CS 430 Lecture 12 Activities

Opening Questions

1. What do you think the issue we need to handle when deleting a node from a red-black tree? How does red-black delete differ from a BST delete? If the actaul node deleted is red, then you're done. If the node you're deleting is black, then you potentially could have two red nodes in a row. If you color the deleted nodes child black (if it was red), then you're done. There would only be 0 or 1 children, not 2. If the child was already black, you add an extra "blackness" to temporarily fix the black-height problem, meaning the node counts for 2 black nodes.

Red-Black Tree Delete

- Think of V as having an "extra" unit of blackness. This extra blackness must be absorbed into the tree (by a red node), or propagated up to the root and out of the tree. There are four cases – our examples and "rules" assume that V is a left child. There are symmetric cases for V as a right child.

Terminology in Examples

- The node just deleted was U
- The node that replaces it is V, which has an extra unit of blackness
- The parent of V is P
- The sibling of V is S

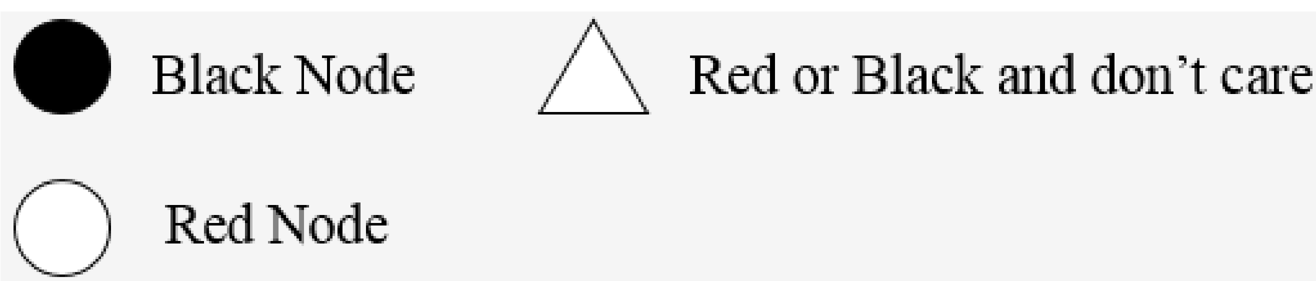


Figure 12.1: Red Black Tree Graphics Definitions

- V's sibling, S is Red
  - Rotate S around P
  - Swap the colors of S & P
- NOT a terminal case – One of the other cases with now apply
- All other cases apply when S is Black

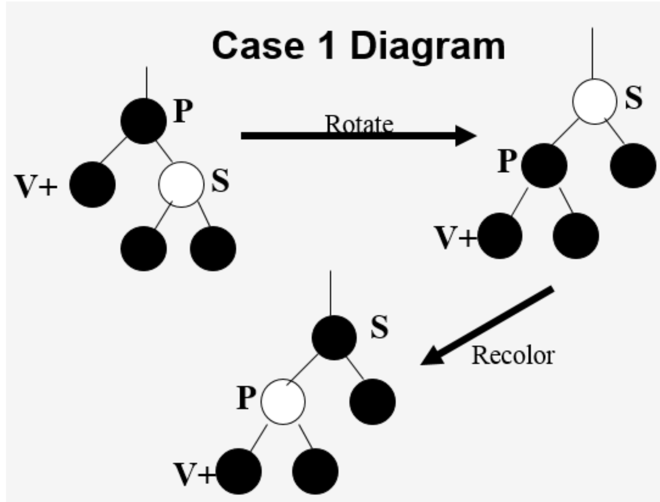


Figure 12.2: Red Black Tree Delete Case #1

- V's sibling, S is black and has two black children.
  - Recolor S to be Red
  - P absorbs V's extra blackness
    - If P was Red, make it black, we're done
    - If P was Black, it now has extra blackness and problem has been propagated up the tree

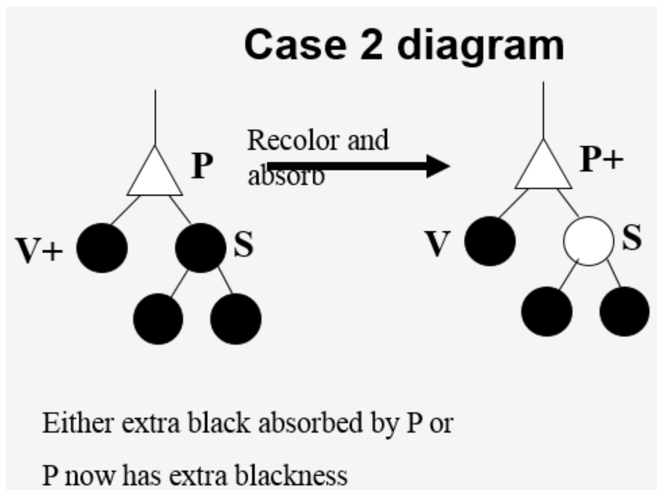


Figure 12.3: Red Black Tree Delete Case #2

- S is black
- S's RIGHT child is RED (Left child either color)
  - Rotate S around P
  - Swap colors of S and P, and color S's Right child Black
- This is the terminal case – we're done

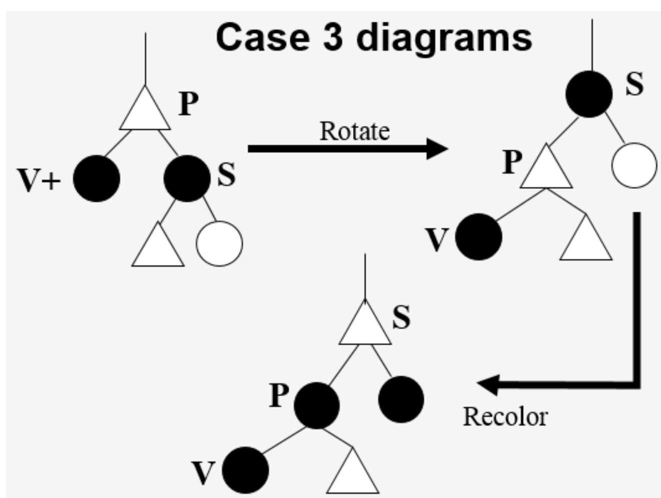


Figure 12.4: Red Black Tree Delete Case #3

- S is Black, S's right child is Black and S's left child is Red
  - Rotate S's left child around S
  - Swap color of S and S's left child
  - Now in case 3

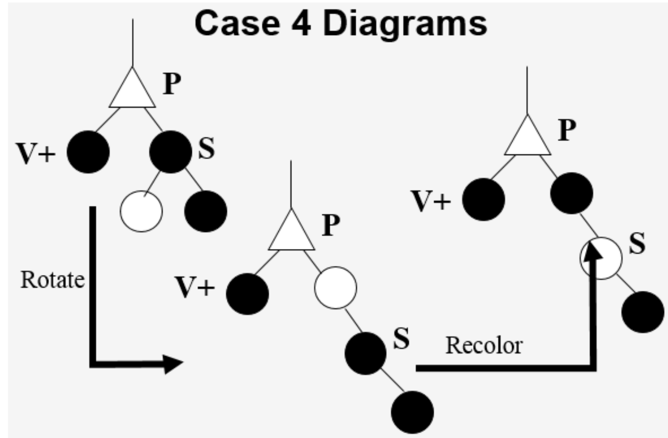


Figure 12.5: Red Black Tree Delete Case #4

Red Black Visualization:

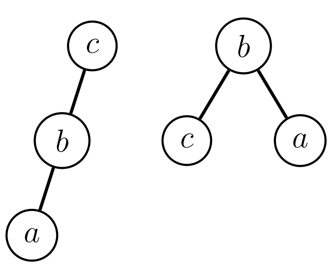
- http://gauss.ececs.uc.edu/RedBlack/redblack.html
- https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

AVL Trees

An AVL tree is a special type of binary tree that is always "partially" balanced. The criteria that is used to determine the "level" of "balanced-ness" is the difference between the heights of sub-trees of every node in the tree. The "height" of the tree is the "number of levels" in the tree. AN AVL tree is a special binary tree in which the difference between the height of the right and left sub-trees (of any node) is never more than one.

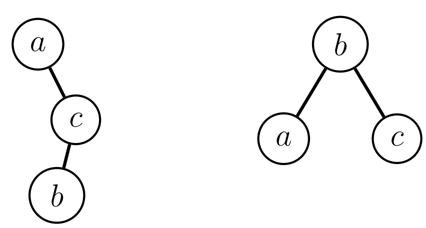
1. How do you think we could keep track of the height of the right and left sub-trees of every node?
2. If we find an imbalance, how can we correct it without adding any significant cost to the insert of delete?

Single Rotations
The imbalance is left-left (or right-right)



Perform single right rotation at c (R-rotation)
Similar idea for single left rotation (L-rotation)

Double Rotations
The imbalance is left-right (or right-left)



Perform right rotation at c then left rotation at a (RL-rotation)
Similar idea for left rotation then right rotation (LR-Rotation)

AVL Visualization: https://visualgo.net/bn/bst