CÁC GIẢI THUẬT TRÊN CÂY AVL

I. KHAI BÁO

```
typedef … KeyType;
struct Node{
     KeyType Key;
     struct Node* Left;
     struct Node* Right;
     int height; //Chiều cao
};
typedef struct Node *Tree;
```

II. TÌM KIẾM

Tương tự giải thuật tìm kiếm trên cây tìm kiếm nhị phân.

III. THÊM KHÓA K VÀO CÂY AVL

Ý tưởng giải thuật theo lối đệ quy như sau:

- Thực hiện thêm bình thường vào cây tìm kiếm nhị phân.
  - Mỗi nút mặc nhiên có chiều cao ban đầu là 1
- Mỗi khi một nút được thêm vào, chiều cao nút cha của nó tăng 1. Để ý là chiều cao của tổ tiên các nút đó cũng được cập nhật trong quá trình quay lui của đệ quy
- Cân bằng lại nút thêm vào (các nút tổ tiên được cân bằng trong quá trình quay lui lại)

```
oHệ số cân bằng = height(cây con trái) – height(cây con phải)
oIF (Hệ số cân bằng > 1 && k<khóa của nút con bên trái)
     → rightRotate(nút đó)
o IF (Hệ số cân bằng > 1 && k>khóa của nút con bên trái)
     → leftrightRotate(nút đó)
oIF (Hệ số cân bằng < -1 && k>khóa của nút con bên phải)
     → leftRotate(nút đó)
oIF (Hệ số cân bằng < -1 && k<khóa của nút con bên phải)
     → rightleftRotate(nút đó)
```

IV. XÓA KHÓA K KHỎI CÂY AVL

- Thực hiện xóa khóa k khỏi cây tìm kiếm nhị phân. Gọi nút gốc của cây sau khi xóa là root.
- Tính lại chiều cao của root
- Hệ số cân bằng tại root **balance** = **height**(root left child) – **height**(root right child)
- IF (balance > 1 && Hệ số cân bằng tại *root left child* >=0)
  → **rightRotate**(root)
- IF (balance > 1 && Hệ số cân bằng tại *root left child* <0)
  → **leftrightRotate**(root)
- IF (balance < -1 && Hệ số cân bằng tại *root right child* <=0)
  → **leftRotate**(root)
- IF (balance < -1 && Hệ số cân bằng tại *root right child* >0)
  → **rightleftRotate**(root)

  → root