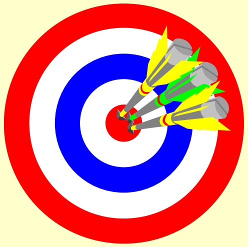
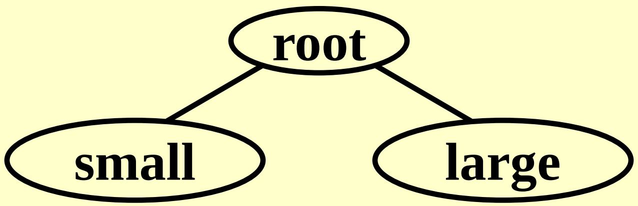
AVL Trees, Splay Trees, and Amortized Analysis

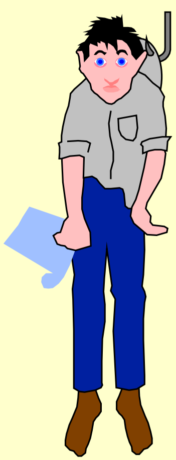
AVL Trees



Target : **Speed up searching (with insertion and deletion)**

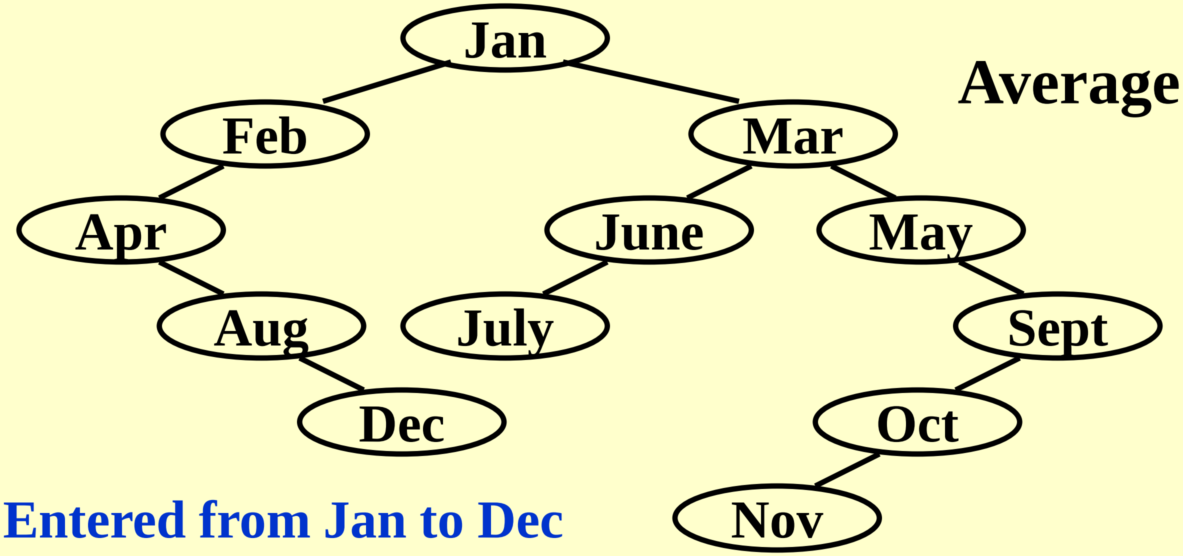
Tool : **Binary search trees**





Problem : **Although T_p = O(height), but the height can be as bad as O(N).**

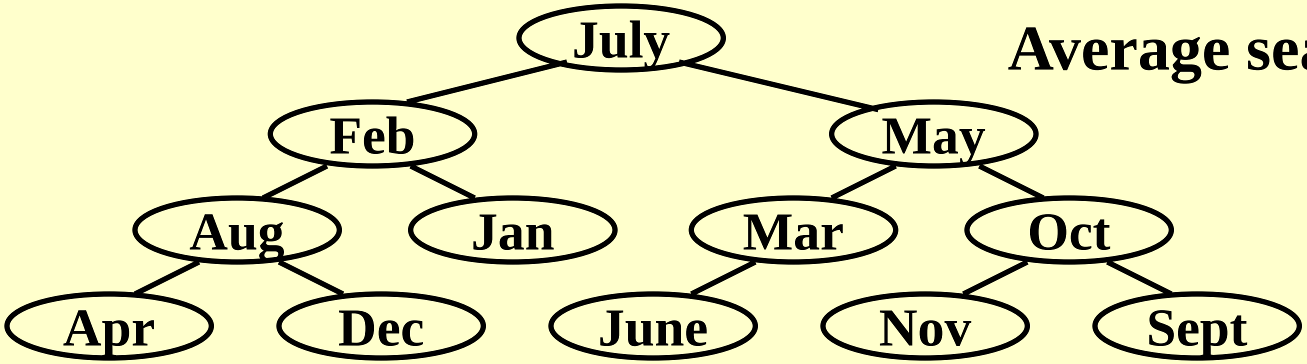
〖Example〗 2 binary search trees obtained for the months of the year



Average search time = 3.5

Average search time of the skew tree = 6.5

Entered from Jan to Dec



Average search time = 3.1

A balanced tree

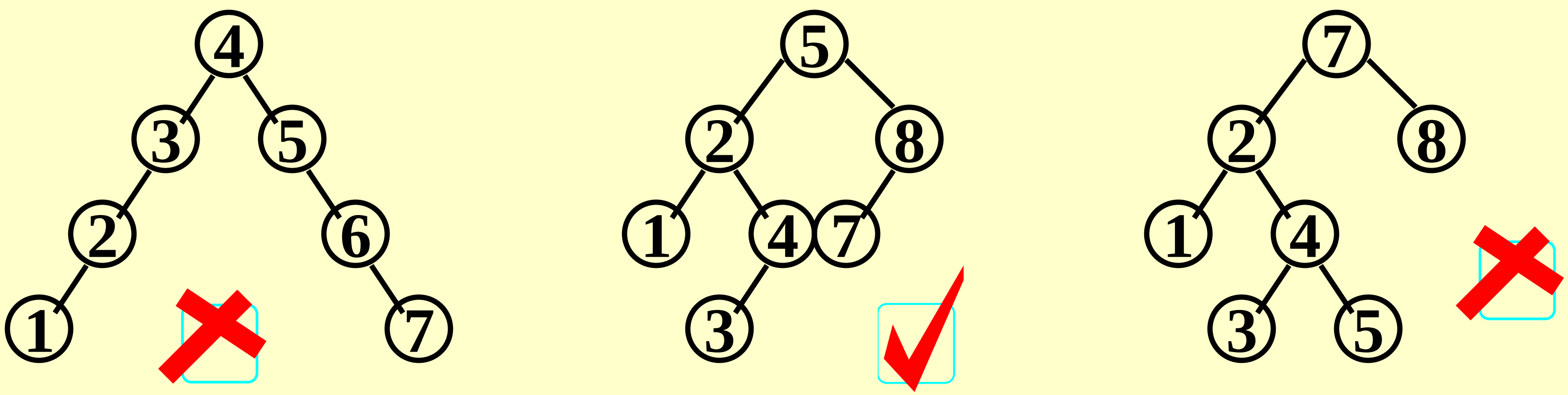
Adelson-Velskii-Landis (AVL) Trees (1962)

【**Definition**】**An empty binary tree is height balanced. If T is a nonempty binary tree with T_L and T_R as its left and right subtrees, then T is height balanced iff**

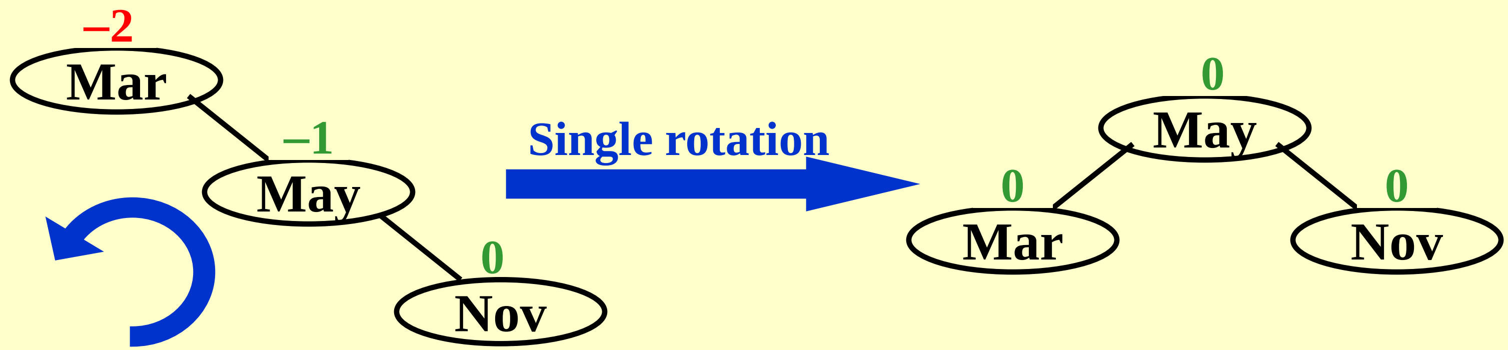
(1) T_L and T_R are height balanced, and

(2) $| h_L - h_R | \le 1$ where h_L and h_R are the heights of T_L and T_R, respectively.

【**Definition**】**The balance factor $BF(\text{ node }) = h_L - h_R$. In an AVL tree, $BF(\text{ node }) = -1, 0,$ or 1.**

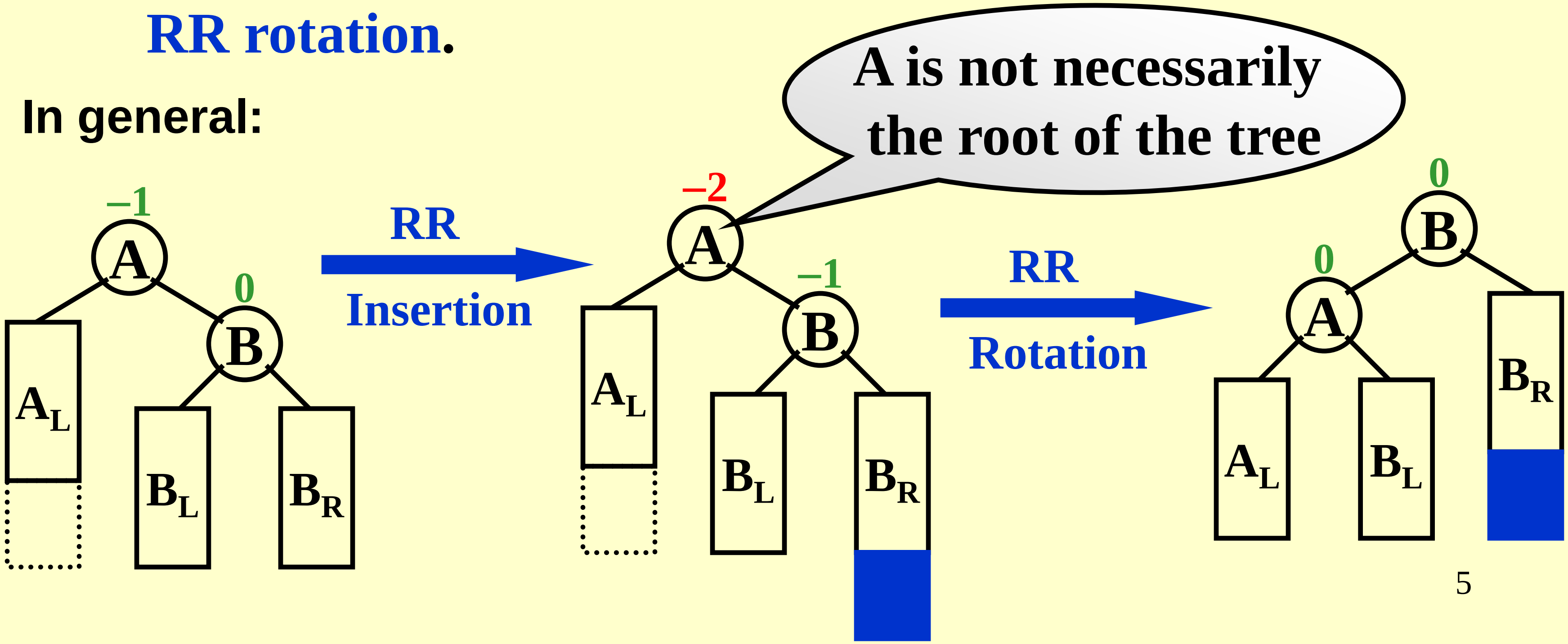


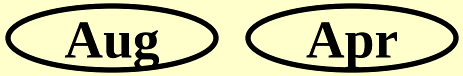
〖Example〗 Input the months Mar May Nov

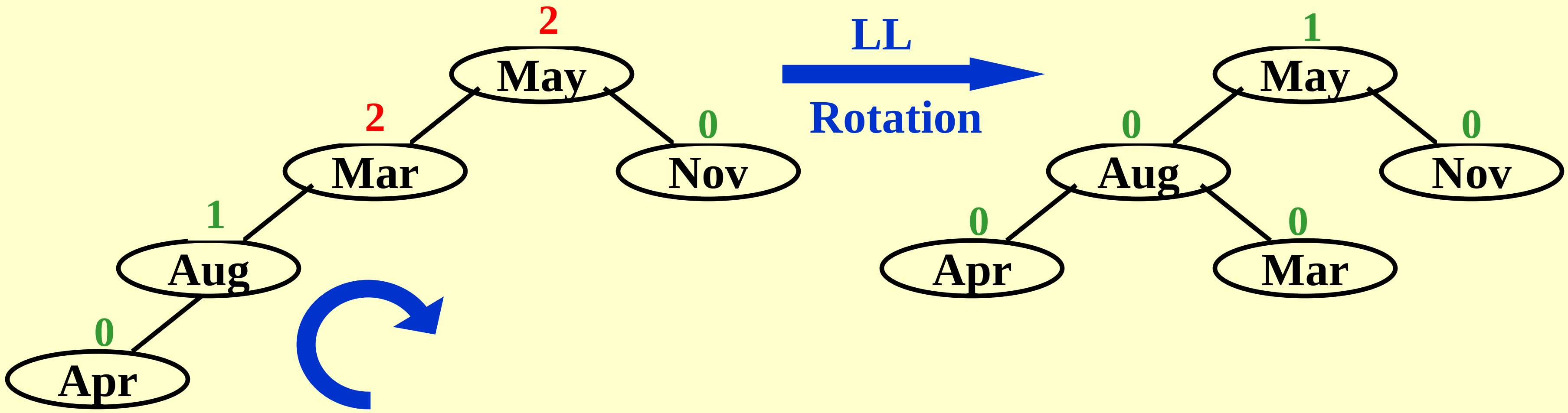


The trouble maker Nov is in the right subtree's right subtree of the trouble finder Mar. Hence it is called an RR rotation.

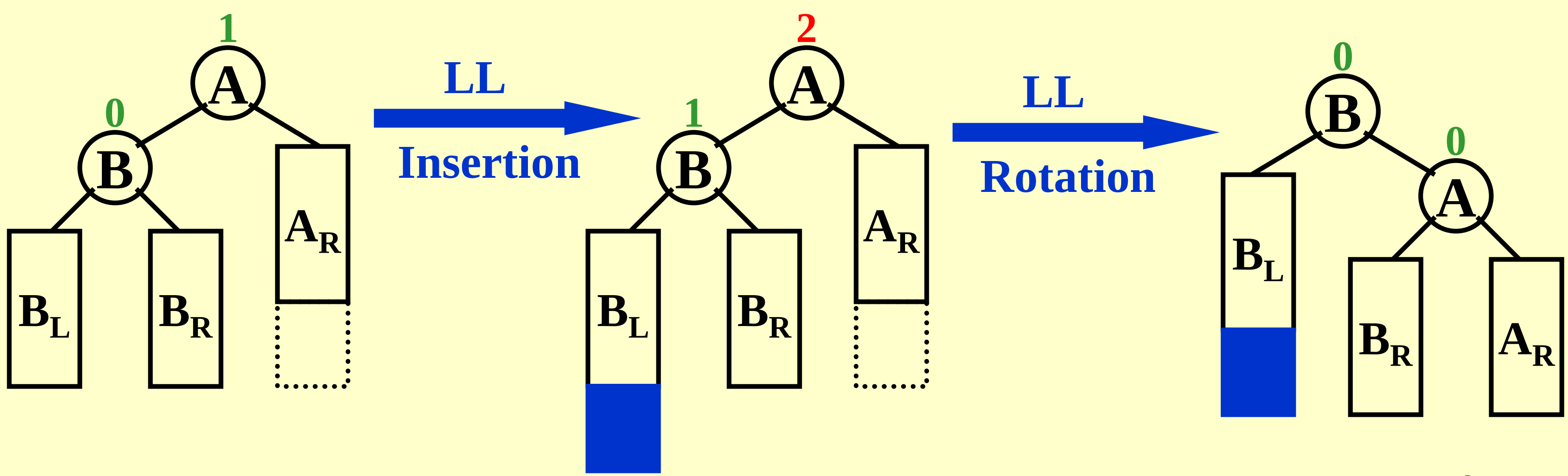
In general:



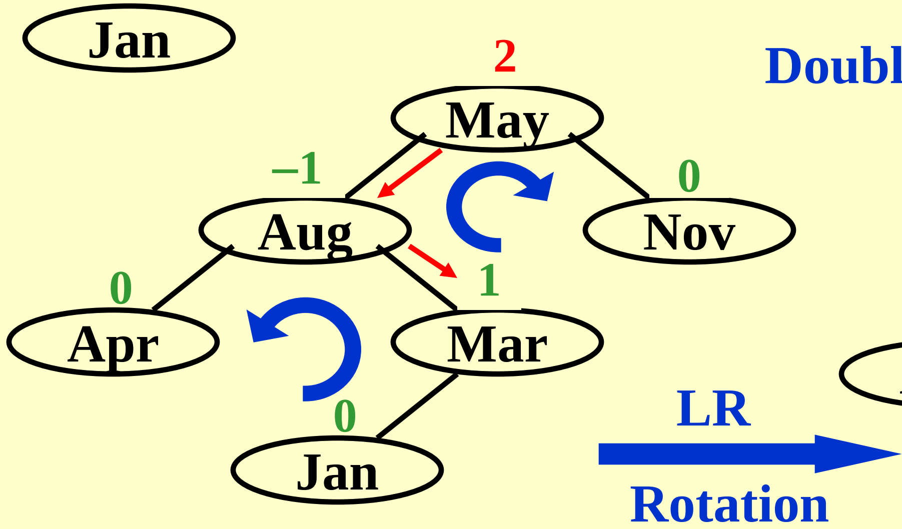

Aug
Apr

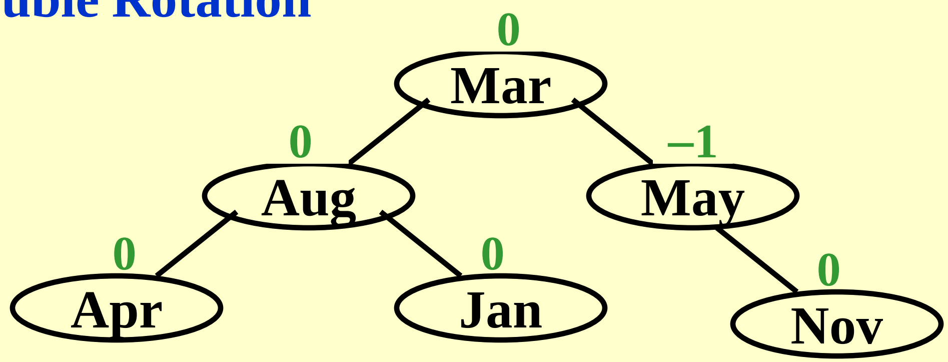

2
May
2
Mar
0
Nov
1
Aug
0
Apr
LL
Rotation
1
May
0
Aug
0
Nov
0
Apr
0
Mar

In general:

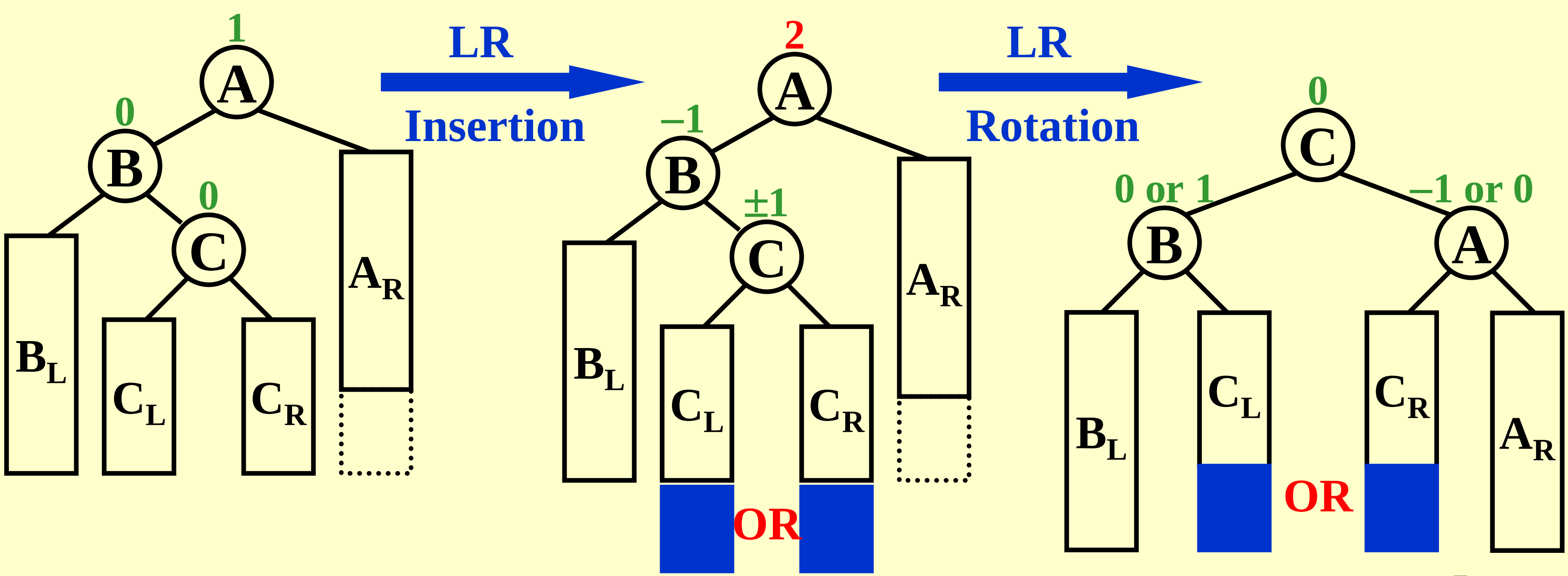

1
A
0
B
A_R
B_L
B_R
LL
Insertion
2
A
1
B
A_R
B_L
B_R
LL
Rotation
0
B
0
A
B_L
B_R
A_R

Double Rotation

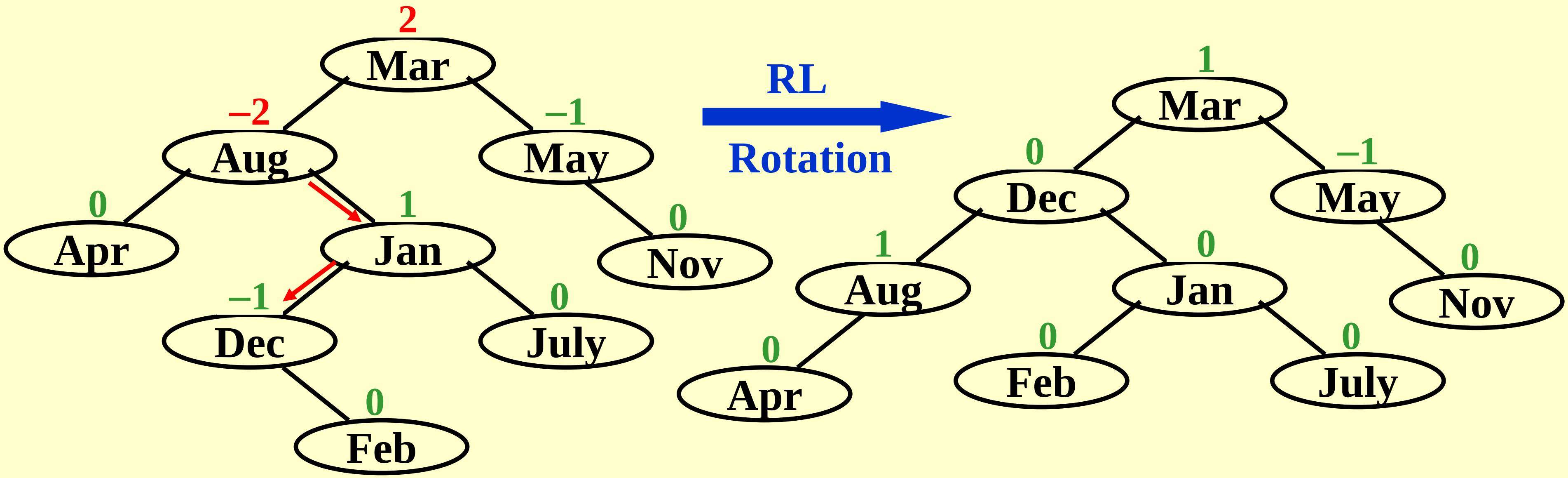




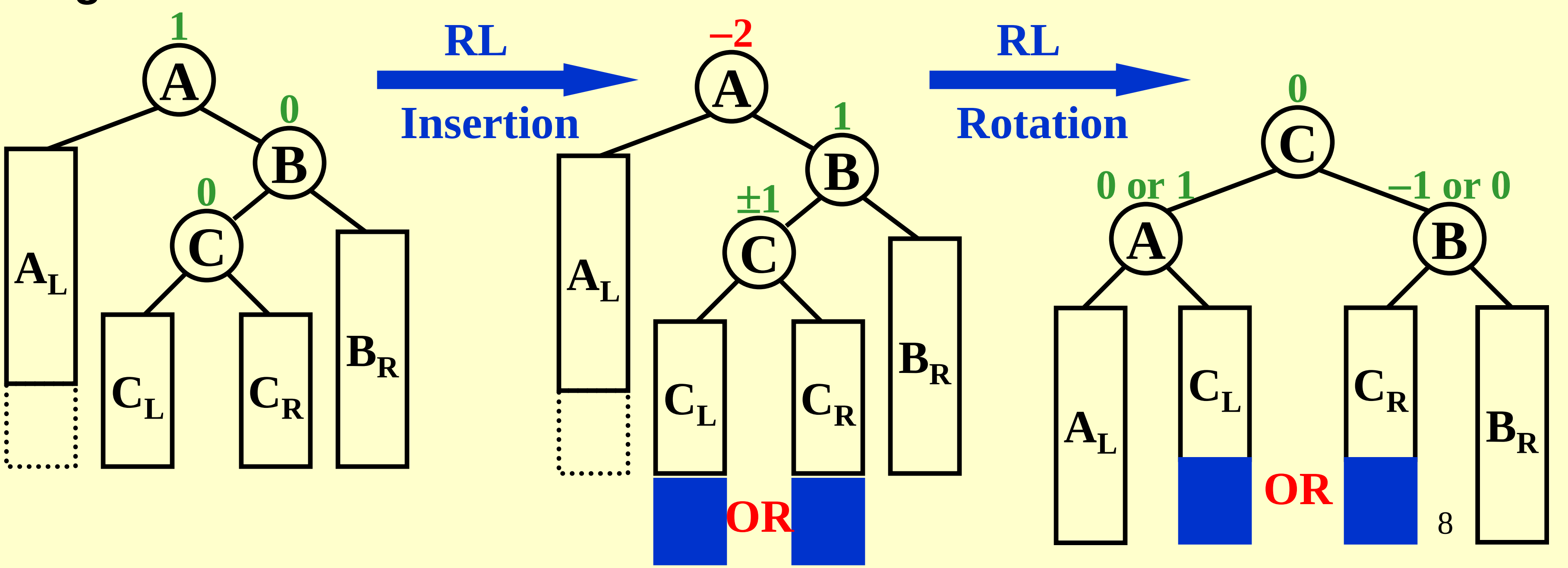
In general:

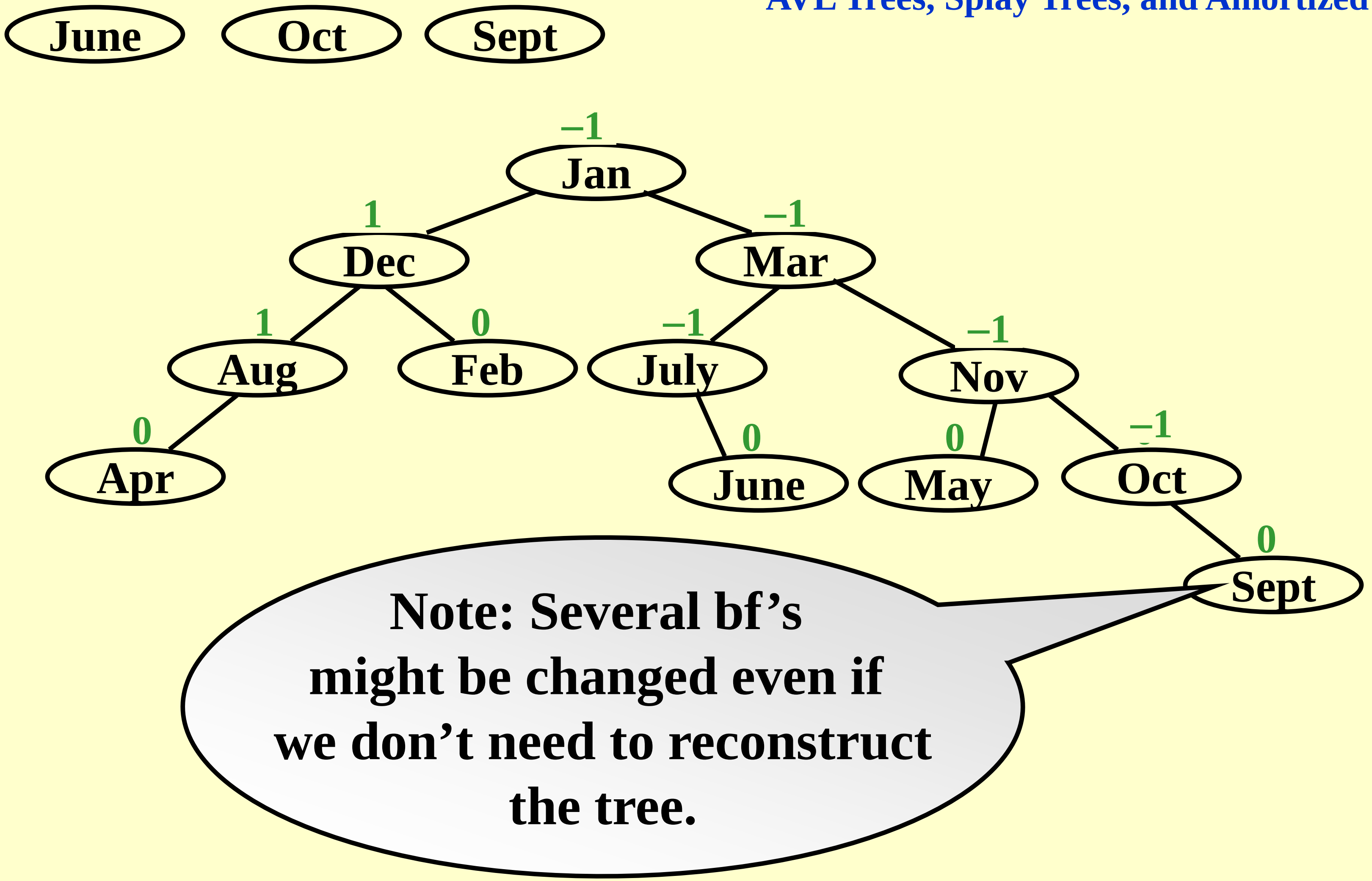



Dec
July
Feb


Mar
2
Aug
–2
May
–1
Apr
0
Jan
1
Nov
0
Dec
–1
July
0
Feb
0
RL
Rotation
Mar
1
Dec
0
May
–1
Aug
1
Jan
0
Nov
0
Apr
0
Feb
0
July
0

In general:

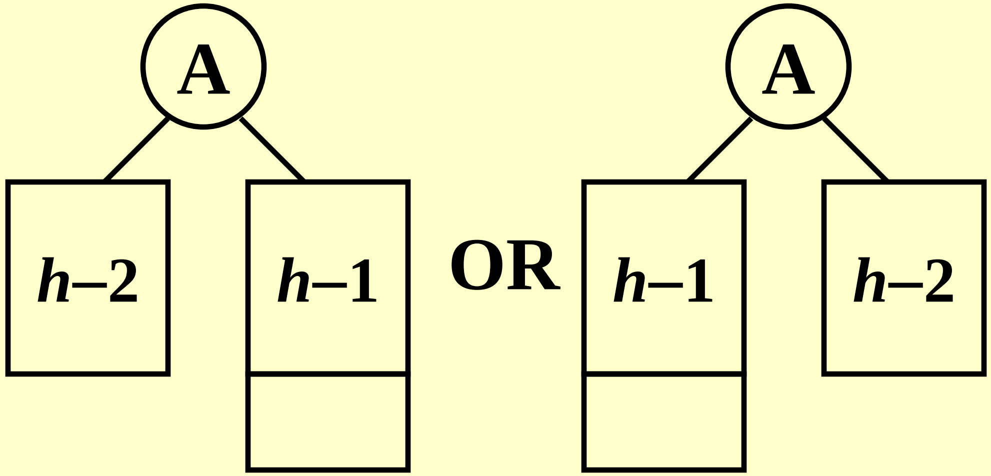

A
1
B
0
C
0
A_L
C_L
C_R
B_R
RL
Insertion
A
–2
B
1
C
±1
A_L
C_L
C_R
B_R
OR
RL
Rotation
C
0
A
0 or 1
B
–1 or 0
A_L
C_L
C_R
B_R
OR



Another option is to keep a *height* field for each node.

Read the declaration and functions in [1] Figures 4.42 – 4.48

Let n_h be the minimum number of nodes in a height balanced tree of height h. Then the tree must look like



$$\Rightarrow \quad n_h = n_{h-1} + n_{h-2} + 1$$

Fibonacci numbers:

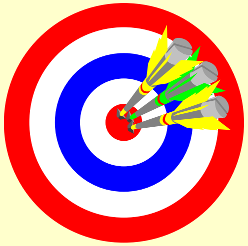
$$F_0 = 0,\ F_1 = 1,\ F_i = F_{i-1} + F_{i-2} \quad \text{for } i > 1$$

$$\Rightarrow \quad n_h = F_{h+2} - 1, \text{ for } h \geq 0$$

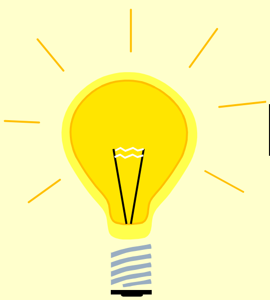
Fibonacci number theory gives that $F_i \approx \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^i$

$$\Rightarrow \quad n_h \approx \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{h+2} - 1 \qquad \Rightarrow \quad h = O(\ln n)$$

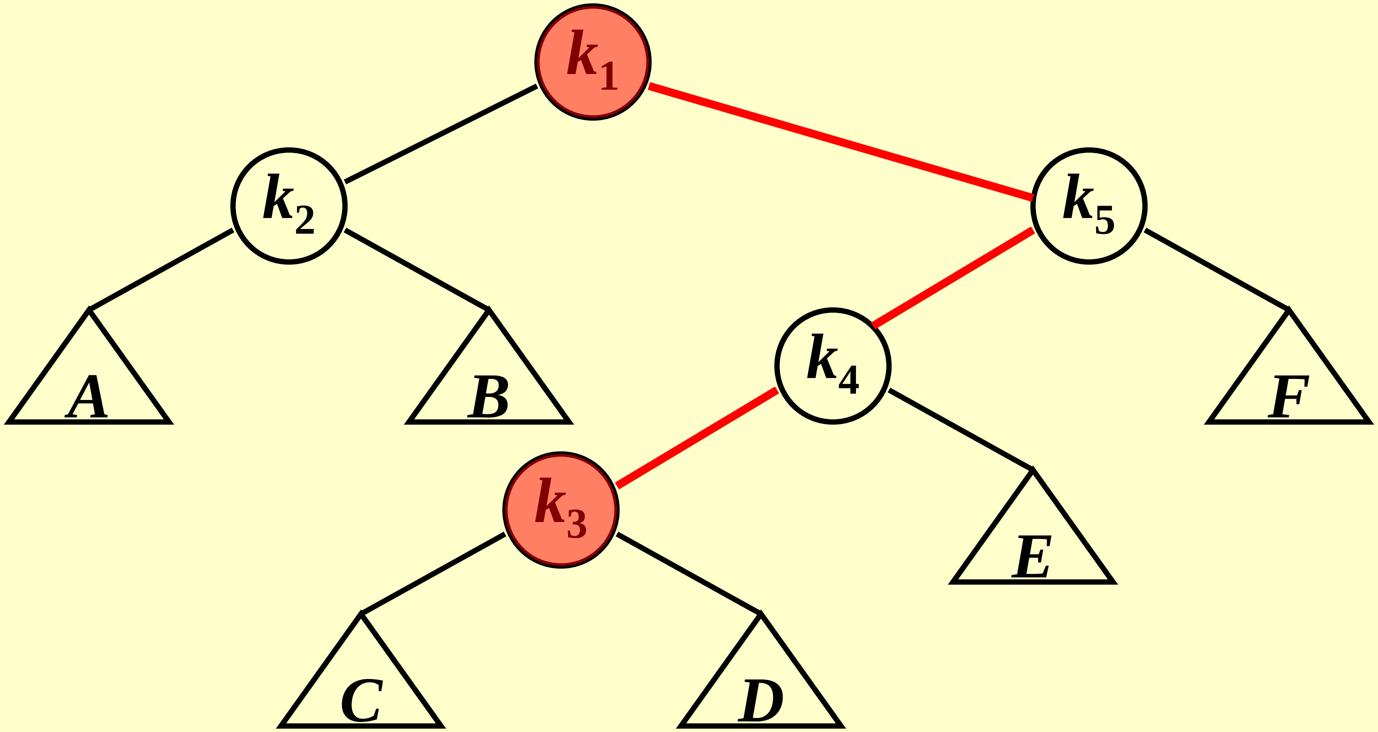
Splay Trees

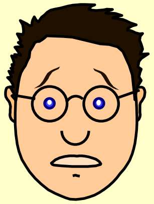


Target : **Any *M* consecutive tree operations starting from an empty tree take at most O(*M* log *N*) time.**



Idea : **After a node is accessed, it is pushed to the root by a series of AVL tree rotations.**

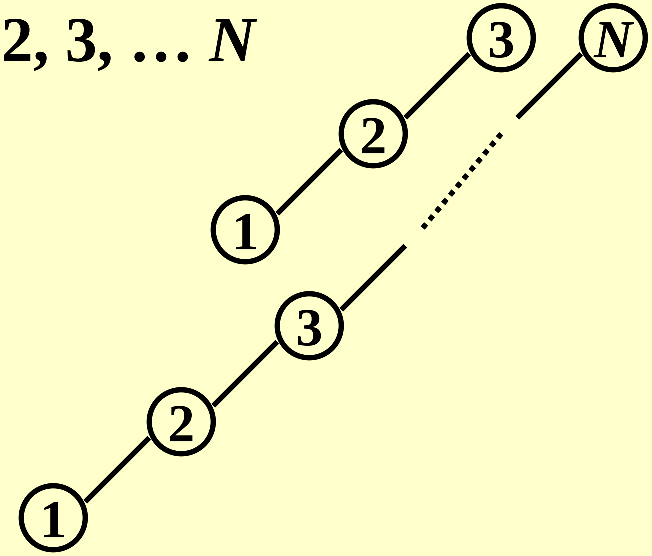




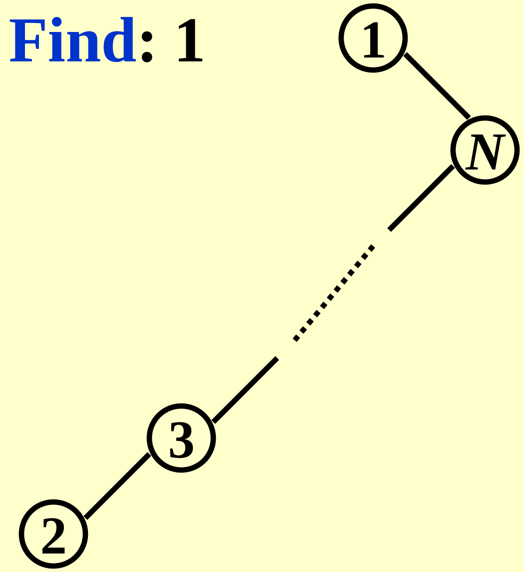
Does NOT work!

An even worse case:

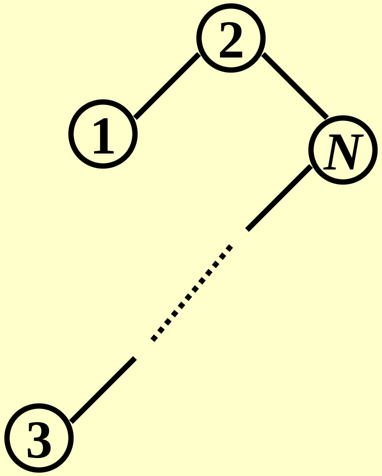
Insert: 1, 2, 3, … N



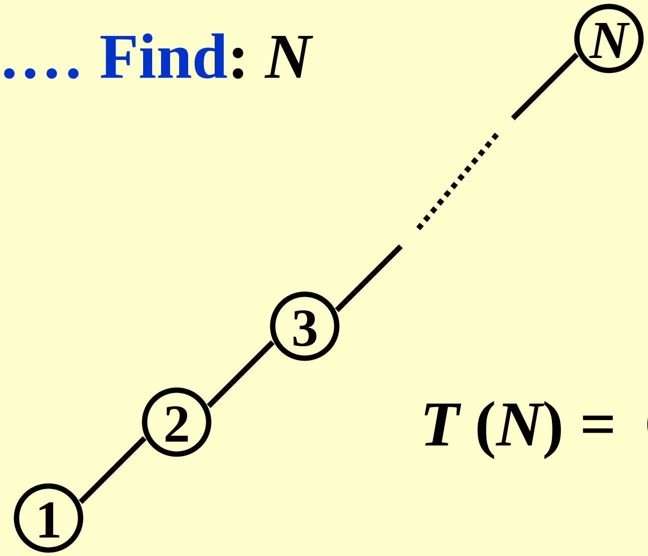
Find: 1



Find: 2



…… Find: N



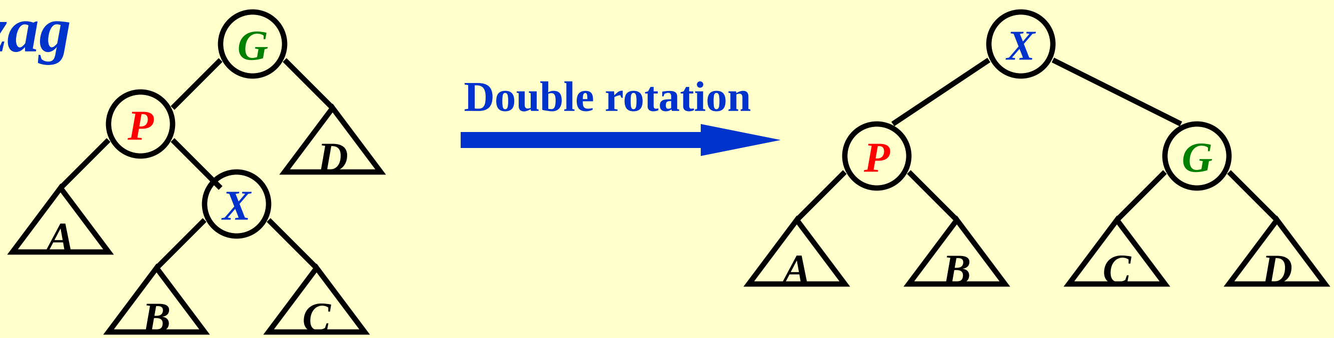
$$T(N) = O(N^2)$$

Try again -- For any nonroot node *X* , denote its parent by *P* and grandparent by *G* :

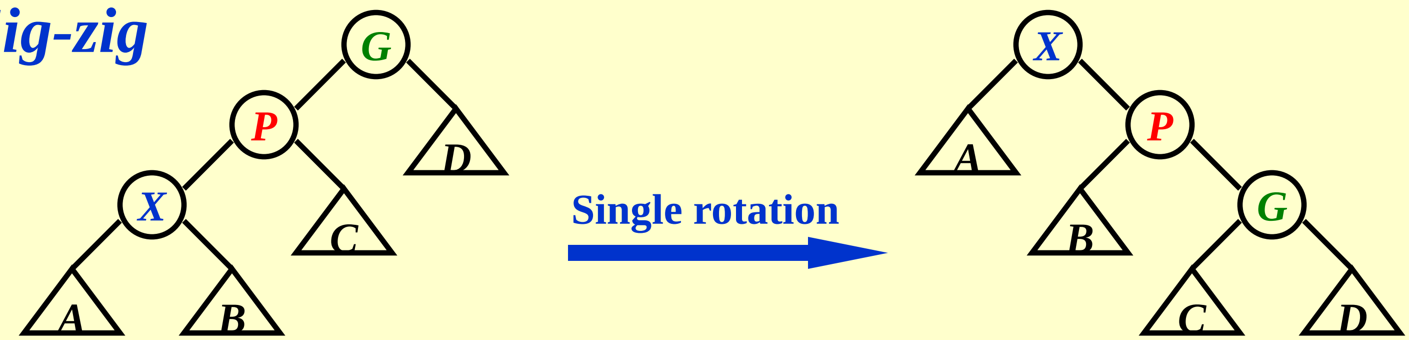
Case 1: *P* is the root → Rotate *X* and *P*

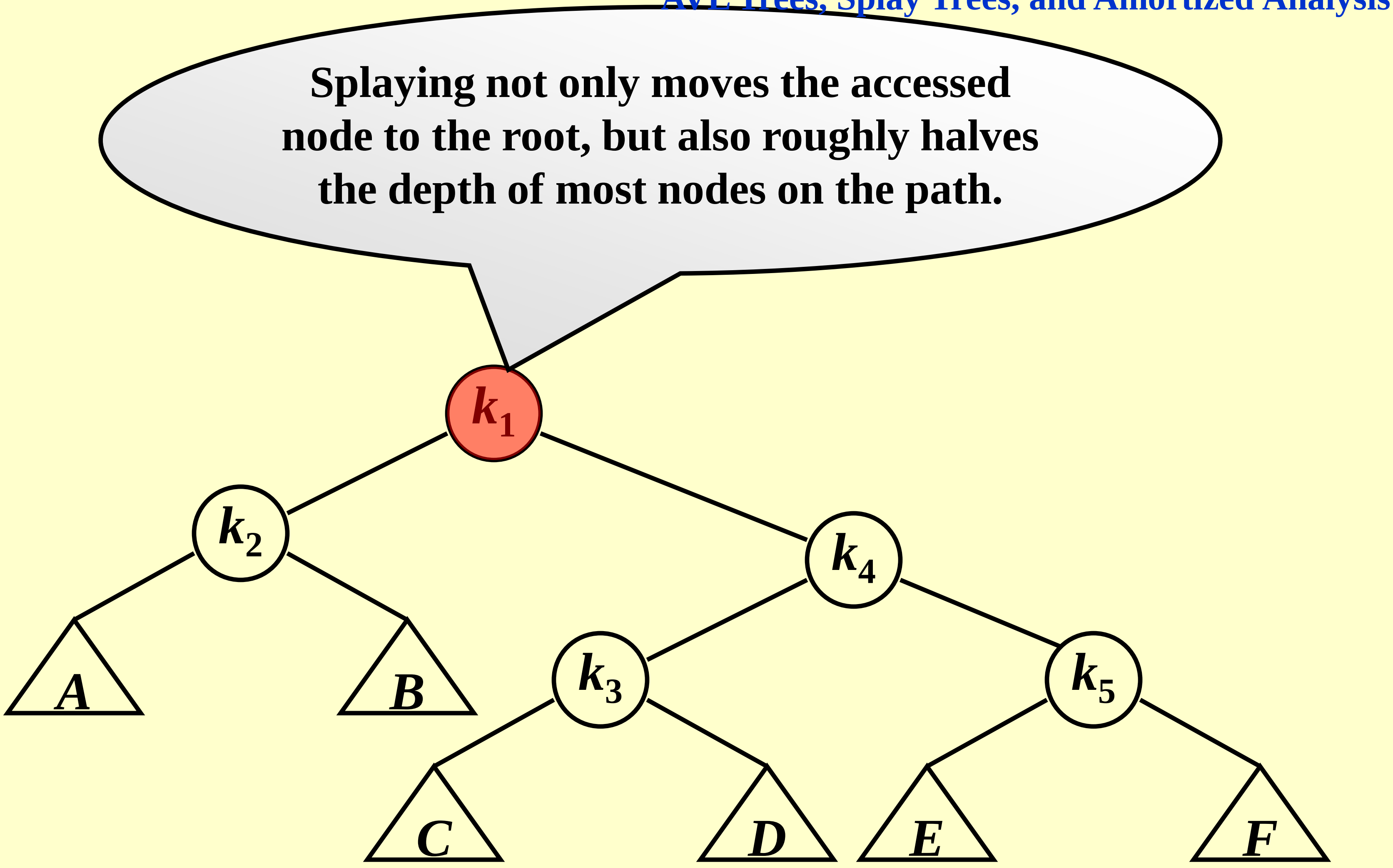
Case 2: *P* is not the root

Zig-zag

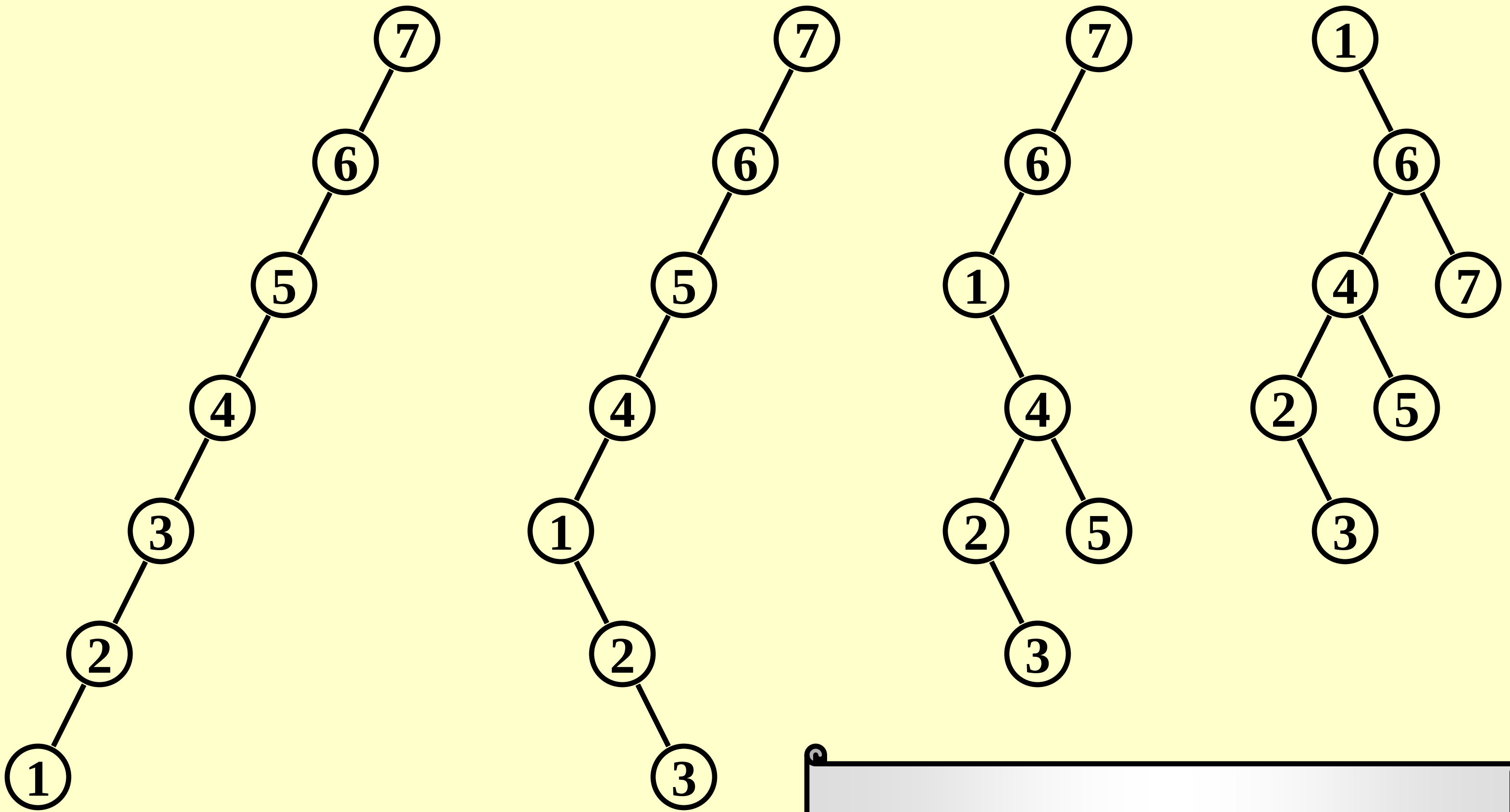


Zig-zig




Splaying not only moves the accessed node to the root, but also roughly halves the depth of most nodes on the path.
k_1
k_2
k_4
k_3
k_5
A
B
C
D
E
F

Insert: 1, 2, 3, 4, 5, 6, 7 **Find**: 1



Read the 32-node example given in Figures 4.52 – 4.60

Deletions:

- **Step 1:** **Find X ;**

  X will be at the root.

- **Step 2:** **Remove X ;**

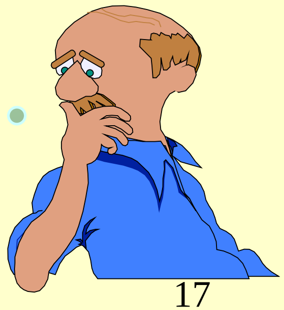
  There will be two subtrees T_L and T_R .

- **Step 3:** **FindMax (T_L) ;**

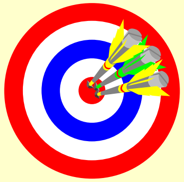
  The largest element will be the root of T_L , and *has no right child.*

- **Step 4:** **Make T_R the right child of the root of T_L .**

Are splay trees really better than AVL trees?



Amortized Analysis



Target: **Any M consecutive operations take at most $O(M \log N)$ time.**

-- *Amortized* time bound

worst-case bound $\geq$ amortized bound $\geq$ average-case bound

Probability is *not* involved

- ☞ **Aggregate analysis**
- ☞ **Accounting method**
- ☞ **Potential method**

☞ Aggregate analysis

Idea : **Show that for all n, a sequence of n operations takes *worst-case* time $T(n)$ in total. In the worst case, the average cost, or *amortized cost*, per operation is therefore $T(n)/n$.**

〖**Example**〗 **Stack with MultiPop(int k, Stack S)**

```
Algorithm  {
   while ( !IsEmpty(S) && k>0 ) {
      Pop(S);
      k - -;
   } /* end while-loop */
}
```

$T = \min(\text{sizeof}(S), k)$

Consider a sequence of n Push, Pop, and MultiPop operations on an initially empty stack.

$$\text{sizeof}(S) \le n$$

$$T_{amortized} = O(n)/n = O(1)$$

☞ Accounting method

Idea: **When an operation's *amortized cost* $\hat{c}_i$ exceeds its *actual cost* c_i, we assign the difference to specific objects in the data structure as *credit*. Credit can help *pay* for later operations whose amortized cost is less than their actual cost.**

Note: For all sequences of *n* operations, we must have

$$\sum_{i=1}^{n}\hat{c}_i \geq \sum_{i=1}^{n} c_i$$

$$T_{amortized} = \frac{\sum_{i=1}^{n}\hat{c}_i}{n}$$

〖Example〗 Stack with MultiPop(int k, Stack S)

c_i for **Push**: 1 ; **Pop**: 1 ; and **MultiPop**: min (sizeof(S), k)

$\hat{c}_i$ for **Push**: 2 ; **Pop**: 0 ; and **MultiPop**: 0

Starting from an empty stack —— *Credits* for

Push: +1 ; **Pop**: –1 ; and **MultiPop**: –1 for each +1

sizeof(S) ≥ 0 ➡ *Credits* ≥ 0

$$\Rightarrow O(n) = \sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$$

$$\Rightarrow T_{amortized} = O(n)/n = O(1)$$

☞ Potential method

Idea : **Take a closer look at the *credit* --**

$$\hat{c}_i - c_i = Credit_i = \Phi(D_i) - \Phi(D_{i-1})$$

Potential function

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} \left(c_i + \Phi(D_i) - \Phi(D_{i-1}) \right)$$

$$= \left(\sum_{i=1}^{n} c_i \right) + \underbrace{\Phi(D_n) - \Phi(D_0)}_{\geq 0}$$

In general, a good potential function should always assume its minimum at the start of the sequence.

〖Example〗 **Stack with MultiPop(int k, Stack S)**

D_i = the stack that results after the i-th operation

$\Phi(D_i)$ = the number of objects in the stack D_i

$$\Phi(D_i) \geq 0 = \Phi(D_0)$$

Push: $\Phi(D_i) - \Phi(D_{i-1}) = (sizeof(S)+1) - sizeof(S) = 1$

$\Rightarrow \hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + 1 = 2$

Pop: $\Phi(D_i) - \Phi(D_{i-1}) = (sizeof(S)-1) - sizeof(S) = -1$

$\Rightarrow \hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 - 1 = 0$

MultiPop: $\Phi(D_i) - \Phi(D_{i-1}) = (sizeof(S)-k') - sizeof(S) = -k'$

$\Rightarrow \hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = k' - k' = 0$

$$\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} O(1) = O(n) \geq \sum_{i=1}^{n} c_i \Rightarrow T_{amortized} = O(n)/n = O(1)$$

〖Example〗 Splay Trees: $T_{amortized} = O(\log N)$

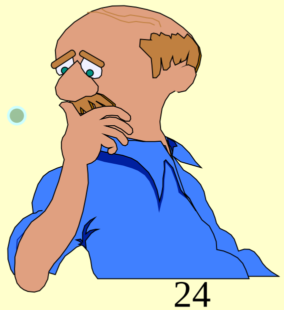
D_i = the root of the resulting tree

$\Phi(D_i)$ = must increase by at most $O(\log N)$ over n steps, AND will also cancel out the number of rotations (*zig*:1; *zig-zag*:2; *zig-zig*:2).

$$\Phi(T) = \sum_{i \in T} \log S(i)$$ where $S(i)$ is the number of descendants of i (i included).

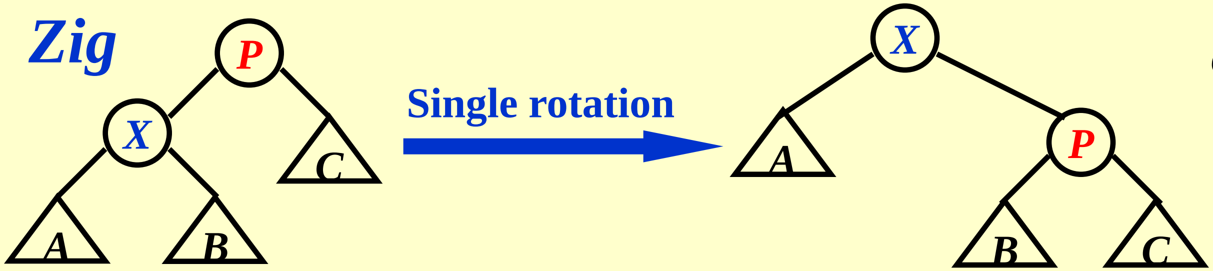
Rank of the subtree
≈ Height of the tree

Why not simply use the heights of the trees?



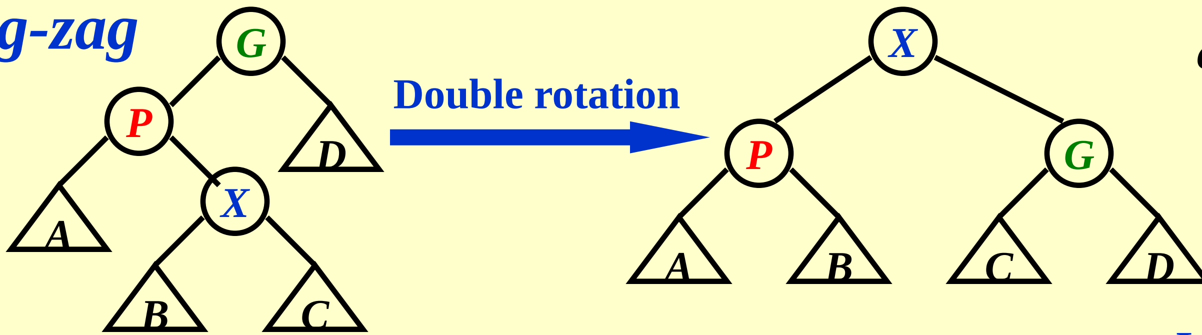
$$\Phi(T) = \sum_{i \in T} Rank(i)$$

Zig



$$\begin{aligned} \hat{c}_i &= 1 + R_2(X) - R_1(X) \\ &\quad + \underline{R_2(P) - R_1(P)} \\ &\le 1 + R_2(X) - R_1(X) \end{aligned}$$

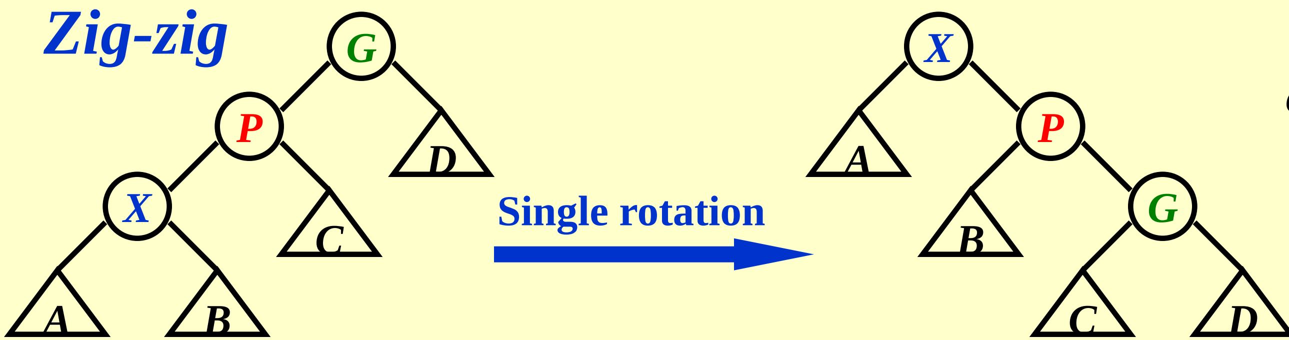
Zig-zag



$$\begin{aligned} \hat{c}_i &= 2 + \cancel{R_2(X)} - \underline{R_1(X)} \\ &\quad + R_2(P) - \underline{R_1(P)} \\ &\quad + R_2(G) - \cancel{R_1(G)} \\ &\le 2\left(R_2(X) - \underline{R_1(X)}\right) \end{aligned}$$

Lemma 11.4 on [1] p.448

Zig-zig



$$\begin{aligned} \hat{c}_i &= 2 + R_2(X) - R_1(X) \\ &\quad + R_2(P) - R_1(P) \\ &\quad + R_2(G) - R_1(G) \\ &\le 3\left(R_2(X) - R_1(X)\right) \end{aligned}$$

【Theorem】 **The amortized time to splay a tree with root T at node X is at most $3(R(T) - R(X)) + 1 = O(\log N)$.**

Reference:

Data Structure and Algorithm Analysis in C (2nd Edition)：Ch.4，p.106-128； Ch.11，p.447-451；*M.A.Weiss著、陈越改编，人民邮件出版社，2005*

Introduction to Algorithms, 3rd Edition: Ch.17，p. 451-478； *Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. The MIT Press. 2009*