AVL Tree

Monday, 4 September 2023 8:29 AM

AVL Trees

AVL → Has ① search invariant, ② height balanced

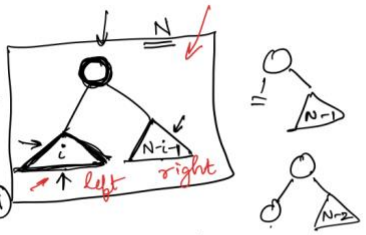
BST → Add Del Search

Avg case log(n) log(n) log(n)

→ Proof for avg. O(log n) in BST ✓

Let $D(N) = \sum_{i=1}^{N} d(i)$ = sum of depth of nodes in a tree (d(i): node)

Also, $D(N) = \underbrace{D(i)}_{\text{left subtree}} + \underbrace{D(N-i-1)}_{\text{right subtree}} + (N-1)$ — ①



avg: $D'(i) = \frac{1}{N} \sum_{j=0}^{N-1} D(j)$ — ② (sum)

→ we consider all values of i are equally likely

avg: $D'(N-i-1) = \frac{1}{N} \sum_{j=0}^{N-1} D'(j)$ — ③

$D(0) + D(1) + D(2) + \dots = \sum_{i=0}^{2} d(i)$

$D'(N) = \frac{2}{N} \sum_{0}^{N-1} D'(j) + (N-1)$ — ④

$N\,D'(N) = 2 \sum_{j=0}^{N-1} D'(j) + N(N-1)$

$(N-1)\,D'(N-1) = 2 \sum_{j=0}^{N-2} D'(j) + (N-1)(N-2)$

$N\,D'(N) - (N-1)\,D'(N-1) = 2\,D'(N-1) + 2(N-1)$

$N\,D'(N) = (N+1)\,D'(N-1) + 2(N-1)$ — ⑤

both sides

$N\,D'(N) - (N-1)\,D'(N-1) = 2\,D(N-1)$

$$N\,D'(N) = (N+1)\,D'(N-1) + 2(N-1) \quad \text{— (5)}$$

divide by $N(N+1)$ both sides

$$\frac{D'(N)}{N+1} = \frac{D'(N-1)}{N} + \frac{2(N-1)}{N(N+1)} \quad \text{— (6)}$$

$$\frac{D'(N-1)}{N} = \frac{D'(N-2)}{N-1} + \frac{2(N-2)}{(N-1)N}$$

$$\frac{D(N-2)}{N-1} = \frac{D(N-3)}{N-2} + \frac{2(N-3)}{(N-2)(N-1)}$$

$$= 2\sum \frac{N}{(N+1)(N+2)}$$

$$= 2\sum \left[\frac{2}{(N+2)} - \frac{1}{(N+1)}\right] = \frac{N}{(N+1)(N+2)}$$

$$= 2\left\{\sum \frac{2}{N+2} - \sum \frac{1}{N+1}\right\}$$

$$\frac{D(N)}{(N+1)} = 2\{2\ln(N) - \ln(N)\}$$

$$\sum \frac{1}{N}$$

$$\int \frac{1}{N} = \ln N$$

$$N\,D(N) = \log(N)$$

(N+1)

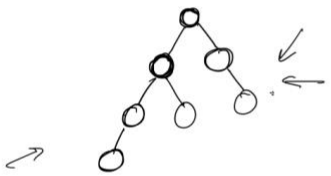
$\frac{D(N)}{(N+1)} \approx 2 \log(N)$

AVL Tree → balanced BST → worst case $O(\log N)$

Let N_h ⇒ minimum number of nodes in an AVL Tree of height h

N = 4

h	N_h
0	1
1	2
2	4
3	7



recursion

$N_h = N_{h-1} + N_{h-2} + 1$

$T(N) = T(N-1) + T(N-2) + 1$ → $2^{K/2}$ → 2^K

$N_h > 2^{h/2}$ or $N_h < 2^h$

$h/2 < \log_2 N_h$

$h < 2 \log_2 N_h$ → $\log N$

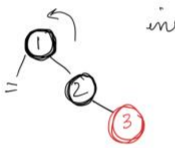
$O(\log N)$

$N_h > \frac{\phi^h}{\sqrt{5}}$ $\quad$ $h < 1.440 \log_2 N$

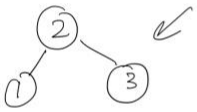
→ Insertion in AVL Tree

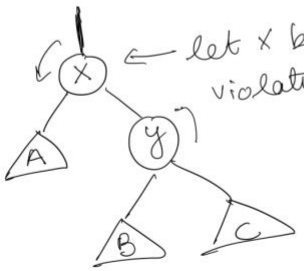
Rotations

left heavy
right heavy



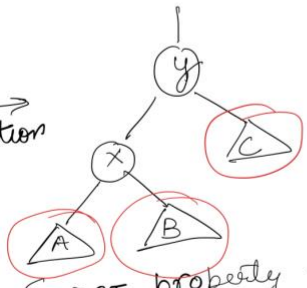
insert (3)
left rotation →





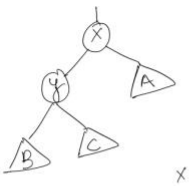
← let X be where violation occurs

left rotation →

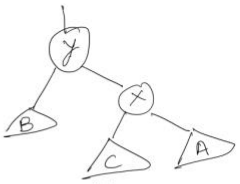


BST property is intact

[X → right = Y → left
Y → left = X]



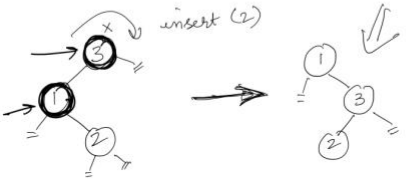
right rotation →

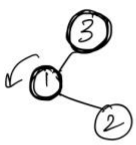


X → left = y → right
y → right = X

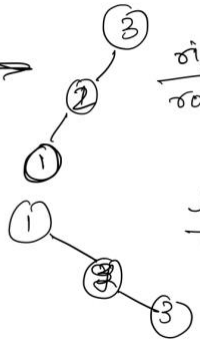
Zig Zag

insert (2)

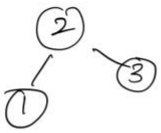


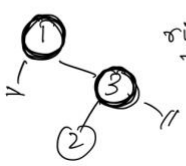


left rotation (1) →



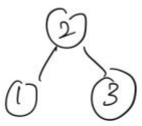
right rot (3) →

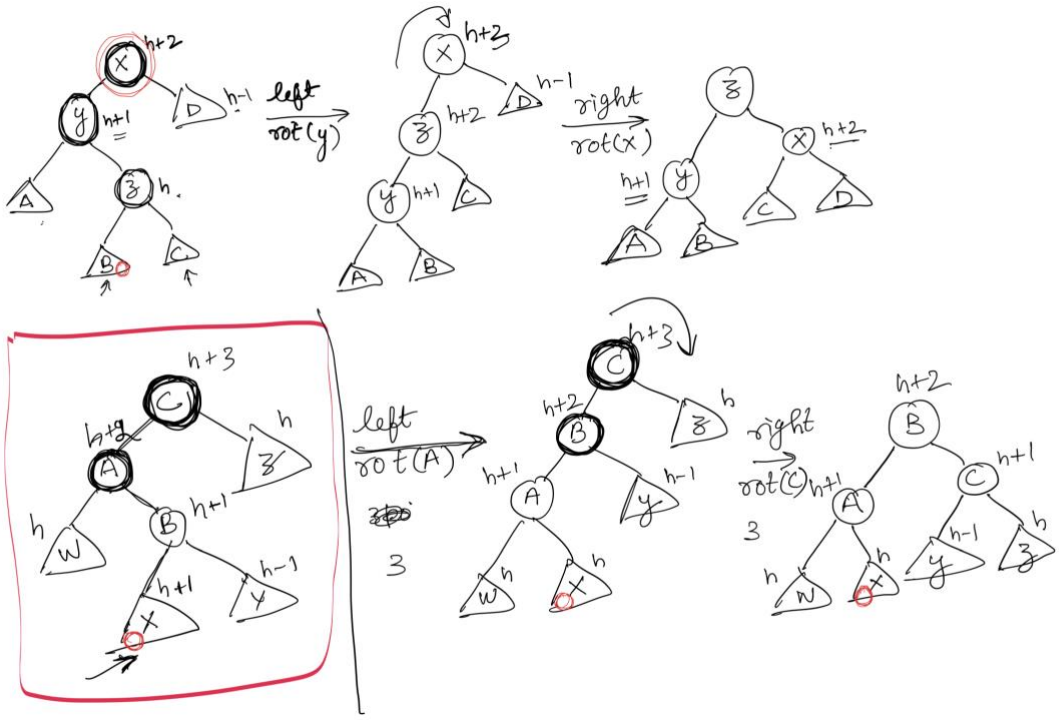




right rot (3) →

left rot (1) →




left
rot(y)
right
rot(x)
left
rot(A)
right
rot(C)