Computer Science 331

Introduction to Red-Black Trees

Mike Jacobson

Department of Computer Science
University of Calgary

Lecture #14

Outline

Definition of a Red-Black Tree

A **red-black tree** is a binary tree that can be used to implement the “Dictionary” ADT (also “SortedSet” and “SortedMap” interfaces from the JCF)

- **Internal Nodes** are used to store elements of a dictionary.
- **Leaves** are called “NIL nodes” and do not store elements of the set.
- Every internal node has two children (either, or both, of which might be leaves).
- The smallest red-black tree has size one (single NIL node).
- If the leaves (NIL nodes) of a red-black tree are removed then the resulting tree is a binary search tree.

Red-Black Properties

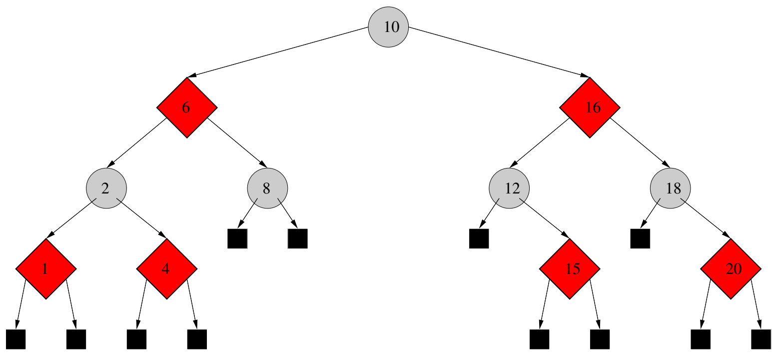
A binary search tree is a *red-black* tree if it satisfies the following:

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 both its children are black.
5. For each node, all paths from the node to descendant leaves contain the same number of black nodes.

Why these are useful:

- height is in $\Theta(\log n)$ in the worst case (tree with n internal nodes)
- worst case complexity of search, insert, delete are in $\Theta(\log n)$

Example



- "Black" internal nodes are drawn as circles
- "Red" nodes are drawn as diamonds
- NIL nodes (leaves) are drawn as black squares

Implementation Details

Example: Figure 13.1 on page 275 of the Cormen, Leiserson, Rivest, and Stein book.

- The color of a node can be represented by a Boolean value (eg, true=black, false=red), so that only one bit is needed to store the color of a node
- To save space and simplify programming, a single sentinel can replace all NIL nodes.
- The "parent" of the root node is pointed to the sentinel as well.
- An "empty" tree contains one single NIL node (the sentinel)

Black-Height of a Node

The **black-height** of a node x, denoted $\text{bh}(x)$, is the number of black nodes on any path from, but not including, a node x down to a leaf.

Example: In the previous red-black tree,

- The black-height of the node with label 2 is:
- The black-height of the node with label 4 is:
- The black-height of the node with label 6 is:
- The black-height of the node with label 8 is:
- The black-height of the node with label 10 is:

Note: Red-Black Property #5 implies that $\text{bh}(x)$ is well-defined for each node x.

The Main Theorem

Theorem 1

If T is a red-black tree with n nodes then the height of T is at most $2\log_2(n+1)$.

Outline of proof:

- prove a *lower bound* on tree size in terms of black-height
- prove an *upper bound* on height in terms of black-height of the tree
- combine to prove main theorem

Bounding Size Using Black-Height

Lemma 2

For each node x, the subtree with root x includes at least $2^{bh(x)} - 1$ nodes.

Method of Proof: mathematical induction on height of the subtree with root x (using the strong form of mathematical induction)

- Base case: prove that the claim holds for subtrees of height 0
- Inductive step: prove, for all $h \geq 0$, that if the lemma is true for all subtrees with height at most $h - 1$ then it also holds for all subtrees with height h.

Base Case ($h = 0$)

Inductive Step

Let h be an integer such that $h \geq 0$.

Inductive Hypothesis: Suppose the claimed result holds for every node y such that the height of the tree with root y is *less than* h.

Let x be a node such that the height of the tree T_x is h.

Let n be the number of nodes of T_x.

Required to Show: $n \geq 2^{\text{bh}(x)} - 1$ holds for T_x, assuming the inductive hypothesis.

Notation for Inductive Step

b	Black-height of x
b_L	Black-height of left child of x
b_R	Black-height of right child of x
T_x	Subtree with root x
h	Height of T_x
h_L	Height of left subtree of T_x
h_R	Height of right subtree of T_x
n	Size of T_x
n_L	Size of left subtree of T_x
n_R	Size of right subtree of T_x

Useful Properties Involving Size and Height

$n = n_L + n_R + 1$. The n nodes of T_x are:

- the n_L nodes of the left subtree of T_x
- the n_R nodes of the right subtree of T_x
- one more node — the root x of T_x

$h = 1 + \max(h_L, h_R)$, so $h_L \leq h - 1$ and $h_R \leq h - 1$

- height of any tree (including T_x) is the maximum length of any path from the root to any leaf
- it follows by this definition that $h = 1 + \max(h_L, h_R)$
- the remaining inequalities are now easily established

Useful Property Involving Black-Height

$b_L \geq b - 1$ and $b_R \geq b - 1$.

Case 1: x has color red

- both children of x have color black (Red-Black Property #4)
- Red-Black Property #5 implies that $b_L = b_R = b - 1$.

Case 2: x has color black.

- children of x could each be either red or black
- $b_L \geq b - 1$, because by the definition of "black-height"

$$b_L = \begin{cases} b & \text{if the left child of } x \text{ is red} \\ b - 1 & \text{if the left child of } x \text{ is black.} \end{cases}$$

- an analogous argument shows that $b_R \geq b - 1$

Proof of Inductive Step

Bounding Height Using Black-Height

Lemma 3

If T is a red-black tree then $bh(r) \geq h/2$ where r is the root of T and h is the height of T.

Proof.

Assume that T has height h :

-
-

□

Proof of the Main Theorem

Theorem 4

If T is a red-black tree with n nodes then the height of T is at most $2\log_2(n+1)$.

Proof.

Let r be the root of T. The two Lemmas state that:

$$n \geq 2^{\mathrm{bh}(r)} - 1 \quad \text{and} \quad bh(r) \geq h/2$$

Putting these together yields:

$$\Rightarrow \qquad \Rightarrow$$

as required. □

Searching in a Red-Black Tree

Searching in a red-black tree is *almost* the same as searching in a binary search tree.

Difference Between These Operations:

- leaves are NIL nodes that do not store values
- thus, unsuccessful searches end when a leaf is reached instead of when a `null` reference is encountered

Worst-Case Time to Search in a Red-Black Tree:

-

What's Next?

Unfortunately, *insertions* and *deletions* are more complicated because we need to preserve the "Red-Black Properties."

We will discuss these operations during the next two lectures.

Reference: To read ahead, please see

Chapter 13 of *Introduction to Algorithms*
(on reserve in the library)

for more information about red-black trees.

Section 11.3 of the text discusses insertion, and Chapter 11 (programming problem 6) discusses deletion.