Chapter 13. Red Black Trees

Introduction to Algo.

~~Recall~~.

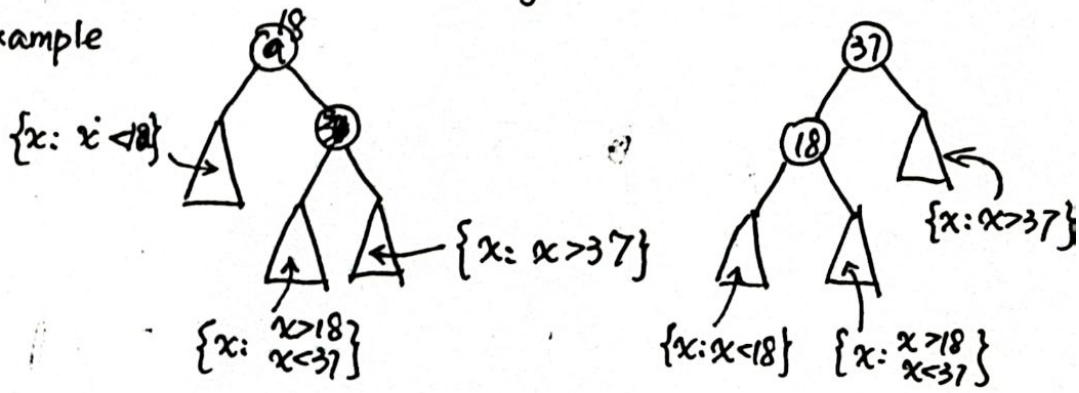
§ 13.1 Structure to keep balance

Recall. $\boxed{\text{AVL Tree}}$

- Assume we have an inbalanced BST, try to fix it.

▷ By BST's property; there are 2 types.

for example



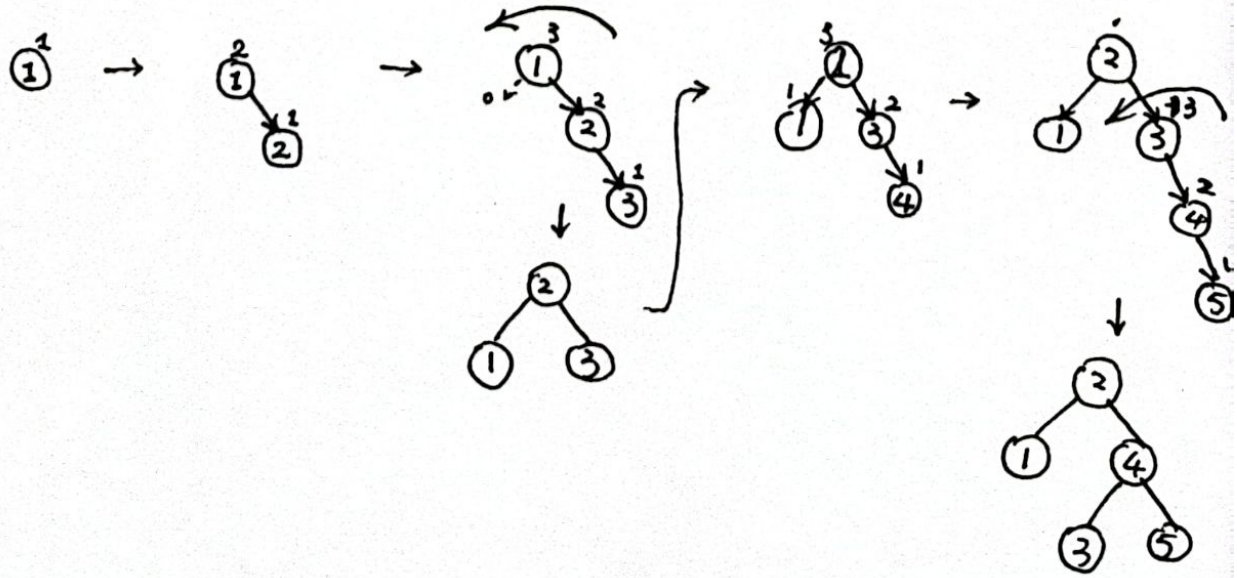
▷ How to balance?

- Balanced ≠ every level full
- AVL trees denote balanced by the ~~diff~~ $\boxed{\text{height}}$ diff of left subtree and right subtree is at most 1.

  distance to furthest node

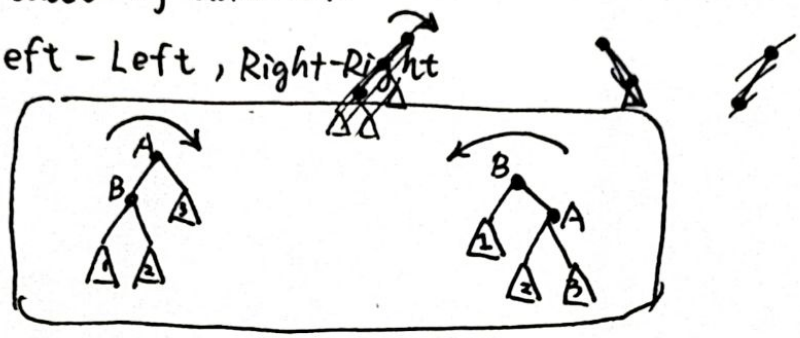
- Rotation will be performed to make a deeper subtree more shallow.

▷ Example. Add 1, 2, 3, 4, 5.



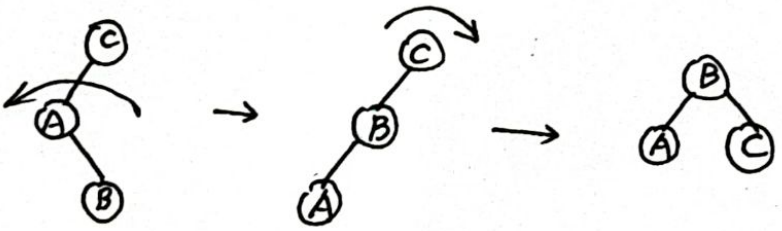
▷ Four cases of add ~~and~~

- Left - Left , Right-Right

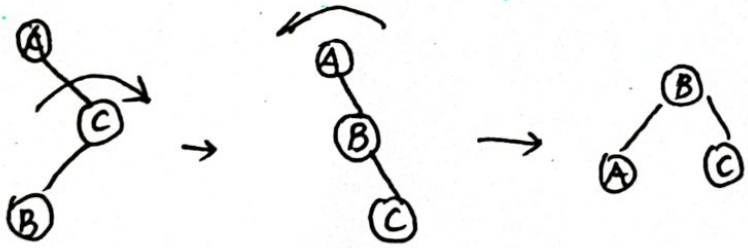


3 nodes forms a chain. Single rotation.

- Left-Right / Right-Left rotation



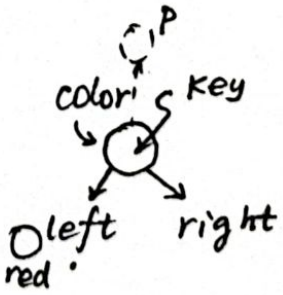
Left-Right



Right-left.

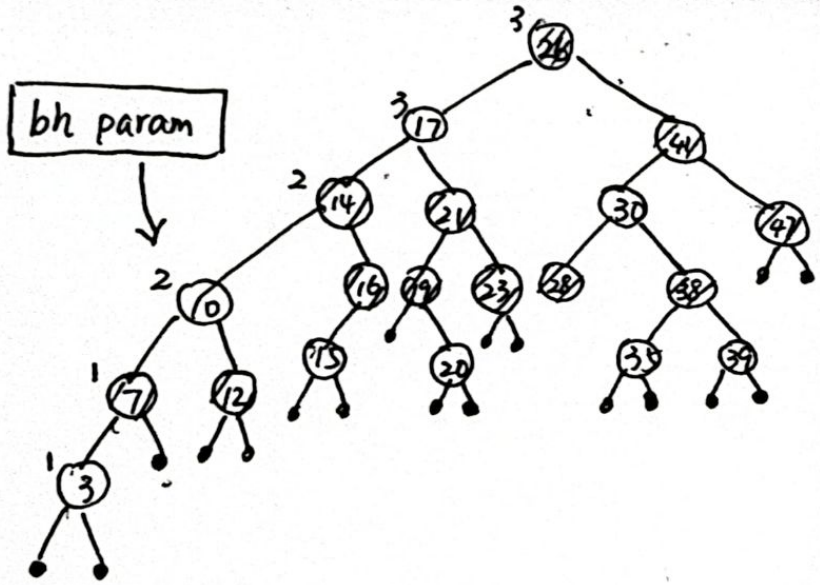
▷ Deletion. Perform as normal BST, then rotate.

Red Black Tree



1. Red-black property:
   - Every node is either ● Black or ○ red.
   - The root is ●.
   - every leaf (NIL) is ●
   - if a node is red, both its children is ● black.
   - for each node, all simple paths from the node to decent leaves contains same no. of black nodes.

That is: insert at most 2n of depth (inbalance). instead of AVL's diff factor 1.

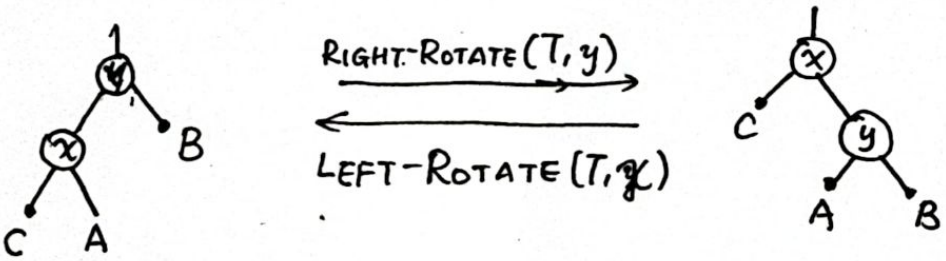


- at least $h/2$,

  $n \geq 2^{\frac{h}{2}} - 1$.

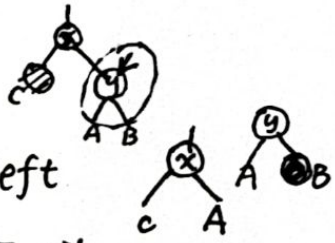
  $\Rightarrow h \leq 2\lg(n+1)$.

2. Rotation to fix ~~inequality~~ inbalance



LEFT-ROTATE

```
y = x.right
x.right = y.left
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
elif 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.
```



3. On insert

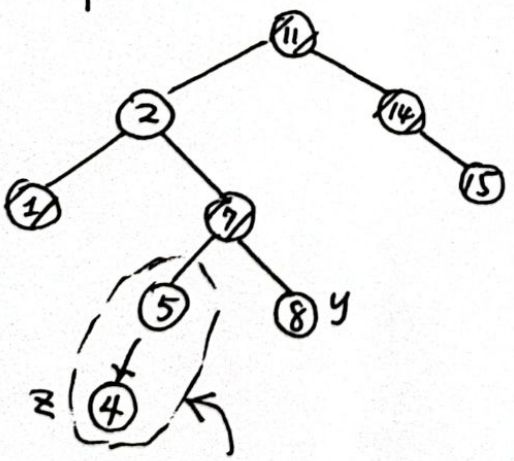
```
RB-INSERT(T, z)
  y = T.nil
  x = T.root
  while x ≠ T.nil
      y = x
      if z.key < x.key
          x = x.left
      else x = x.right
  z.p = y
  if y = T.nil
      T.root = z
  else if z.key < y.key
      y.left = z
  else y.right = z
  z.left = T.nil
  z.right = T.nil
  z.color = RED
  RB-INSERT-FIXUP(T, z)
```

Normal BST insertion

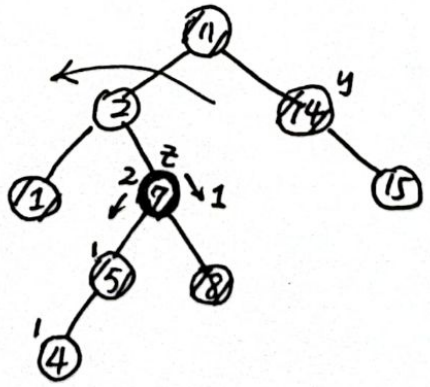
initinal as red node

How do we fix the coloring?

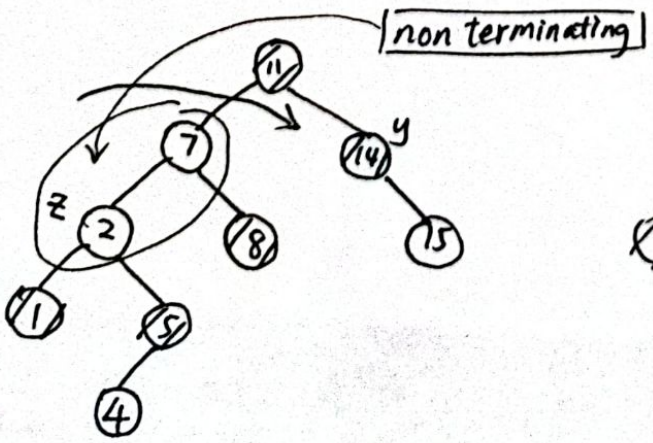
Example.

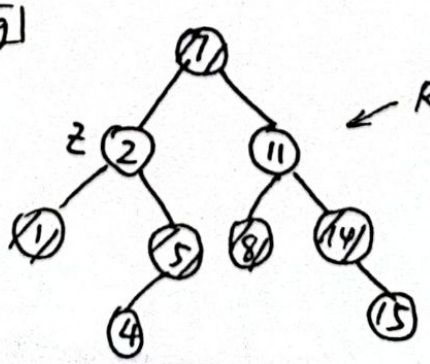


2 reds in a row



Two red, black split





Recoloring

Observation: To fix 2 nodes in a row, maintain bh.

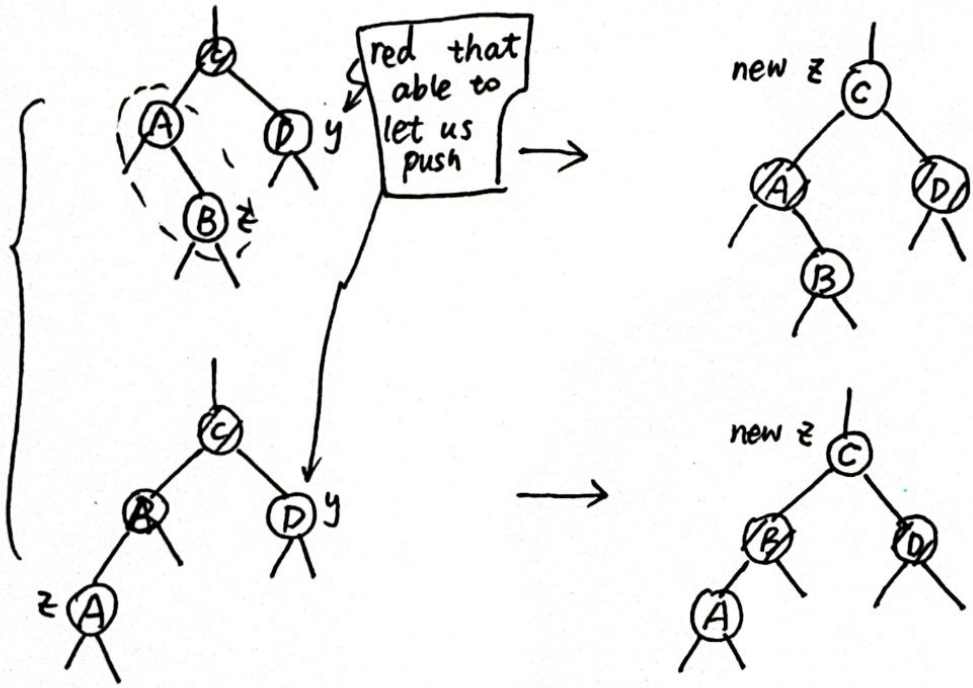
- Recoloring : 
- Rotation : includes simbling

~~In each fix step~~

Look at the small cases : See At most 2 <u>red in a row</u>

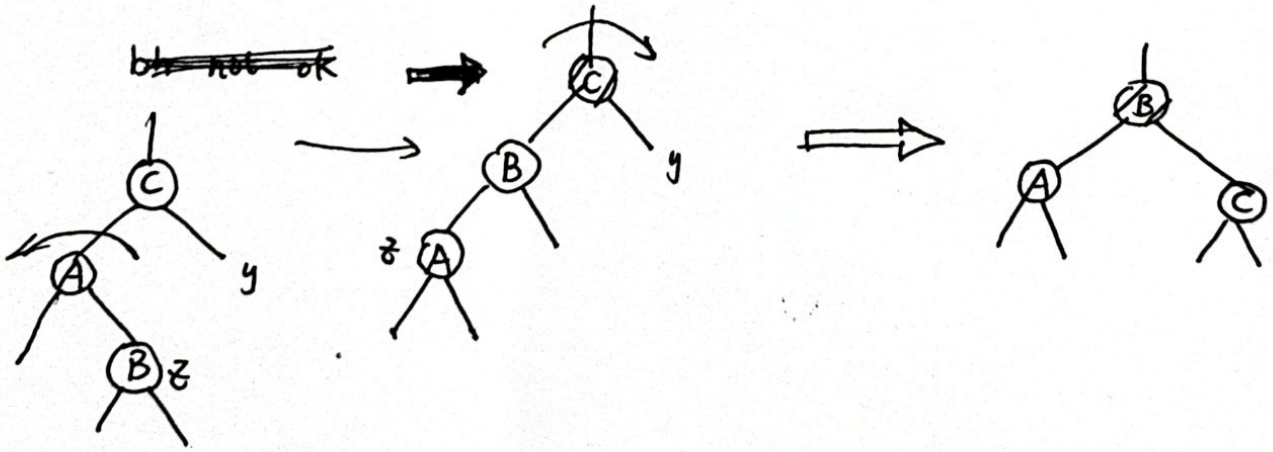
(we will see at the end)

bh ok



CASE 1.
z's uncle y is red

~~bh not ok~~



CASE 2. z's uncle y is black and z is a right child

CASE 3. z's uncle y is black and z is a left child

<u>Time: $\Theta(n)$.</u>

RB-INSERT-FIXUP

```
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
            y.color ← Black
            z.p.p.color ← Red
        elif z = z.p.right
            z ← z.p
            LEFT-ROTATE(T, z)
            z.p.color = black
            z.p.p.color = red
            RIGHT-ROTATE(T, z.p.p)
    else  same as ↑ with right and left exchanged.
```

Case 1

Case 2

Case 3