LAB 06 – CÂY NHỊ PHÂN TÌM KIẾM CÂN BẰNG (AVL)

1. HẠN CHẾ CỦA CÂY NHỊ PHÂN TÌM KIẾM

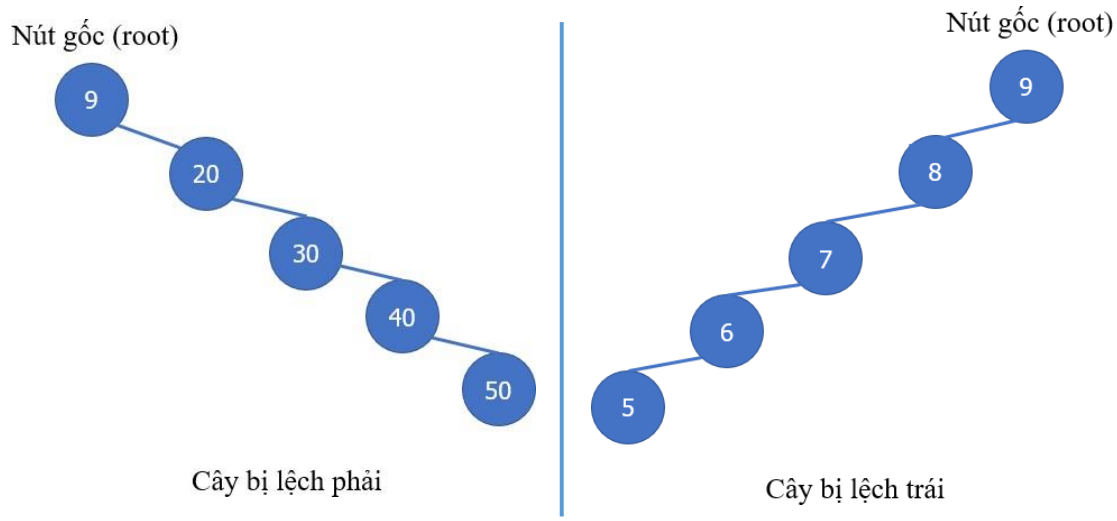
Cấu trúc dữ liệu cây nhị phân tìm kiếm là tập hợp các nút có mối liên hệ với nhau.

Mỗi cây có 1 nút gốc và có các cây con, liên kết thông qua quan hệ cha con.

Tính chất của cây nhị phân tìm kiếm:

- Tất cả các nút bên trái nút gốc phải có giá trị nhỏ hơn nút gốc.
- Tất cả các nút bên phải nút gốc phải có giá trị lớn hơn nút gốc.
- Cây con trái và cây con phải của nút gốc cũng là cây nhị phân tìm kiếm.

Nếu trường hợp các giá trị thêm vào cây tăng dần hoặc giảm dần, dẫn đến việc các thao tác tìm kiếm hay thêm, xóa giá trị không còn nhanh nữa.



Trong trường hợp lệch trái hay lệch phải:

- Chiều cao của cây con trái so với cây con phải lớn hơn 1
- Thao tác thêm, xóa hay tìm kiếm có chi phí tuyến tính O(n) so với mục tiêu ban đầu đề ra là O(logn)
- Không tận dụng hết các lợi ích của cấu trúc dữ liệu cây nhị phân

2. CÂY NHỊ PHÂN TÌM KIẾM CÂN BẰNG (AVL)

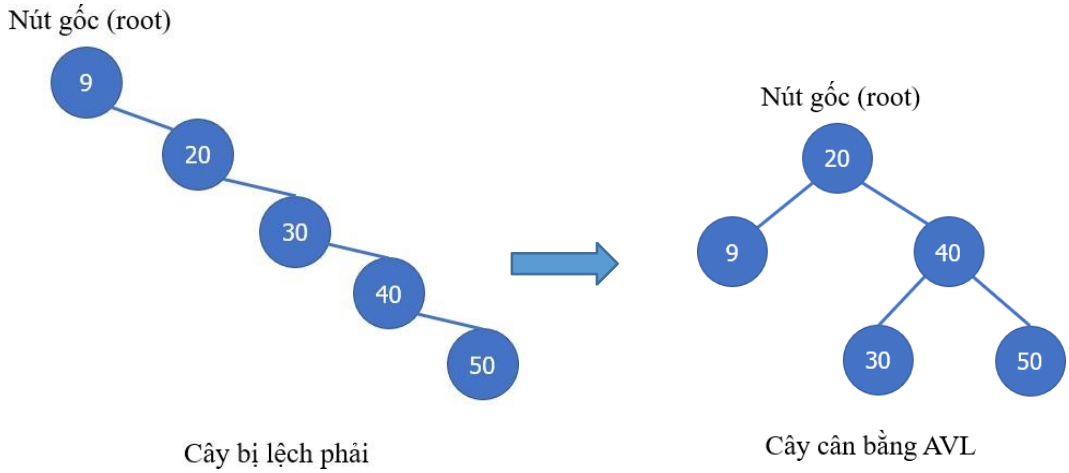
Để làm cho cây không còn bị lệch, cần phải cân bằng lại cây trong những trường hợp trên.

Cây AVL bổ sung việc cân bằng cây bằng cách kiểm tra các trường hợp mất cân bằng dựa trên chiều cao của cây và tiến hành các thao tác xoay cây.

Chiều cao của cây: số liên kết từ nút gốc đến nút lá cộng 1.

```
 //Hàm tính chiều cao cây
int height(NODE* pRoot) {
    if (pRoot == NULL) return 0;// cây rỗng,chiều cao của cây = 0
    else return 1 + max(height(pRoot->pLeft), height(pRoot->pRight));
 }
```

Hình minh họa trường hợp cân bằng cây



3. CÁC THAO TÁC CƠ BẢN VÀ XỬ LÝ MẤT CÂN BẰNG

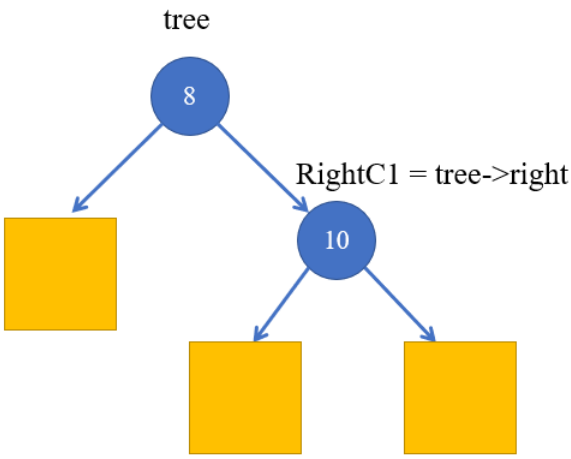
a. ***<u>Các thao tác xoay cây:</u>*** Cây AVL định nghĩa 2 thao tác chính để hỗ trợ cân bằng lại cây nhị phân tìm kiếm. Bao gồm:

- Xoay trái tại một nút
- Xoay phải tại một nút

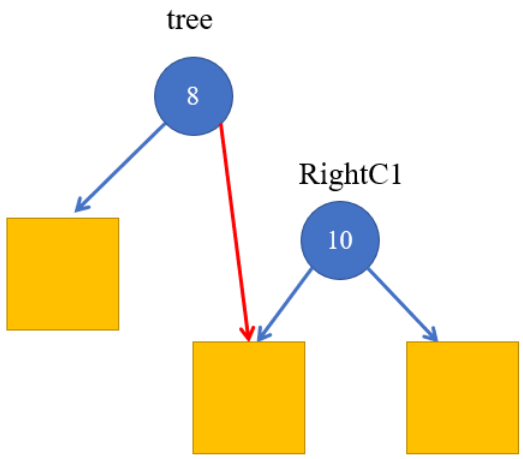
➢ Thao tác xoay trái tại một nút:

- Bước 1. Gán nút con bên phải để giữ liên kết
- Bước 2: Gán liên kết bên phải của nút cha là liên kết bên trái của nút con.
- Bước 3: Gán liên kết bên trái của nút con là nút cha
- Bước 4: Cập nhật lại nút cha mới

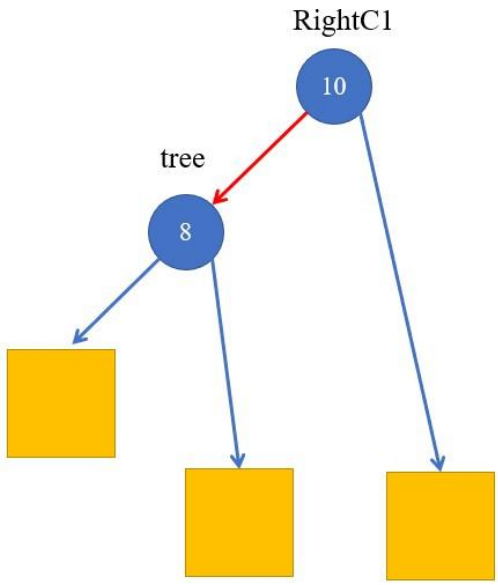
Hình minh họa các bước của thao tác xoay trái tại nút tree



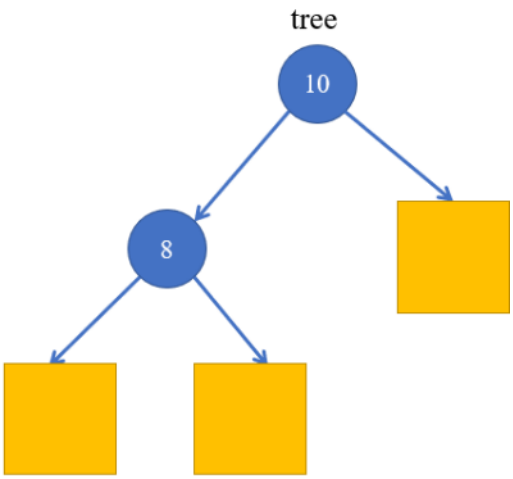
1. Gán nút **RightC1 = tree->right** để giữ liên kết



2. Gán liên kết phải của tree bằng liên kết trái của RightC1
Tree->right = RightC1->left



3. Gán liên kết trái của RightC1 là tree (nút con lên làm cha, nút cha xuống làm con bên trái)
RightC1->left = tree



4. Gán lại nút cha

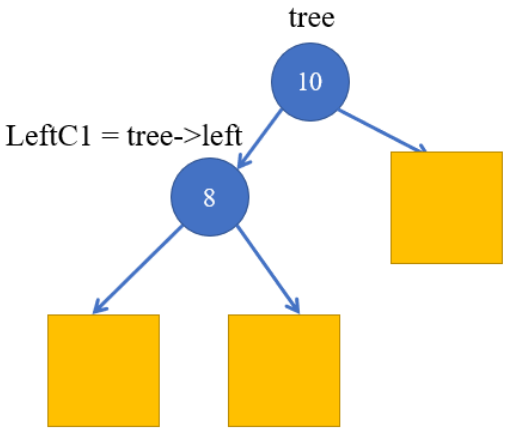
tree = RightC1

Code tham khảo:

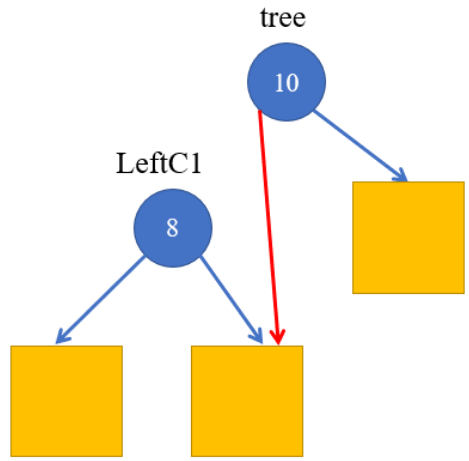
```
//Hàm xoay trái
void rotateLeft(NODE*& tree) {
    NODE* RightC1 = tree ->pRight;
    tree ->pRight = RightC1 ->pLeft;
    RightC1 ->pLeft = tree;
    tree = RightC1;
}
```

- Thao tác xoay phải tại một nút: (ngược lại với xoay trái)
  - Bước 1. Gán nút con bên trái để giữ liên kết
  - Bước 2: Gán liên kết bên trái của nút cha là liên kết bên phải của nút con.
  - Bước 3: Gán liên kết bên phải của nút con là nút cha
  - Bước 4: Cập nhật lại nút cha mới

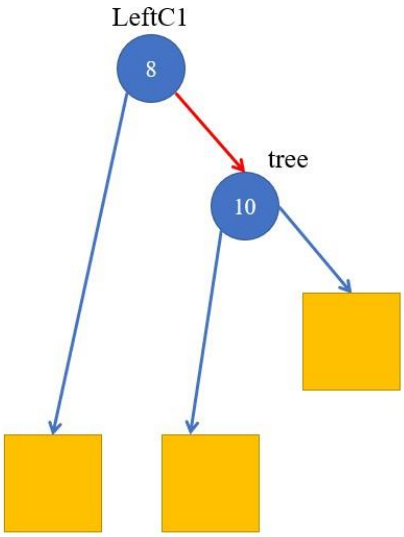
Hình minh họa các bước của thao tác xoay phải tại nút tree



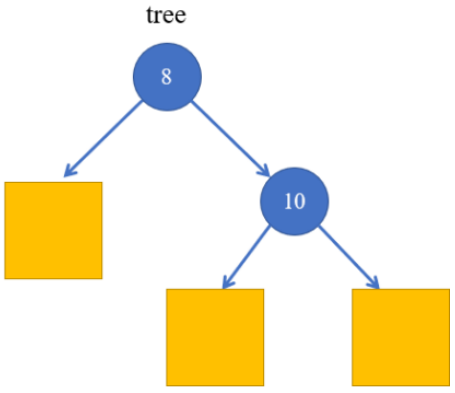
1. Gán nút **LeftC1 = tree->left** để giữ liên kết



2. Gán liên kết trái của tree bằng liên kết phải của LeftC1
Tree->left = LeftC1->right



3. Gán liên kết phải của LeftC1 là tree (nút con lên làm cha, nút cha xuống làm con bên trái)
LeftC1->right = tree



4. Gán lại nút cha
tree= LeftC1

Sinh viên hoàn thành đoạn code sau:

```
//Hàm xoay phải
void rotateRight(NODE*& tree) {


}
```

b. *Các trường hợp mất cân bằng và cách xử lý*

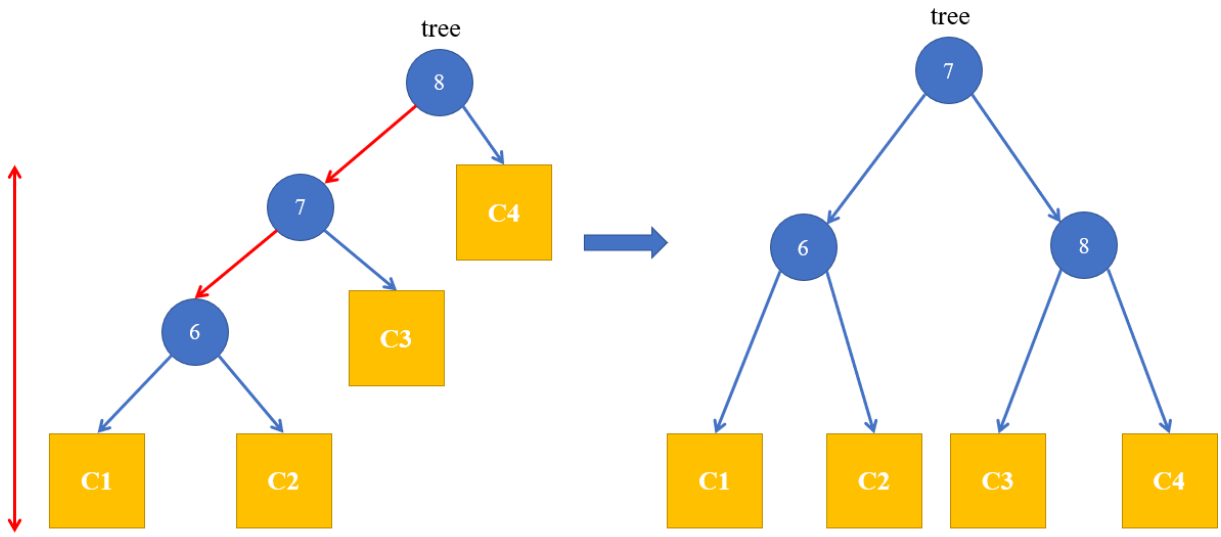
Có 4 trường hợp mất cân bằng xảy ra, dựa theo vị trí, bao gồm:

- Mất cân bằng trái – trái
- Mất cân bằng trái – phải
- Mất cân bằng phải – phải
- Mất cân bằng phải – trái

Cách xử lý:

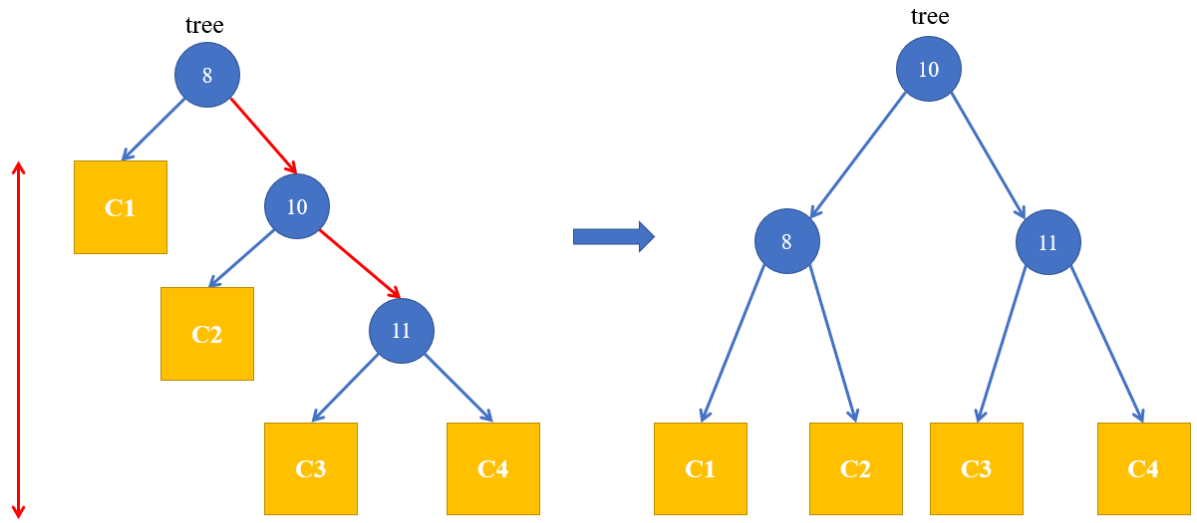
- Nếu cùng hướng: quay ngược hướng 1 lần tại nút bị mất cân bằng
  - Mất cân bằng **trái – trái**: quay **phải** 1 lần tại nút bị mất cân bằng
  - Mất cân bằng **phải – phải**: quay **trái** 1 lần tại nút bị mất cân bằng
- Nếu ngược hướng: quay 2 lần và quay cùng hướng, lần 1 tại nút con của nút bị mất cân bằng (vị trí cùng hướng với hướng quay), lần 2 tại nút cha bị mất cân bằng.
  - Mất cân bằng **trái – phải**: quay **trái** tại nút **con trái** của nút mất cân bằng, sau đó quay **phải** tại nút mất cân bằng
  - Mất cân bằng **phải – trái**: quay **phải** tại nút **con phải** của nút mất cân bằng, sau đó quay **trái** tại nút mất cân bằng

Minh họa trường hợp mất cân bằng trái – trái:



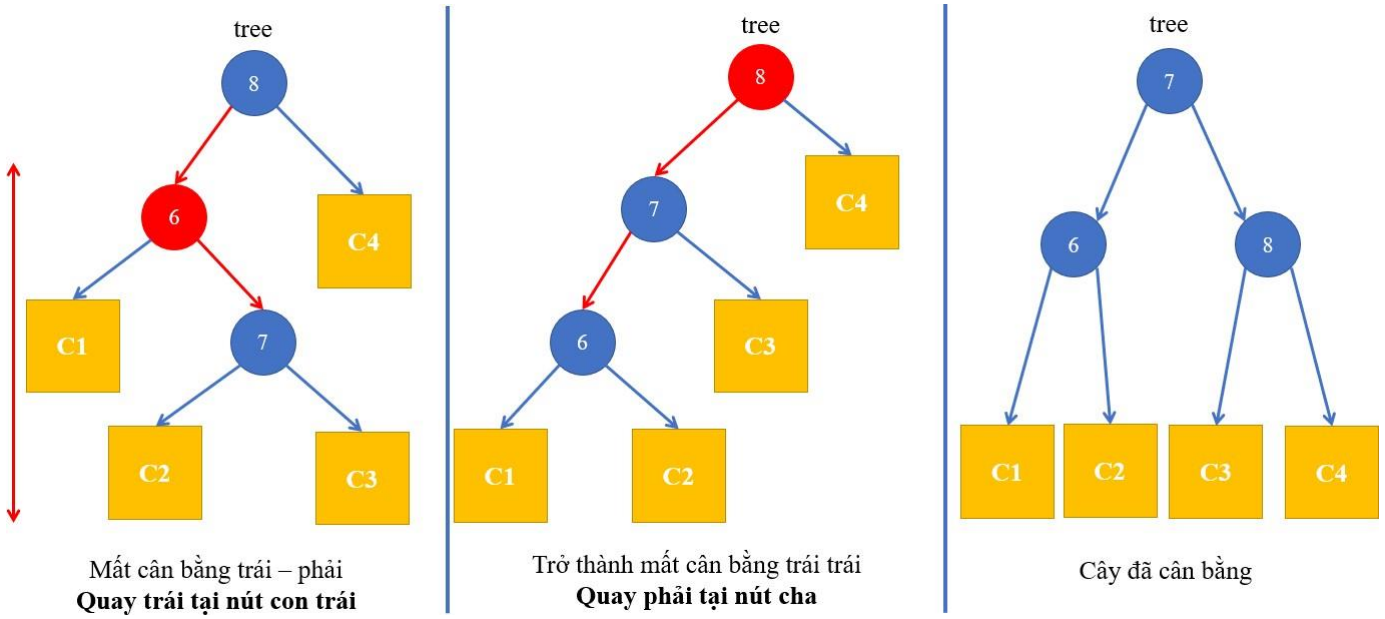
Mất cân bằng trái trái, quay phải 1 lần tại nút tree

Minh họa trường hợp mất cân bằng phải – phải:



Mất cân bằng phải phải, quay trái 1 lần tại nút tree

Minh họa trường hợp mất cân bằng trái – phải:

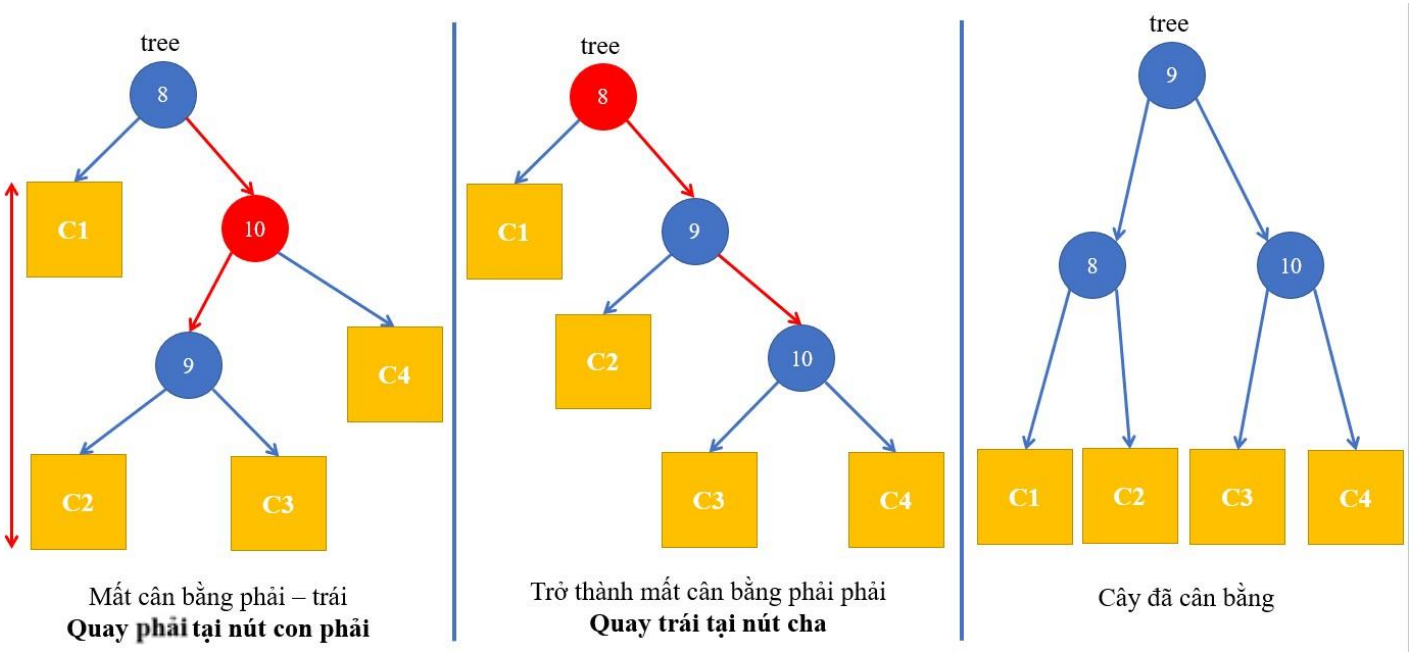


Mất cân bằng trái – phải
Quay trái tại nút con trái

Trở thành mất cân bằng trái trái
Quay phải tại nút cha

Cây đã cân bằng

Minh họa trường hợp mất cân bằng phải – trái:



Tại mỗi thao tác thêm hoặc xóa dữ liệu, cần phải kiểm tra mất cân bằng và cân bằng lại cây từ nút đó.

Nút mất cân bằng khi chiều cao của cây con trái và chiều cao của cây con phải của nó lệch nhau hơn 1

```
 //Sinh viên hoàn thành Hàm cân bằng
void balancing(NODE*& pRoot) {
    int leftHeight = height(pRoot->pLeft); //chiều cao của cây con trái
    int rightHeight = height(pRoot->pRight); //chiều cao của cây con phải
    if (abs(leftHeight - rightHeight) > 1) // nếu chiều cao của cây con trái và chiều cao
của cây con phải lệch nhau hơn 1=> cây mất cân bằng, cân bằng lại cây theo 4 trường hợp
       {
        if (leftHeight > rightHeight) {
           //TODO: code here
           //nếu chiều cao của cây con trái của con trái >= chiều cao của cây con phải của
       con trái

                //Left-Left=>xoay phải (1)

            //Ngược lại:
                //Left-Right=> xoay trái tại con trái, xoay phải tại pRoot  (2)

        }
        else {
            //TODO: code here
           //nếu chiều cao của cây con phải của con phải >= chiều cao của cây con trái của
       con phải

                //Right-Right => xoay trái (3)

           //Ngược lại:
                //Right-Left => Xoay phải tại con phải, xoay trái tại pRoot (4)

        }
    }
}
```

4. BÀI TẬP THỰC HÀNH

Sinh viên nộp bài "Lab06.cpp":

Dựa vào file BST.cpp là chương trình C++ cài đặt cấu trúc dữ liệu cây nhị phân tìm kiếm (Lab05) hoặc bài LAB05 sinh viên đã làm hãy cập nhật code để hoàn thành cây nhị phân tìm kiếm cân bằng AVL với các yêu cầu sau:

- Thêm node, tìm kiếm node với giá trị k, xóa node có giá trị x
- Xuất các giá trị trên cây theo thứ tự giảm dần
- Tính tổng tất cả giá trị của cây
- Tính tổng giá trị của tất cả nút **lá** trên cây

Gợi ý tên các hàm:

```
void rotateLeft(NODE*& ) // xoay trái
void rotateRight(NODE*& ) //xoay phải
void balancing(NODE*& ) // cân bằng
void insertNode(NODE*&, int) //thêm 1 node vào cây, phải cân bằng lại sau khi thêm
void deleteNode(NODE*&, int) //xóa 1 node khỏi cây, phải cân bằng lại sau khi xóa
void outputDecrease (NODE*) //xuất theo thứ tự giảm dần
int SumAllNodes(NODE*) //Tổng cả cây
int SumLeaves(NODE*)  //Tổng node lá
```

- HẾT-