Cây AVL

Adelson-Velskii & Landis

Ts. Nguyễn Đức Thuần
BM Hệ thống Thông Tin

Cây AVL

- Nhận xét: Cây nhị phân tìm kiếm BST
  - Độ cao của phụ thuộc vào thứ tự chèn nút vào cây
    - Ví dụ., Chèn 1, 2, 3, 4, 5, 6, 7 vào 1 cây BST
    - Vấn đề: Sự cân bằng của cây "balance" – Cây bị suy biến thành 1 danh sách liên kết !!
  - Tất cả các thao tác trên cây BST có độ cao h nhận độ phức tạp O(h) ở đây log n <= h <= n-1, trường hợp xấu nhất các thao tác trên BST là O(n)

- Câu hỏi: Có thể thực hiện bất chấp thứ tự việc chèn các giá trị (khóa) để độ cao của cây BST là log(n)? Nói cách khác, chúng ta có thể giữ cây BST cân bằng?

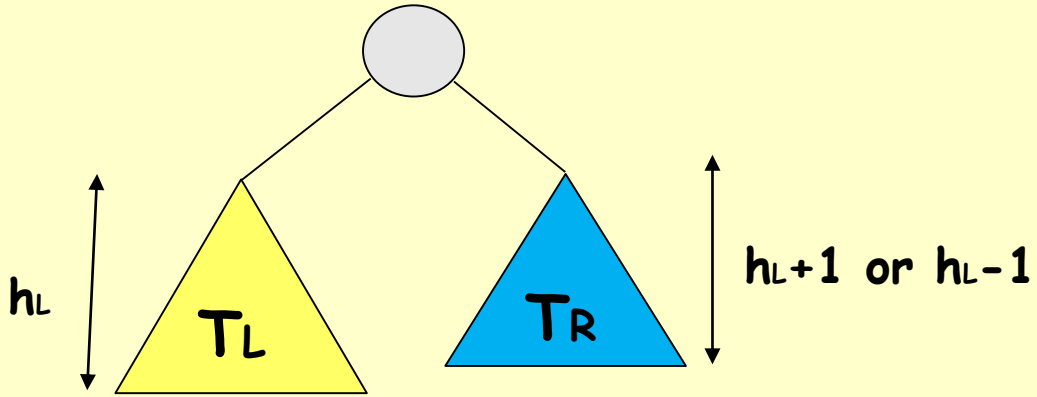
Cây có chiều cao cân bằng

- Nhiều thuật toán hiệu năng cao để cân bằng cây nhị phân (BST) tìm kiếm để các thao tác trên cây nhị phân tìm kiếm nhanh hơn

  - Adelson-Velskii and Landis (AVL) trees (1962)
  - Splay trees and other self-adjusting trees (1978)
  - B-trees and other multiway search trees (1972)
  - Red-Black trees (1972)

AVL Trees: Định nghĩa

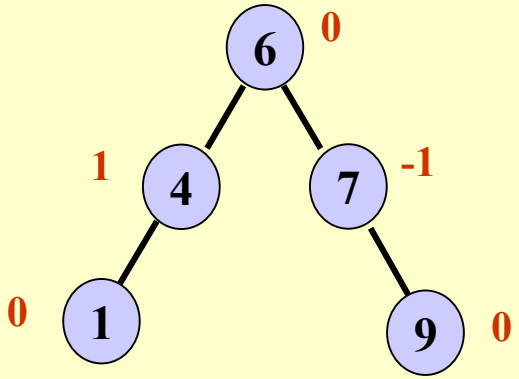
1. Tất cả cây rỗng là cây AVL
2. Nếu T là cây BST không rỗng với T_L and T_R là các cây con trái và phải thì T là cây AVL nếu và chỉ nếu
   1. T_L và T_R là các cây AVL
   2. $|h_L – h_R| <= 1$, ở đây h_L và h_R là chiều cao tương ứng của T_L và T_R



Cây AVL

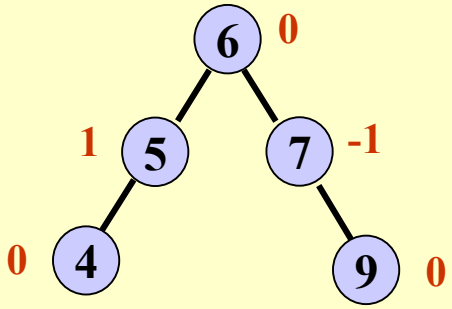
- Cây AVL là cây nhị phân tìm kiếm có chiều cao được cân bằng
- Yếu tố cân bằng của 1 nút = chiều cao(left subtree) – chiều cao(right subtree)
- Một cây căn bằng các yếu tố cân bằng tại mỗi nút là 1 trong 3 giá trị –1, 0, or 1
- Tại mọi nút chiều cao cây con trái và chiều cao cây con phải sai kém không quá 1 đơn vị

Một AVL Tree

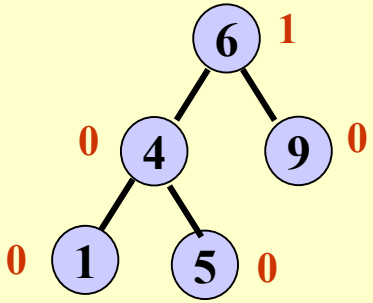


Số màu đỏ là các yếu tố căn bằng

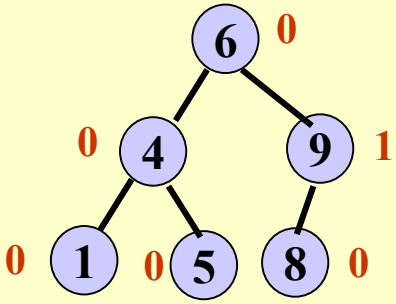
Cây AVL :Ví dụ



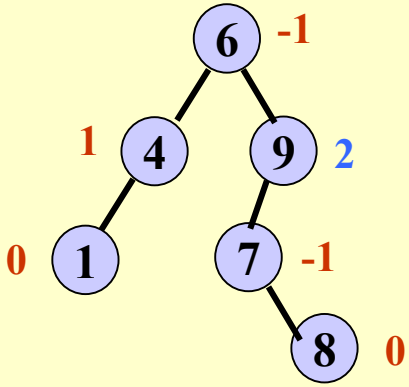
AVL Tree



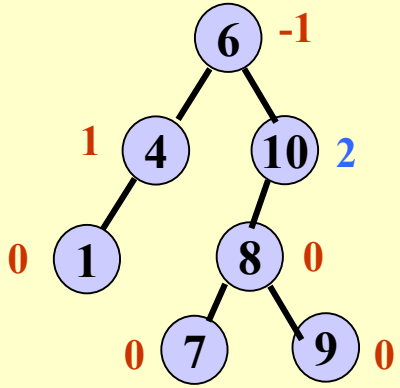
AVL Tree



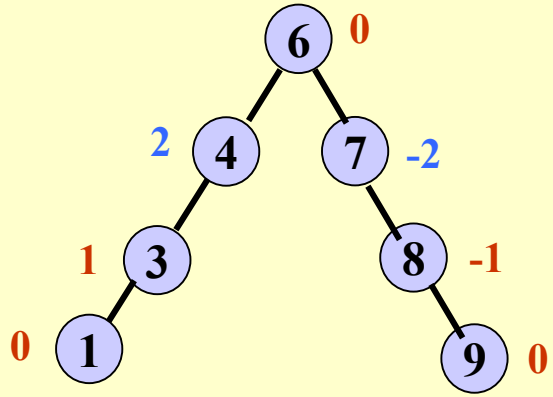
AVL Tree



Non-AVL Tree



Non-AVL Tree



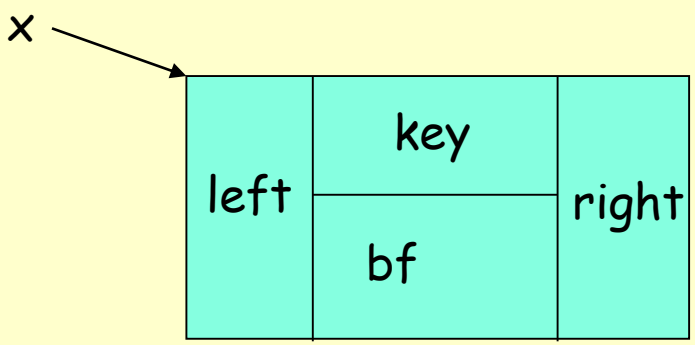
Non-AVL Tree

Số màu đỏ là các yếu tố cân bằng
Red numbers are Balance Factors

Cài đặt AVL Trees

- Khai báo tại mỗi nút

(bf: yếu tố cân bằng)

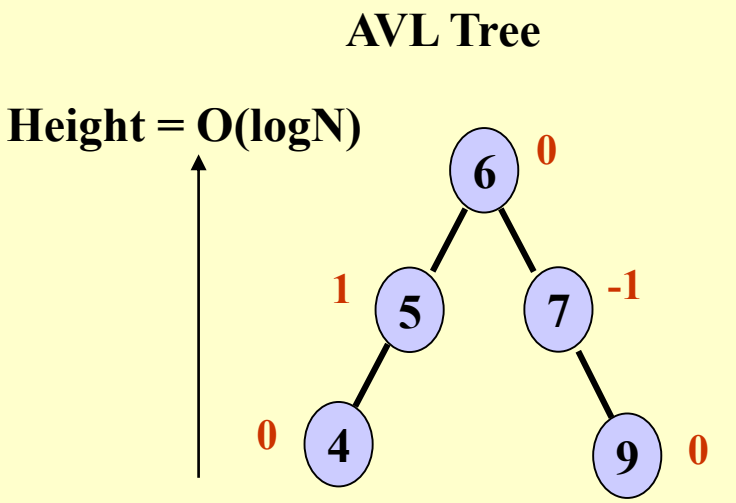



```
C++ Declaration

struct AVLTreeNode{
    int key;
    int bf;
    AVLTreeNode left;
    AVLTreeNode right;
};
```

- Balance factor (bf) của x = chiều cao cây con trái của x – chiều cao cây con phải của x
- Trong 1 cây AVL, bf $\in$ {-1, 0, 1}

Một vài thông tin về AVL Trees



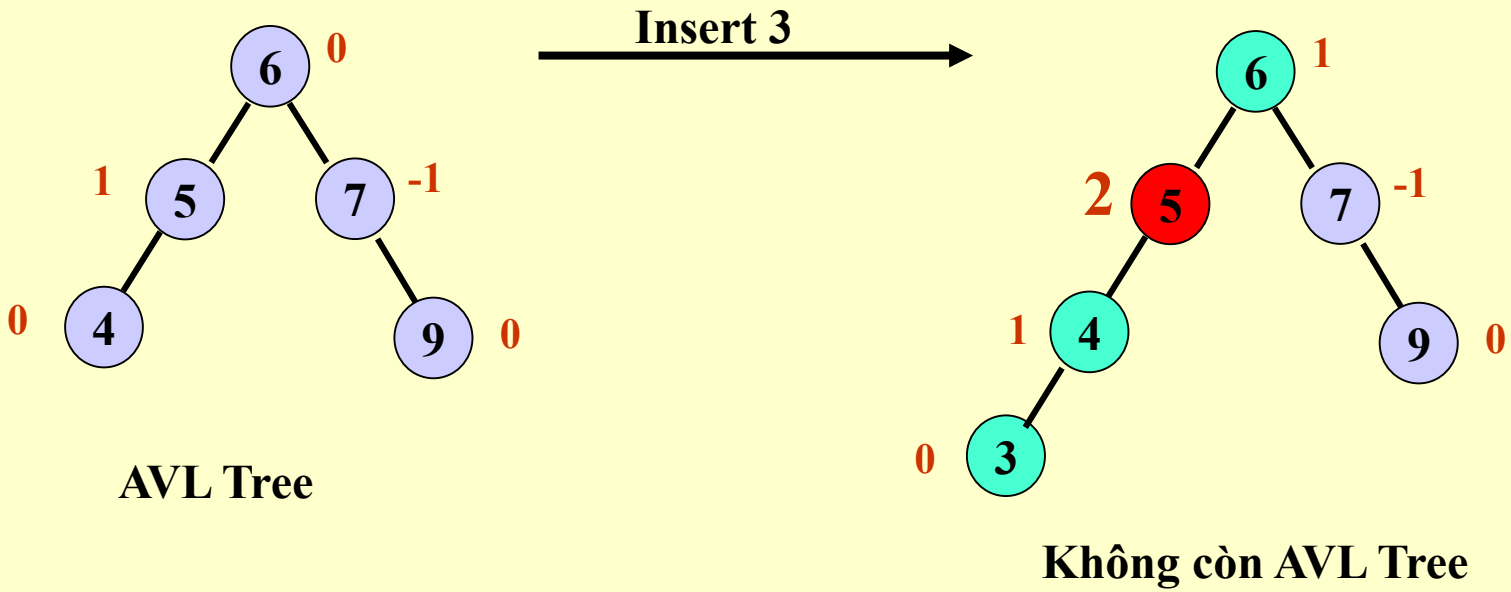
Các số màu đỏ là yếu tố cân bằng

- Có thể chứng minh:Chiều cao của 1 cây AVL có n nút luôn là

  $O(\log n)$

Ưu nhược điểm AVL Trees

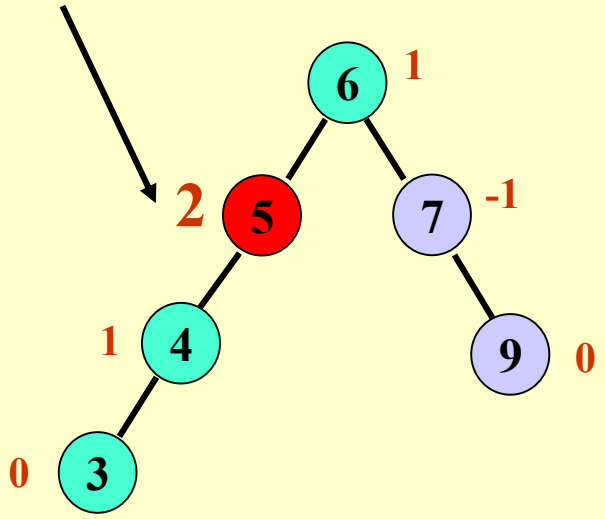
- Ưu:
  - Độ phức tạp thuật toán tìm kiếm O(h) = O(logn)
- Nhược:
  - Việc chèn và xóa nút có thể làm cây mất cân bằng!



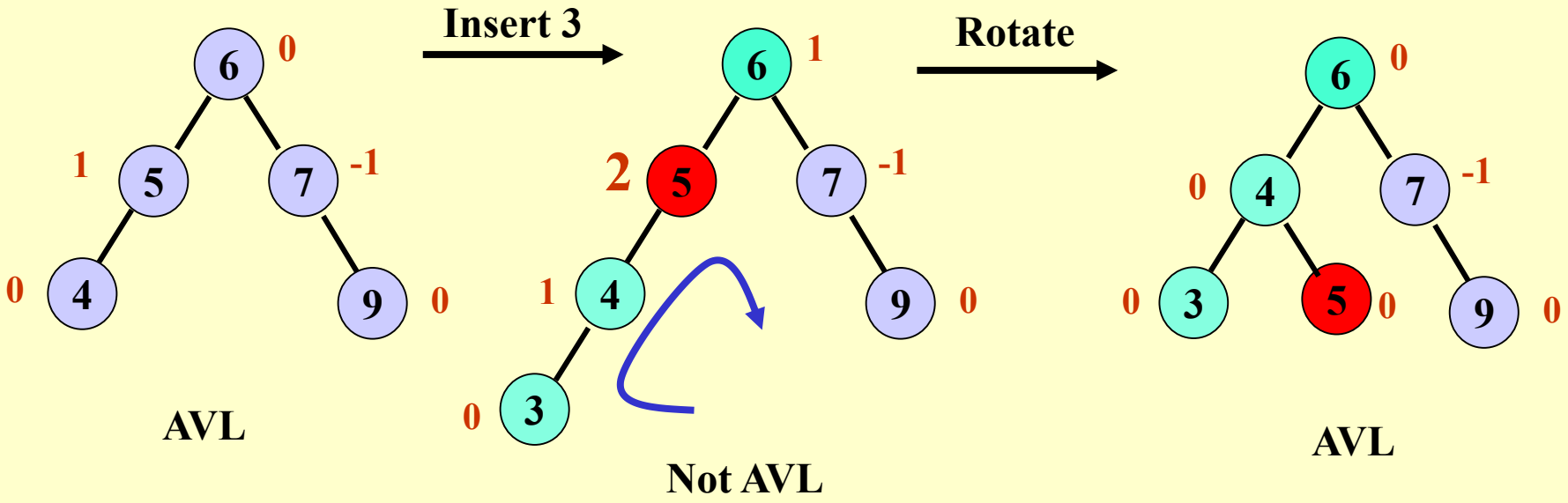
Phục hồi cân bằng trong cây AVL

- Vấn đề: Chèn có thể là nguyên nhân làm yếu tố cân bằng tại 1 số nút trở thành 2 or –2, trên đường từ chèn mới đến nút gốc

- Ý tưởng: Sau khi chèn 1 nút mới
  1. Cập nhật lại các yếu tố căn bằng theo đường (path) truy xuất
  2. Nếu 1 nút có yếu tố cân bằng = 2 hay –2, thực hiện các phép quay (rotation) để hiểu chỉnh cân bằng



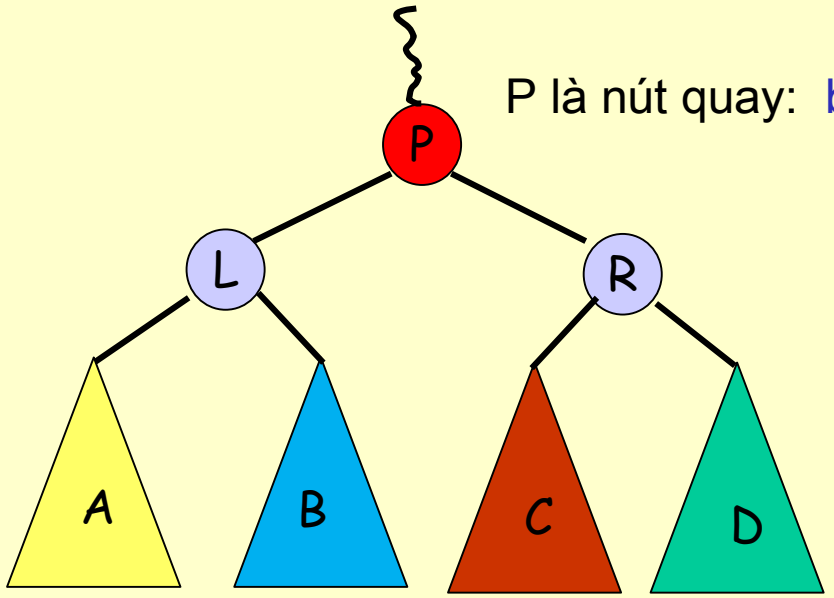
Phục hồi cân bằng: Ví dụ



Sau khi chèn một nút mới:

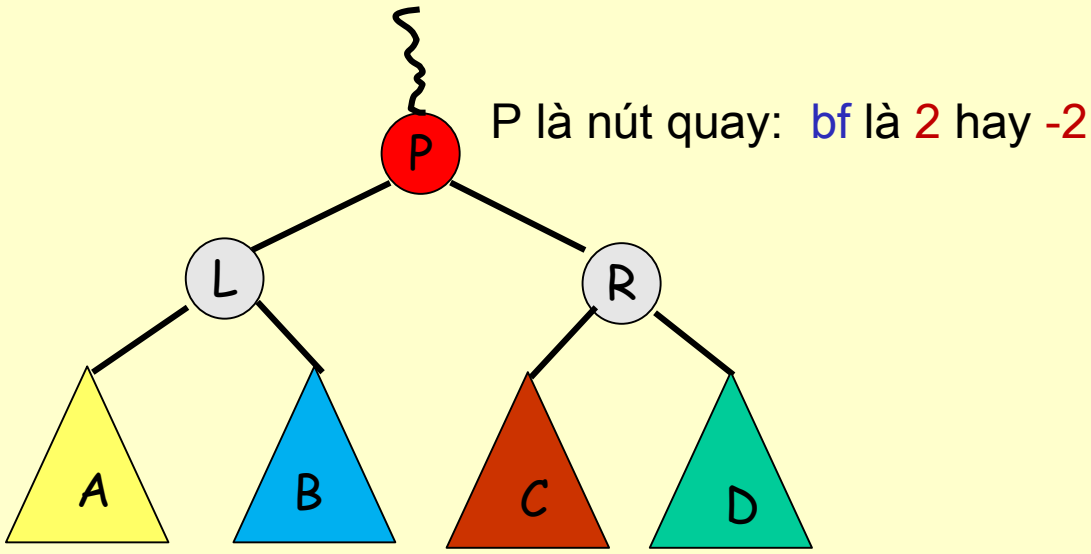
1. Cập nhật lại các yếu tố căn bằng theo đường (path) truy xuất
2. Nếu 1 nút có yếu tố cân bằng = 2 hay –2, thực hiện các phép quay (rotation) để hiểu chỉnh cân bằng

Chèn trên cây AVL (1)



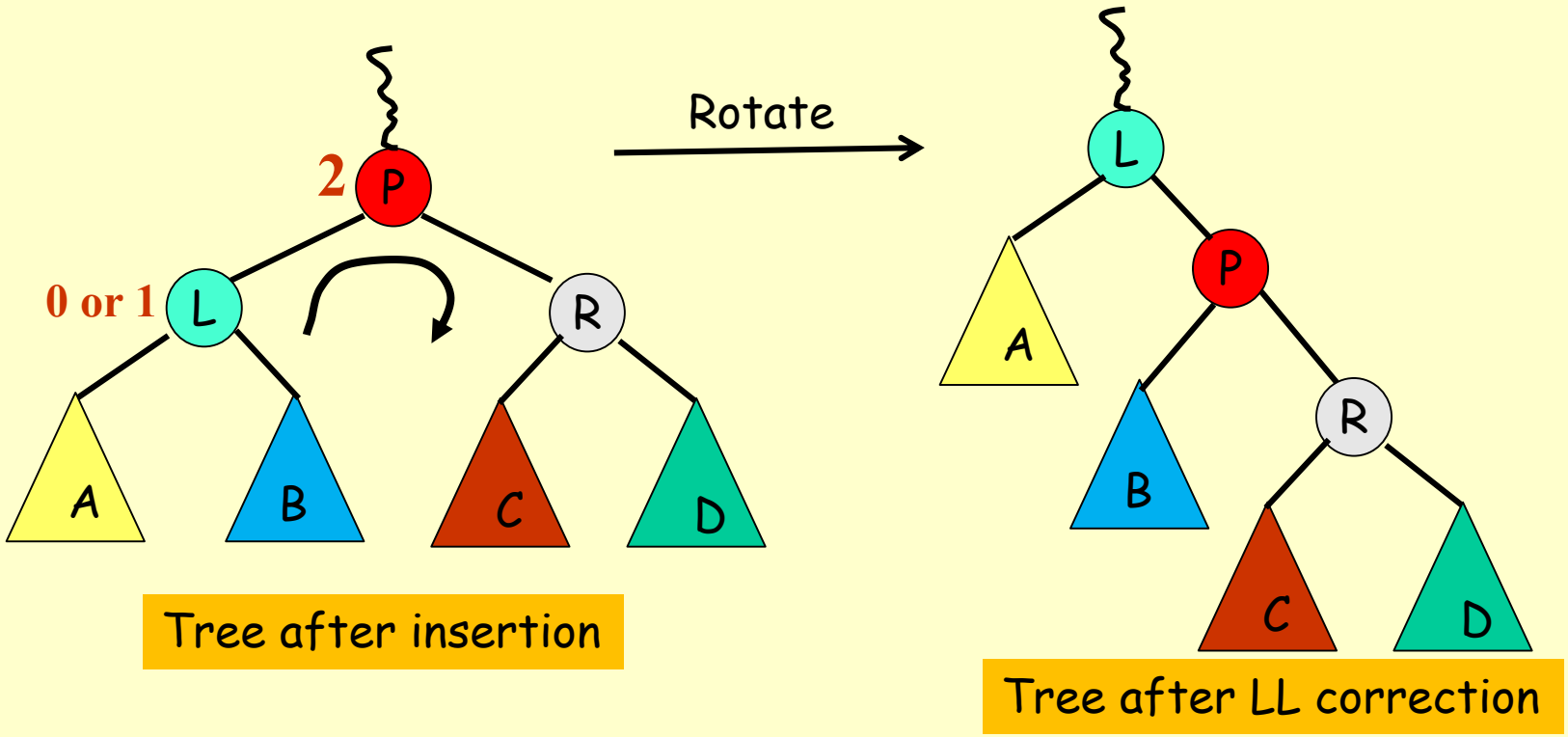
- Nút cần hiệu chỉnh là P.
  - P được gọi là nút quay (pivot node)
  - P là nút đầu tiên có bf là 2 hay -2 kể từ gốc sau khi chèn nút mới

Chèn trên cây AVL(2)



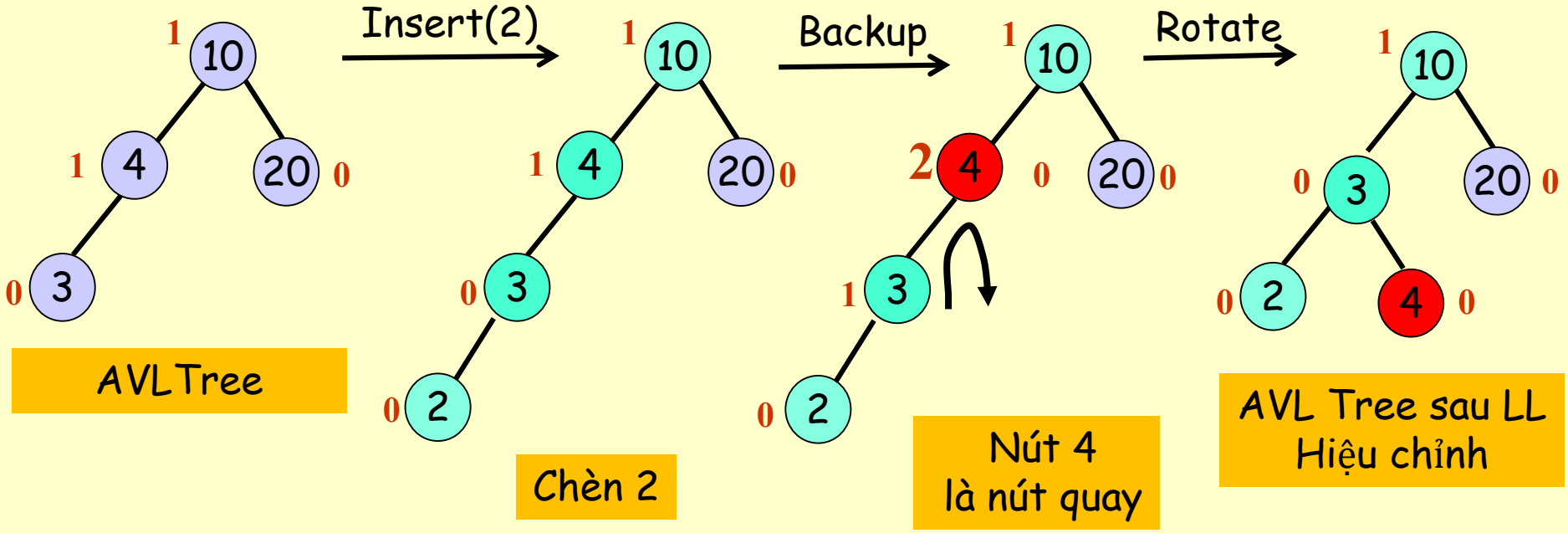
- Có 4 trường hợp:
  - Các trường hợp bên ngoài (yêu cầu quay đơn - single rotation)
    1. Chèn vào cây con trái của nút con trái của P (LL Imbalance).
    2. Chèn vào cây con phải của nút con phải của P(RR Imbalance.)
  - Các trường hợp bên trong (yêu cầu quay kép - double rotation)
    3. Chèn vào cây con trái của nút con phải của P (RL Imbalance)
    4. Chèn vào cây con phải của nút con trái của P (LR Imbalance)

LL mất cân bằng và hiệu chỉnh



- LL mất căn bằng: Đã chèn vào cây con trái của nút trái của P (Chèn vào cây con A)
  - bf của P là 2
  - bf của L là 0 hay 1
- Hiệu chỉnh: Quay đơn (Single rotation) phải quanh P

LL mất cân bằng và hiệu chỉnh(1)

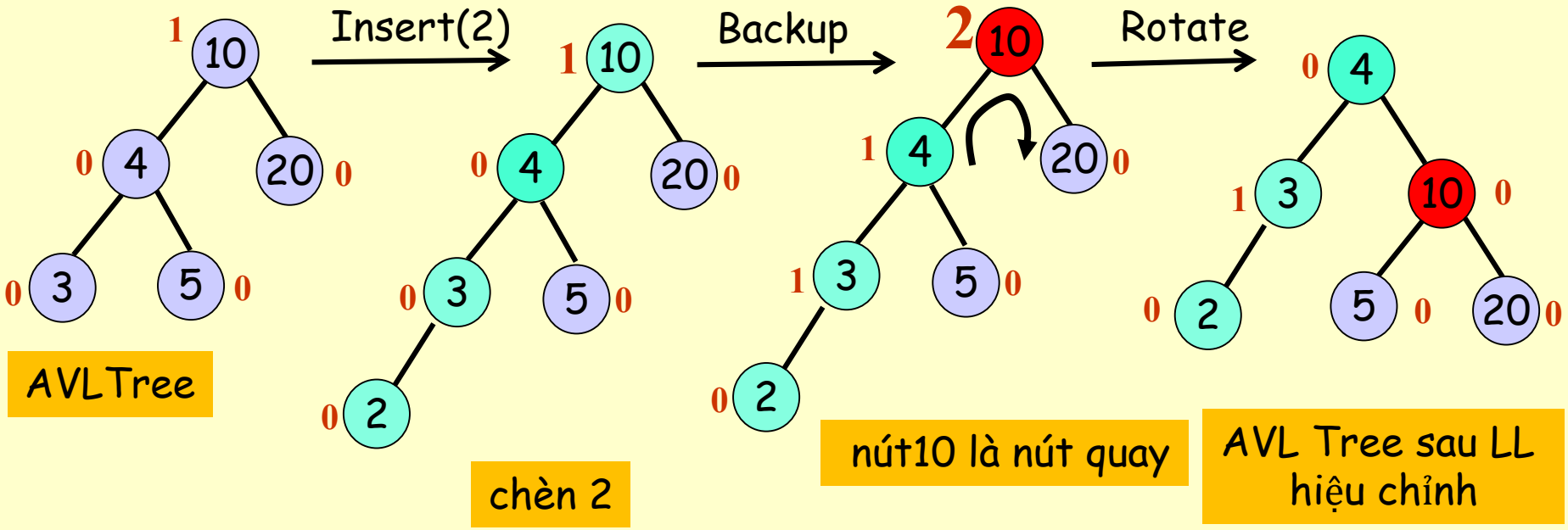


Cập nhật lại yếu tố cân bằng

Phân lớp loại mất cân bằng

- LL mất cân bằng:
  - bf của P(4) là 2
  - bf của L(3) là 0 hay 1

LL mất cân bằng và hiệu chỉnh(2)

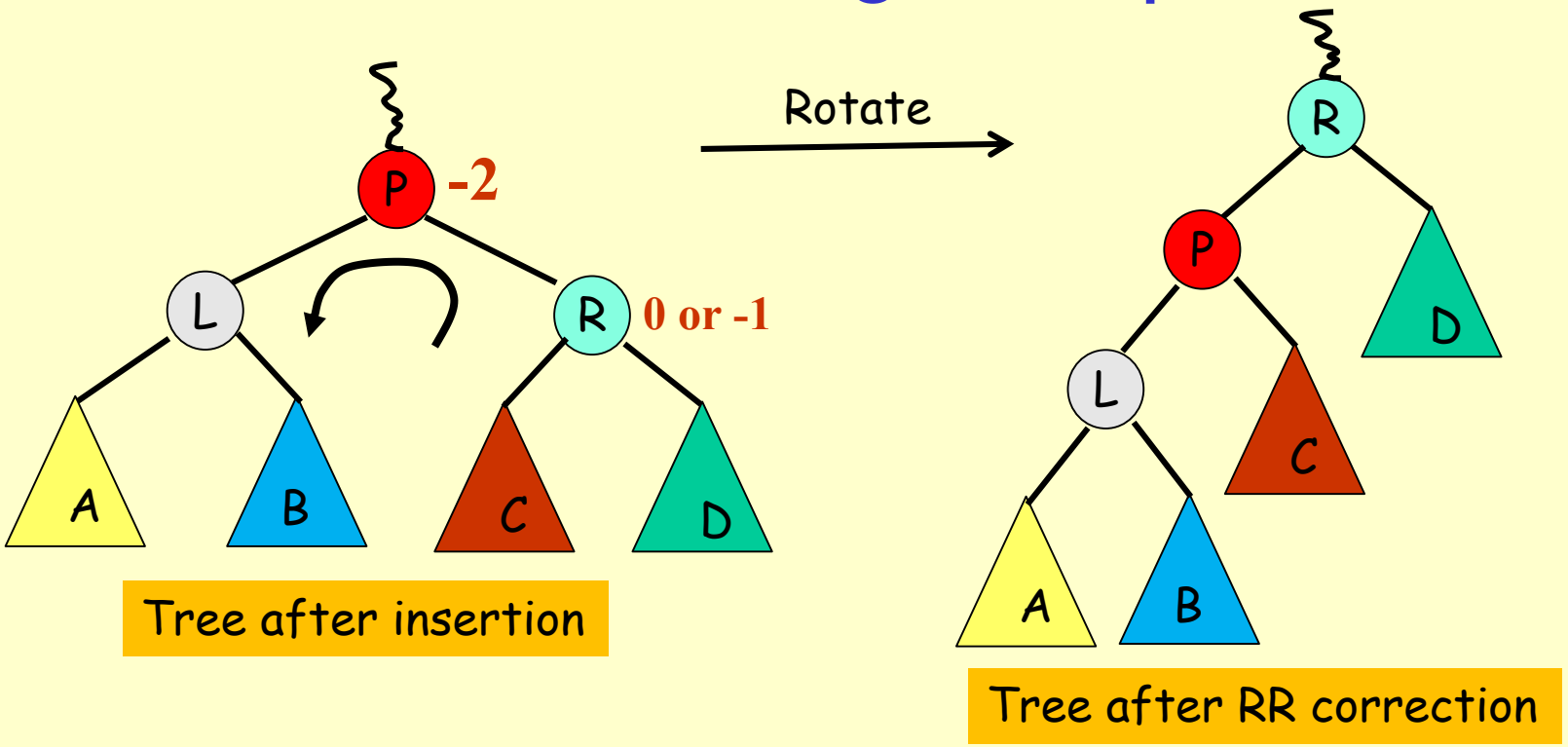


Hiệu chỉnh lại các yếu tố cân bằng

Phân lớp loại mất cân bằng

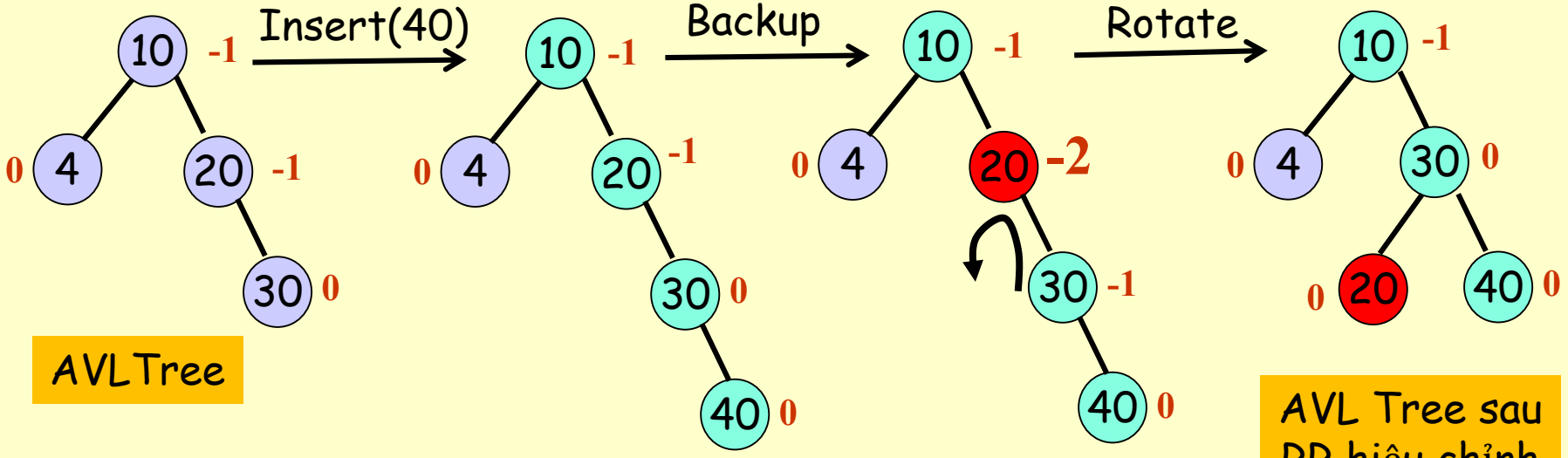
- LL mất cân bằng:
  - bf của P(10) là 2
  - bf của L(4) là 0 hay 1

RR mất cân bằng và hiệu chỉnh



- RR mất cân bằng: Chèn vào cây con phải của nút phải của P
  - bf của P là -2
  - bf của R là 0 hay -1
- Hiệu chỉnh: Quay đơn (Single rotation) bên trái quanh P

RR mất cân bằng và hiệu chỉnh (1)



AVLTree

Chèn 40

Nút 20
Là trục quay

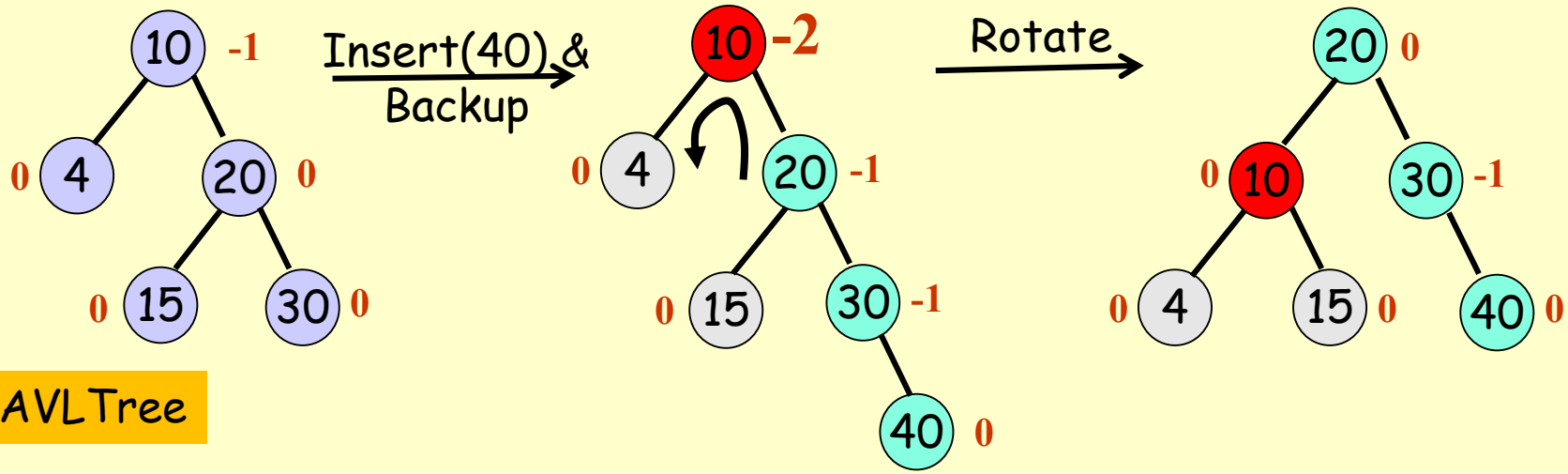
AVL Tree sau
RR hiệu chỉnh

Hiệu chỉnh lại các yếu tố cân bằng

Phân lớp loại mất cân bằng

- RR mất cân bằng :
  - bf của P(20) là -2
  - bf của R(30) là 0 hay-1

RR mất cân bằng và hiệu chỉnh (2)



AVLTree

Chèn 40

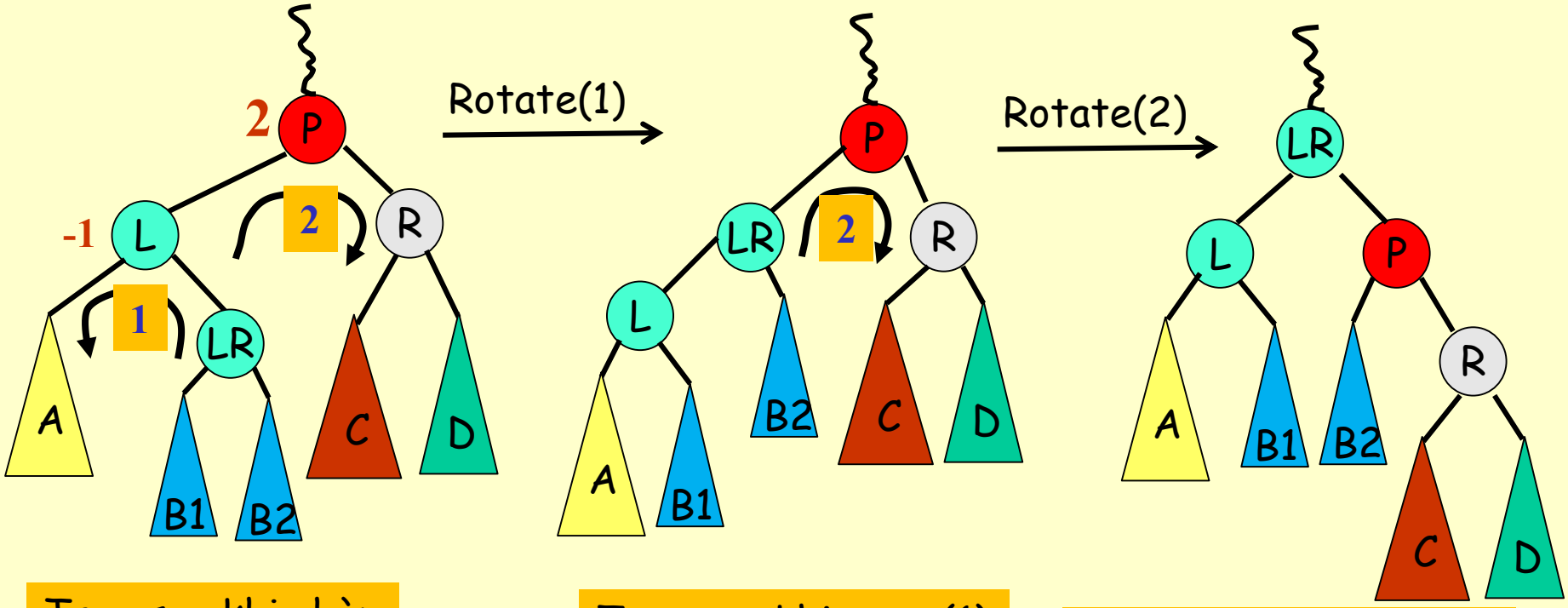
AVL Tree sau RR
Hiệu chỉnh

Sau khi hiệu chỉnh các yếu tố cân bằng, nhận thấy nút 10 là nút quay

Phân lớp loại mất cân bằng

- RR mất cân bằng:
  - bf của(10) là -2
  - bf của (20) lài 0 hay -1

LR mất cân bằng và hiệu chỉnh



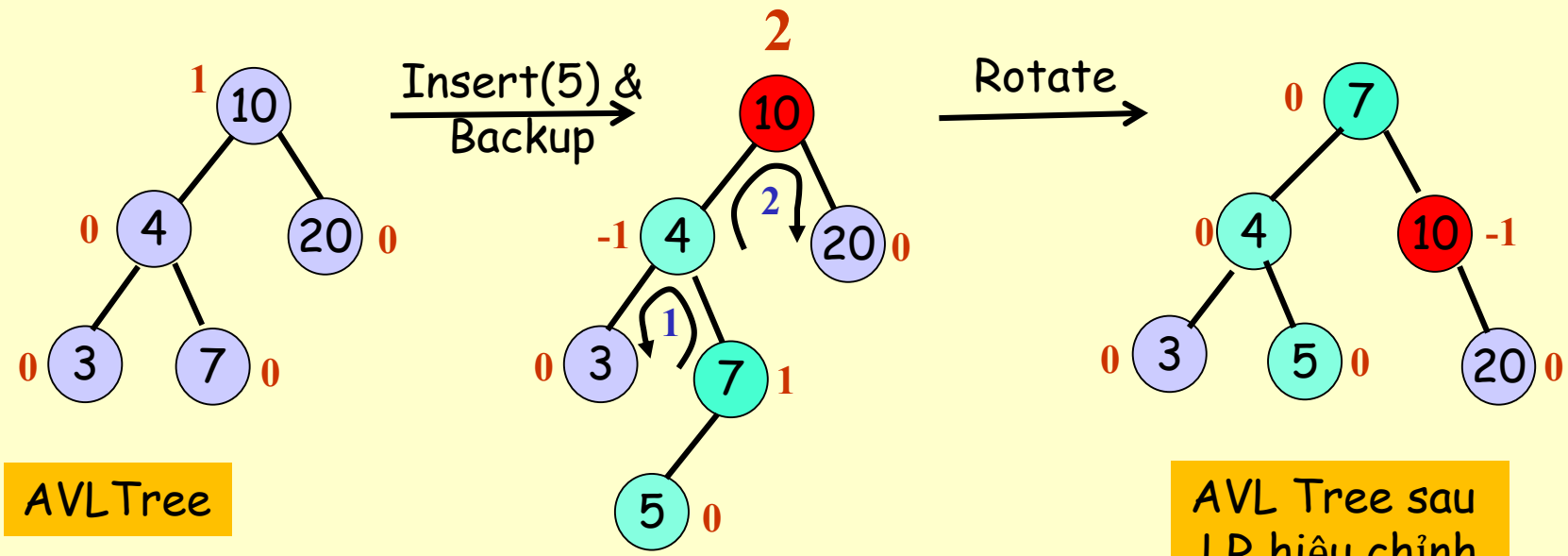
Tree sau khi chèn

Tree sau khi quay (1)

Tree sau hiệu chỉnhLR

- LR mất căn bằng : Đã chèn vào cây con phải của nút trái của P (chèn LR)
  - bf của P là 2
  - bf của L là -1
- Hiệu chỉnh: Quay kép (Double rotation) quanh L & P

LR mất cân bằng và hiệu chỉnh



AVLTree

Chèn 5

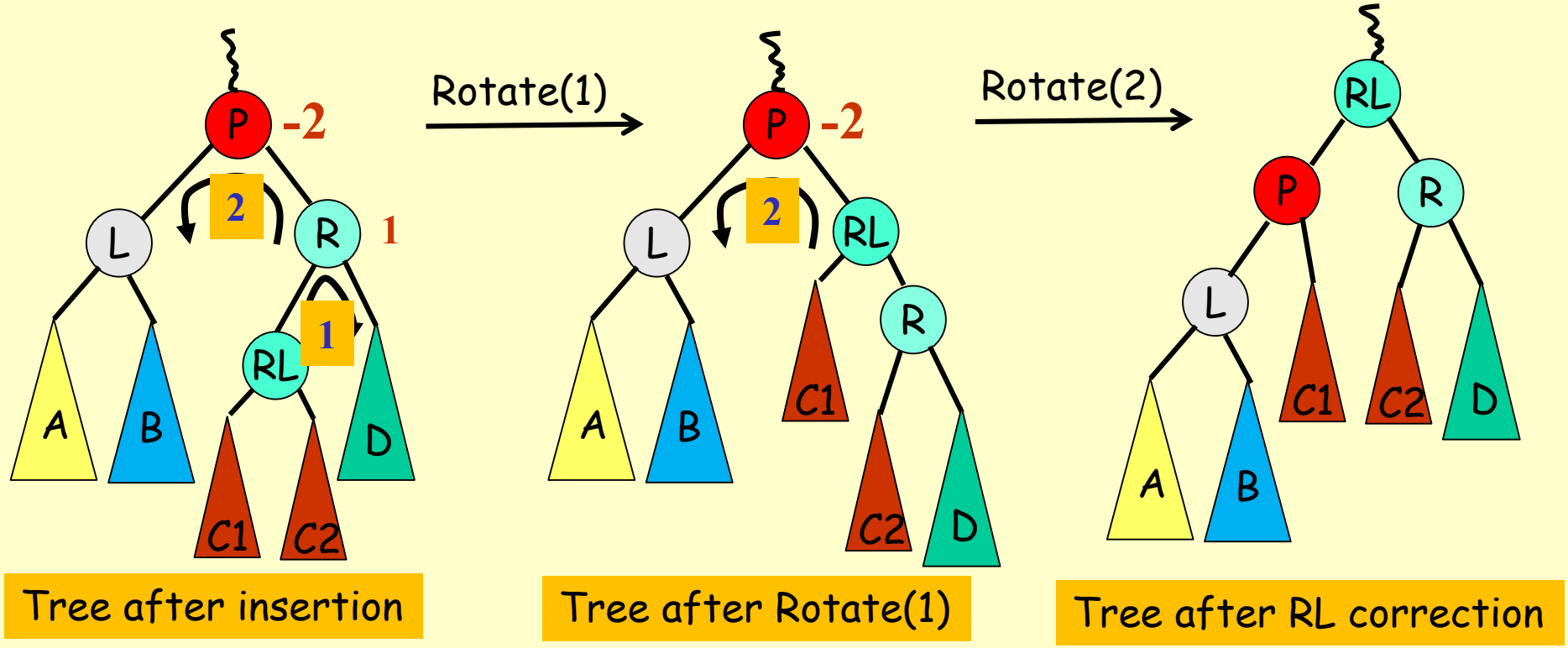
AVL Tree sau LR hiệu chỉnh

Sau khi hiệu chỉnh các yếu tố cân bằng, nhận thấy nút 10 là nút quay

Phân lớp loại mất cân bằng

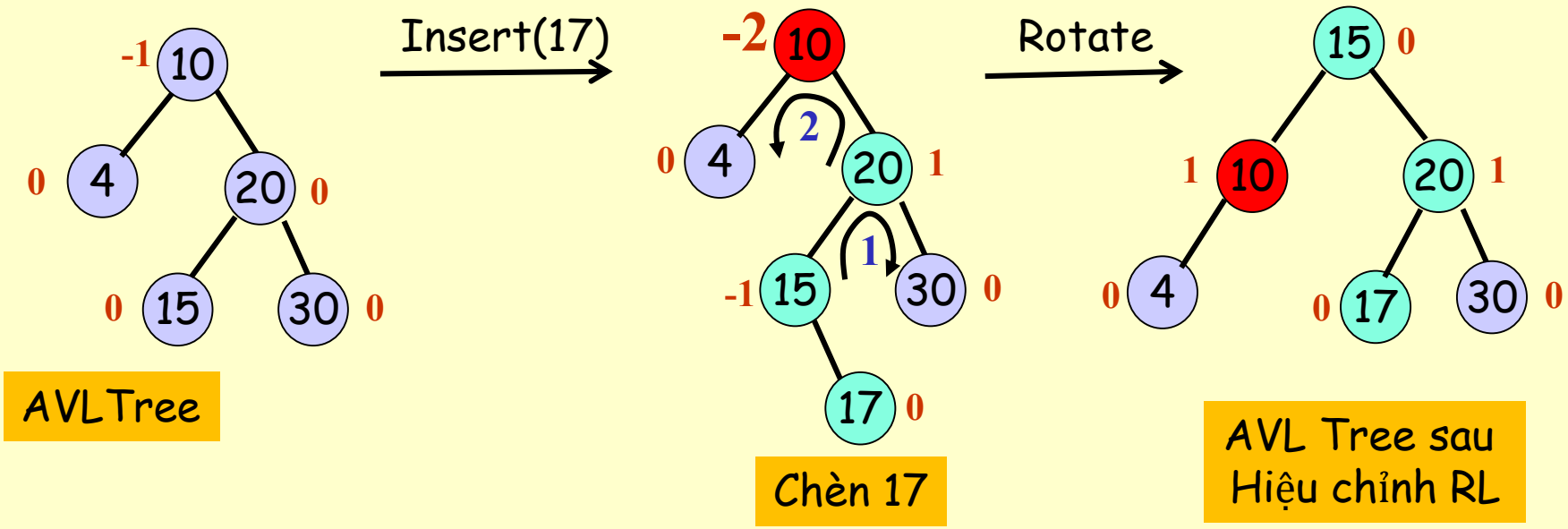
- LR mất cân bằng:
  - bf của P(10) là 2
  - bf của L(4) là -1

RL mất cân bằng và hiệu chỉnh



- RL mất cân bằng: Đã chèn vào cây con trái của nút phải của P (chèn RL)
  - bf của P là -2
  - bf của L là 1
- RL hiệu chỉnh: Quay kép (Double rotation) quanh R & P

RL mất cân bằng và hiệu chỉnh



Sau khi hiệu chỉnh các yếu tố cân bằng, nhận thấy nút 10 là nút quay

Phân lớp loại mất cân bằng

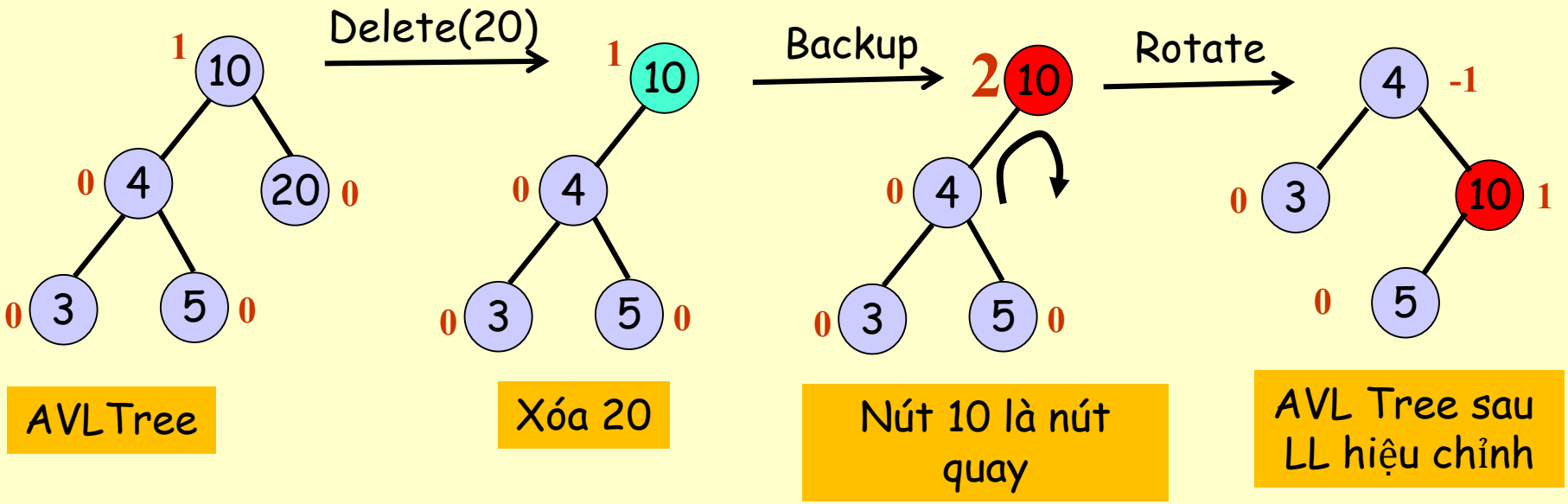
- RL mất cân bằng
  - bf của L(10) là -2
  - bf của R(20) là 1

Xóa

- Xóa tương tự chèn

- Đầu tiên theo dõi thường xuyên các nút bị xóa trên BST, giữ các nút trên đường của nút bị xóa
- Sau khi xóa 1 nút cập nhật lại các yếu tố cân bằng
  - Nếu sự mất cân bằng được phát hiện, thực hiện khôi phục lại cây AVL
  - Có thể phải làm nhiều hơn một vòng quay như phải sao lưu các cây

Xóa (1)

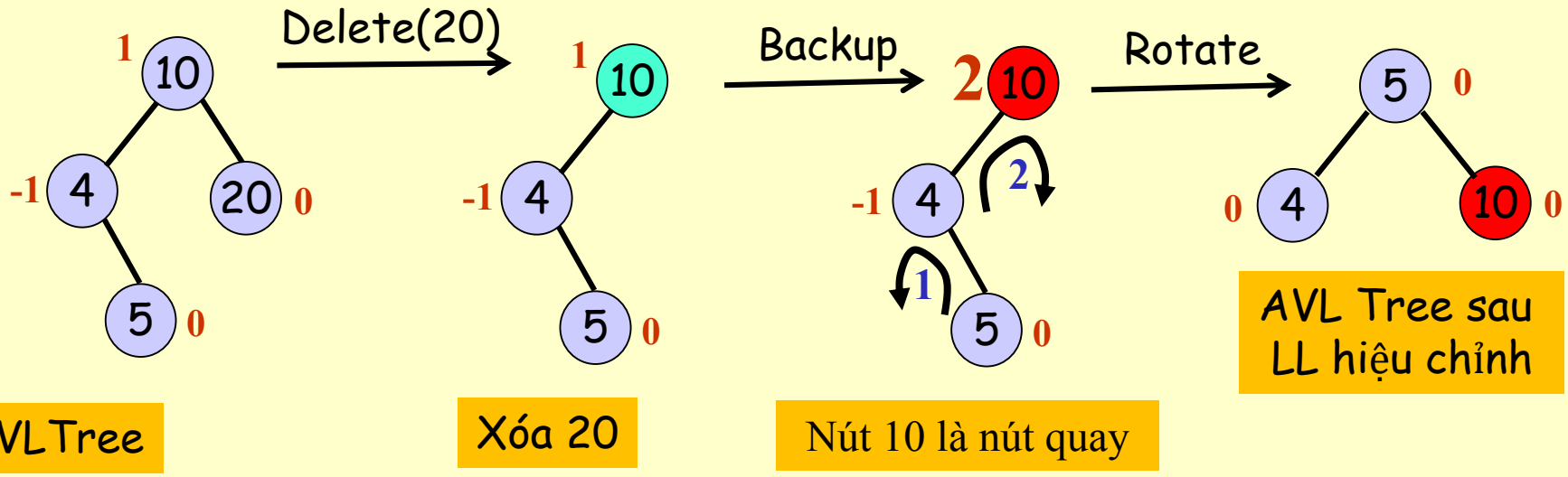


Hiệu chỉnh các yếu tố cân bằng

Phân lớp loại mất cân bằng

- LL mất cân bằng:
  - bf của P(10) là 2
  - bf của L(4) là 0 hay 1

Xóa (2)

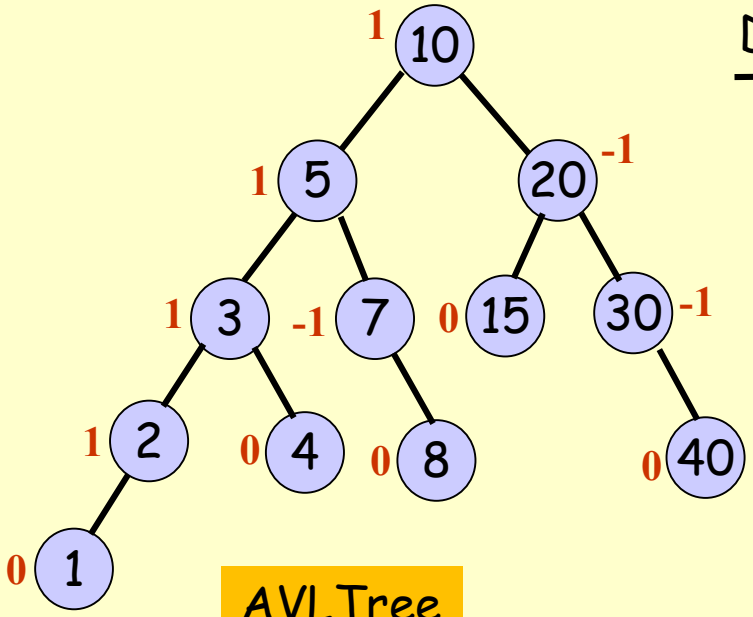


Hiệu chỉnh các yếu tố cân bằng

Phân lớp loại mất cân bằng

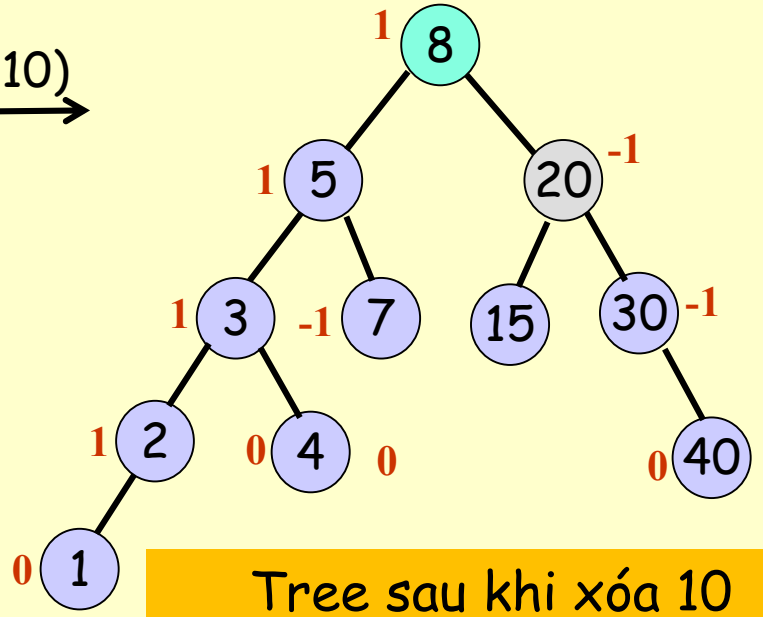
- LR mất cân bằng:
  - bf của P(10) là 2
  - bf của L(4) là -1

Xóa (3)



AVLTree

Delete(10)

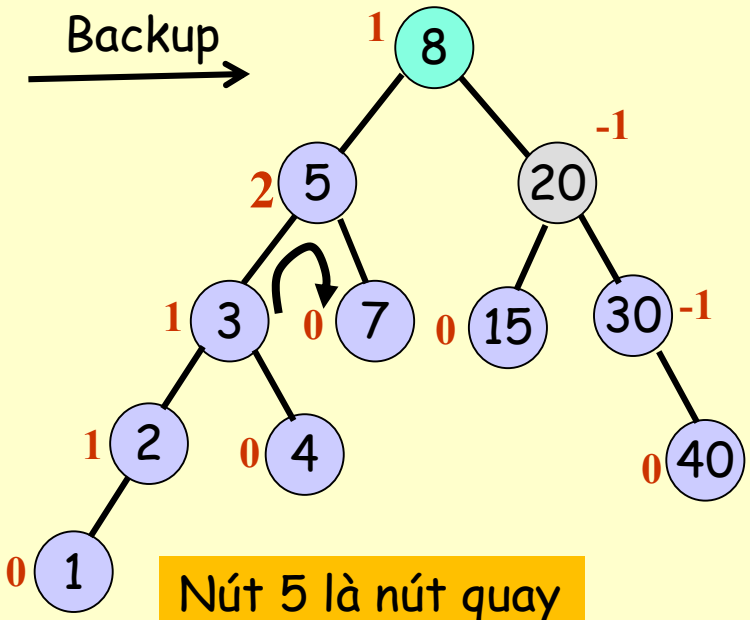


Tree sau khi xóa 10

Chúng ta phải chép 8 vào vị trí của 10 và xóa 8

Cập nhật lại các yếu tố cân bằng

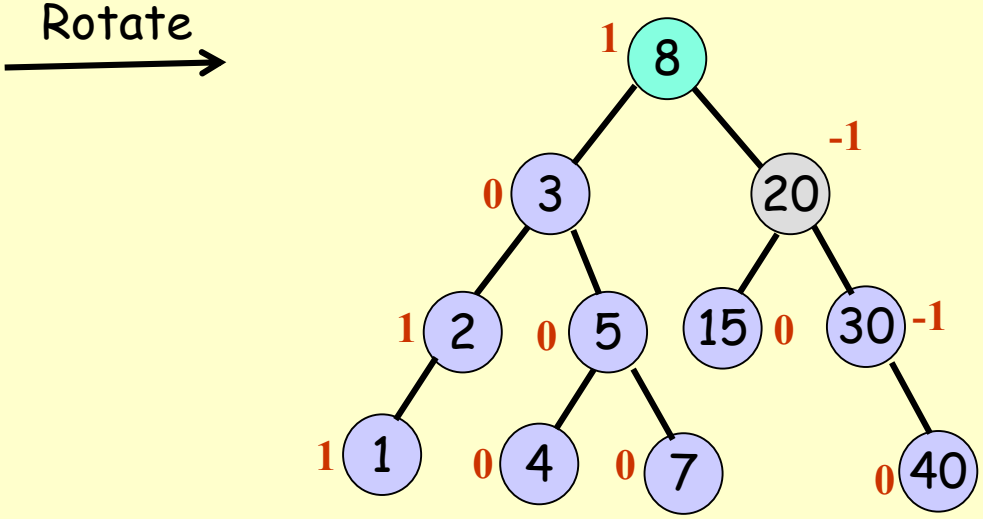
Xóa (3) – tiếp



Nút 5 là nút quay

Phân lớp loại mất cân đối

- LL mất cân bằng:
  - bf của P(5) là 2
  - bf của L(3) là 1



Nút 5 là nút quay

Đây là AVL tree

Cập nhật lại các yếu tố cân bằng

Cám ơn đã theo dõi!