on slides from D. Plaisted (UNC)
The running time of insertions in BST trees with n nodes is:

• $\Omega(\log(n))$
• $\Theta(\log(n))$
• $O(\log(n))$
• $O(n)$
• $\Omega(n)$
Which assertion(s) are true?

• Rotations preserve BST properties ✔
• Rotations preserve AVL tree properties ✗
• AVL properties can be restored using rotations ✔
• In the worst case, a rotation has a O( log n ) running time ✗
How should we modify BST sort to sort numbers in decreasing order?

- Use post-order traversal
- Reverse the order of recursive calls in in-order traversal
- Use an AVL tree instead of a BST
**Definition:** An AVL tree is a BST such that the heights of the two child subtrees of any node differ by at most one.

- AVL trees are self-balanced binary search trees.
- Insert, Delete & Search take $O(\log n)$ in average and worst cases.
Recap lecture 3

Rotations preserve the BST property.

Proof: elements in B are $\geq x$ and $\leq y$...
Insert in AVL trees

Right rotation at 27

36

12

57

12

57

8

43

20

43

15

27

15

27
Insert in AVL trees

Left rotation at 43

RotateLeft(T, 43)
Insert in AVL trees

Right rotation at 57

\[ \text{RotateRight}(T, 57) \]
Red-black trees: Overview

• Red-black trees are a variation of binary search trees to ensure that the tree is balanced.
  – Height is $O(lg n)$, where $n$ is the number of nodes.

• Operations take $O(lg n)$ time in the worst case.

• Invented by R. Bayer (1972).

Red-black Tree

• Binary search tree + 1 bit per node: the attribute color, which is either red or black.

• All other attributes of BSTs are inherited:
  – key, left, right, and parent.

• All empty trees (leaves) are colored black.
  – Note: We can use a single sentinel, nil, for all the leaves of red-black tree $T$, with $color[nil] = \text{black}$. The root’s parent is also $nil[T]$. 
Red-black Properties

1. Every node is either red or black.
2. The root is black.
3. Every leaf \((nil)\) is black.
4. If a node is red, then its children are black (i.e. no 2 consecutive red nodes).
5. For each node, all paths from the node to descendant leaves contain the same number of black nodes (i.e. same black height).
Red-black Tree – Example

Note: every internal node has two children, even though nil leaves are not usually shown.
Height of a Red-black Tree

• Height of a node:
  – \( h(x) = \) number of edges in the longest path to a leaf.

• Black-height of a node \( x \), \( bh(x) \):
  – \( bh(x) = \) number of black nodes (including \( nil[T] \)) on the path from \( x \) to leaf, not counting \( x \).

• Black-height of a red-black tree is the black-height of its root.
  – By Property 5, black height is well defined.
Height of a Red-black Tree

- **Height** $h(x)$:
  #edges in a longest path to a leaf.

- **Black-height** $bh(x)$:
  # black nodes on path from $x$ to leaf, *not counting* $x$.

- **Property**: $bh(x) \leq h(x) \leq 2 \cdot bh(x)$
Bound on RB Tree Height

**Lemma 1:** Any node $x$ with height $h(x)$ has a black-height $bh(x) \geq h(x)/2$.

**Proof:** By property 4, $\leq h/2$ nodes on the path from the node to a leaf are red. Hence $\geq h/2$ are black. ■
**Bound on RB Tree Height**

**Lemma 2:** The subtree rooted at any node \( x \) contains \( \geq 2^{bh(x)} - 1 \) internal nodes.

**Proof:** By induction on height of \( x \).

- **Base Case:** Height \( h(x) = 0 \) \( \Rightarrow \) \( x \) is a leaf \( \Rightarrow bh(x) = 0 \). Subtree has \( \geq 2^0 - 1 = 0 \) nodes.

- **Induction Step:**
  - Each child of \( x \) has height \( h(x) - 1 \) and black-height either \( b(x) \) (child is red) or \( b(x) - 1 \) (child is black).
  - By ind. hyp., each child has \( \geq 2^{bh(x) - 1} - 1 \) internal nodes.
  - Subtree rooted at \( x \) has \( \geq 2 \cdot (2^{bh(x) - 1} - 1) + 1 = 2^{bh(x)} - 1 \) internal nodes. (The +1 is for \( x \) itself) \( \blacksquare \)
Bound on RB Tree Height

**Lemma 1:** Any node $x$ with height $h(x)$ has a black-height $bh(x) \geq h(x)/2$.

**Lemma 2:** The subtree rooted at any node $x$ has $\geq 2^{bh(x)} - 1$ internal nodes.

**Lemma 3:** A red-black tree with $n$ internal nodes has height at most $2 \lg(n+1)$.

**Proof:**
- By lemma 2, $n \geq 2^{bh} - 1$,
- By lemma 1, $bh \geq h/2$, thus $n \geq 2^{h/2} - 1$.
- $\Rightarrow h \leq 2 \lg(n + 1)$. 

Insertion in RB Trees

• Insertion must preserve all red-black properties.
• Should an inserted node be colored Red? Black?
• Basic steps:
  – Use BST Tree-Insert to insert a node \( x \) into \( T \).
    • Procedure \textbf{RB-Insert}(x).
  – Color the node \( x \) red.
  – Fix the new tree by (1) re-coloring nodes, and (2) performing rotation to preserve RB tree property.
    • Procedure \textbf{RB-Insert-Fixup}.
InserRon

\[
\begin{align*}
\text{RB-Insert}(T, z) & \\
1. & y \leftarrow \text{nil}[T] \\
2. & x \leftarrow \text{root}[T] \\
3. & \text{while } x \neq \text{nil}[T] \\
4. & \quad \text{do } y \leftarrow x \\
5. & \quad \quad \text{if } \text{key}[z] < \text{key}[x] \\
6. & \quad \quad \quad \text{then } x \leftarrow \text{left}[x] \\
7. & \quad \quad \quad \text{else } x \leftarrow \text{right}[x] \\
8. & \quad p[z] \leftarrow y \\
9. & \quad \text{if } y = \text{nil}[T] \\
10. & \quad \quad \text{then } \text{root}[T] \leftarrow z \\
11. & \quad \quad \text{else if } \text{key}[z] < \text{key}[y] \\
12. & \quad \quad \quad \text{then } \text{left}[y] \leftarrow z \\
13. & \quad \quad \quad \text{else } \text{right}[y] \leftarrow z \\
\end{align*}
\]

\text{RB-Insert}(T, z) \text{ Contd.}

14. \quad \text{left}[z] \leftarrow \text{nil}[T] \\
15. \quad \text{right}[z] \leftarrow \text{nil}[T] \\
16. \quad \text{color}[z] \leftarrow \text{RED} \\
17. \quad \text{RB-Insert-Fixup } (T, z)

\text{Regular BST insert + color assignment + fixup.}
Insert RB Tree – Example
Insert RB Tree – Example

Insert(T, 15)
Insert RB Tree – Example

Recolor 10, 8 & 11
Insert RB Tree – Example

Right rotate at 18
Insert RB Tree – Example

Right rotate at 18 (parent & child with conflict are aligned)
Insert RB Tree – Example

Left rotate at 7
Insert RB Tree – Example

Left rotate at 7
Insert RB Tree – Example

Recolor 10 & 7 (root must be black!)
RB-Insert-Fixup \((T, z)\)

1. while \(\text{color}[p[z]] = \text{RED}\) 
2. do if \(p[z] = \text{left}[p[p[z]]]\) 
3. then \(y \leftarrow \text{right}[p[p[z]]]\) 
4. if \(\text{color}[y] = \text{RED}\) 
5. then \(\text{color}[p[z]] \leftarrow \text{BLACK} \quad \text{// Case 1}\) 
6. \(\text{color}[y] \leftarrow \text{BLACK} \quad \text{// Case 1}\) 
7. \(\text{color}[p[p[z]]] \leftarrow \text{RED} \quad \text{// Case 1}\) 
8. \(z \leftarrow p[p[z]] \quad \text{// Case 1}\)
RB-Insert-Fixup\((T, z)\) (Contd.)

9. \(\text{else if } z = \text{right}[p[z]] \quad \text{\scriptsize // color}[y] \neq \text{RED}\)

10. \(\text{then } z \leftarrow p[z] \quad \text{\scriptsize // Case 2}\)

11. \(\text{LEFT-ROTATE}(T, z) \quad \text{\scriptsize // Case 2}\)

12. \(\text{color}[p[z]] \leftarrow \text{BLACK} \quad \text{\scriptsize // Case 3}\)

13. \(\text{color}[p[p[z]]] \leftarrow \text{RED} \quad \text{\scriptsize // Case 3}\)

14. \(\text{RIGHT-ROTATE}(T, p[p[z]]) \quad \text{\scriptsize // Case 3}\)

15. \(\text{else (if } p[z] = \text{right}[p[p[z]]])\) (same as 10-14)

16. \(\text{with “right” and “left” exchanged}\)

17. \(\text{color}[\text{root}[T]] \leftarrow \text{BLACK}\)
Case 1 – uncle y is red

\[ p[p[z]] \] (z’s grandparent) must be black, since z and \( p[z] \) are both red and there are no other violations of property 4.

- Make \( p[z] \) and y black \( \Rightarrow \) now z and \( p[z] \) are not both red. But property 5 might now be violated.
- Make \( p[p[z]] \) red \( \Rightarrow \) restores property 5.
- The next iteration has \( p[p[z]] \) as the new z (i.e., z moves up 2 levels).

z is a right child here. Similar steps if z is a left child.
Case 2 – y is black, z is a right child

- Left rotate around \( p[z] \), \( p[z] \) and \( z \) switch roles \( \Rightarrow \) now \( z \) is a left child, and both \( z \) and \( p[z] \) are red.
- Takes us immediately to case 3.
Case 3 – $y$ is black, $z$ is a left child

- Make $p[z]$ black and $p[p[z]]$ red.
- Then right rotate right on $p[p[z]]$ (in order to maintain property 4).
- No longer have 2 reds in a row.
- $p[z]$ is now black $\Rightarrow$ no more iterations.
Algorithm Analysis

• $O(lg\ n)$ time to get through RB-Insert up to the call of RB-Insert-Fixup.

• Within RB-Insert-Fixup:
  – Each iteration takes $O(1)$ time.
  – Each iteration but the last moves $z$ up 2 levels.
  – $O(lg\ n)$ levels $\Rightarrow O(lg\ n)$ time.
  – Thus, insertion in a red-black tree takes $O(lg\ n)$ time.
  – Note: there are at most 2 rotations overall.
Correctness

Loop invariant:

• At the start of each iteration of the `while` loop,
  
  – $z$ is red.

  – There is at most one red-black violation:
    
    • Property 2: $z$ is a red root, or
    
    • Property 4: $z$ and $p[z]$ are both red.
Correctness – Contd.

- **Initialization**: ✓
- **Termination**: The loop terminates only if $p[z]$ is black. Hence, property 4 is OK. The last line ensures property 2 always holds.
- **Maintenance**: We drop out when $z$ is the root (since then $p[z]$ is sentinel $nil[T]$, which is black). When we start the loop body, the only violation is of property 4.
  - There are 6 cases, 3 of which are symmetric to the other 3. We consider cases in which $p[z]$ is a left child.
  - See cases 1, 2, and 3 described above.
Further Readings


See Chapter 13 for the complete proofs & deletion