CSE221

Lecture 15:
Red-Black Trees

Hyungon Moon

Acknowledgment: The content of this file is based on the slides of the textbook as well as the slides provided by Prof. Won-Ki Jeong.

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Red-Black Trees

- Self-balancing binary search tree
- Guarantee O(log n) insertion, search, delete
- Definition
  - Binary search tree that every node is colored either red or black
  - Leaf nodes do not contain data
    - External nodes
  - Satisfy the properties

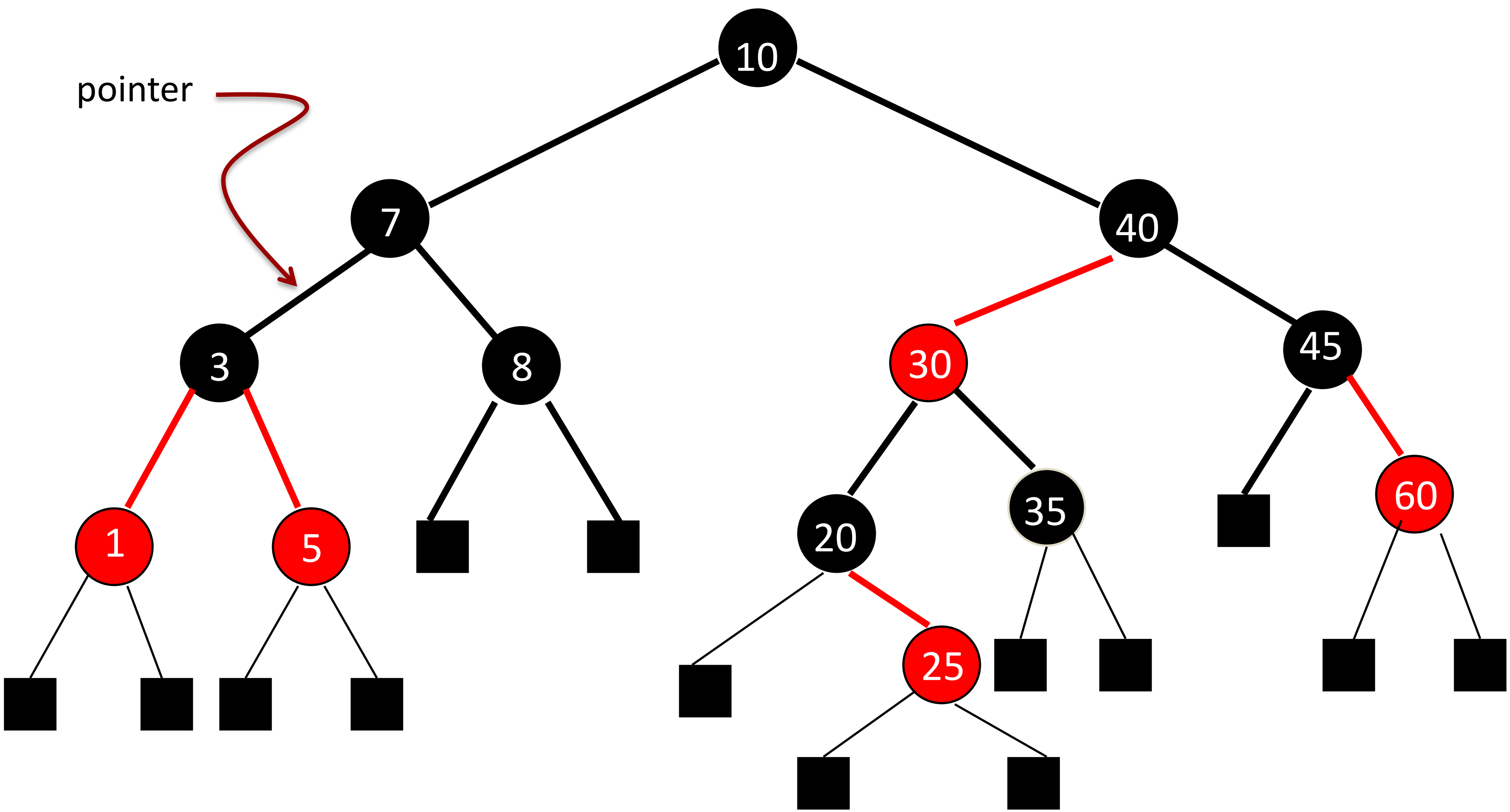
Red-Black Tree Properties

- The root and all external nodes are <u>black</u>
- No root-to-external-node path has two consecutive red nodes
  – (=) Red node must have two black children
- All root-to-external-node paths have the same number of black nodes

Red-Black Tree Properties (Pointer)

- Pointers from an internal node to an external node are black
- No root-to-external-node path has two consecutive red pointers
- All root-to-external-node paths have the same number of black pointers

Example Red-Black Tree

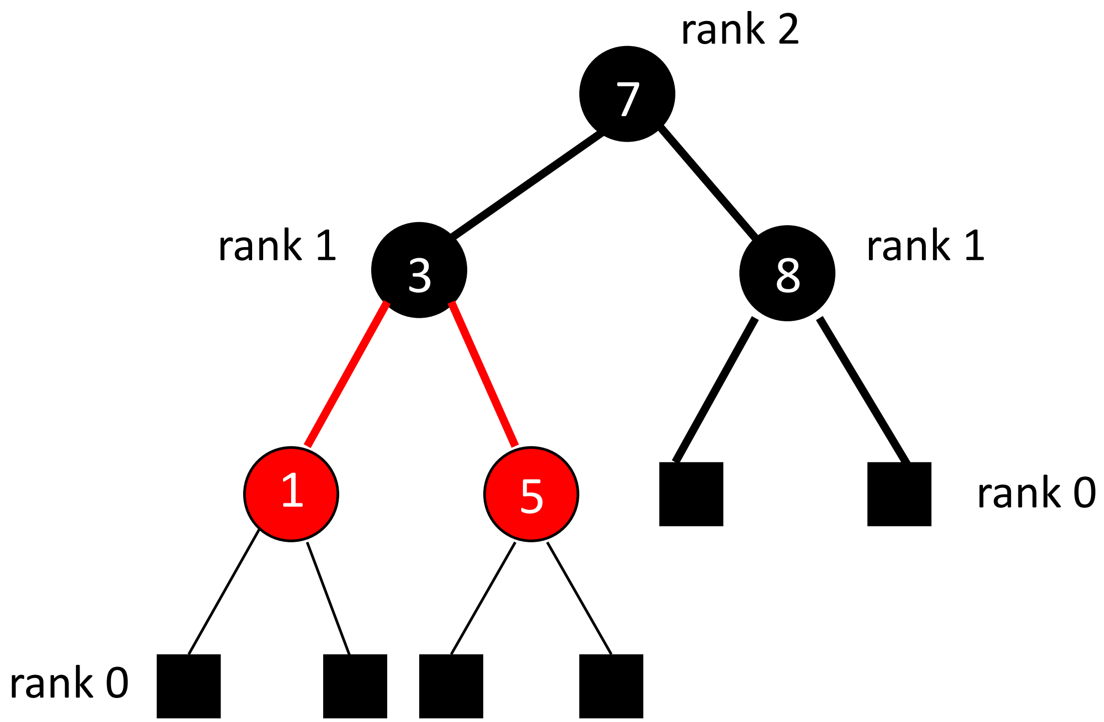


Properties

- If P and Q are two root-to-external-node paths in a red-black tree, then length(P) <= 2*length(Q)
- (=) longest path length is bounded by 2*shortest path length
  - Shortest path : B-B-B-....-B
  - Longest path : B-R-B-R...-B
  - Number of B must be same for all paths by definition, so the above statement is true

Properties

- Rank : # of black pointers on any path from a node to any external node



Properties

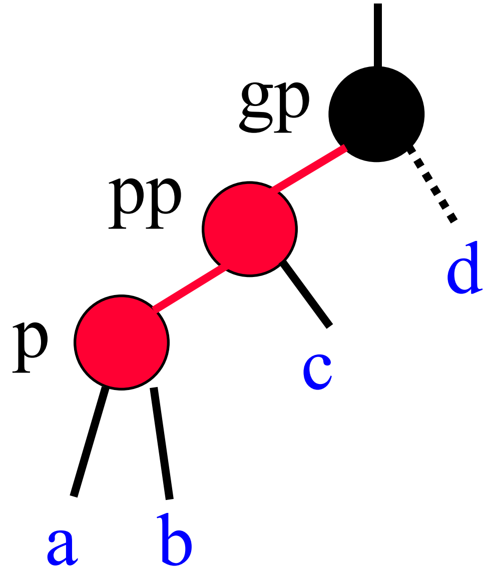
- h : height, r : rank of the root, n : # of nodes
- $h \leq 2r$
- $n \geq 2^r - 1$
- $h \leq 2\log_2(n+1) \Rightarrow h = O(\log n)$

Inserting

- Same as regular binary search tree insertion
  - Except for coloring is needed.
- How to color a new node?
  - If the tree was empty, new node is root so assign black
  - If the tree was not empty, assign black causes increase one black node in the path : NO!
    - b/c violate same # of black nodes for all paths, difficult resolve
  - If the tree was not empty, assign red may cause two consecutive red nodes in the path : OK!
    - Can be resolved by rotation and color flips

Classification Of 2 Red Nodes/Pointers

pp: parent
gp : grand parent

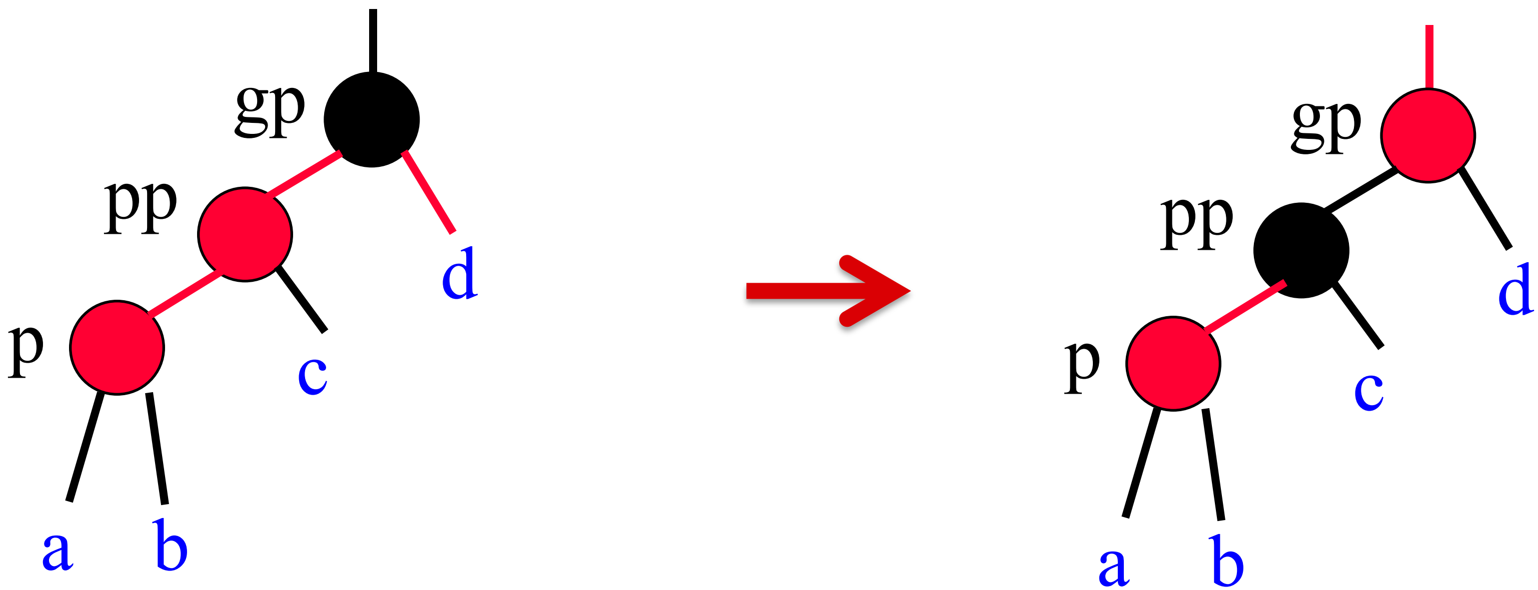


LLb example

- XYz
  - X => relationship between gp and pp.
    - pp is left child of gp => X = L.
  - Y => relationship between pp and p.
    - p is right child of pp => Y = R.
  - z = b (black) if d = null or a black node
  - z = r (red) if d is a red node

XYr

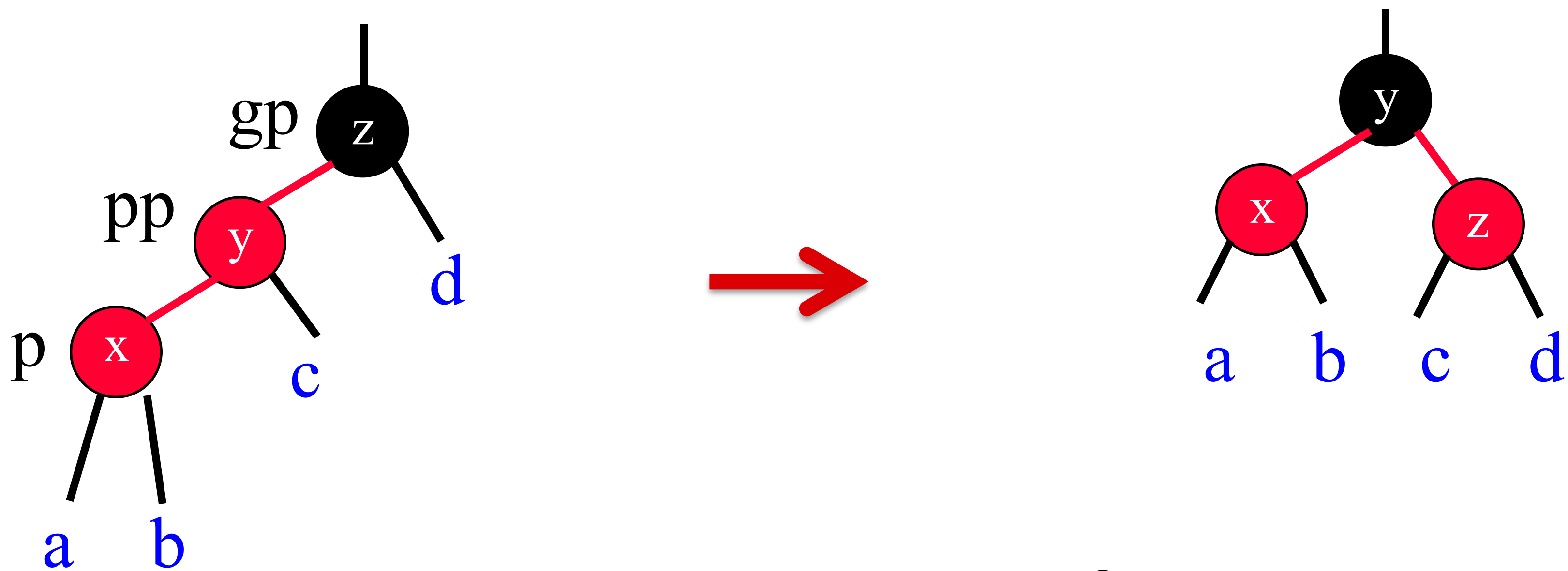
- Color flip



- Flip color of pp, gp, d and pointers of gp
- Flip color d to black is ok b/c gp is also flipped
- Reapply transformation to gp by p = gp

LLb

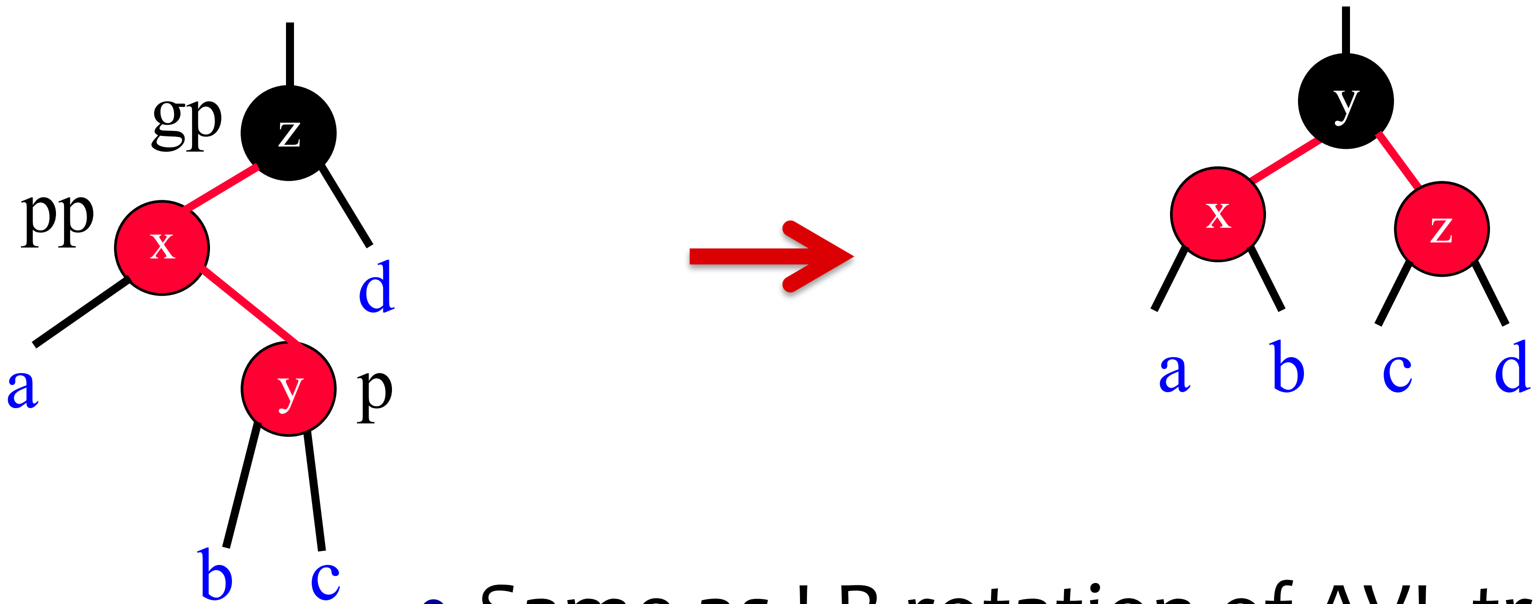
- Rotate



- Same as LL rotation of AVL tree
- Filp color of pp and gp after rotation
- No need to check parent b/c root color is not changed

LRb

- Rotate

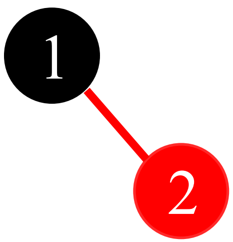


- Same as LR rotation of AVL tree
- Flip color of p and gp
- RRb and RLb are symmetric

Insert Example

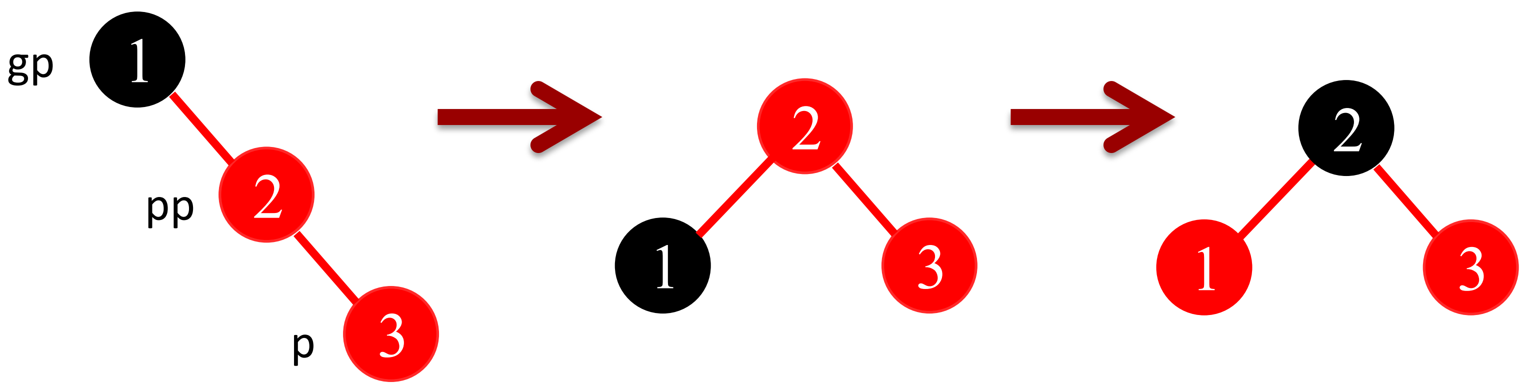
- Insert 1
  - Root

- Insert 2
  - Red



Insert Example

- Insert 3
  - RRb

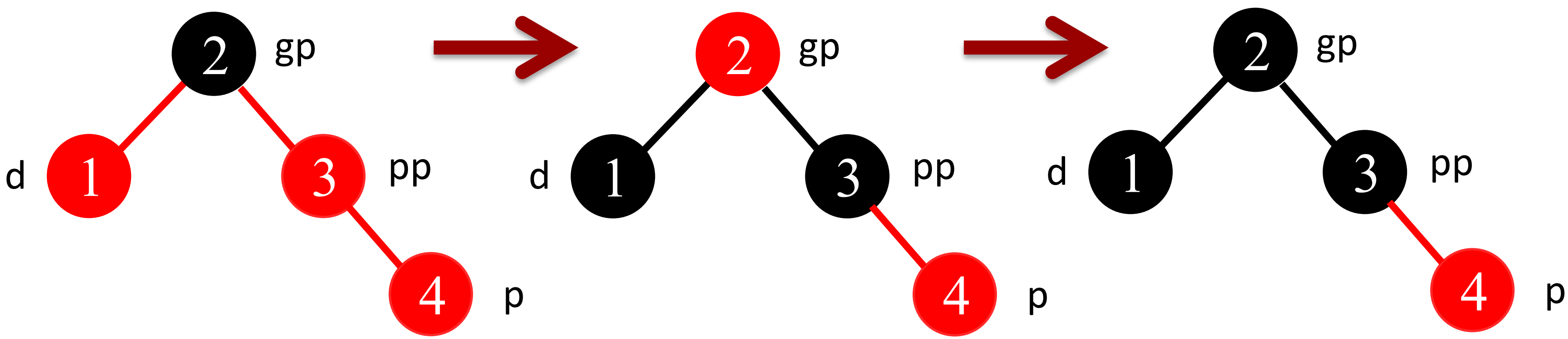


left rotate

flip color of gp & pp

Insert Example

- Insert 4
  - RRr

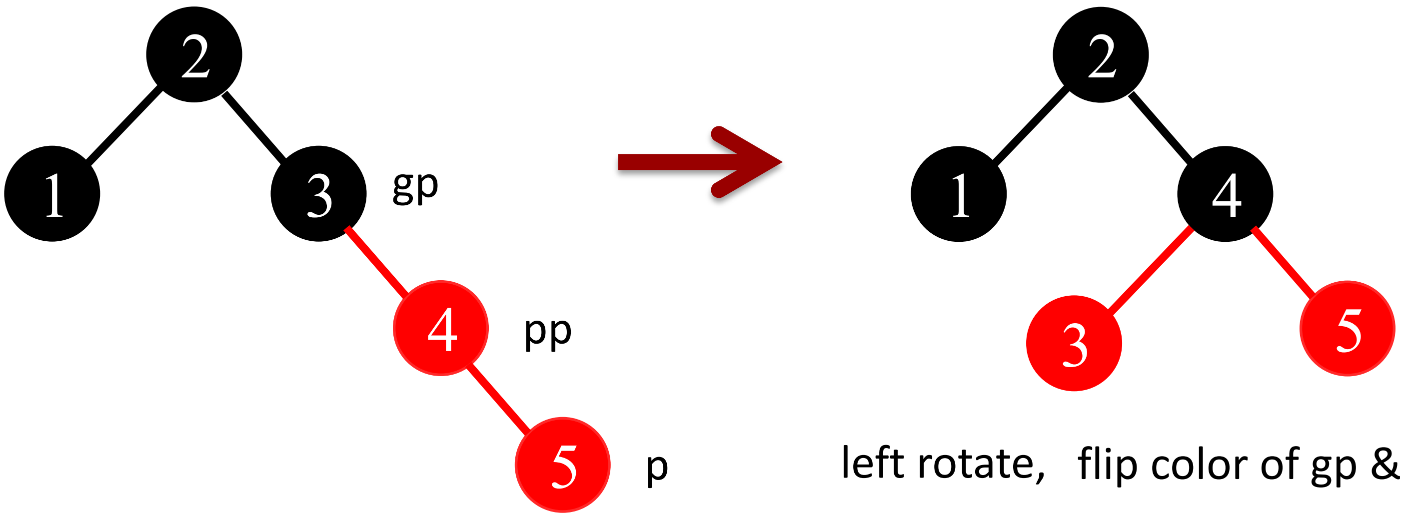


flip gp, pp, d color

flip gp back to black (root)

Insert Example

- Insert 5
  - RRb

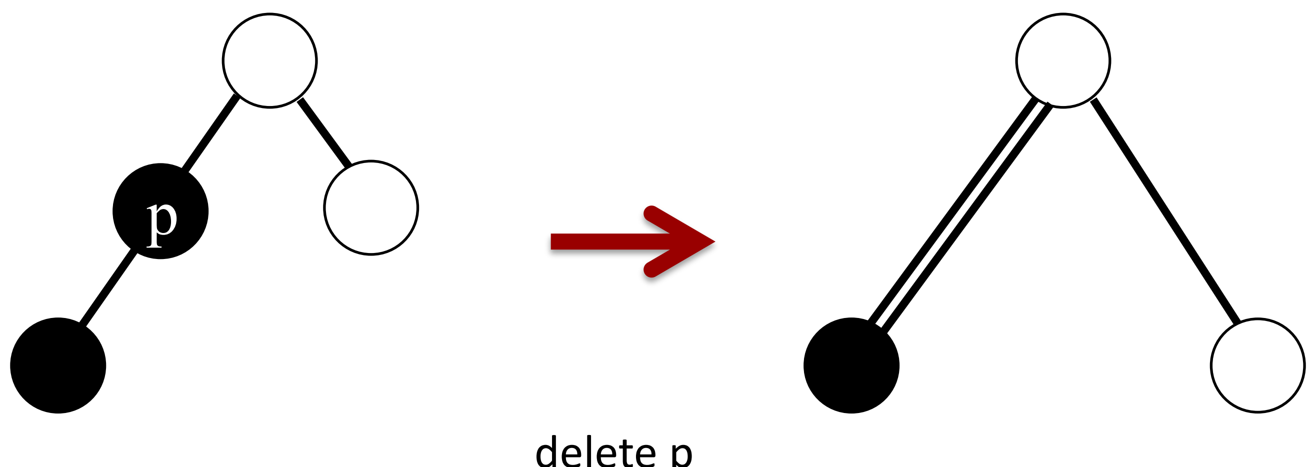


left rotate, flip color of gp & pp

Delete

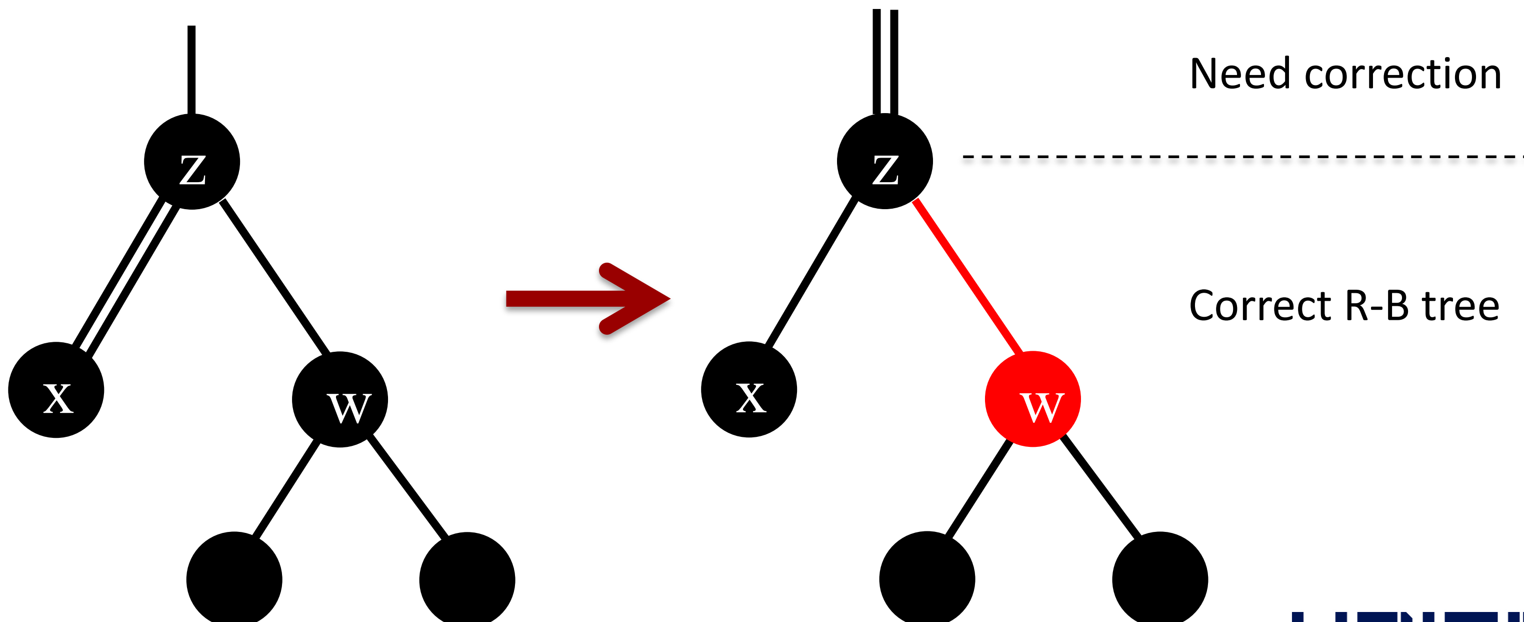
- Similar to insert, but more complicated
- Delete black will violate red-black property
  - Path passing through deleted node will have less number of black nodes
  - Double black pointer

white circle : any color can be placed



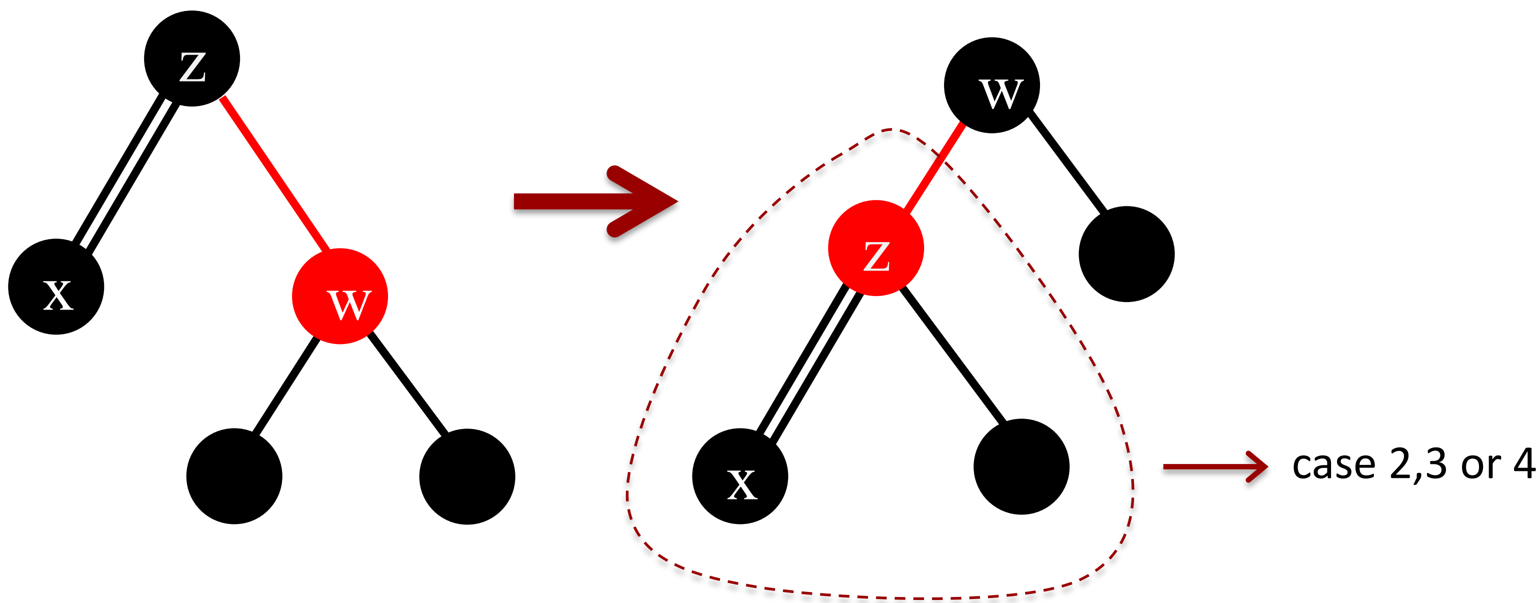
Delete

- Case 0 : w and its two children are black
  - Make w red
  - If z is black, then move up double black pointer. x = z, z = z->parent and restart (below z is ok)



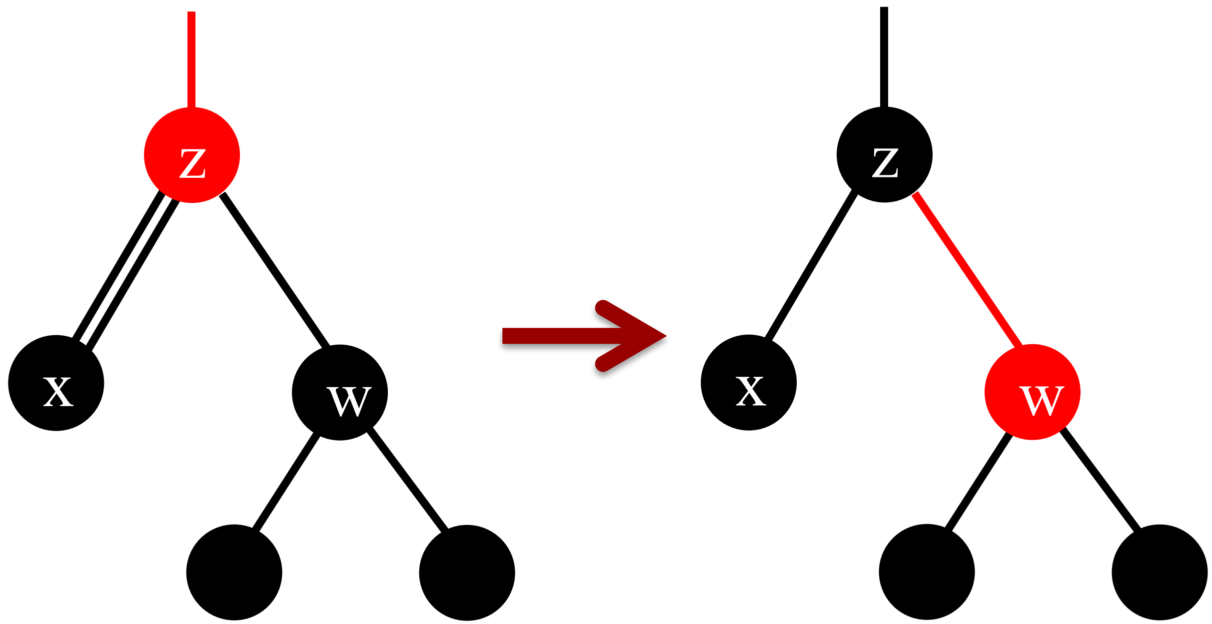
Delete

- Case 1 : z is black and w is red
  - Left-rotate at z and exchange colors of z & w
  - Go to case 2, 3, or 4 for subtree of z



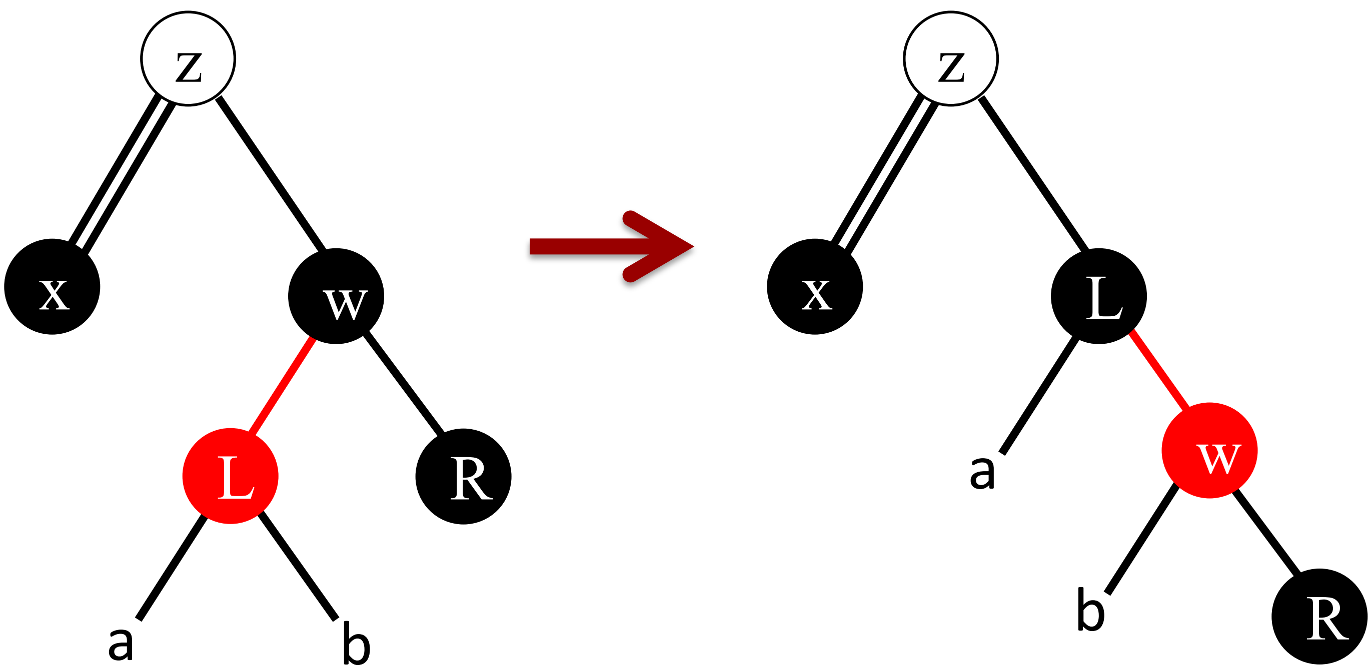
Delete

- Case 2 : w and its two children are black
  - Make w red
  - If z is red, then make z black and remove double pointer. Done.



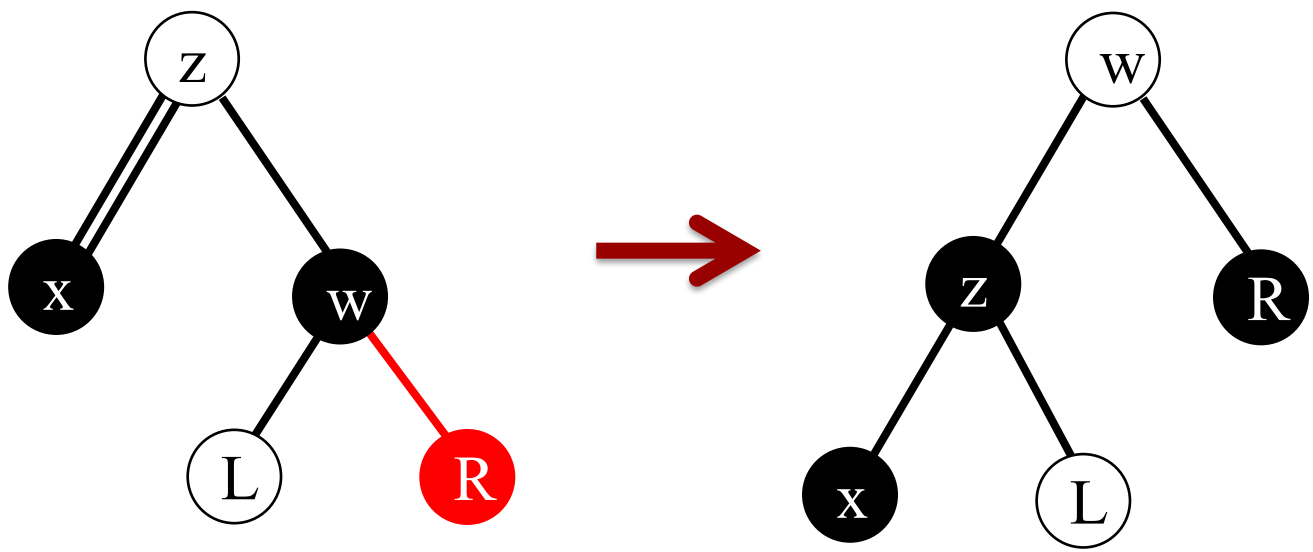
Delete

- Case 3 : w and its right child are black while its left child is red
  – Right-rotate at w and exchange colors of w and its left child. Go to case 4.



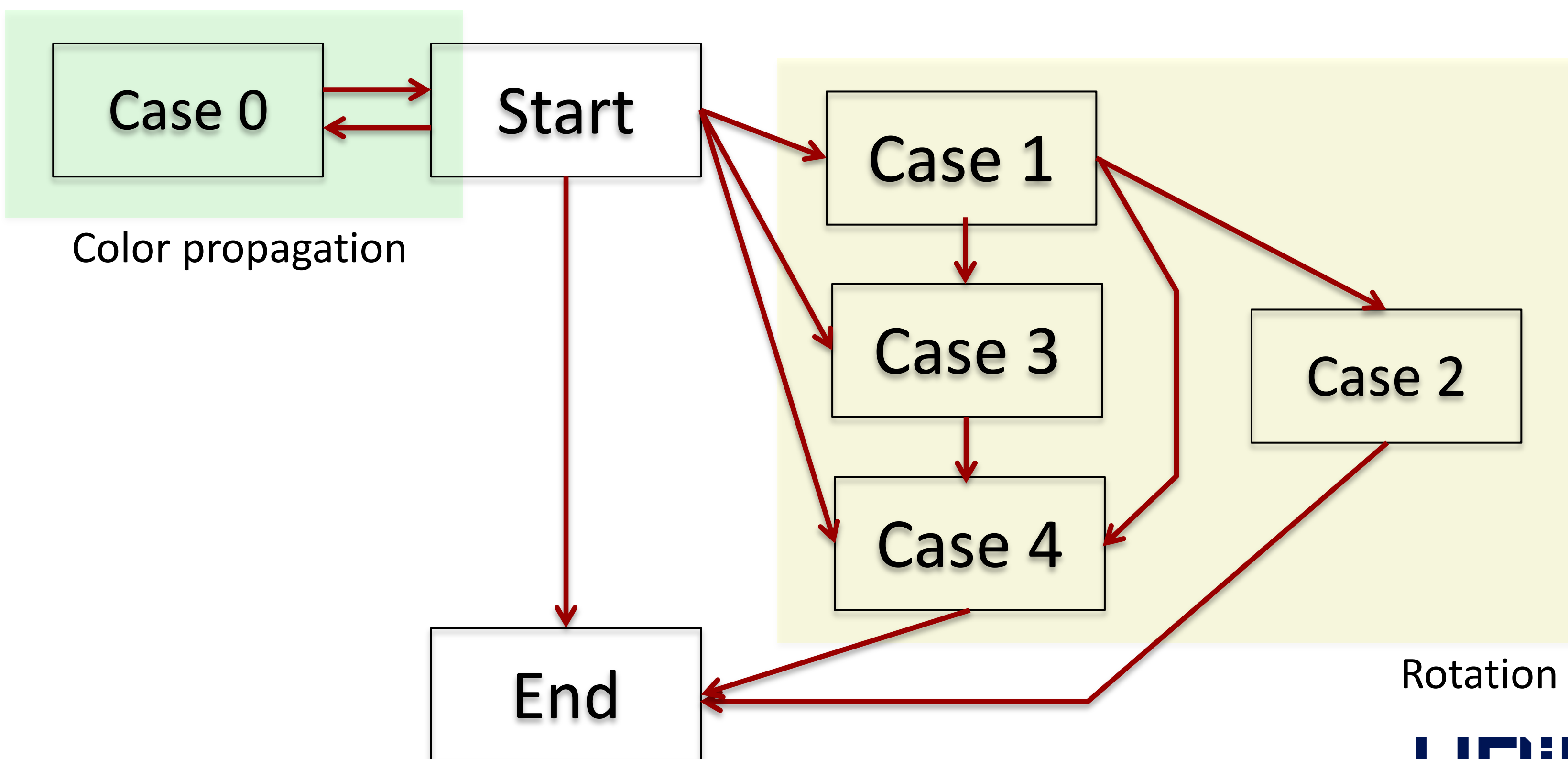
Delete

- Case 4 : w is black and its right child is red
  - Left rotate at z, exchange colors of z & w
  - Remove double black pointer, change R to black
  - Done



Delete Workflow

- At most 3 rotations are needed
- Color exchange can propagate log n times



Discussion

- Red-Black trees use color as balancing information instead of height as in AVL trees
- Insertion/deletion may cause a perturbation (if two consecutive red nodes exist)
- Perturbation is either
  - resolved locally (rotations), or
  - propagated to a higher level in the tree by recoloring (color flip)
- O(1) for a rotation or O(log n) color flips
- Total time: O(log n)

Questions?