Data Structures

Insertion In A Red-Black Tree

Design and Analysis of Algorithms I

High-Level Plan

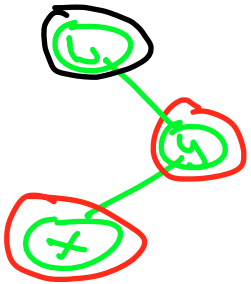
Idea for Insert / Delete: proceed as in a normal binary search tree, then recolor and/or perform rotations until invariants are restored.

Insert(x): ① insert x as usual (makes x a leaf)

② try coloring x red

③ if x's parent y is black, done.

④ else y is red. ⇒ y has a black parent w

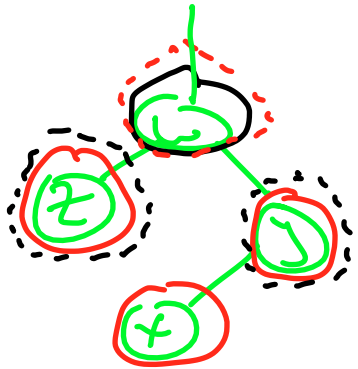


rock and hard place-
if we color red, we might have 2 reds in row
if we color black, we might introduce additional black in root-null path
start with coloring red. check parent. else color black

Insertion

2 cases, either w has other child red or (black or no other child)

Case 1



Case 1: the other child z of x's grandparent w is also red.

=> recolor y, z black and w red

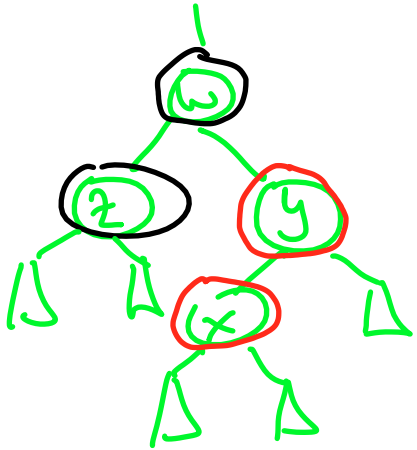
[key point: does not break invariant (4)]

=> either restores invariant (3) or propagates the double red upward

=> can only happen $O(\log n)$ times

[if you reach the root, recolor it black => preserves invariant (4)]

Case 2



Case 2: let x,y be the current double-red, x the deeper node.
Let w = x's grandparent.
Suppose w's other child (≠y) is NULL or is a black node z.

Exercise/Case analysis (details omitted): can eliminate double-red [⇒ all invariants satisfied] in O(1) time via 2-3 rotations + recolorings.