CPSC 223
Algorithms & Data Abstract Structures

Lecture 21:
Red-black tree
2-3 trees

Today …
• Red-Black Trees (cont.)
• 2-3 Trees (if time) [Ch 12: 651-670]
Balance Constraints

1. Every node is assigned a color (either Red or Black)
2. The root is always Black
3. The children of a Red node are always Black
4. Every path from the root to a NULL node has the same number of Black nodes

Inserting Nodes (Bottom Up)

If the inserted node’s parent is Red …
• We have to consider several cases
  – Let …
    \( X \) = inserted node
    \( P \) = parent
    \( S \) = sibling of \( P \)
    \( G \) = grandparent of \( X \)
  – Note that \( G \) must be Black!
• If \( S \) is Red
• If \( S \) is Black
  – If \( X \) is an outside grandchild
  – If \( X \) is an inside grandchild

NOTE: Because we are possibly propagating up, we consider the general case where \( X \) may have subtrees \( A \) and \( B \)
Sibling is Red

- If S is Red
  - Change P and S to Black
  - Change G to Red
  - Repeat with G as the new X

Outside Grandchildren

If S is Black and X is an outside grandchild
  - “Outside” grandchild means X is left-left or right-right

- Do a single rotation to the right (i.e., with G’s left)
- Change G to Red and P to Black
- We are done … Why?
  - The original root (G) was Black

This case assumes the children of X are Black (i.e., X is an internal node)
Inside Grandchildren

If $S$ is Black and $X$ is an inside grandchild

- “Inside” grandchild means $X$ is left-right or right-left

• Do a double rotation
• Change $X$ to Black and $G$ to Red
• We are done rebalancing

Inserting Nodes Top Down

• We want to avoid having to rotate “up” the tree
  - We don’t need to when $S$ is Red
• To avoid iterating back up the tree, we make sure that when we insert a node, the sibling $S$ is not Red

• One the way down
  - As we see a node $X$ with two Red children …
  - we change $X$ to Red and its children to Black (color flip)
  - If $X$ and $X$'s parent are Red, we apply single or double Rotation

  - While this increases the total number of Black nodes … it does not change the number of Black nodes on a path
Inserting Nodes Top Down

- For example, let's insert “9” into this tree

Start Here ➔ 30

15

10

8

12

Traverse Here ➔ 15

30

70

10

25

8

12

Now we need to do a
Single Rotation at “30” ...
Inserting Nodes Top Down

- For example, let’s insert “9” into this tree

We’re here →

15
10
8 12 25 70

Traverse here →

15
10
8 12 25 70
9

And insert “9”

Removing Nodes

- Recall that in a binary search tree we only remove the inorder successor
  - Copying the inorder successor does not violate the Red Black constraints
  - Removing the inorder successor might

- Simple cases:
  - If the node is Red, we can just remove it
  - If it is Black with a Red child … remove it and change the child’s color to Black
Removing Nodes

• Simple cases:
  – If the node is Red, we can just remove it
  – If it is Black with a Red child … remove it and change the child’s color to Black

![Diagram of simple cases](image1)

Remove X
NOTE: P must be Black

Remove X
Change N to Black

Removing Nodes

• The complicated case
  – The node X (to be deleted) and its child are Black
  – In this case, we end up with one fewer Black node on a path … and so the tree must be re-balanced
  – First, delete X and replace with inorder successor N
  – Now we need to rebalance N

![Diagram of complicated case](image2)

Remove X
(the inorder successor)
Removing Nodes – Bottom Up

• **Case A**: If \( N \) is the new root of the tree we are done
• **Case B**: The sibling \( S \) is Red
  – Flip colors of \( P \) and \( S \)
  – Rotate left at \( P \)
  – Now rebalance \( N \) with Cases D, E, or F

![Diagram showing Case A and Case B]

Now \( N \) has a Black sibling and Red parent ... so we go to cases 3-5 at \( N \)

Removing Nodes – Bottom Up

• **Case C**: if parent \( P \) and sibling \( S \) and \( S \)'s children are Black
  – Change \( S \) to Red
  – All paths through \( P \) now have one less Black node
  – So we now have to re-balance at \( P \) !!!

![Diagram showing Case C]

CPSC 223 – Fall 2010
Removing Nodes – Bottom Up

- **Case D**: If parent \( P \) is Red, but Sibling \( S \) and \( S \)'s children are Black
  - Change \( P \) to Black
  - Change \( S \) to Red
  - This doesn’t change number of Black nodes going through \( S \)
  - It adds a Black node on the path going through \( N \)

We are done rebalancing …

Removing Nodes – Bottom Up

- **Case E**: If \( N \)'s sibling \( S \) is black, \( S \)'s right child is Black, but \( S \)'s left (inner) child is Red
  - Change \( S \)'s left child to Black
  - Rotate \( S \) to the right
  - Then consider Case F …
Removing Nodes – Bottom Up

- **Case F**: If the sibling $S$ is Black and $S$’s right child is Red
  - Change $S$ to the color of $P$
  - Change $P$ to Black
  - Change the right child to Black
  - Rotate $P$ to the left
  - $N$ now has an additional Black node as an ancestor

We are done rebalancing ...