Red Black Trees

Pseudocode for Operation Insert

Left Rotate

LEFT-ROTATE(T, x)

```
y = x.right                    // set y
x.right = y.left               // turn y's left subtree into x's right subtree
if y.left ≠ T.nil
    y.left.p = x
y.p = x.p                      // link x's parent to y
if x.p == T.nil
    T.root = y
elseif x == x.p.left
    x.p.left = y
else x.p.right = y
y.left = x                     // put x on y's left
x.p = y
```

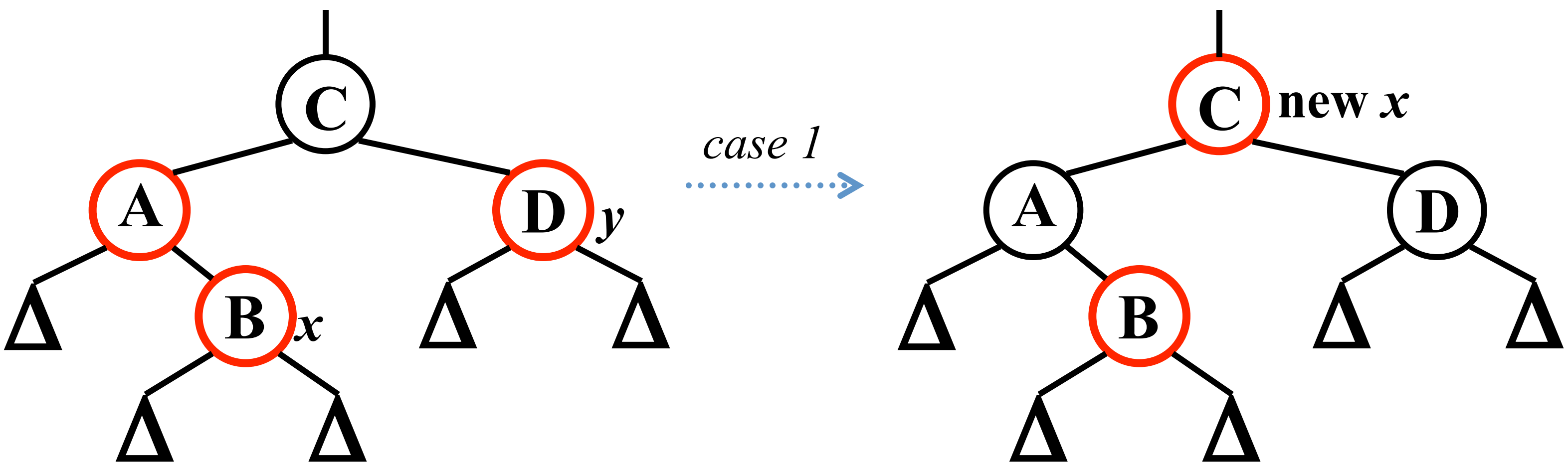
Red-Black Trees: Insertion

- Insertion: the basic idea
  - Insert *x* into tree, color *x* red
  - Only r-b property #3 might be violated (if x.p red)
    - If so, move violation up tree until a place is found where it can be fixed
  - Total time will be O(log *n*)

RB Insert: Case 1

```
if (y.color == RED)
    x.p.color = BLACK;
    y.color = BLACK;
    x.p.p.color = RED;
    x = x.p.p;
```

- Case 1: "uncle" is red
- In figures below, all Δ's are equal-black-height subtrees

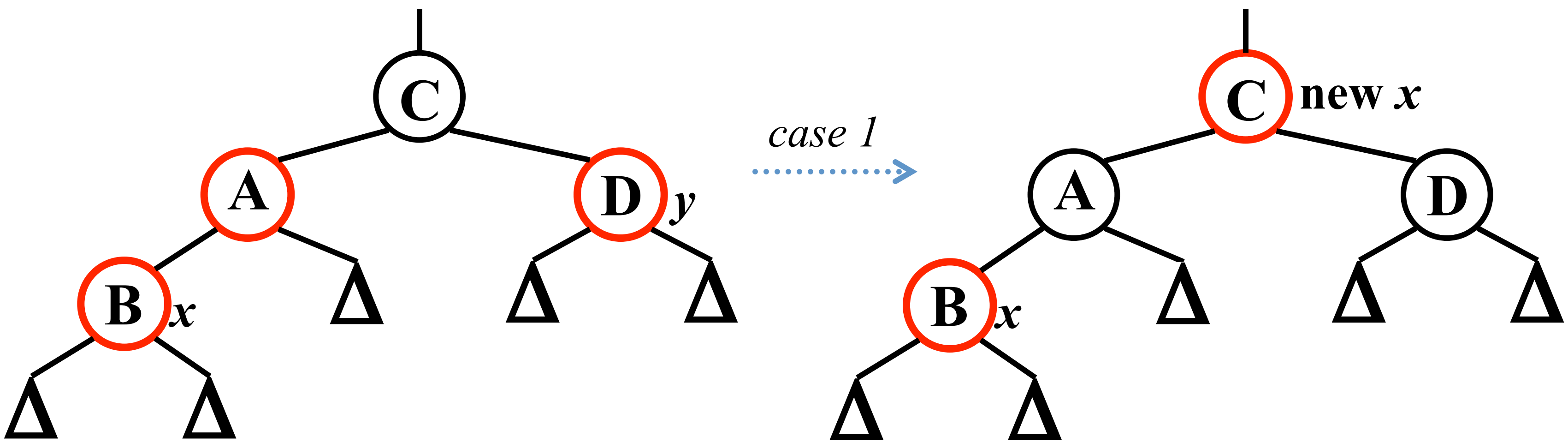


*Change colors of some nodes, preserving #4: all downward paths have equal **bh**.*
The while loop now continues with x's grandparent as the new x

RB Insert: Case 1's symmetrical

```
if (y.color == RED)
    x.p.color = BLACK;
    y.color = BLACK;
    x.p.p.color = RED;
    x = x.p.p;
```

- Case 1: "uncle" is red
- In figures below, all Δ's are equal-black-height subtrees

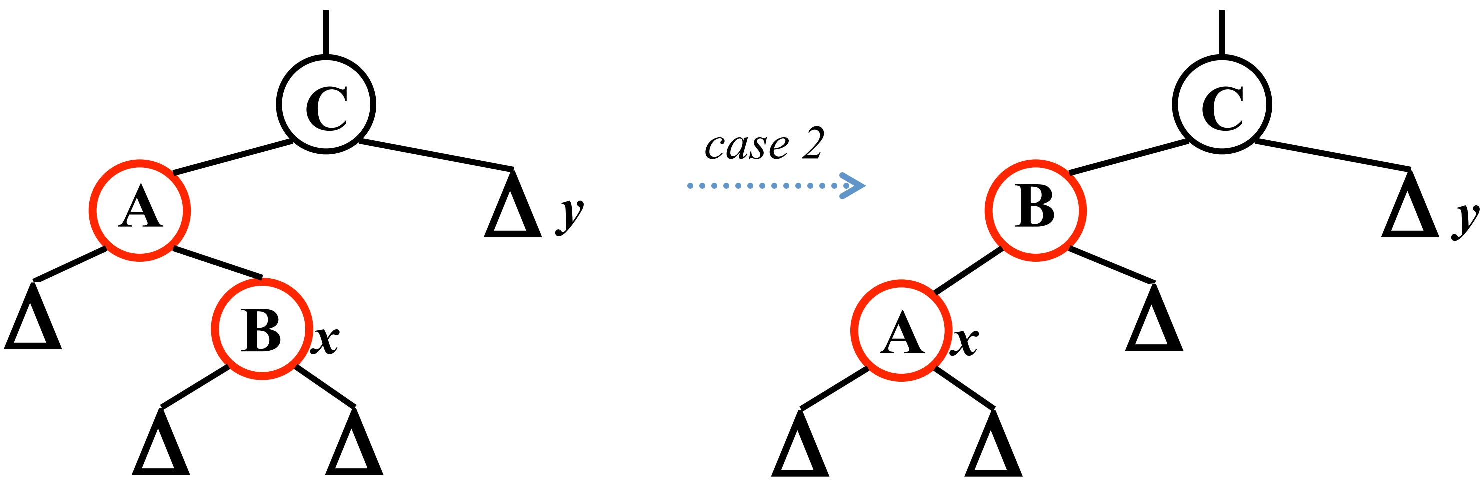


Same action whether x is a left or a right child

RB Insert: Case 2

```
if (x == x.p.right)
    x = x.p;
    leftRotate(x);
// continue with case 3 code
```

- Case 2:
  - “Uncle” is black
  - Node *x* is a right child
- Transform to case 3 via a left-rotation

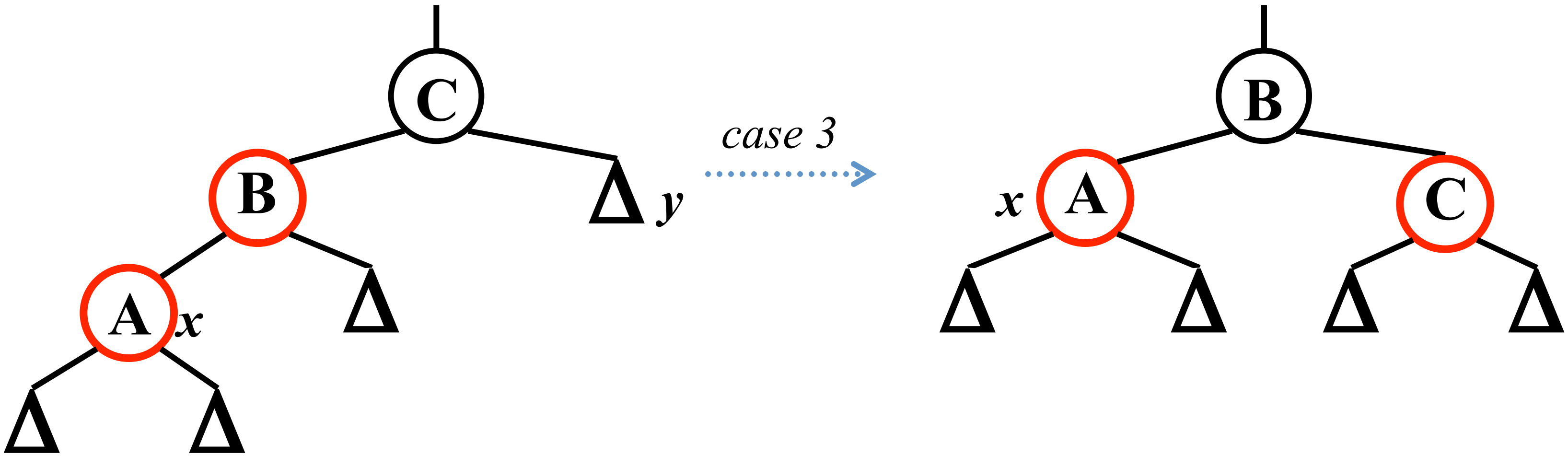


Transform case 2 into case 3 (x is left child) with a left rotation
This preserves property# 4: all downward paths contain same number of black nodes

RB Insert: Case 3

```
x.p.color = BLACK;
x.p.p.color = RED;
rightRotate(x.p.p);
```

- Case 3:
  - “Uncle” is black
  - Node *x* is a left child
- Change colors; rotate right



Perform some color changes and do a right rotation

Again, preserves property #4: all downward paths contain same number of black nodes

RB Insert: Cases 4-6

- Cases 1-3 hold if *x*'s parent is a left child
- If *x*'s parent is a right child, cases 4-6 are symmetric (swap left for right)

```
Rb-Insert(z)
  BST-Insert(z);
  z.color = RED;
  // Move violation of #3 up tree, maintaining #4 as invariant:

while z.p.color == RED
    if z.p == z.p.p.left
        y = z.p.p.right
        if y.color == RED
            z.p.color = BLACK                    // case 1
            y.color = BLACK                      // case 1
            z.p.p.color = RED                    // case 1
            z = z.p.p                            // case 1
        else if z == z.p.right
                z = z.p                          // case 2
                LEFT-ROTATE(T, z)                // case 2
            z.p.color = BLACK                    // case 3
            z.p.p.color = RED                    // case 3
            RIGHT-ROTATE(T, z.p.p)               // case 3
    else (same as then clause
            with "right" and "left" exchanged)
```