Red-Black-Trees

Binary Search Tree

- Average case and worst case Big O for
  - insertion
  - deletion
  - access
- Can balanced be guaranteed?

Red Black Trees

- A BST with more complex algorithms to ensure balance
- Each node is labeled as Red or Black.
- Path: A unique series of links (edges) traverses from the root to each node.
  - The number of edges (links) that must be followed is the path length
- In Red Black trees paths from the root to elements with 0 or 1 child are of particular interest

black-height

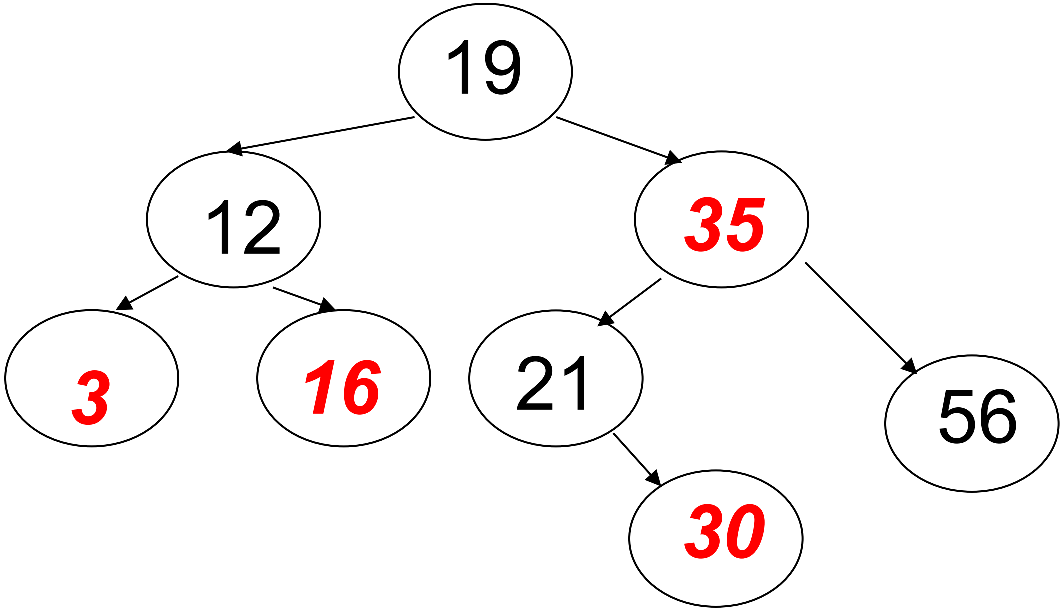
- The **black-height** of a node x, denoted bh(x), is the number of black nodes on any path from, but not including, a node x down to a leaf.

Red Black Tree Rules

1. Every node is colored either Red or black
2. The root is black
3. If a node is red its children must be black.
4. Every path from a node to a null link must contain the same number of black nodes

Example of a Red Black Tree

- The root of a Red Black tree is black
- Every other node in the tree follows these rules:
  - Rule 3: If a node is Red, all of its children are Black
  - Rule 4: The number of Black nodes must be the same in all paths from the root node to null nodes



- Prove :for each node x in red black tree, the Subtree with root x includes at least $2^{bh(x)}-1$ *internal nodes.*
  - The **basis** is when height of the tree is 0, which means that x is a leaf node and therefore bh(x) = 0 and the subtree rooted at node x has $2^{bh(x)}-1$ = 2^0-1 = 1-1 = 0 nodes.
  - We Assume the subtree with root x and height h has $2^{bh(x)}-1$internal nodes then we can show that the subtree with root x with height h+1 has $2^{bh(x)}-1$ internal nodes:

Prove(Continue)

For any non-leaf node x (height > 0) we can see that the black height of any of its two children is at least equal to bh(x)-1 or bh(x).

By applying the assumption above we conclude that each child has at least $2^{[bh(x)-1]}-1$ internal nodes, accordingly node v has at least

$$2^{[bh(x)-1]}-1 + 2^{[bh(x)-1]}-1 + 1 = 2^{bh(x)}-1$$

internal nodes, which ends the proof.

- Is there any relationship between the height of a red black tree and its internal nodes?
  - A red-black tree with n internal nodes has height at most 2lg(n+1)
  - Proof?

$n \geq 2^{\mathrm{bh}(root)} - 1$

$n \geq 2^{h/2} - 1$

$\lg(n+1) \geq h/2$

$h \leq 2 \lg(n + 1)$

Thus $h = O(\lg n)$

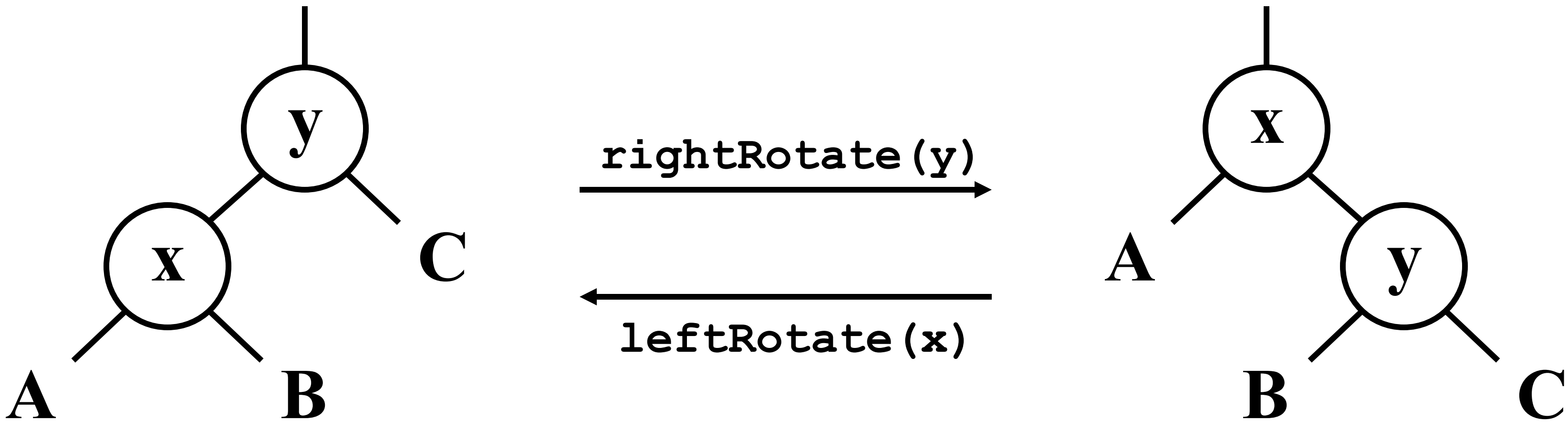
- Is there any relationship between number of leaves and number of internal nodes?
- See some examples of red-black trees
  - Number of leaves = Number of internal nodes +1?
  - Can you prove this by induction?

Red-Black Trees: Insertion

- Insertion: the basic idea
  - Insert *x* into tree, color *x* red
  - Only rule 3 might be violated (if parent of x is red)
    - If so, move violation up tree until a place is found where it can be fixed
  - Total time will be O(lg *n*)

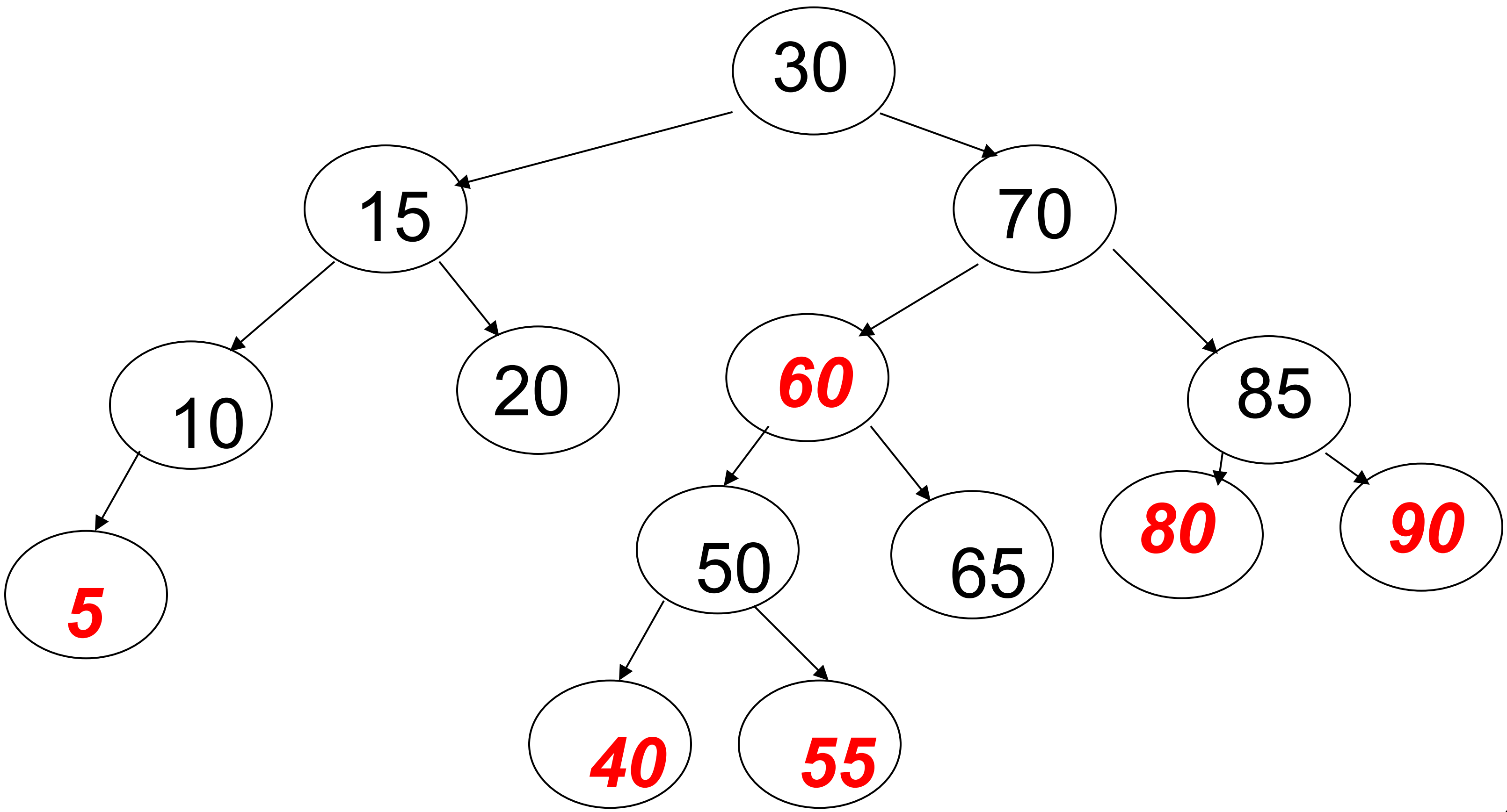
RB Trees: Rotation

- Our basic operation for changing tree structure is called *rotation*:



Insertions with Red Parent - Child

Must modify tree when insertion would result in Red Parent (using color changes and rotations)



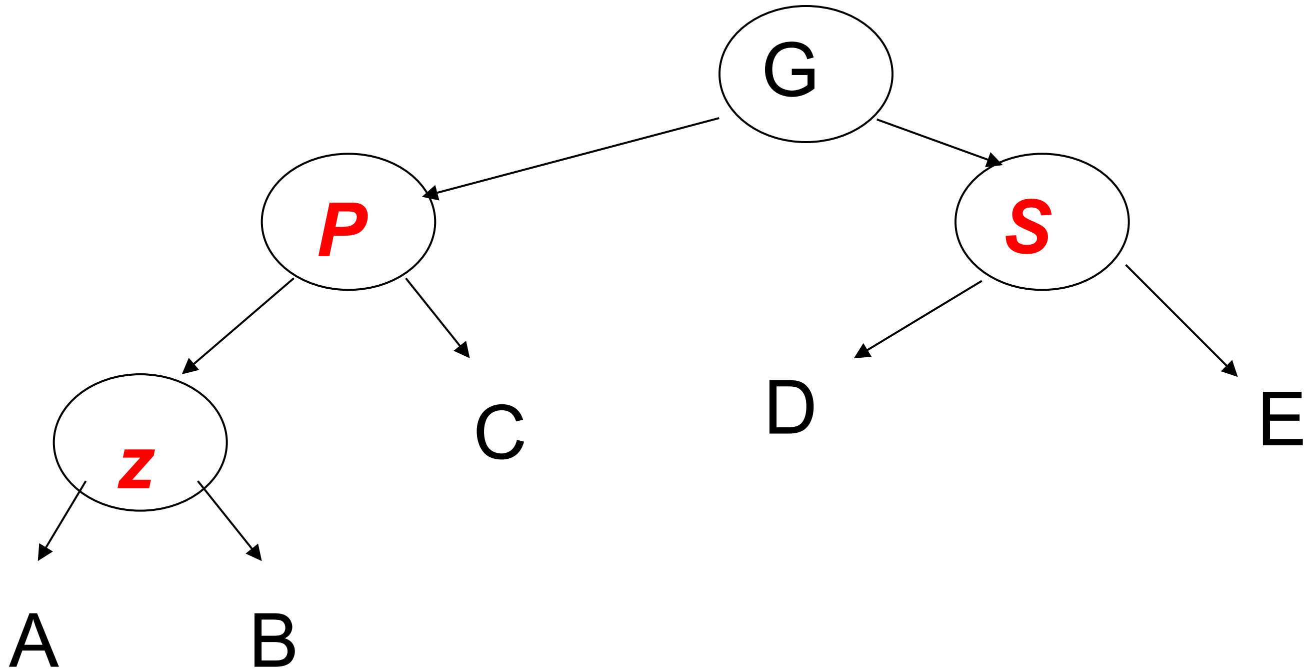
Insertion Cases

1. Parent of z is a left child; sibling y of parent of z is red.
   (a) z is a left child.
   (b) z is a right child.
2. Parent of z is a left child; sibling y of parent of z is black. z is a right child.
3. Parent of z is a left child; sibling y of parent of z is black. z is a left child.

RB Insert: Case 1: Red uncle

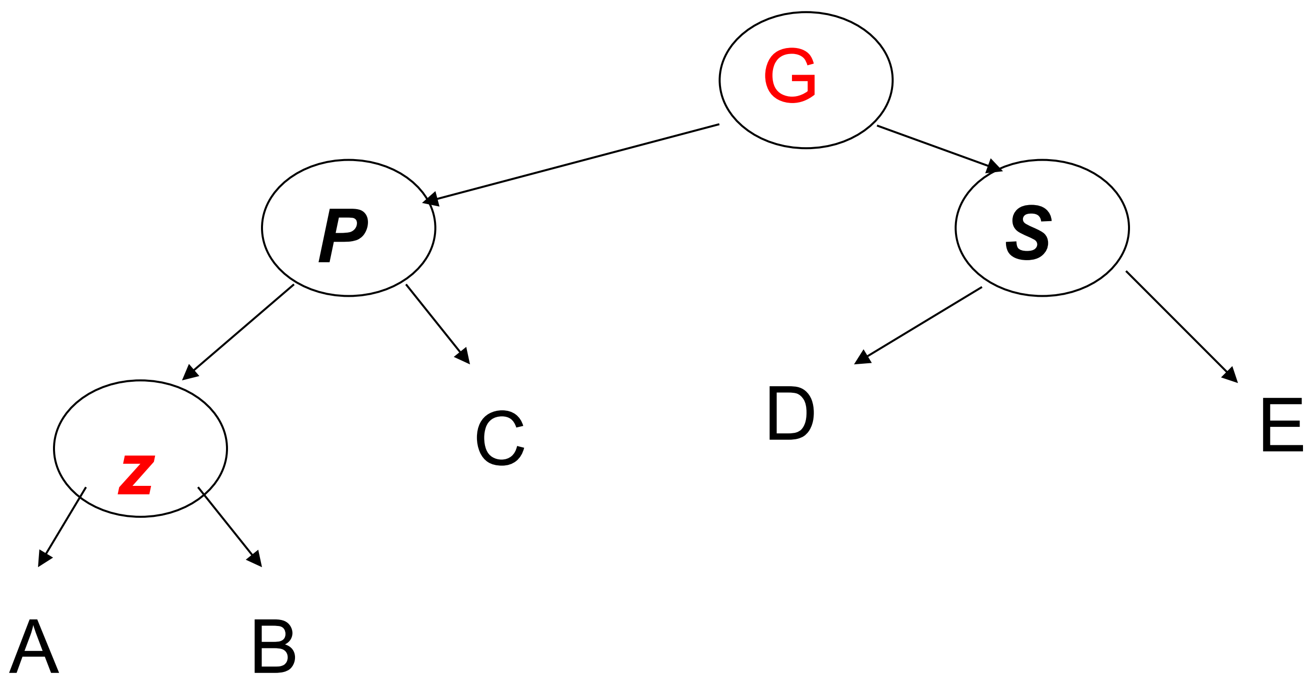
Change colors of some nodes, preserving rule 4: all downward paths have equal bh

The while loop now continues with z's grandparent as the new z



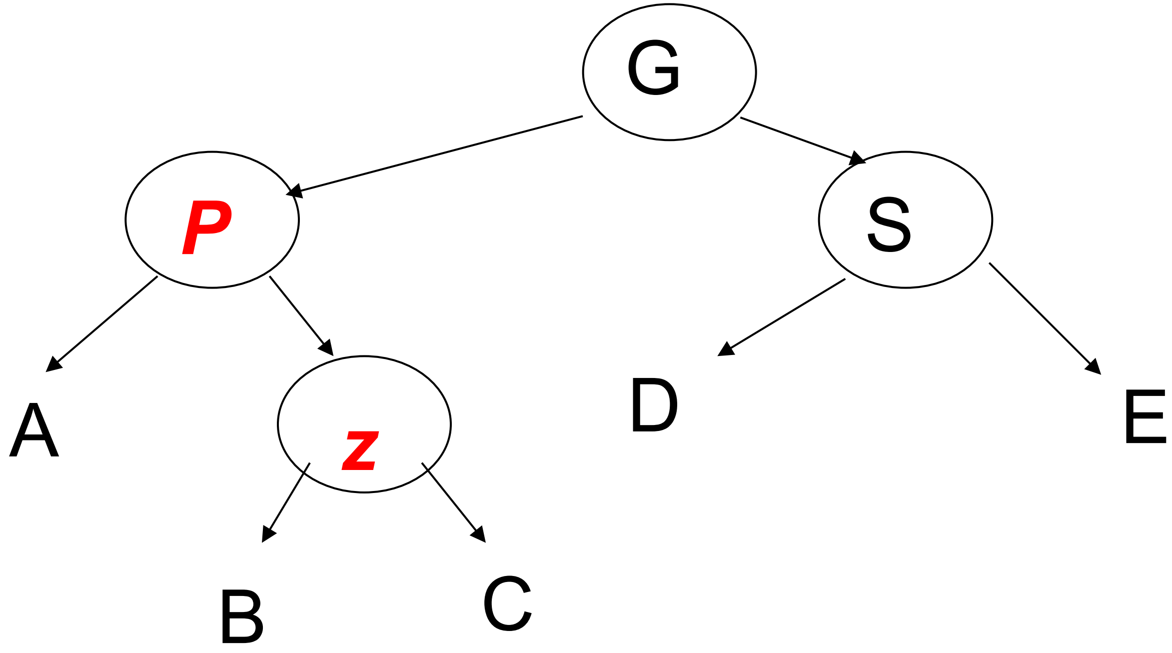
Fixing Tree when S is Red

- make appropriate color changes

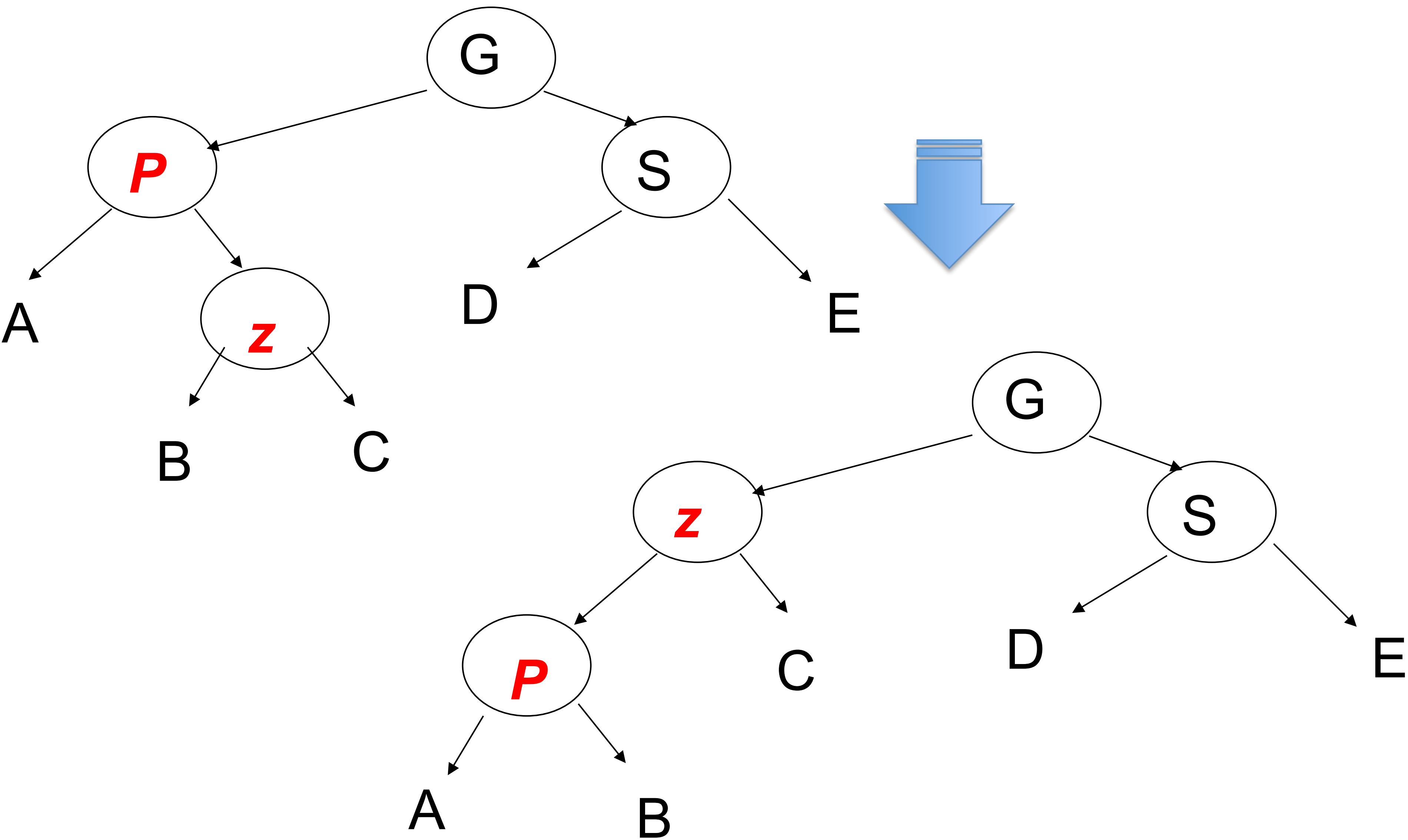


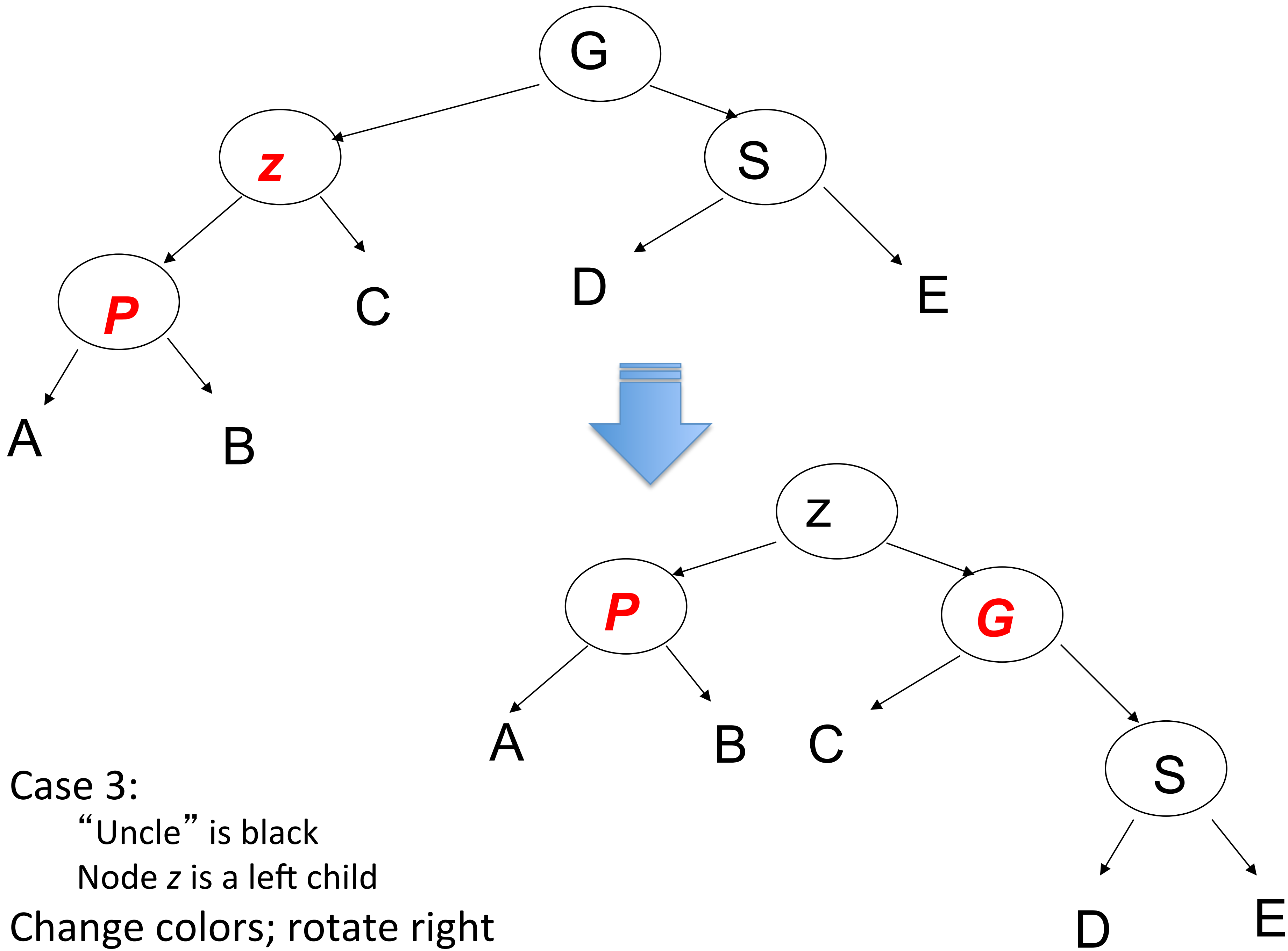
Case 2: Black Unlce

- Case 2:
  - "Uncle" is black
  - Node *z* is a right child
- Transform to case 3 via a left-rotation



Left Rotation





Case 3:

“Uncle” is black

Node *z* is a left child

Change colors; rotate right

Red-Black Tree Visualization

- https://www.cs.usfca.edu/~galles/visualization/RedBlack.html