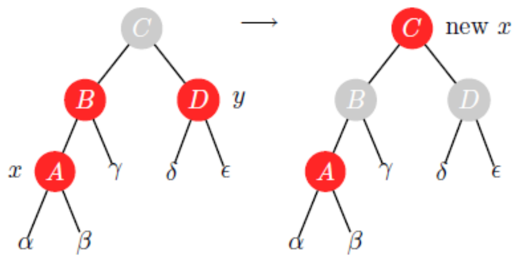
CS 430 Introduction to Algorithms – Fall 2017
Recitation Class Note 5
Xiaolang Wang

(All parts of this note are based on Professor Reingold's lecture note)

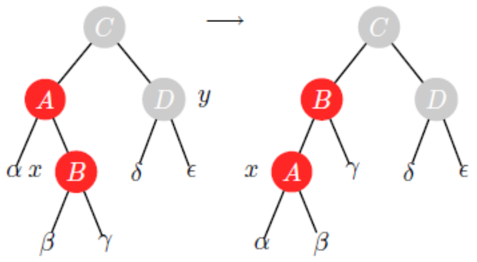
1. **2-3-4 trees and red-black trees**
   - Motivation of using red-black trees: height-balance, avoiding worst case in binary search (Also AVL trees)
   - Properties of red-black trees:
     - ❖ Node red or black
     - ❖ Nil-leaf is black
     - ❖ Root is black
     - ❖ No two red nodes are adjacent
     - ❖ Each path from root to leaf passes same number of black nodes (same black depth)
   - A 2-3-4 tree:
     - ❖ Each node has two, three or four children
     - ❖ 1 data 2 children; 2 data 3 children; 3 data 4 children
     - ❖ Each leaf has the same depth
   - Insertion and deletion can be done while maintaining the property.
   - We want to show red-black tree has $\Theta(\log n)$ height though 2-3-4 trees
   - If there are n data in a 2-3-4 tree, how tall can the tree be?
     - ❖ Tallest: binary tree: $\lg n - 1$
     - ❖ Shortest: each node has four children: $\log_4(\frac{n}{3} \times \frac{3}{4}) = \frac{1}{2}\lg n - 1$
     - ❖ $\Theta(\log n)$ height
   - If we can find transformations between 2-3-4 trees and red-black trees which change the tree's height by at most a constant factor, then red-black trees must also have $\Theta(\log n)$ height:
     - ❖ Each node in 2-3-4 trees mapped to a set of adjacent nodes with at most one black node
     - ❖ 2-node: a black node (data)
     - ❖ 4-node: a black node (data) with 2 red children (data)
     - ❖ 3-node: a black node (data) with a red children (data)
   - The height of red-black tree is at most $2\lg n - 2$.

2. **Insertions and deletion in red-black trees**
   - Insertion
     - ❖ Insert as in a binary search tree, then color it red.
     - ❖ What kind of violation we might have here?

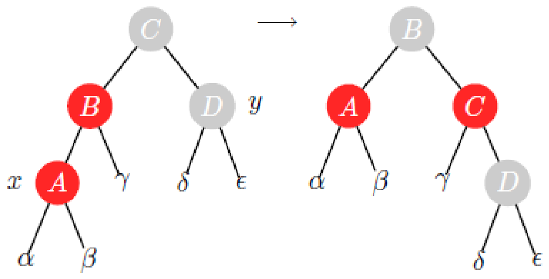
1. x has a red uncle y. In this case recoloring is sufficient to restore the red-black properties. Make x's parent and uncle black and x's grandparent red.



2. x has black uncle y and x is the right child. In this case, rotate the edge between x and its parent. This makes x's parent its left child and results in a Case 3.



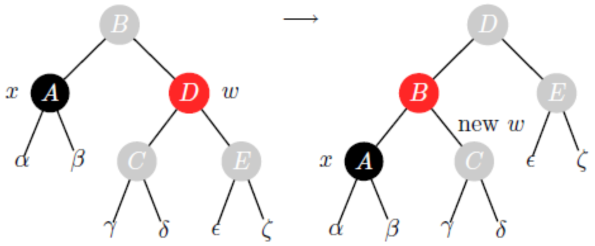
3. x has black uncle y and x is the left child. In this case rotate the edge between x's parent and grandparent. Also color x's parent black and its old grandparent red. This restores the red-black properties.



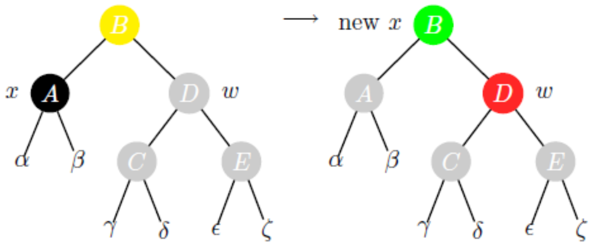
Since only local changes are made in each of these cases, insertion can be done in $O(\log n)$ time.

- Deletion
  - ❖ Delete as in a binary search tree.
  - ❖ What kind of violation we might have if we delete a black node?
  - ❖ Way to fix: double black node.

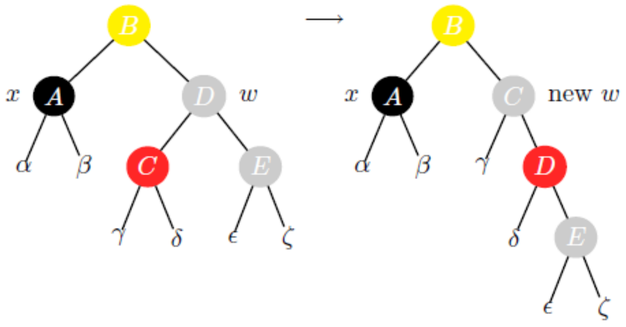
1. Double-black x has a red sibling w. In this case, rotate the edge between x and its parent so that this sibling becomes x's grandparent. Also recolor so that x's parent is red and its old sibling is black. This converts a case 1 situation into one of the other cases.



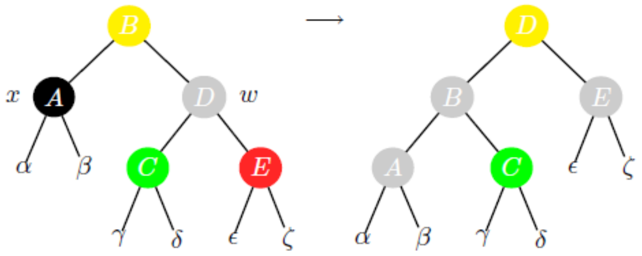
2. Double-black x has a black sibling w and black nephews. In this case, make x single black and the sibling red. To compensate for this, make x's parent black if it was red and double black if it was already black. This either solves the problem or moves the double black node up toward the root.



3. Double-black x has a black sibling w, a red left nephew, and a black right nephew. In this case, rotate the edge between the red nephew and the sibling so that the nephew becomes x's sibling. Also swap the colors of the nephew and sibling. This gives a situation handled by case 4.



4. Double-black x has a black sibling w and red right nephew. In this case, rotate the edge between x's sibling and x's parent so that the former sibling becomes x's new grandparent. Then recolor so that the former right nephew becomes black, x becomes single black, x's parent becomes black, and x's former sibling gets the previous color of x's parent. This restores the red-black properties.



Again, since only local changes are made at each step and the double black node moves upward, deletion is accomplished in $O(\log n)$ time.