http://www.cs.unh.edu/~ruml/cs758

1 handout: slides
Red-Black Trees

• Red-Black Trees
• BST Deletion
• Single Child
• Immed. Succ.
• Deep Succ.
• Break

Deletion Fixup
Red-Black Trees

node: data, left, right, parent, color

1. every node is either red or black
2. the root is black
3. (consider nil to be black)
4. both children of a red node are black
5. from any node, all paths to leaves have the same ‘black height’
4 cases of delete(z):

1. no left child, or no kids: substitute right subtree at parent.
2. no right child: substitute left subtree at parent.
3. successor y is z’s right child:
   (a) substitute y for z
   (b) attach z’s left subtree as y’s left subtree
4. successor y is deeper:
   (a) substitute y’s right subtree for y
   (b) attach z’s right subtree as y’s right subtree
   (c) as above, substitute y for z
   (d) as above, attach z’s left subtree as y’s left subtree

What if it’s a red-black tree?
Cases 1 and 2: Single Child

1. every node is either red or black
2. the root is black
3. (consider nil to be black)
4. both children of a red node are black
5. from any node, all paths to leaves have the same ‘black height’

deleting \( z \) with single child \( x \)

1. \( x \) takes \( z \)’s place
1. every node is either red or black
2. the root is black
3. (consider nil to be black)
4. both children of a red node are black
5. from any node, all paths to leaves have the same ‘black height’

deleting $z$ with single child $x$

1. $x$ takes $z$’s place
2. book uses $y$ for $z$ for short code
3. if $y (= z)$ was black, we have ‘extra black’ at $x$, so call fixup routine at $x$
Case 3: Two Children, Successor is Child

1. every node is either red or black
2. the root is black
3. (consider nil to be black)
4. both children of a red node are black
5. from any node, all paths to leaves have the same ‘black height’

deleting \(z\), successor \(y\) is right child

1. \(y\) takes \(z\)’s place and color
2. attach \(z\)’s left subtree as \(y\)’s left subtree
Case 3: Two Children, Successor is Child

1. every node is either red or black
2. the root is black
3. (consider nil to be black)
4. both children of a red node are black
5. from any node, all paths to leaves have the same ‘black height’

Deleting $z$, successor $y$ is right child

1. $y$ takes $z$’s place and color
2. attach $z$’s left subtree as $y$’s left subtree
3. if $y$ was black, we need ‘extra black’ at $y$’s right child $x$, so call fixup routine at $x$
Case 4: Two Children, Successor is Deeper

deleting \( z \), successor \( y \) is deep down

1. substitute \( y \)'s right child \( x \) for \( y \)
2. attach \( z \)'s right subtree as \( y \)'s right subtree
   as in simpler case:
3. \( y \) takes \( z \)'s place and color
4. attach \( z \)'s left subtree as \( y \)'s left subtree
5. if \( y \) was black, we need ‘extra black’ at \( x \), so call fixup
   routine at \( x \)
- asst 4: write verifier
- schedule: next week
Red-Black Trees

Deletion Fixup

■ Fix-up Loop
■ Case 1
■ Case 2
■ Case 3
■ Case 4
■ Complexity
■ EOLQs

Red-Black Tree Deletion Fixup
need to find a red node to make black

when $x$ red or root, color black and terminate

$x$ is non-root black node.
assume $x$ is a left child (other cases symmetric).
Must have sibling $w$, since $x$ holds ‘extra blackness’.
4 cases:

1. $w$ is red
2. $w$ and both its children are black
3. $w$ is black, its right child is black, its left child is red
4. $w$ is black, its right child is red

fix-up loop invariant: black heights of fringe (greek) nodes unchanged
Case 1

Case 1: \( w \) is red. so parent and children must be black.

solution:

1. rotate and recolor to get black sibling (moves red horizontally)
2. fall through to case 2, 3, or 4
Case 2

Red-Black Trees
Deletion Fixup
- Fix-up Loop
- Case 1
- Case 2
- Case 3
- Case 4
- Complexity
- EOLQs

Case 2: \( w \) and both its children are black

Solution:
1. Color \( w \) red. Subtree at parent now ‘black-balanced’.
2. Move \( x \)’s blackness (and \( w \)’s) up the tree.
3. Recur at parent.

If from case 1, \( x \) now red, so will terminate.
case 3: $w$ is black, its right is black, left is red

solution:
1. rotate right and move red over to right child
2. fall through to case 4
case 4: \( w \) is black, its right child is red

solution:

1. rotate and recolor to annihilate red with \( x \)’s black
2. set \( x \) to root to force termination
finding successor is $O(\lg n)$

one fixup iteration is constant time

fixup loops only when moving up, so is $O(\lg n)$

how many rotations are performed?
For example:

- What’s still confusing?
- What question didn’t you get to ask today?
- What would you like to hear more about?

Please write down your most pressing question about algorithms and put it in the box on your way out.

*Thanks!*