UIT
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Data Structures & Algorithms

CÂY CÂN BẰNG – AVL

ĐẶT VẤN ĐỀ

- **Vẽ** cây nhị phân tìm kiếm lập được từ dãy sau theo chiều từ trái sang phải: 1, 3, 5, 6, 7, 9

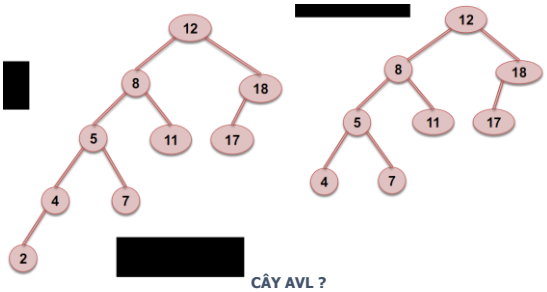
BINARY SEARCH TREE → LIST

Khái niệm cây AVL

- **Cây cân bằng AVL** là **cây nhị phân tìm kiếm** mà tại **mỗi đỉnh của cây**, **độ cao của cây con trái và cây con phải chênh lệch không quá 1**.

- Do G.M. **A**delsen **V**elskii và E.M **L**endis đưa ra vào năm 1962, đặt tên là AVL

Cây AVL – Ví dụ



CÂY AVL ?

Cấu trúc cây AVL

- **Cây cân bằng AVL** là **cây nhị phân tìm kiếm** được bổ sung một **giá trị cho biết sự cân bằng của cây**.

```
struct node
{
  KDL      Key;
  struct node *pLeft;
  struct node *pRight;
  int bal;
};
typedef struct node TNode;

typedef TNode* AVL;
```

Trong đó **bal là chỉ số cân bằng**

Cấu trúc cây AVL

- **Chỉ số cân bằng của một node**: là **hiệu số** của **chiều cao cây con phải** và **cây con trái** của nó.
- Đối với cây AVL: **chỉ số cân bằng của mỗi nút** chỉ có thể mang một trong ba giá trị sau
  - ✓ CSCB(p) =**0** Nếu Độ cao của cây phải **=** độ cao cây trái (**EH**)
  - ✓ CSCB(p) =**1** Nếu Độ cao của cây phải **>** độ cao cây trái (**RH**)
  - ✓ CSCB(p) =**-1** Nếu Độ cao của cây phải **<** độ cao cây trái (**LH**)

```
#define EH 0
#define RH 1
#define LH -1
```

Các thao tác trên AVL

- **Thêm** một phần tử vào cây AVL

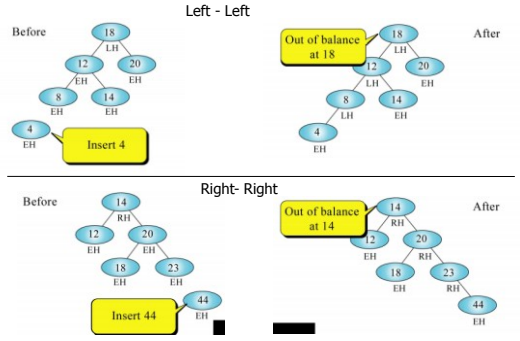
➔ Giống như cây NPTK, tuy nhiên sau khi thêm **phải cân bằng lại cây.**

- **Hủy** một phần tử trên cây AVL.

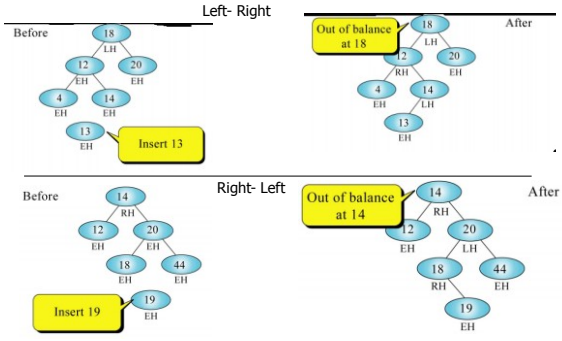
➔ Giống như cây NPTK, tuy nhiên sau khi hủy **phải cân bằng lại cây.**

- **Cân bằng lại một cây vừa bị mất cân bằng (Rotation)**

Các trường trường hợp mất cân bằng

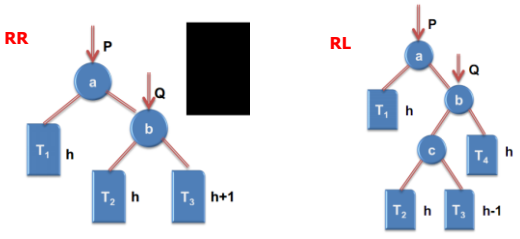


Các trường trường hợp mất cân bằng

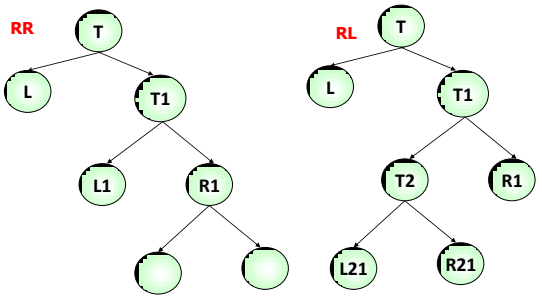


Các trường trường hợp mất cân bằng

- Giả sử tại một node trên cây xảy ra **mất cân bằng phải** (cây **con phải chênh lệch với cây con trái hơn 1**)

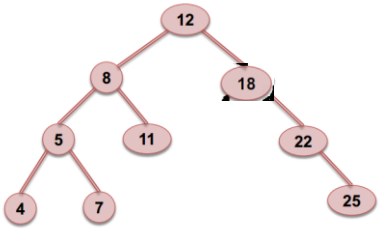


Các trường trường hợp mất cân bằng



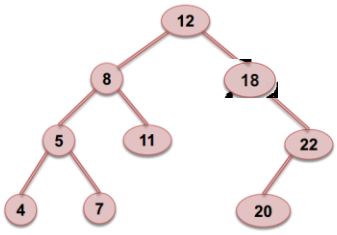
Các trường hợp mất cân bằng

- **Mất cân bằng phải- phải (R-R)**



Các trường hợp mất cân bằng

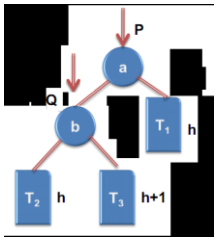
➢ **Mất cân bằng phải- trái(R-L)**



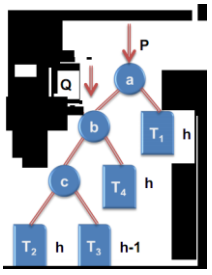
Các trường trường hợp mất cân bằng

➢ Giả sử tại một node trên cây xảy ra **mất cân bằng trái**(cây **con trái chênh lệch với cây con trái hơn 1**)

LL

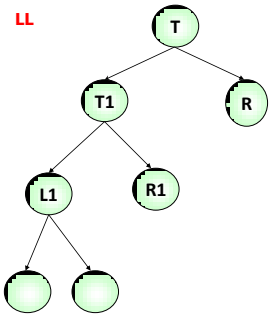


LR

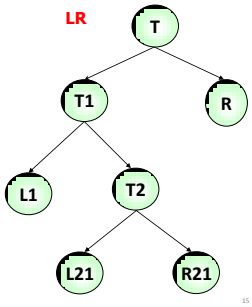


Các trường trường hợp mất cân bằng

LL

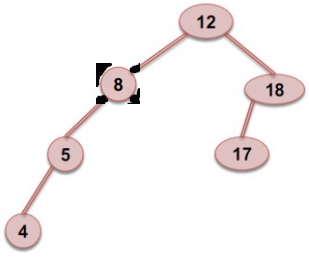


LR



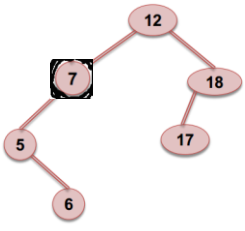
Các trường hợp mất cân bằng

➢ **Mất cân bằng trái trái (L-L)**



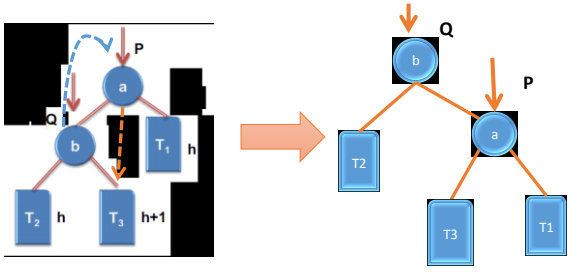
Các trường hợp mất cân bằng

➢ **Mất cân bằng trái- phải (L-R)**



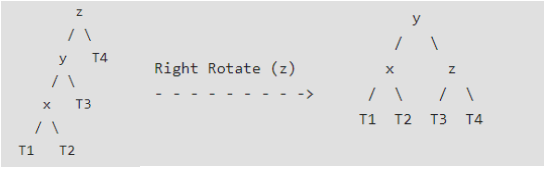
Xử lý các trường hợp mất cân bằng

➢ Mất cân bằng LL ở P ➔ Quay phải



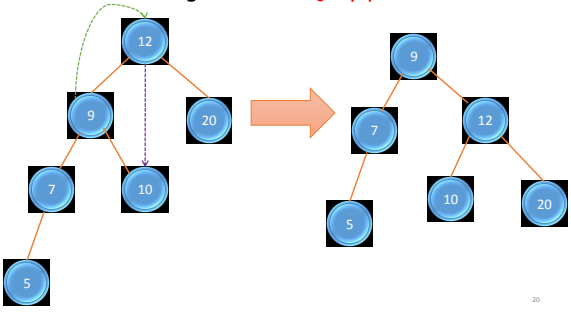
Xử lý các trường hợp mất cân bằng

➢ Mất cân bằng LL ở P ➔ Quay phải



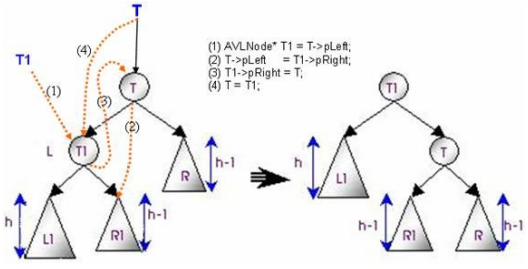
Xử lý các trường hợp mất cân bằng

➢ Mất cân bằng LL ở P ➔ Quay phải



Xử lý các trường hợp mất cân bằng

➢ Mất cân bằng LL ở T ➔ **Quay phải**



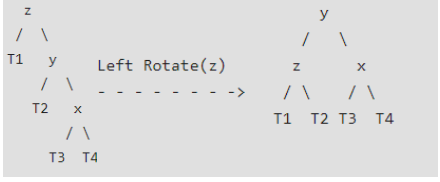
Xử lý các trường hợp mất cân bằng

➢ Mất cân bằng LL ở T ➔ **Quay phải**

```
void rotateLL ( AVLTree &T)
{
        AVLNode * T1 = T-> pLeft ;
        T->pLeft     = T1->pRight ;
        T1-> pRight = T;
        switch( T1-> balFactor )
        {
                case LH: T-> balFactor = EH;
                         T1-> balFactor = EH;
                         break ;
                case EH: T-> balFactor = LH;
                         T1-> balFactor = RH;
                         break ;
        }
        T = T1;
}
```

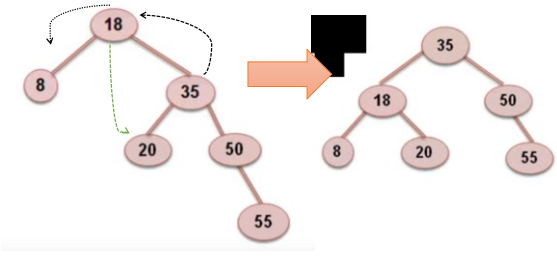

Xử lý các trường hợp mất cân bằng

➢ Mất cân bằng RR ở P ➔ Quay trái



Xử lý các trường hợp mất cân bằng

➢ Mất cân bằng RR ở P ➔ Quay trái



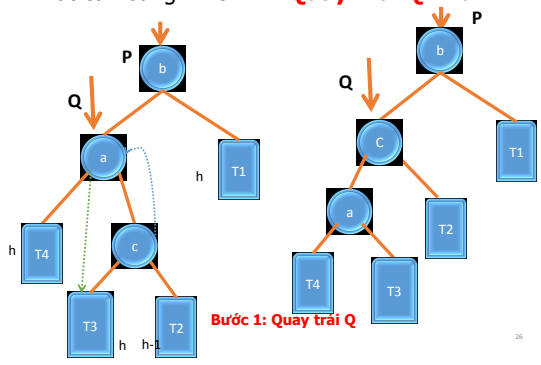
Xử lý các trường hợp mất cân bằng

➢ Mất cân bằng RR ở P ➔ Quay trái

```
void rotateRR(AVLTree &T)
{
        AVLNode * T1 = T-> pRight ;
        T->pRight = T1-> pLeft ;
        T1-> pLeft = T;
        switch ( T1-> balFactor )
        {
                case RH: T-> balFactor = EH;
                        T1-> balFactor = EH;
                        break ;
                case EH: T-> balFactor = RH;
                        T1-> balFactor = LH;
                        break ;
        }
        T = T1;
}
```

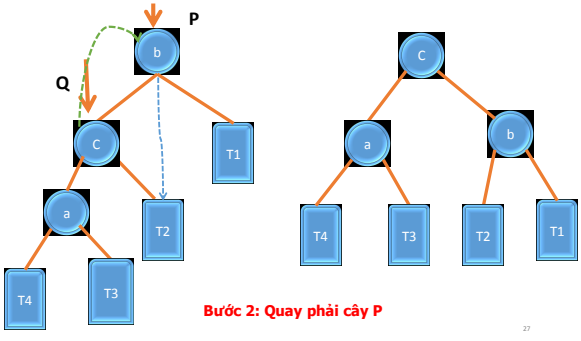

Xử lý các trường hợp mất cân bằng

➢ Mất cân bằng LR ở P ➔ **Quay Trái Q Phải P**



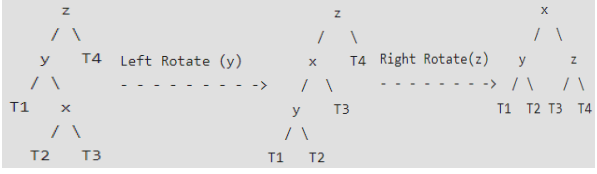
Xử lý các trường hợp mất cân bằng

➢ Mất cân bằng LR ở P ➔ **Quay Trái Q Phải P**



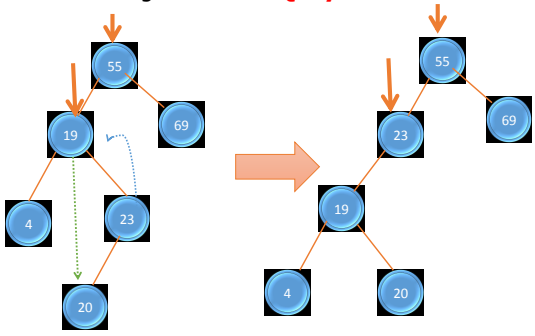
Xử lý các trường hợp mất cân bằng

➢ Mất cân bằng LR



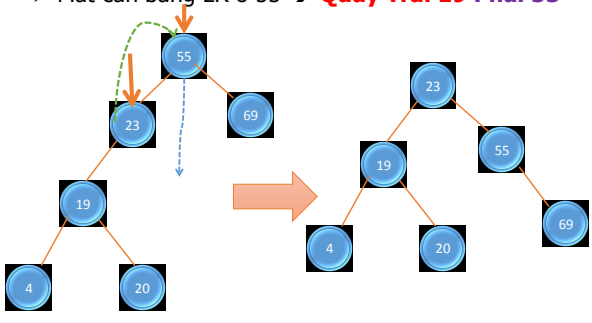
Xử lý các trường hợp mất cân bằng

➢ Mất cân bằng LR ở 55 ➔ **Quay Trái 19 Phải 55**



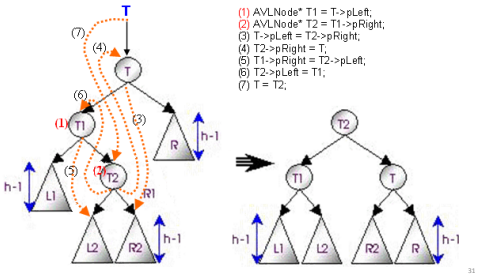
Xử lý các trường hợp mất cân bằng

➢ Mất cân bằng LR ở 55 ➔ **Quay Trái 19 Phải 55**



Xử lý các trường hợp mất cân bằng

➢ Mất cân bằng LR ở T ➔ **Quay Trái T1 Phải T**




```
(1) AVLNode* T1 = T->pLeft;
(2) AVLNode* T2 = T1->pRight;
(3) T->pLeft = T2->pRight;
(4) T2->pRight = T;
(5) T1->pRight = T2->pLeft;
(6) T2->pLeft = T1;
(7) T = T2;
```


Xử lý các trường hợp mất cân bằng

➢ Mất cân bằng LR ở T ➔ **Quay Phải Trái**

```
void rotateLR(AVLTree &T)
{
        AVLNode * T1 = T-> pLeft ;
        AVLNode * T2 = T1-> pRight ;
        T-> pLeft     = T2-> pRight ;
        T2-> pRight = T;
        T1-> pRight = T2-> pLeft ;
        T2-> pLeft     = T1;
        switch (T2-> balFactor )
        {
                case LH: T-> balFactor = RH;
                         T1-> balFactor = EH;
                         break ;
                case EH: T-> balFactor = EH;
                         T1-> balFactor = EH;
                         break ;
                case RH: T-> balFactor = EH;
                         T1-> balFactor = LH;
                         break ;
        }

        T2-> balFactor = EH;
        T = T2;
}
```


Slide được tham khảo từ

- **Slide được tham khảo từ:**
  - Slide CTDL GT, Khoa Khoa Học Máy Tính, ĐHCNTT
  - Congdongcviet.com
  - Cplusplus.com