CSE 100: RED-BLACK TREES
Announcements

• Today is the last day to give PA2 regrades
• Pick your re-graded midterm after class
Red-Black Trees

And now for something not simple at all …
But FAST and used in C++ set implementation
What are the four properties of Red-Black trees?
1. Nodes are either red or black
2. Root is always black
3. No two reds in a row for any root to null path
4. Same no of black nodes in any root to null path
Red-Black Trees

1. Nodes are either red or black
2. Root is always black
3. No two red nodes in a root to leaf path. Children of red node must be black
4. For every node X, every path from X to a null reference must contain the same number of black nodes

No. of black nodes on root to null path is also known as the black height of the Red Black Tree

Black height of this tree is \( \leq 3 \)
Is this an AVL tree?
A. Yes
B. No

Are all red-black trees AVL trees?
Is this an AVL tree? Yes

Are all red black trees AVL trees? B. No
Red-Black Trees

Is this an AVL tree?  
(Not anymore)

Are all red black trees AVL trees?  
No
Why use Red-Black Trees

Fast to insert, slightly longer to find (but still guaranteed $O(\log(N))$)

“Gold standard” for balanced BSTs used in Java Collections and C++ STL
Red-Black trees have height $O(\log N)$

Proof idea: We will alter this tree so that it is in a form that allows us to make assertions about the height of the altered tree (which will NOT be a red-black tree). Then we will relate the height of the real tree to the height of the altered tree, in the worst case.
Height of a red-black tree is always $O(\log N)$.

Sketch of Proof:
1. Consider a Red Black tree with $N$ nodes
2. Merge the red nodes into their black parents
Sketch of Proof:
1. Consider a Red Black tree with N nodes
2. Merge the red nodes into their black parents

What are your observations about this new tree?

Each internal node has 2-4 children (otherwise, invariant 4 would not hold).
All leaf nodes are at the same level.
What is the tightest upper bound on the height of this tree, where $N_{black}$ is the number of nodes in \textit{this} tree?

Leaves are all at the same level
Each internal node has at least 2 children

A. 2
B. $\log_2(N_{black} + 1)$
C. $N_{black}$

$2^{h'} - 1 \leq N_{black}$
$h' \leq \log_2(N_{black} - 1)$

$h' \leq \log_2(N_{black} + 1)$
Red-Black invariants imply balance

Leaves are all at the same level
Each internal node has at least 2 children

Height is at most $\log_2(N+1)$

Based on invariant 3, we can only increase the height of this tree by a factor of 2 (by inserting red nodes between any two black nodes).

Height is at most $2 \cdot \log_2(N+1)$
Now for the fun part… under the hood of insertions

What color will non-root insertions be?
A. Red  ⇒  Sometimes invariant 3 will be broken.
B. Black  ⇒  Invariant 4 will always be broken
Now for the fun part… insertions

Non-root insertions will always be red
Try inserting 13
That wasn’t so bad!

Case 0: Parent was black. Insert new leaf node (red) and you’re done.
Try inserting 3

Case 1: Parent of leaf is red, & sibling of parent (uncle of leaf) is black or non existent
(a) parent is a left child of grandparent, leaf is left child of parent
Insertions: Case 1a

Right AVL rotation, and recolor
Case 1(a) in general

If X's Parent (P) is red, P is a left child of G, X is a left child of P, (and P's sibling (S) is black), then Rotate G right, flip colors of P and G

Why does this work?
Case 1 in general

Same number of black nodes on either side of tree
Roots of subtrees a, b and c (and node S) must be black
X’s and G’s parent is now guaranteed to be black
BST property preserved through AVL rotations
Which insertion can we not handle with the cases we’ve seen so far?

A. 1  B. 7  C. 12  D. 25
Insertions: Case 1b

Case 1: Parent of leaf is red, (& sibling of parent is black or non existent)
(b) parent is a left child of grandparent and leaf is right child of parent

Insert 7
Insertions: Case 1b

Suppose we right rotate 10

Insert 7

Case 1: Parent of leaf is red, (& sibling of parent is black or non existent
(b) parent is a left child of grandparent and leaf is right child of parent
Does this work?
A. Yes! We’re done!
B. The property about red nodes having only black children is violated.
C. The property about having the same number of black nodes on any path from the root through a null reference is violated.
Insertions: Case 1b

Insert 7

Case 1: Parent of leaf is red & sibling of parent is black)
(b) parent is a left child of grandparent, leaf is right child of parent
Insertions: Case 1b

Double rotation!

Insert 7
Insertions: Case 1b

Insert 7

Case 1: Parent of leaf is red & sibling of parent is black)
(b) parent is a left child of grandparent, leaf is right child of parent
Insertions: Case 1b

Case 1: Parent of leaf is red & sibling of parent is black)
(b) parent is a left child of grandparent, leaf is right child of parent

Insert 7
If X’s Parent (P) is red, P is a left child of G, X is a right child of P, (and P’s sibling (S) is black), then rotate P left, then rotate G right, then flip colors of X and G.

Why does this work?
Case 1b in general

Same number of black nodes on either side of tree
Roots of subtrees a, b and c (and node S) must be black
P’s and G’s parent is now black
BST property preserved through AVL rotations
Insertions: Summary, so far

Case 0: The parent of the node you are inserting is black. Insert and you’re done

Case 1: the parent of the node is red, the sibling of the parent is black:

Case 1a: P is left child of G, X is left child of P (single rotate then recolor)
Case 1b: P is left child of G, X is right child of P (double rotate then recolor)

Case 1c: P is right child of G, X is right child of P
Case 1d: P is right child of G, X is left child of P
Insert 1 and then insert 85. Draw the resulting tree.