CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Bài 08: Cây tìm kiếm nhị phân, Cây cân bằng AVL

Ngô Thế Quyền
ngoquyenbg@hus.edu.vn

Ngày 21 tháng 10 năm 2024

Nội dung

Bài toán tìm kiếm

Bài toán tìm kiếm

Cho dãy cách phần tử $a[0..n-1]$ và phần tử x, ta cần tìm vị trí *index* sao cho $a[index] = x$ hoặc trả về giá trị -1 nếu không có phần tử x trong dãy.

Tìm kiếm tuần tự

Tìm kiếm tuần tự - linear seach or sequential search

Thuật toán: Bắt đầu từ phần tử đầu tiên, duyệt qua từng phần tử cho đến khi tìm được phần tử đích hoặc kết luận không tìm được

```
int linearSearch (int arr[], int i)
{
    for (int i = 0; i<arr.length; i++){
        if (arr[i]==x){
            return i;
        }
    }
    return -1;
}
```

Tìm kiếm nhị phân trên mảng được sắp

Tìm kiếm nhị phân - binary search

Ý tưởng: Chia đôi mảng, mỗi lần so sánh phần tử giữa với x, nếu phần tử x nhỏ hơn thì xét nửa trái, ngược lại xét nửa phải

```
int binarySearch(int arr[], int x)
{
    int l = 0, r = arr.length - 1;
    while (l <= r) {
        int m = l + (r - l) / 2;
        if (arr[m] == x)
            return m;
        if (arr[m] < x)
            l = m + 1;
        else
            r = m - 1;
    }
    return -1;
}
```

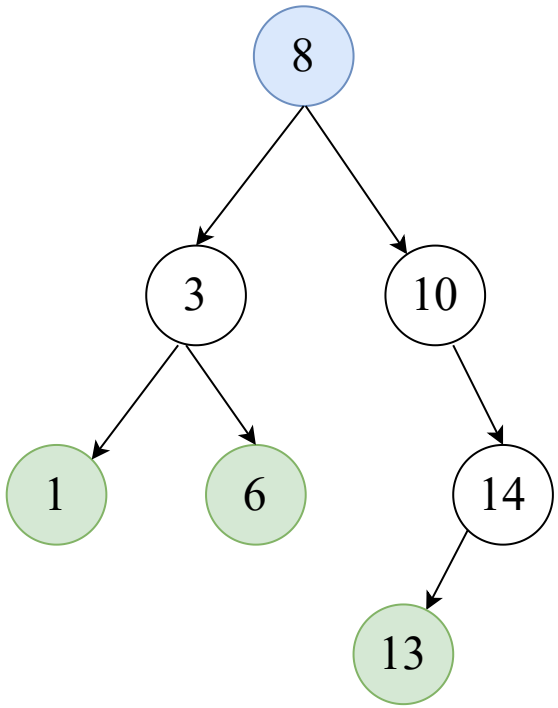
Nội dung

Cây tìm kiếm nhị phân

Định nghĩa cây tìm kiếm nhị phân - Binary Search Tree - BST

Cây tìm kiếm nhị phân là cậy nhị phân mà mỗi nút trong cây đều có một khoá, mọi khoá k đều thoả mãn điều kiện:

- Mọi khoá trên cây con trái đều nhỏ hơn khoá k
- Mọi khoá trên cây con phải đều lớn hơn khoá k



Biểu diễn cây BST

```
class BSTNode {
    int data;
    BSTNode left;
    BSTNode right;
    public BSTNode(int data)
    {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}
```

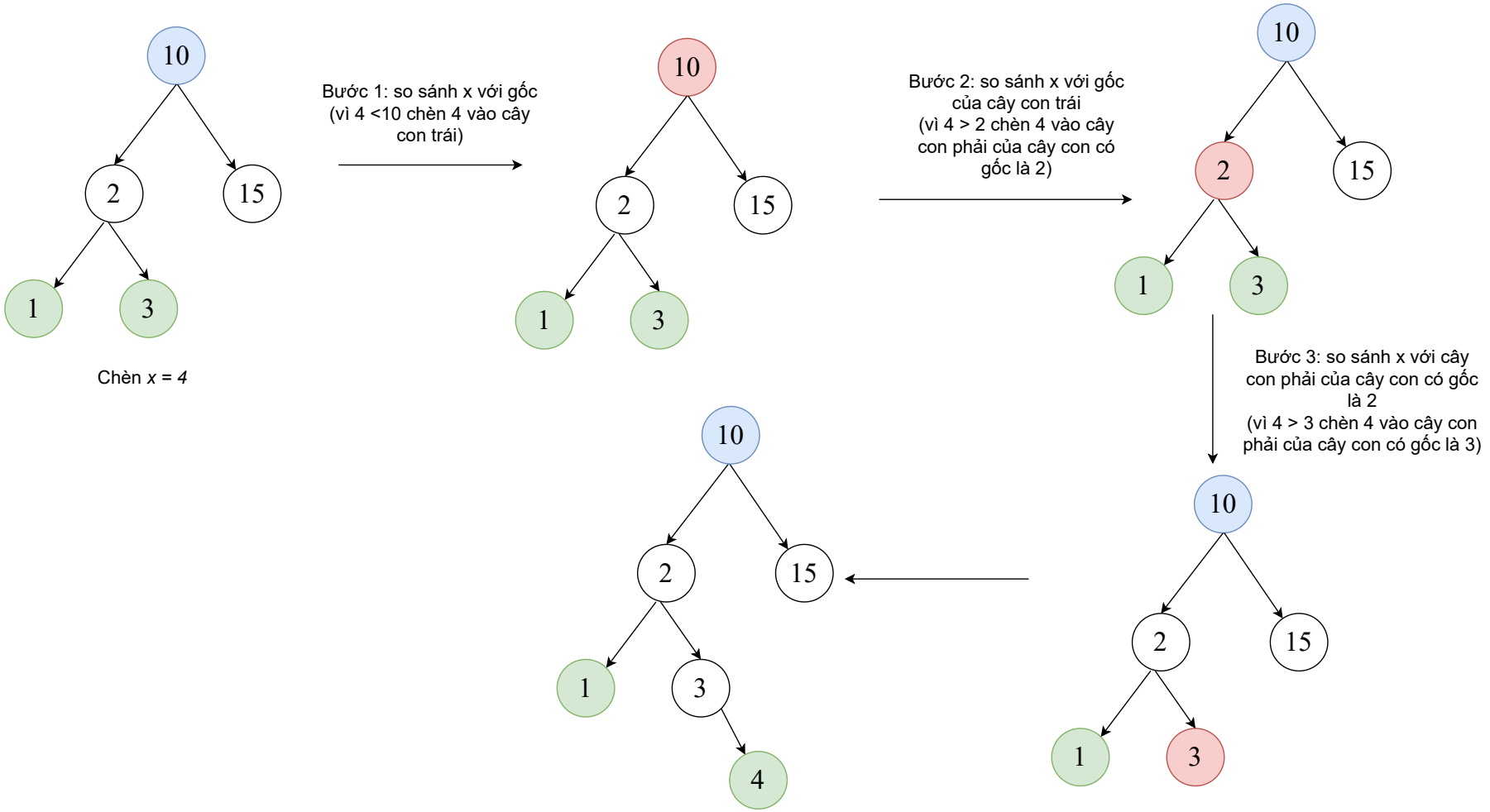
Các thao tác cơ bản trên BST

Chèn phần tử

- Trường hợp 1: Cây rỗng
  - Tạo một nút mới
  - Cho cây con trái và cây con phải của nút mới nhận giá trị **null**
- Trường hợp 2: Trong cây có tồn tại phần tử
  - So sánh giá trị của nút mới thêm vào với gốc.
  - Nếu giá trị mới thêm vào nhỏ hơn gốc thì thêm vào cây con trái (thực hiện đệ quy việc thêm nút ở cây con trái)
  - Nếu giá trị mới thêm vào lớn hơn gốc thì thêm vào cây con phải (thực hiện đệ quy việc thêm nút ở cây con phải)
  - Gắn nút con là nút con của nút cha tìm được. Chú ý là nút mới thêm vào luôn là nút lá

Các thao tác cơ bản trên cây BST

Chèn phần tử



Các thao tác cơ bản trên cây BST

Chèn phần tử

```
BSTNode root;
void insert(int data) {
    root = insertNode(root, data);
}

BSTNode insertNode(BSTNode root, int data) {
    // TH1
    if (root == null) {
        root = new BSTNode(data);
        return root;
    }
    // TH2
    else if (data < root.data)
        root.left = insertNode(root.left, data);
    else if (data > root.data)
        root.right = insertNode(root.right, data);
    return root;
}
```

Các thao tác cơ bản trên cây BST

Xoá phần tử

Khi loại bỏ một nút, cần phải đảm bảo cây thu được vẫn là cây nhị phân tìm kiếm. Có 4 trường hợp có thể xảy ra khi xoá nút

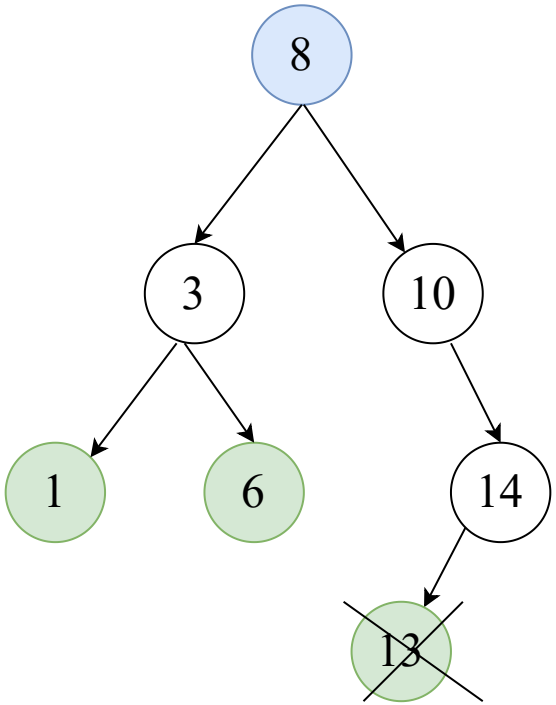
- Trường hợp 1: Nút cần xoá là lá
- Trường hợp 2: Nút cần xoá chỉ có con trái
- Trường hợp 3: Nút cần xoá chỉ có con phải
- Trường hợp 4: Nút cần xoá có hai con

Các thao tác cơ bản trên cây BST

Xoá phần tử

Trường hợp 1: Nút cần xoá là nút lá

Thao tác: Chữa lại nút cha của nút cần xoá có con rỗng

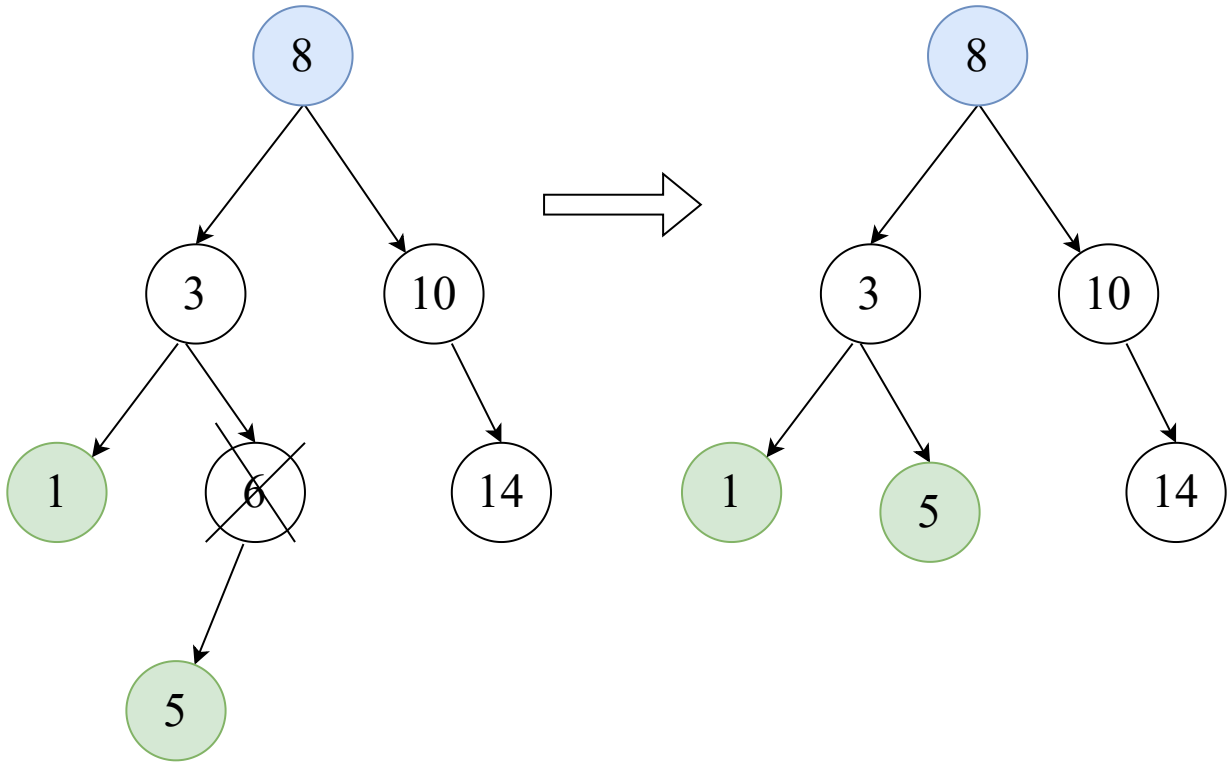


Các thao tác cơ bản trên cây BST

Xoá phần tử

Trường hợp 2: Nút cần xoá có con trái mà không có con phải

Thao tác: Gắn cây con trái của nút cần xoá vào nút cha của nút cần xoá

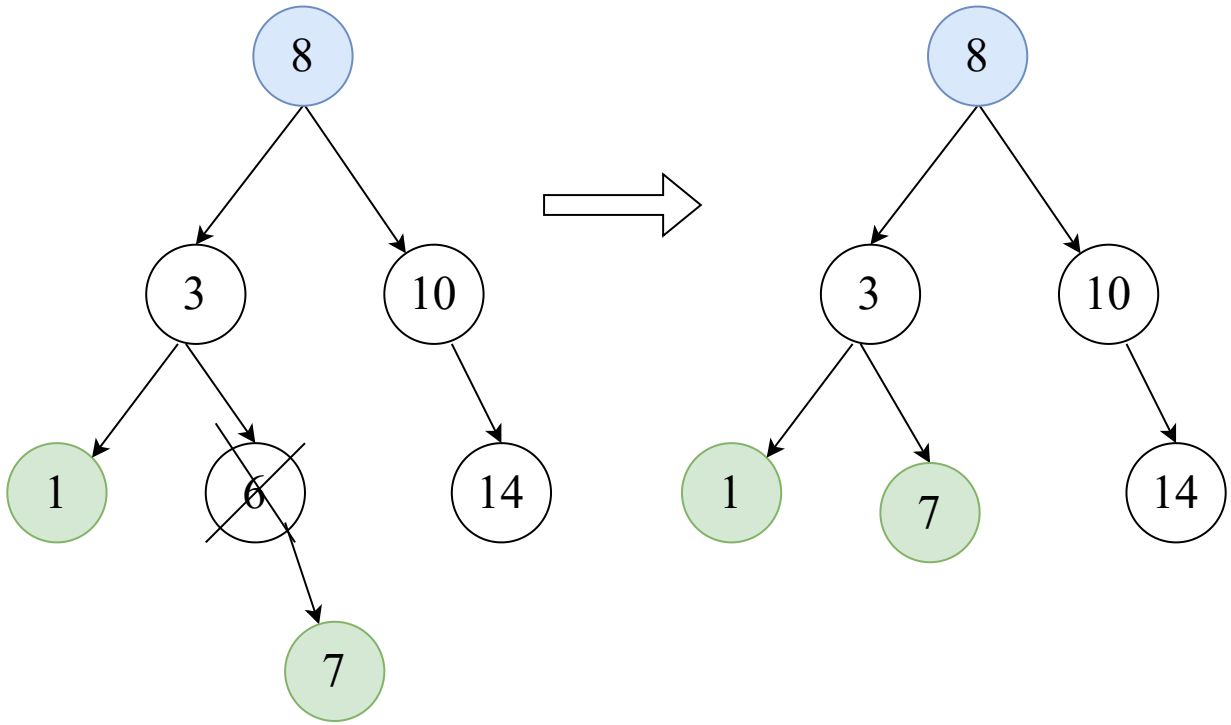


Các thao tác cơ bản trên cây BST

Xoá phần tử

Trường hợp 3: Nút cần xoá có con phải mà không có con trái

Thao tác: Gắn cây con phải của nút cần xoá vào nút cha của nút cần xoá



Các thao tác cơ bản trên cây BST

Xoá phần tử

Trường hợp 4: Nút cần xoá x có đầy đủ hai con

Thao tác:

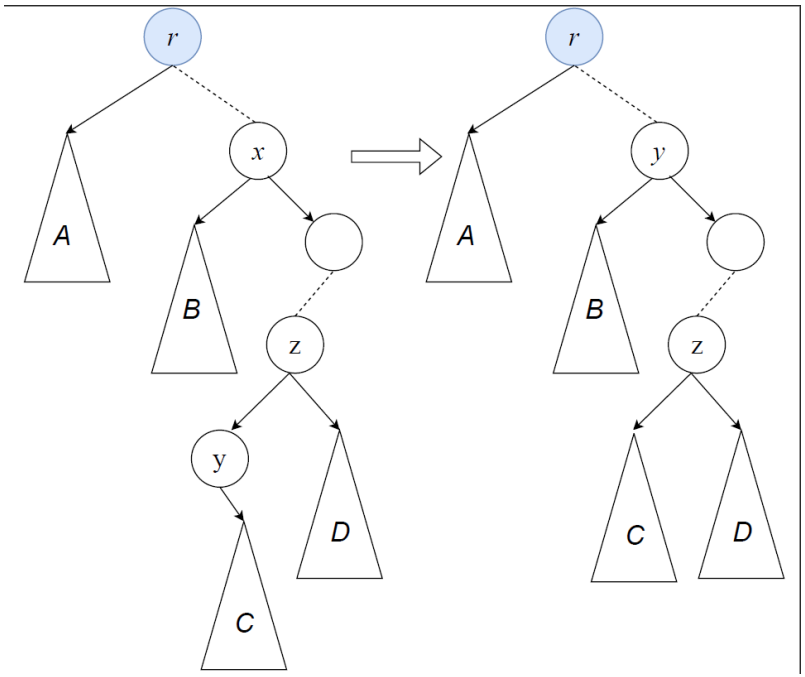
1. Chọn nút y để thế vào chỗ của nút x, nút y sẽ là nút kết tiếp của nút x. Như vậy, y là giá trị nhỏ nhất còn lớn hơn x, nói cách khác, y là giá trị nhỏ nhất của cây con phải của x
2. Nối con phải của y vào cha của y
3. Thay thế y vào nút cần xoá

Các thao tác cơ bản trên cây BST

Xoá phần tử

Trường hợp 4: Nút cần xoá x có đầy đủ hai con

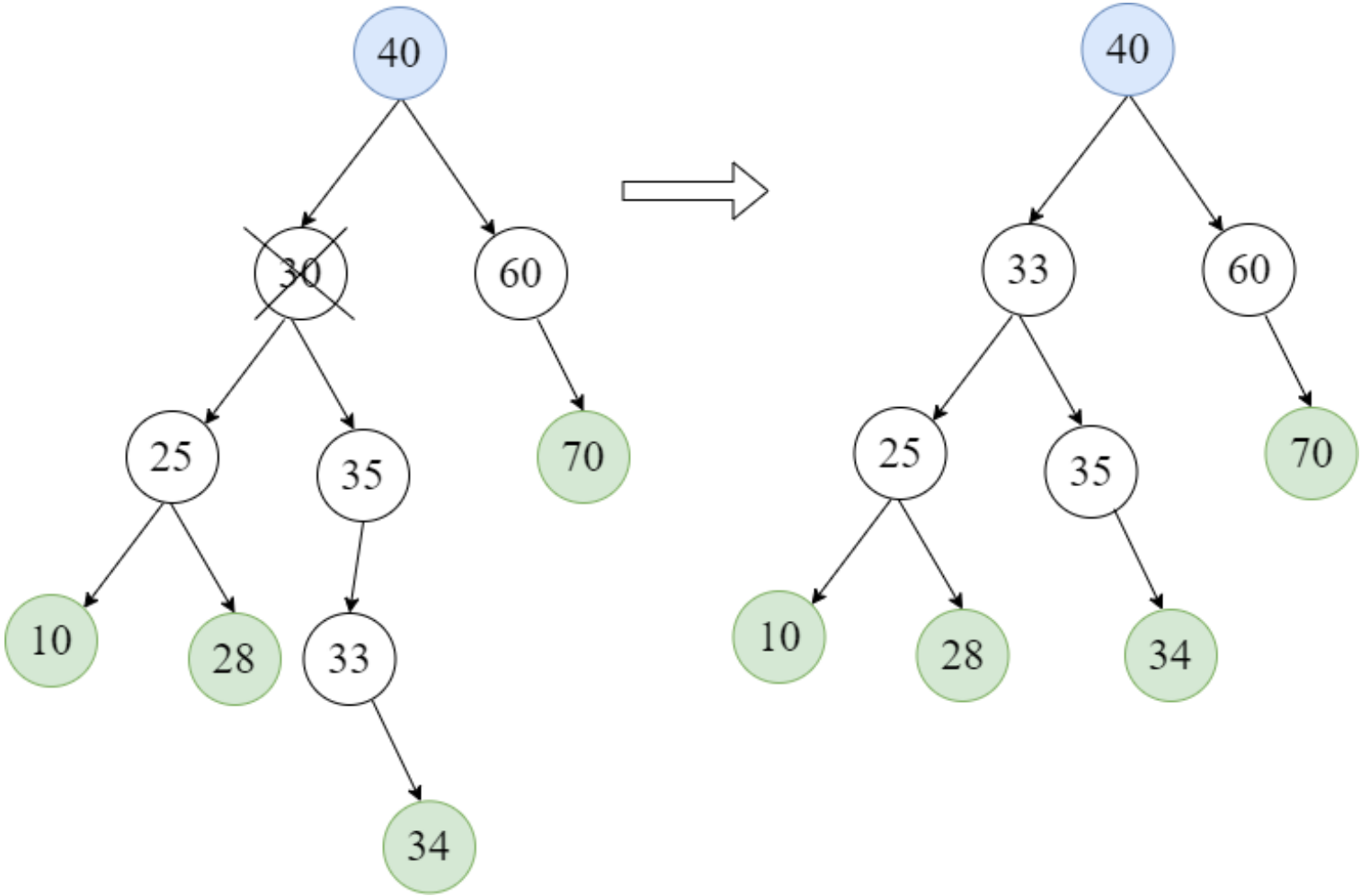
1. Chọn nút y là giá trị nhỏ nhất của cây con phải của x
2. Nối con phải của y vào cha của y
3. Thay thế y vào nút cần xoá



Các thao tác cơ bản trên cây BST

Xoá phần tử

Trường hợp 4: Nút cần xoá x có đầy đủ hai con



Các thao tác cơ bản trên cây BST

Xoá phần tử

```
public BSTNode deleteNode(BSTNode root, int key) {
    if (root == null)
        return root;
    if (key < root.data)
        root.left = deleteNode(root.left, key);
    else if (key > root.data)
        root.right = deleteNode(root.right, key);
    else {
        if (root.left == null)
            return root.right;
        else if (root.right == null)
            return root.left;

        root.data = minValue(root.right);
        root.right = deleteNode(root.right, root.data);
    }
    return root;
}
```

Các thao tác cơ bản trên cây BST

Xoá phần tử

```
int minValue(BSTNode root) {
    int minv = root.data;
    while (root.left != null) {
        minv = root.left.data;
        root = root.left;
    }
    return minv;
}
```

Tìm kiếm trên cây BST

- Nếu khoá cần tìm bằng nút hiện tại
- Nếu khoá cần tìm nhỏ hơn nút hiện tại thì tìm tiếp cây con trái
- Nếu khoá cần tìm lớn hơn nút hiện tại thì tìm tiếp cây con phải

Sắp xếp sử dụng BST

Sắp xếp sử dụng BST

Duyệt cây BST theo thứ tự giữa ra dãy khoá được sắp xếp. Nên ta có thể sử dụng cây BST để giải quyết bài toán sắp xếp:

- Xây dựng cây BST tương ứng với dãy số đã cho bằng cách chèn từng khoá trong dãy vào cây BST
- Duyệt cây BST thu được theo thứ tự giữa để đưa ra dãy được sắp

Phân tích hiệu quả của sắp xếp sử dụng cây BST

- Tình huống trung bình: $O(nlogn)$ vì chèn phần tử thứ i tốn khoản $log_2(i)$ phép so sánh
- Tình huống tồi nhất: $O(n^2)$ bởi vì bổ sung phần tử $i+1$ tốn i phép so sánh. Ví dụ dãy được sắp

Độ phức tạp của việc sắp xếp sử dụng BST

- Độ phức tạp trung bình của các thao tác. Do độ cao trung bình của cây BST là : $h = O(logn)$, từ đó suy ra độ phức tạp trung bình của các thao tác với BST là:
  - Chèn $O(logn)$
  - Xoá $O(logn)$
  - Tìm giá trị lớn nhất/nhỏ nhất $O(logn)$
  - Sắp xếp $O(nlogn)$
- Trường hợp cây bị mất cân đối: chiều cao của cây là $h = n$

Độ phức tạp của việc sắp xếp sử dụng BST

Vấn đề đặt ra

Có cách nào để tạo ra một câh BST sao cho chiều cao của cây là nhỏ nhất có thể, hay nói cách khác chiều cao $h = logn$. Có hai cách tiếp cận

- Luôn giữ cho cây cân bằng tại mọi thời điểm
- Thỉnh thoảng kiểm tra xem cây có mất cân bằng hay không

Cây AVL - AVL (Adelson-Velskii and Landis) Trees

Định nghĩa cây AVL

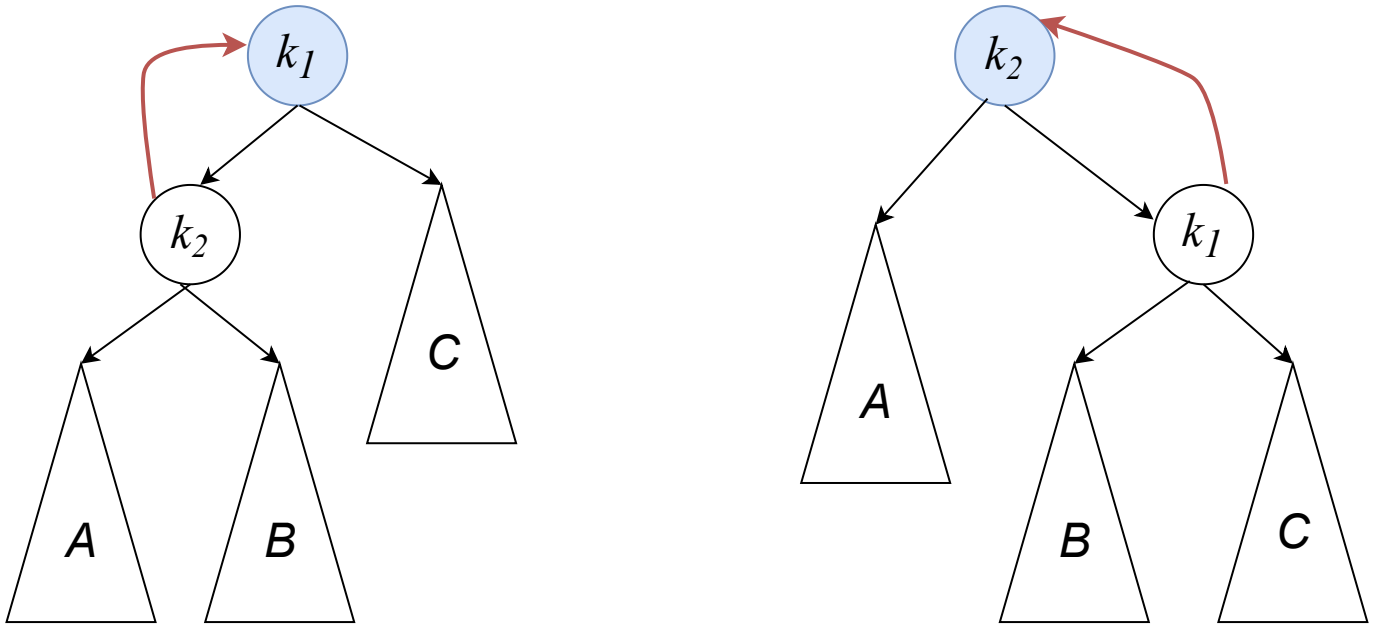
Một cây được gọi là cây AVL nếu:

- Là cây tìm kiếm nhị phân
- Với mỗi nút bất kì, chiều cao của cây con trái và chiều cao của cây con phải không vượt quá 1.
- Cả cây con phải và cây con trái đều là AVL

Hệ số cân bằng - balance factor

Hệ số cân bằng của nút x, ký kiệu là $bal(x)$ là hiều chiều cao của cây con trái trừ đi chiều cao của cây con phải của nút x

Các phép quay cây AVL



Quay phải quanh k_1 Quay trái quanh k_2

Hình 1: Hai phép quay cơ bản không làm mất tính chất của cây AVL

Các phép quay cây AVL

Phép quay phải

```
BSTNode rotateRight(BSTNode x) {
    if (x == null) {
        return x;
    }

    BSTNode xL = x.left;
    BSTNode xLR = xL.right;
    // Perform rotation
    x.left = xLR;
    xL.right = x;
     // Update heights
    x.height = 1 + Math.max(height(x.left), height(x.right));
    xL.height = 1 + Math.max(height(xL.left), height(xL.right));
    return xL;
}
```

Các phép quay cây AVL

Phép quay trái

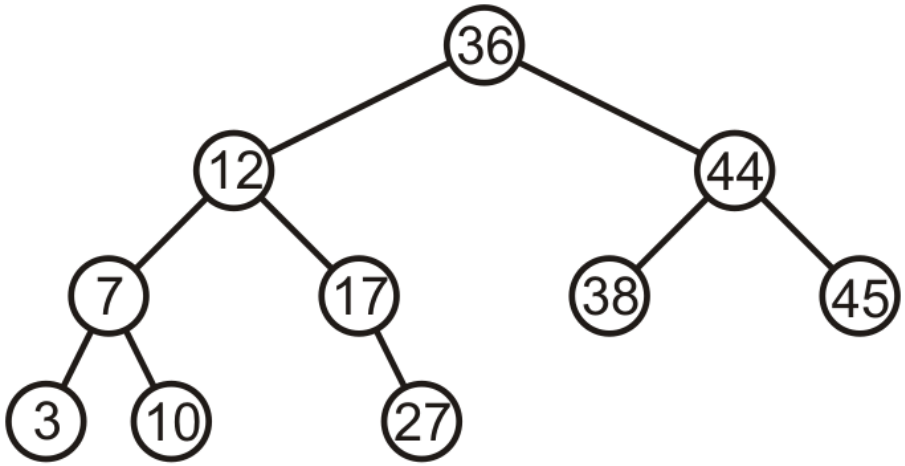
```
BSTNode rotateLeft(BSTNode x) {
    if (x == null) {
        return x;
    }
    BSTNode xR = x.right;
    BSTNode xRL = xR.left;
    // Perform Rotation
    x.right = xRL;
    xR.left = x;
    // Update heights
    x.height = 1 + Math.max(height(x.left), height(x.right));
    xR.height = 1 + Math.max(height(xR.left), height(xR.right));

    return xR;
}
```

Duy trì sự cân bằng của cây

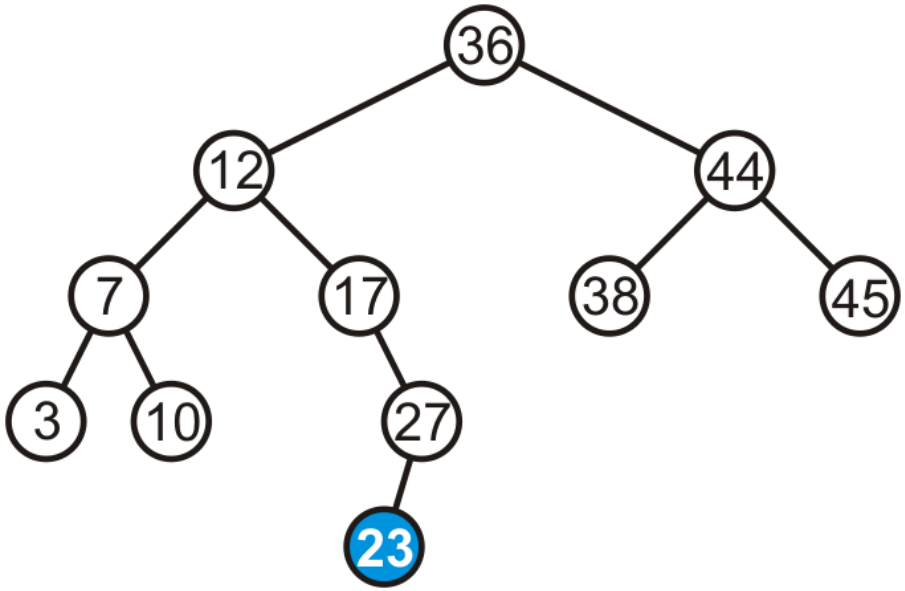
Nếu một cây là cây AVL đã cân bằng, khi chèn một nút có thể làm mất tính chất cân bằng của cây

- Chiều cao của cây con khác nhau 1
- Phép chèn làm tăng chiều cao của cây con sâu hơn lên 1



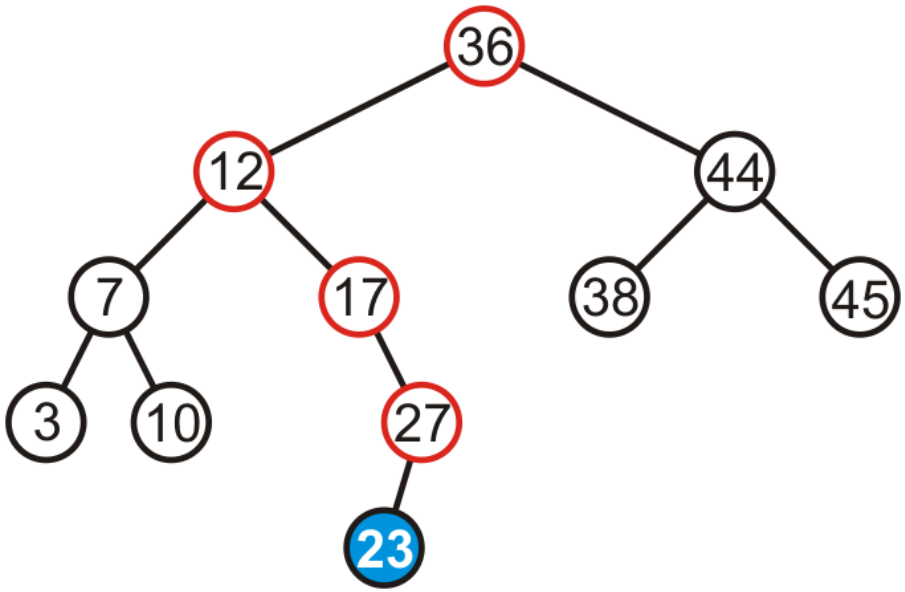
Duy trì sự cân bằng của cây (1)

Chèn nút 23 vào cây



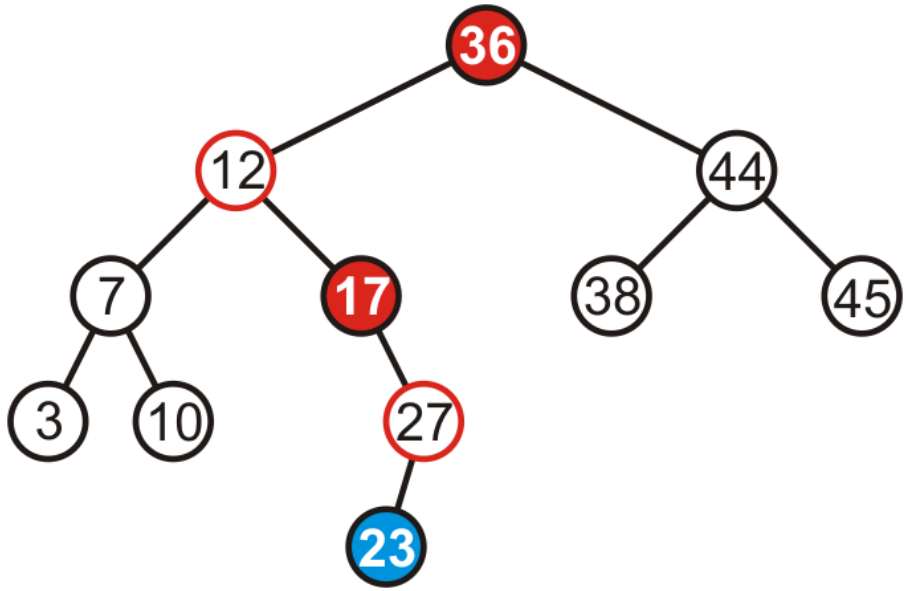
Duy trì sự cân bằng của cây (2)

Chiều cao của một số cây con thay đổi (tăng lên 1)



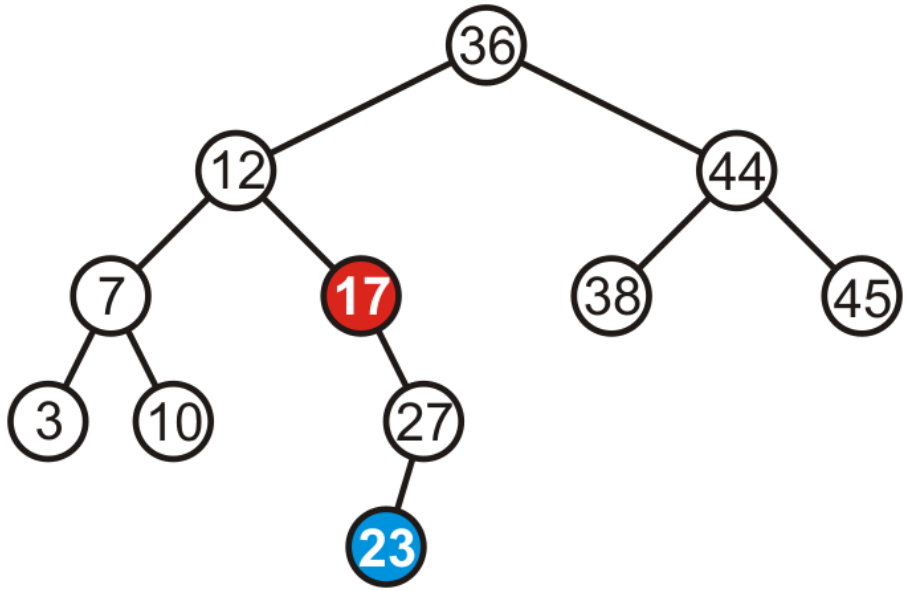
Duy trì sự cân bằng của cây (3)

Hai nút bị mất tính cân bằng: 17 và 36



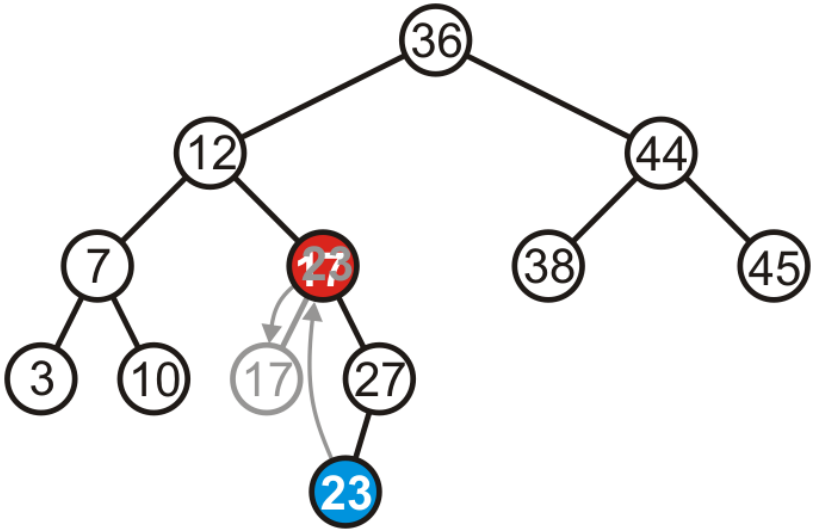
Duy trì sự cân bằng của cây (4)

Duy trì sự cân bằng tại nút 17

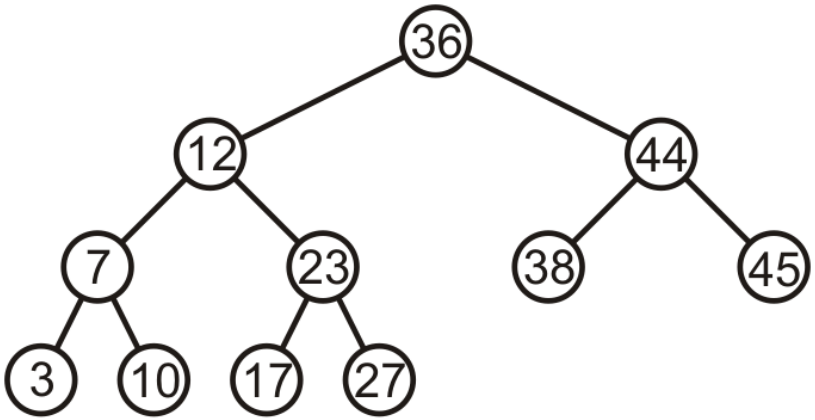


Duy trì sự cân bằng của cây (5)

Đổi nút 23 thế chỗ 17, biến 17 thành con trái của 23

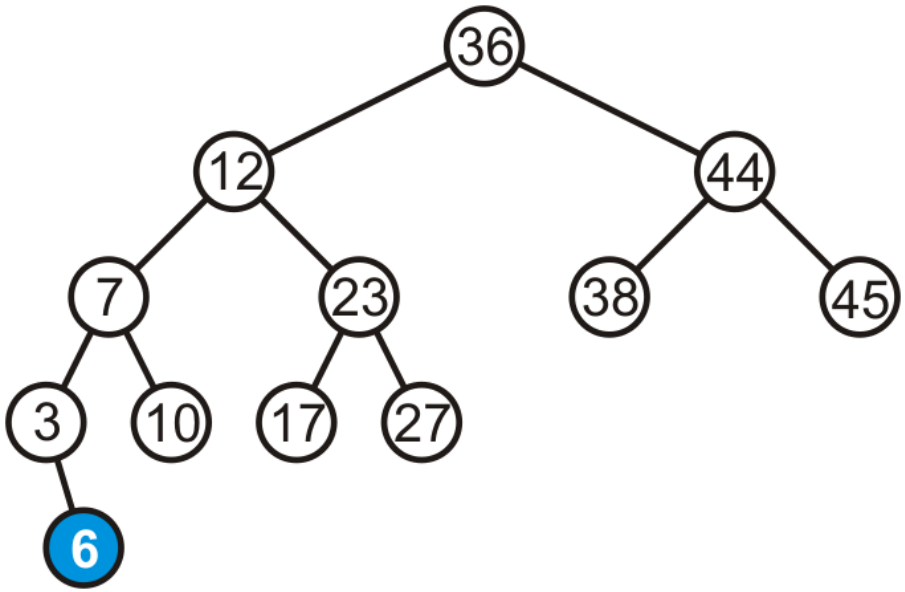


Nút mới đã cân bằng, ngẫu nhiên, nút gốc lúc này cũng cân bằng



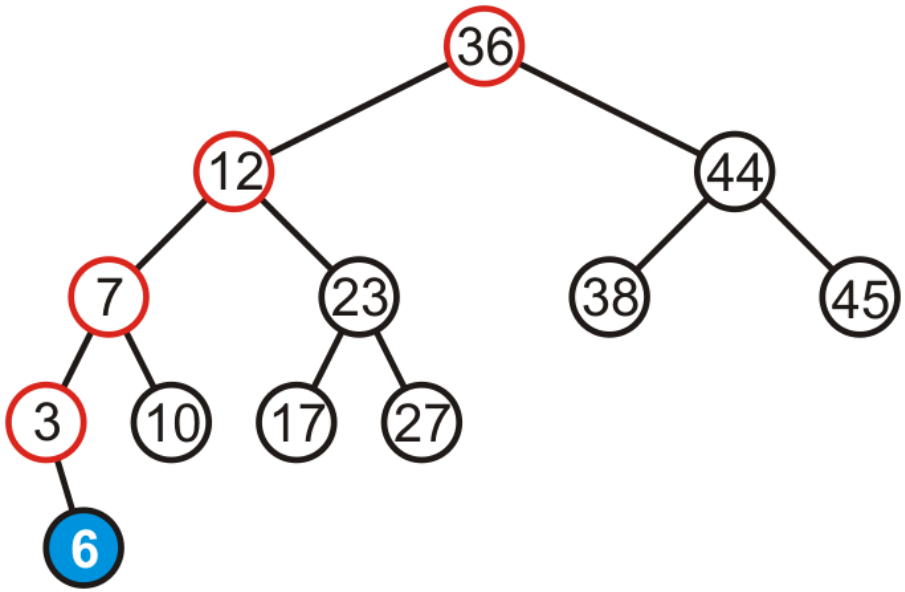
Duy trì sự cân bằng của cây (6)

Chèn thêm 6



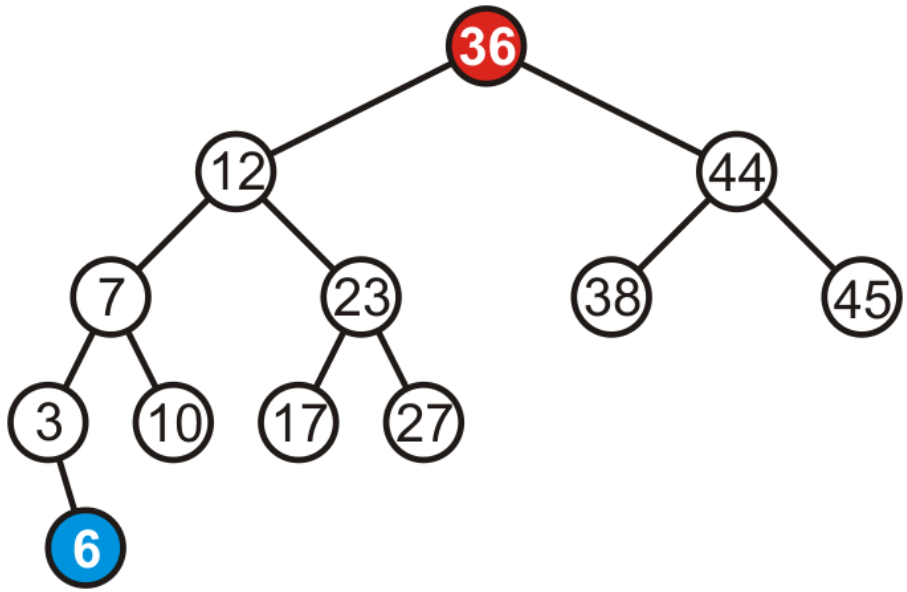
Duy trì sự cân bằng của cây (7)

Chiều cao của một số cây con thay đổi (tăng lên 1)



Duy trì sự cân bằng của cây (8)

Nút gốc mất cân bằng,...

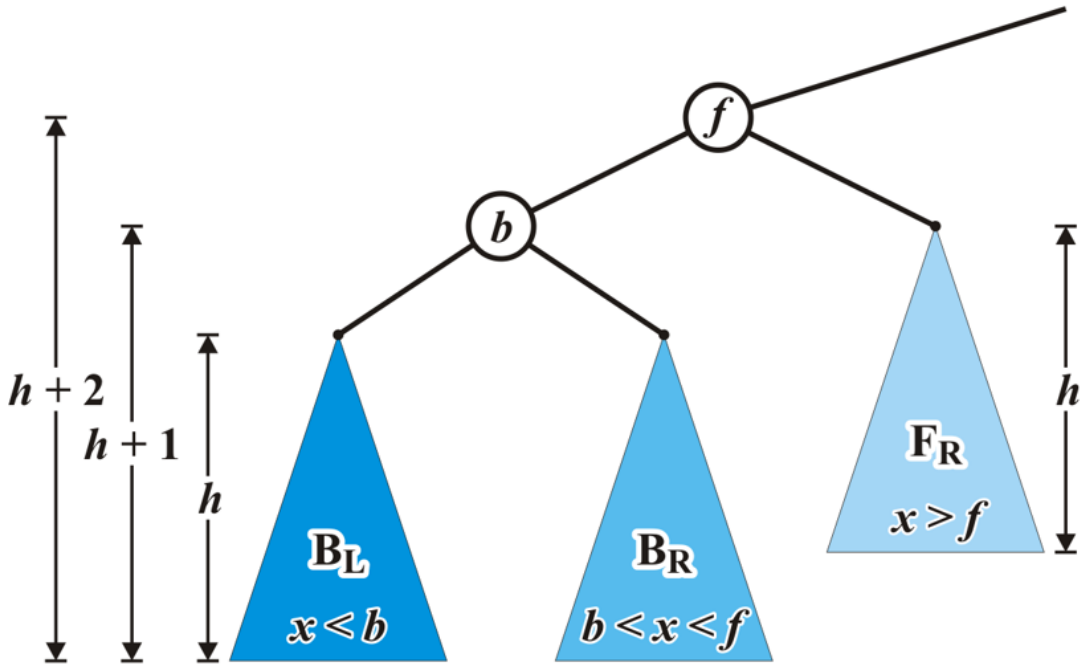


Để cân bằng cây, xét trường hợp tổng quát như sau

Duy trì sự cân bằng của cây

Trường hợp 1: LL

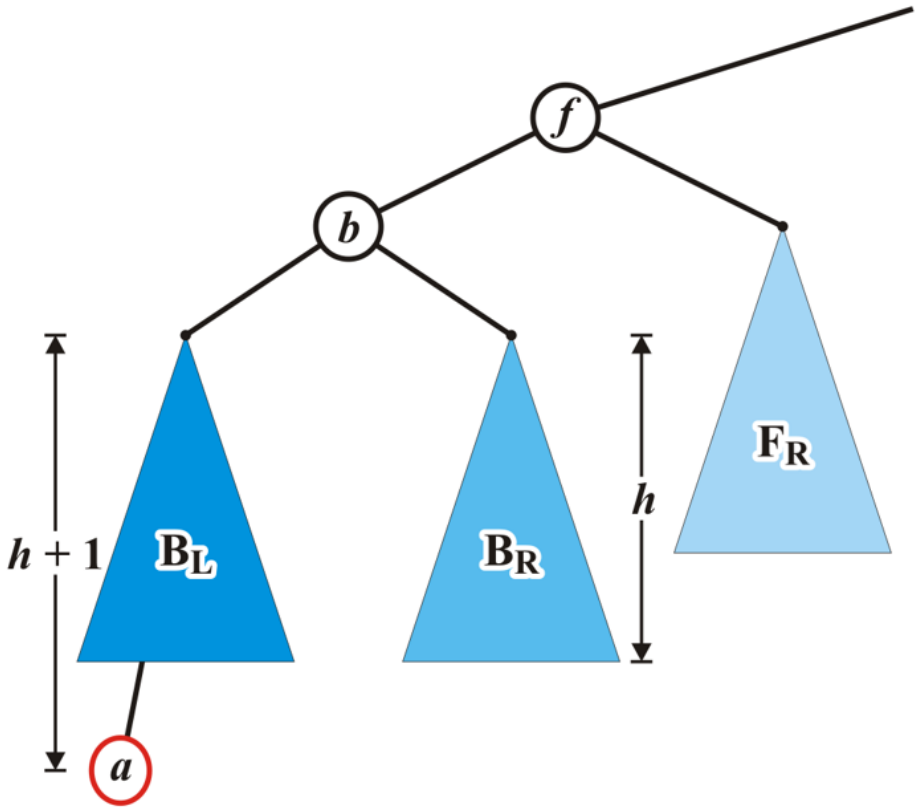
Mỗi hình tam tác màu xanh biểu diễn cho một cây có độ cao h



Duy trì sự cân bằng của cây

Trường hợp 1: LL

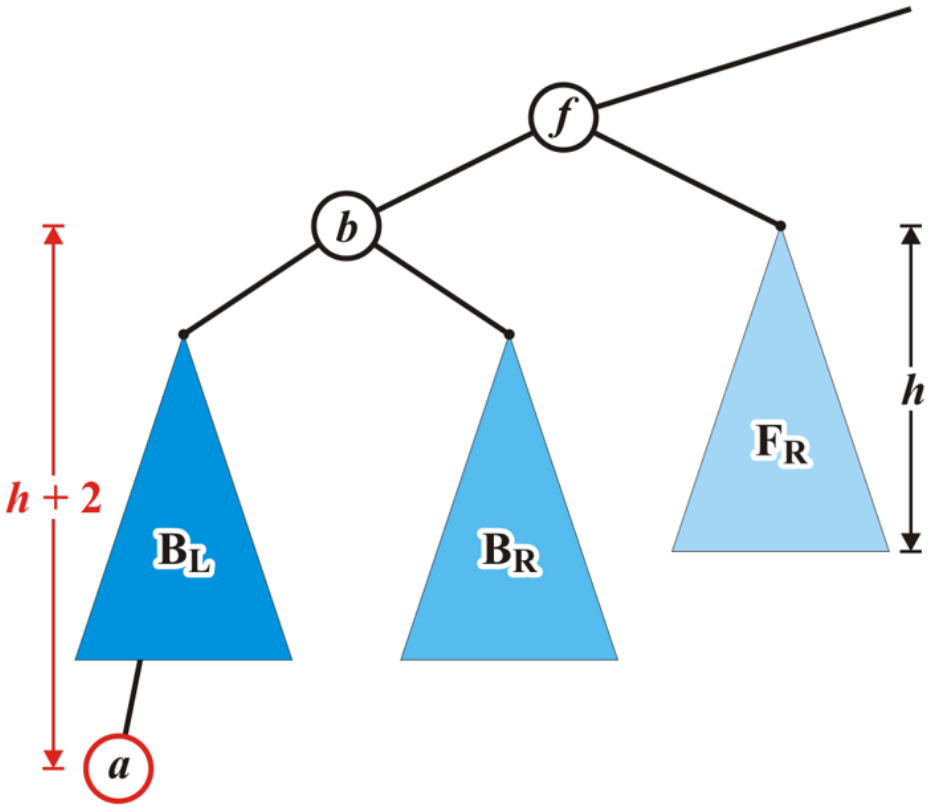
Chèn a vào cây: nếu a được chèn vào cây con trái của cây B_L của b. Nếu B_L cân bằng thì cây có gốc tại b cân bằng



Duy trì sự cân bằng của cây

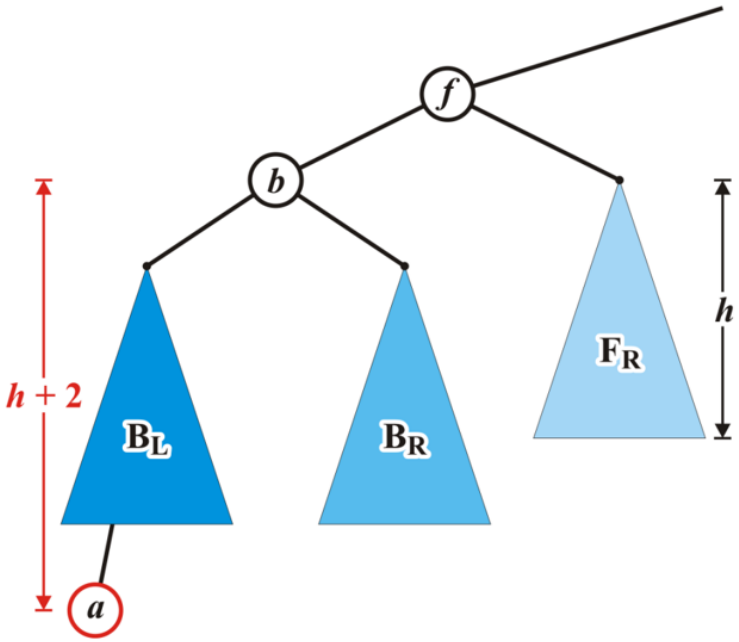
Trường hợp 1: LL

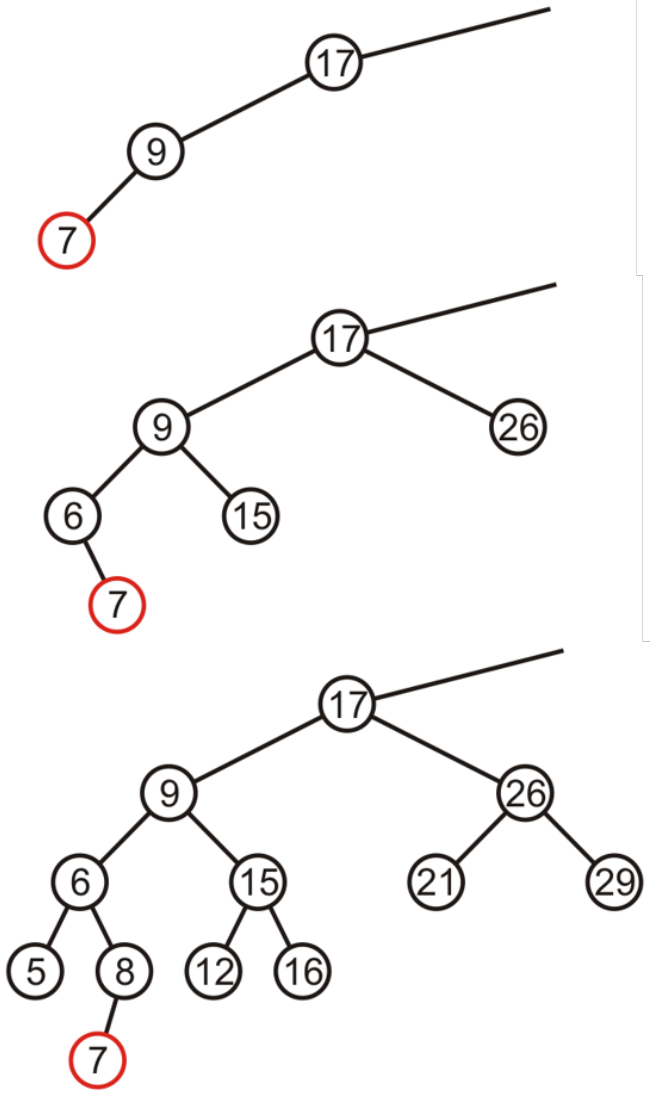
Cây có gốc tại f mất tính cân bằng. Cần phải duy trì sự cân bằng của cây có gốc tại f



Duy trì sự cân bằng của cây

Trường hợp 1: LL



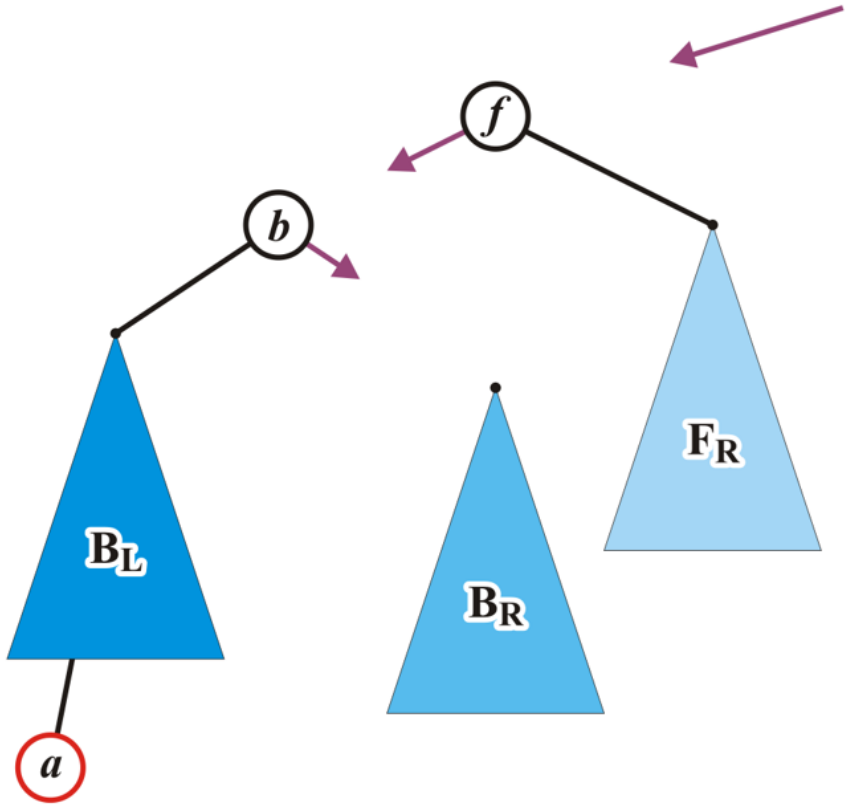


Duy trì sự cân bằng của cây

Trường hợp 1: LL

Cây có gốc tại f bị mất cân bằng

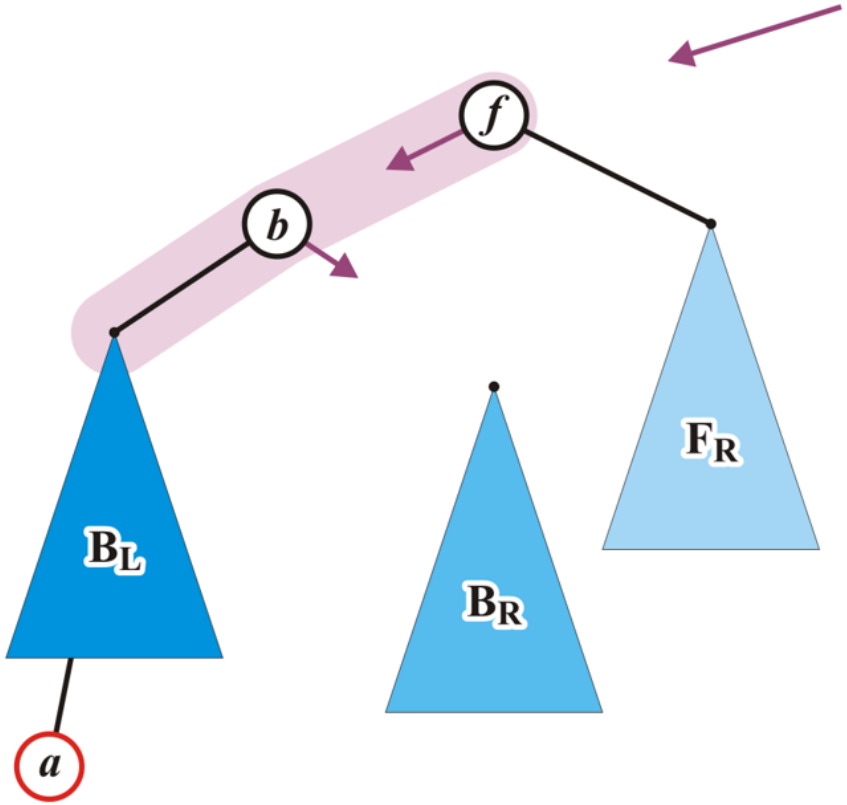
Cân bằng cây bằng cách thực hiện phép quay phải tại f



Duy trì sự cân bằng của cây

Trường hợp 1: LL

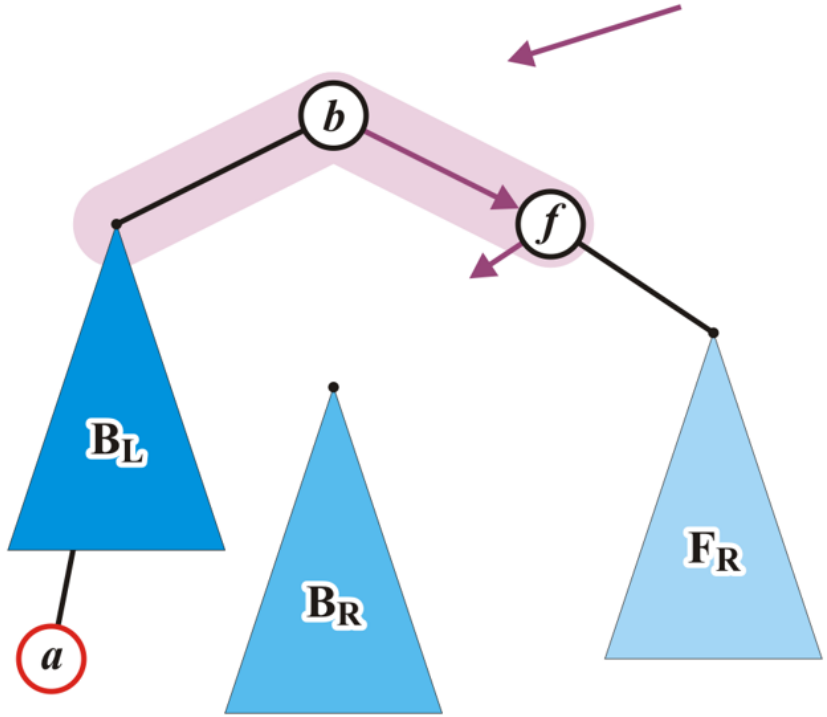
Lấy con phải của b



Duy trì sự cân bằng của cây

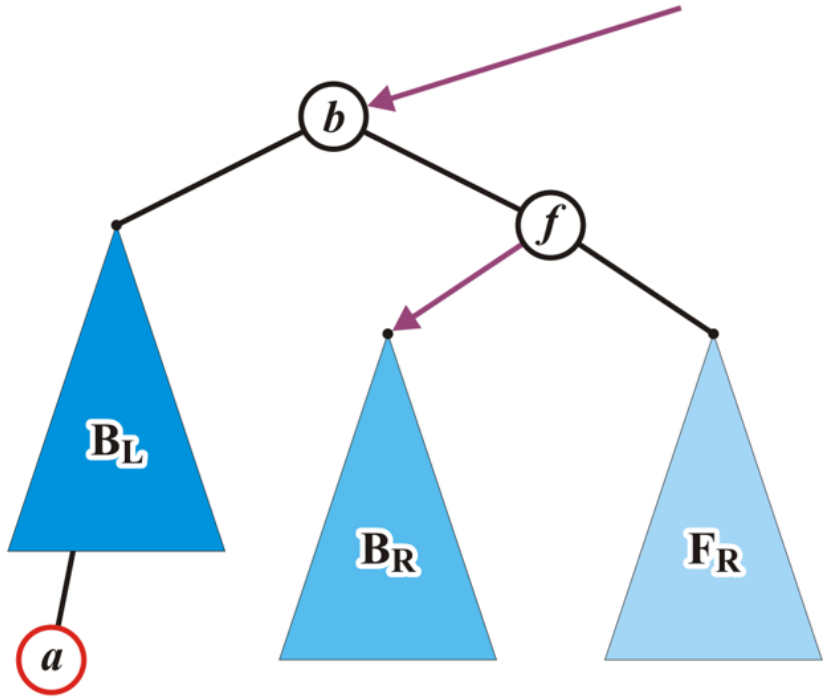
Trường hợp 1: LL

Để b lên làm gốc, f làm con phải của b



Duy trì sự cân bằng của cây

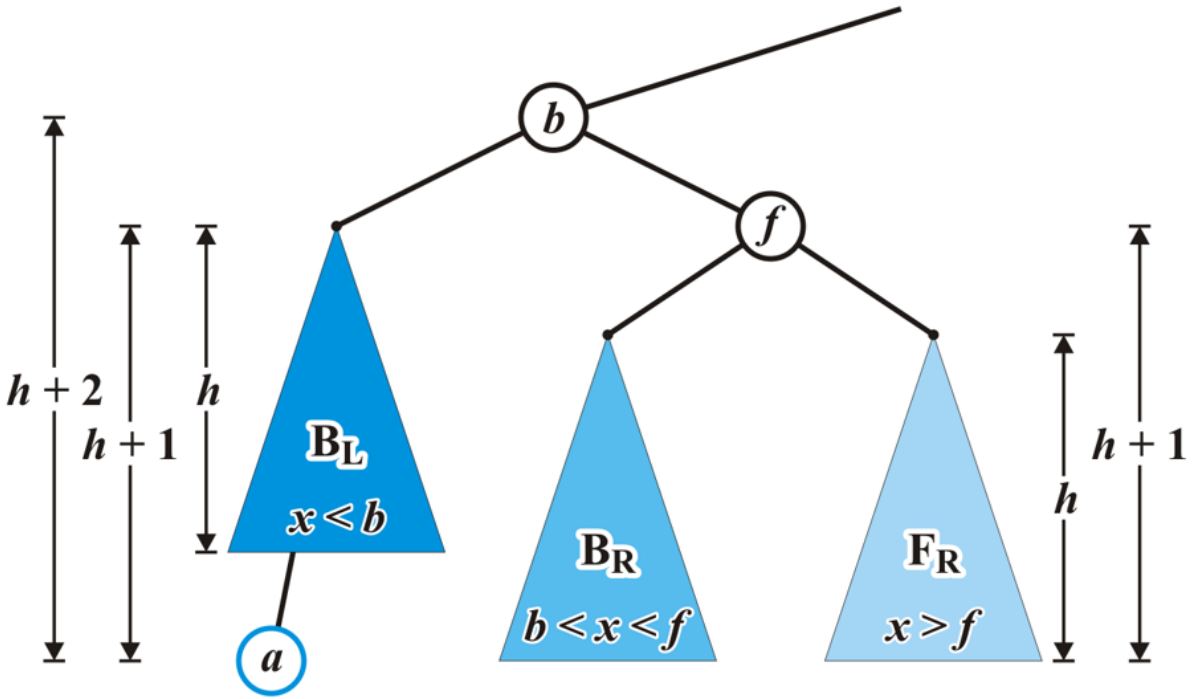
Trường hợp 1: LL



Duy trì sự cân bằng của cây

Trường hợp 1: LL

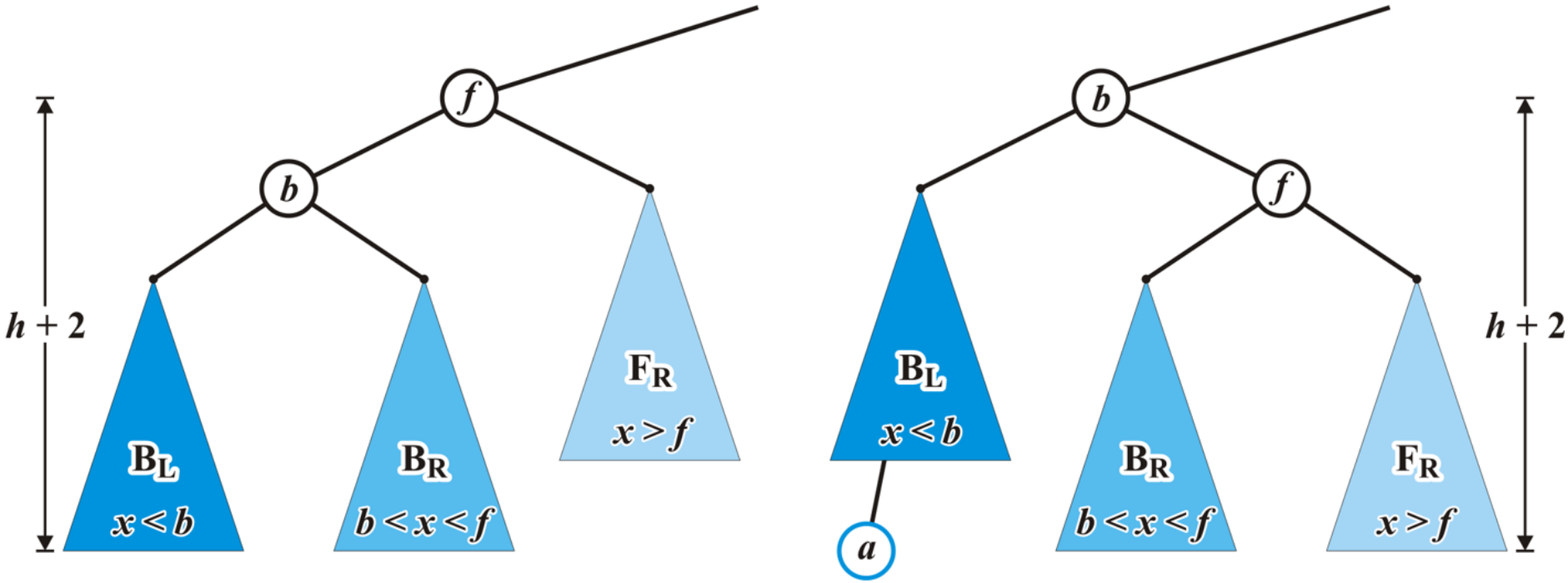
Lấy con phải cũ của b làm con trái của f



Duy trì sự cân bằng của cây

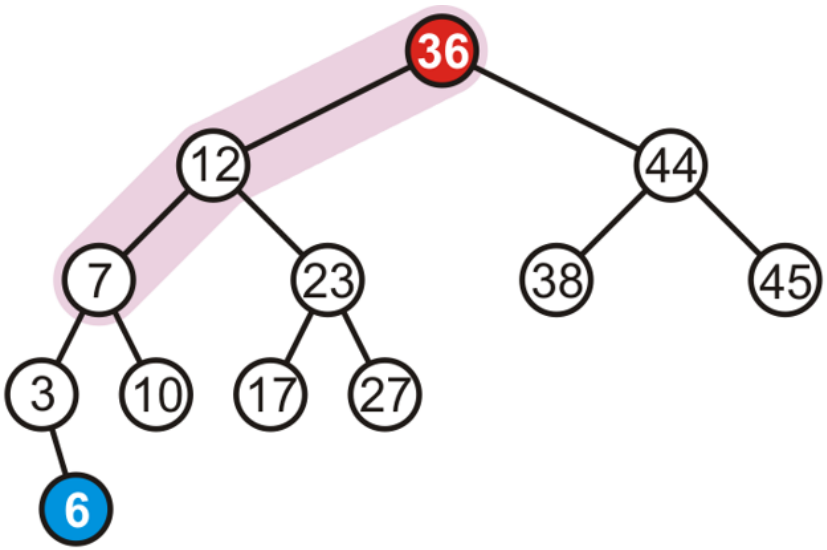
Trường hợp 1: LL

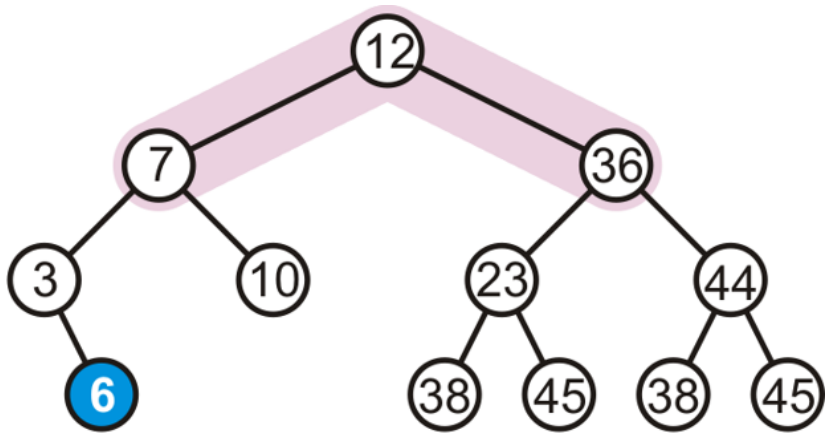
Chiều cao của cây có gốc tại b bằng chiều cao của cây con ban đầu có gốc tại f



Duy trì sự cân bằng của cây

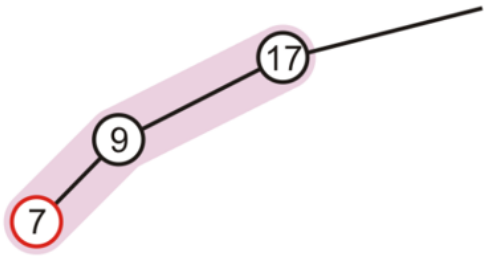
Ví dụ trường hợp 1

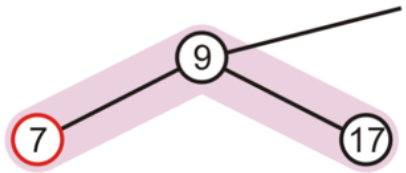


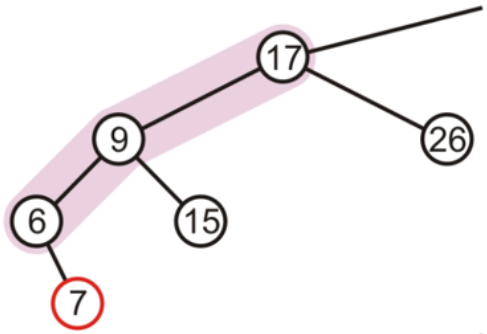


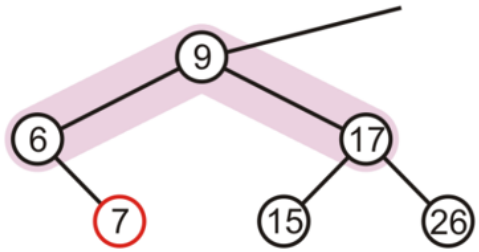
Duy trì sự cân bằng của cây

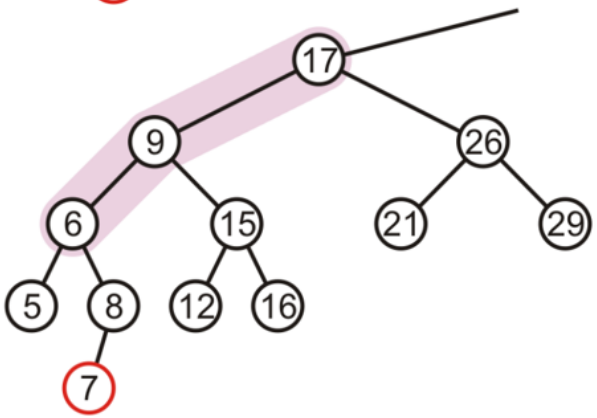
Ví dụ trường hợp 1

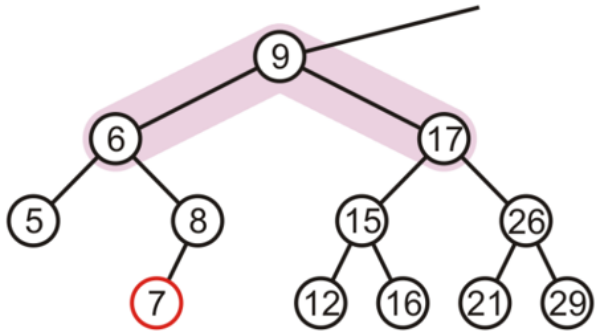








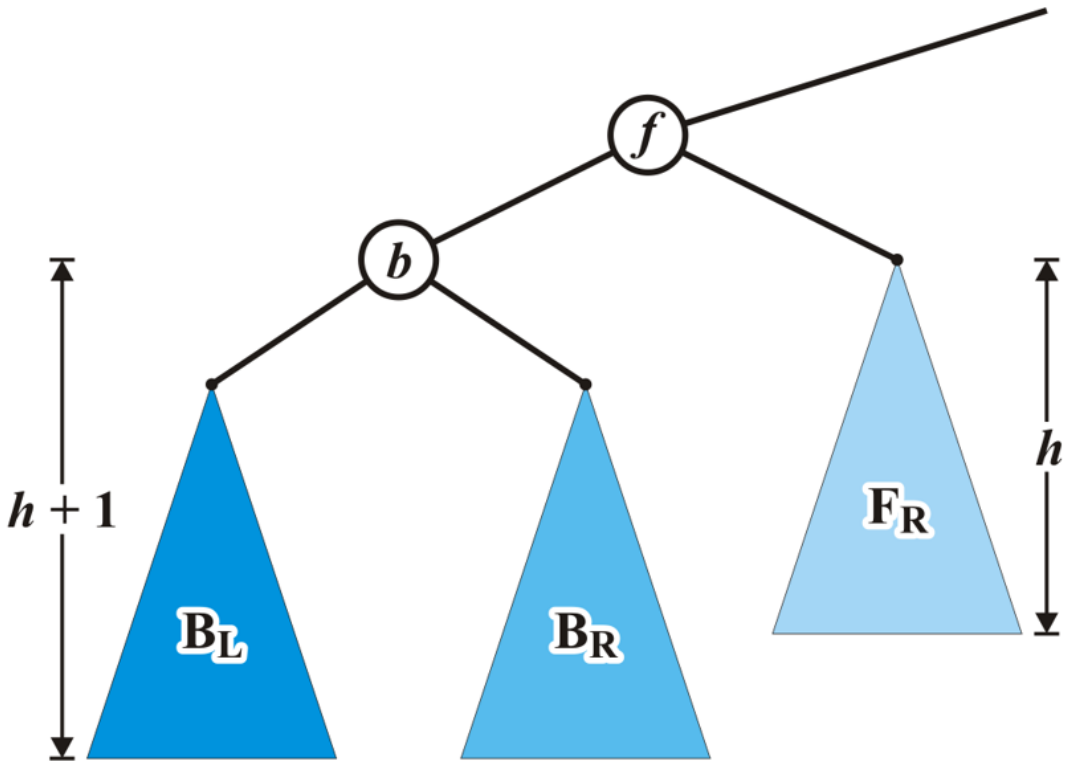




Duy trì sự cân bằng của cây

Trường hợp 2: LR

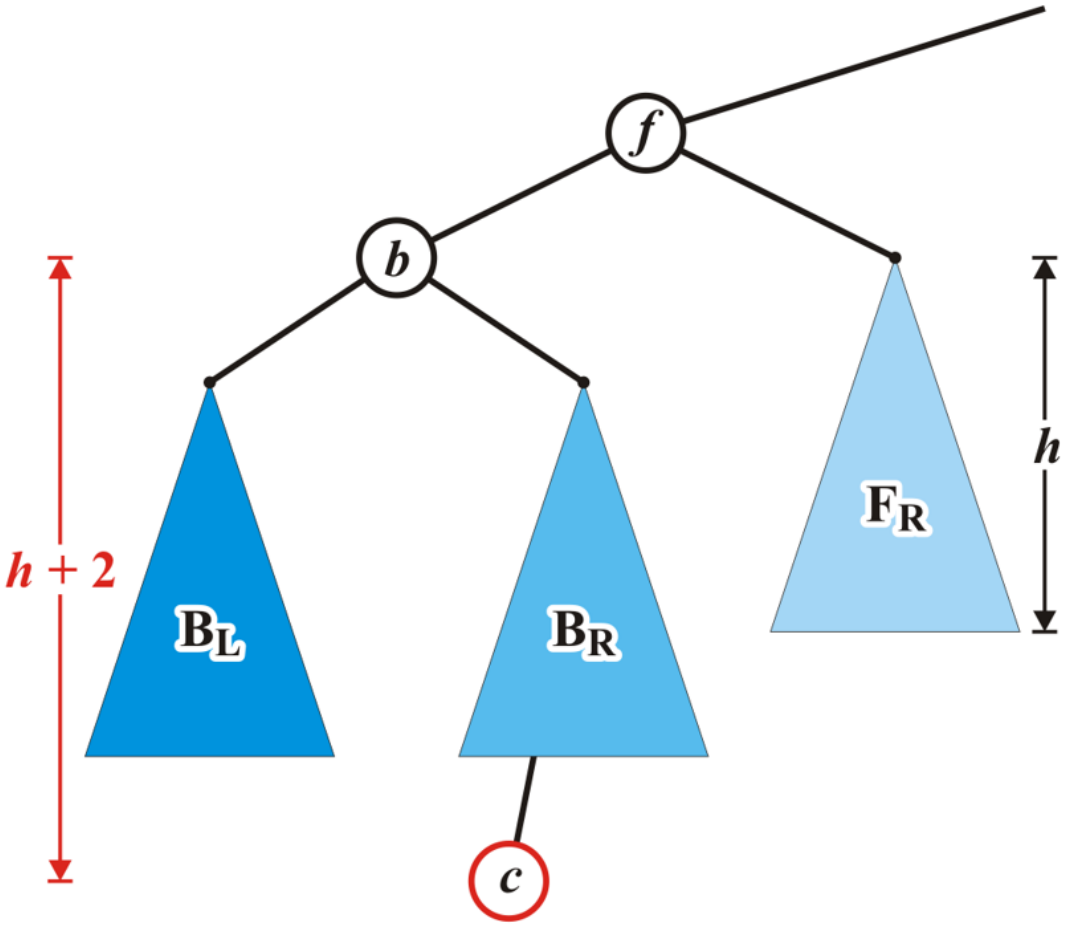
Cây ban đầu



Duy trì sự cân bằng của cây

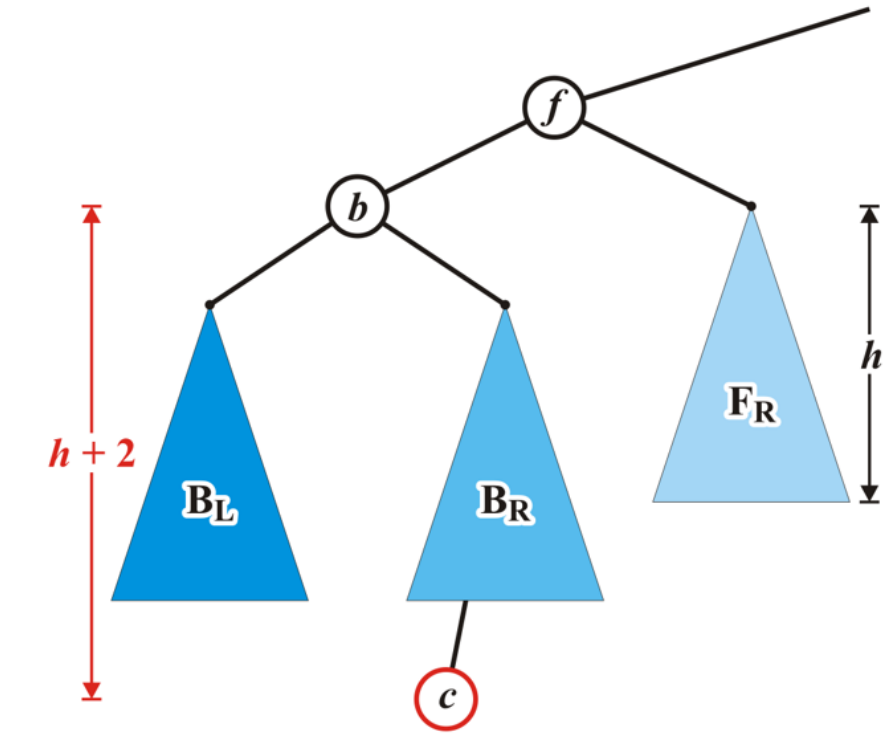
Trường hợp 2: LR

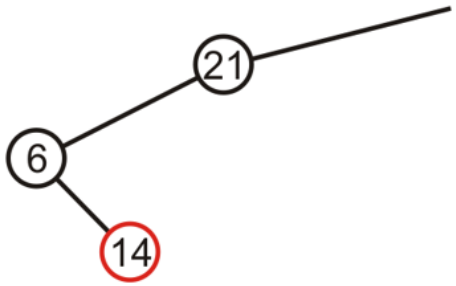
Xem xét trường hợp chèn c với $b < c < f$. Giả sử rằng nếu chèn c sẽ làm tăng chiều cao của cây con B_R. Lúc này f mất cân bằng

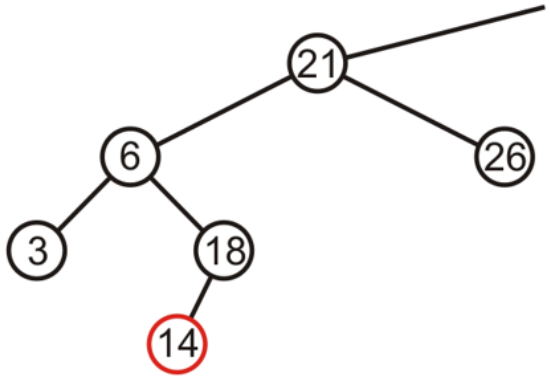


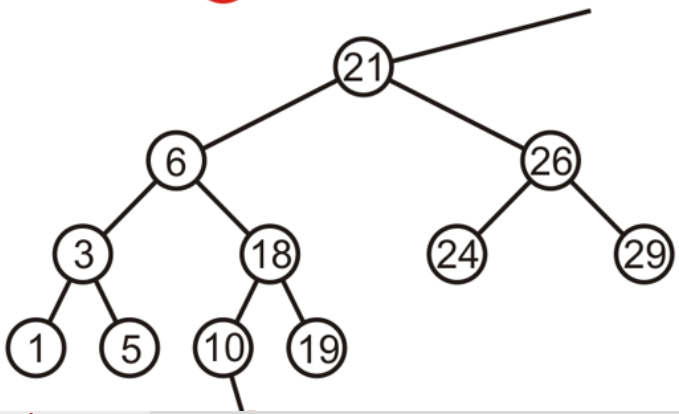
Duy trì sự cân bằng của cây

Trường hợp 2: LR



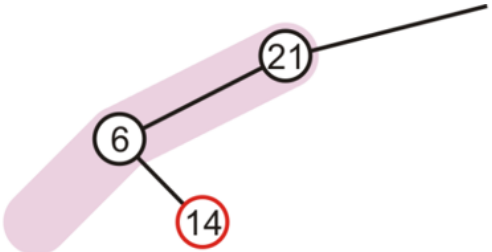


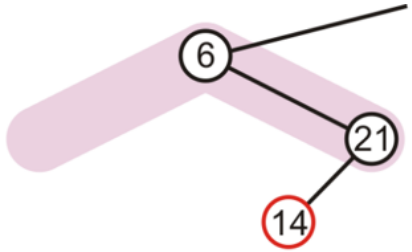


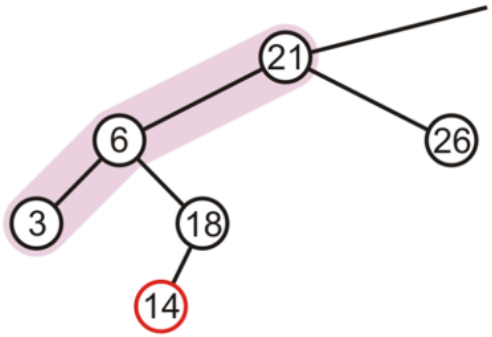


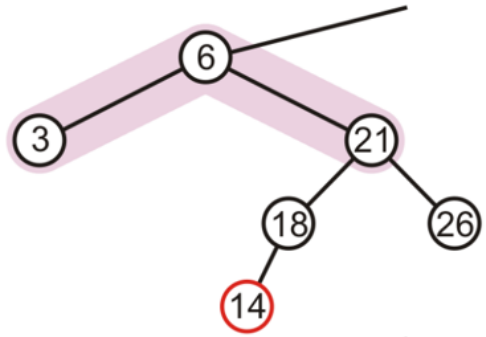
Duy trì sự cân bằng của cây

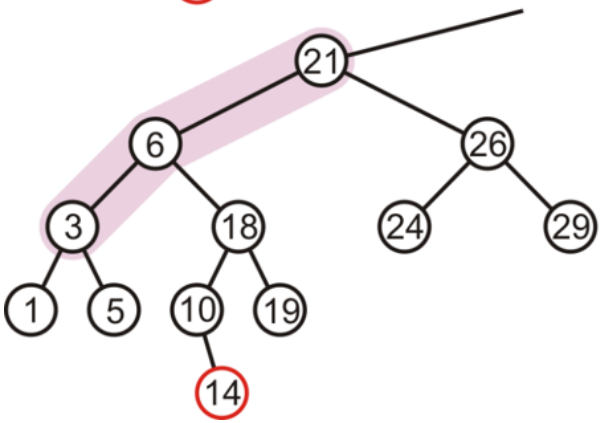
Trường hợp 2: LR

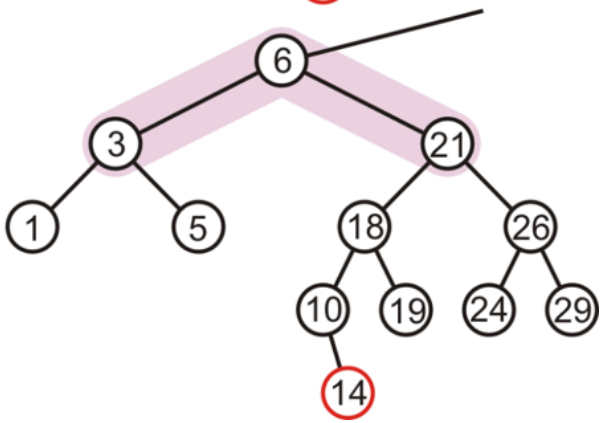






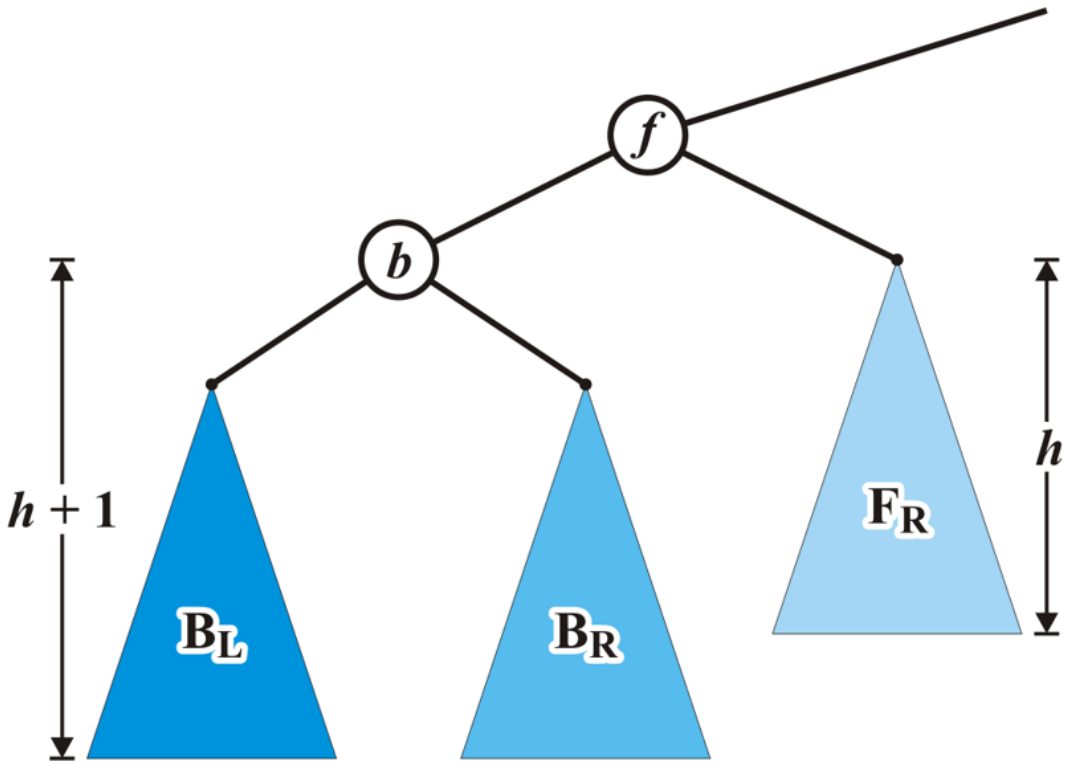






Duy trì sự cân bằng của cây

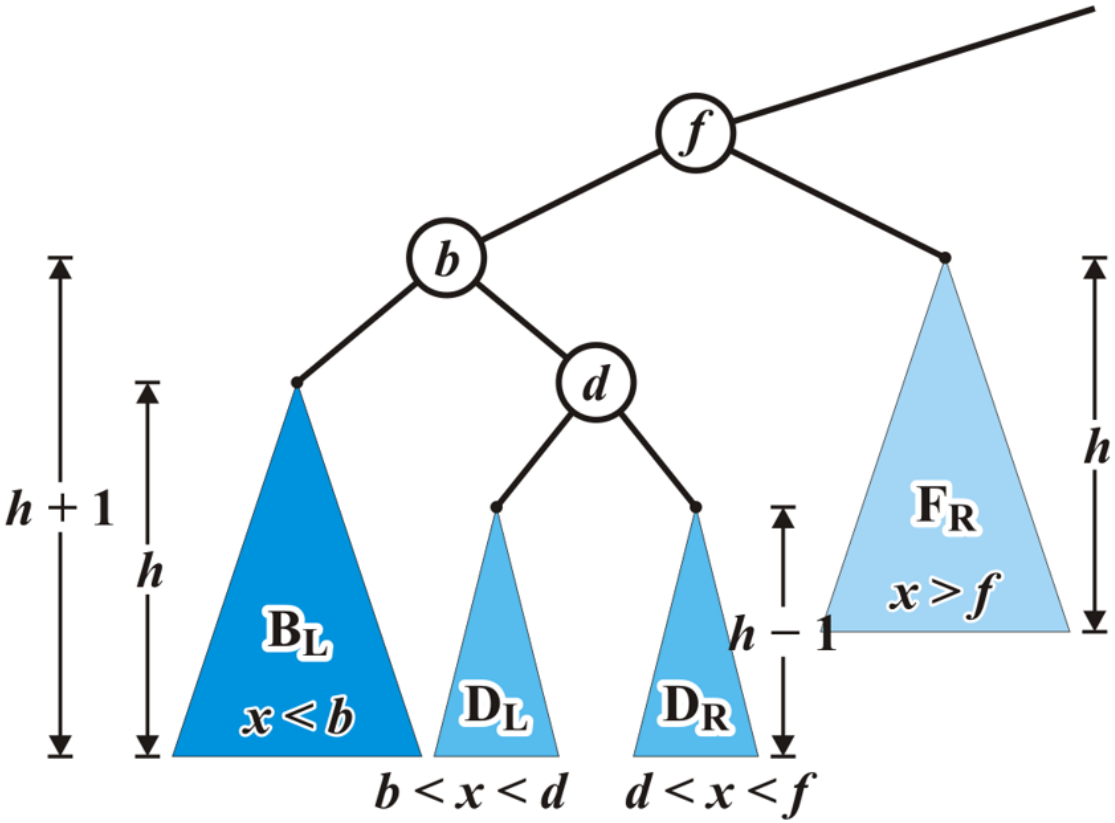
Trường hợp 2: LR



Duy trì sự cân bằng của cây

Trường hợp 2: LR

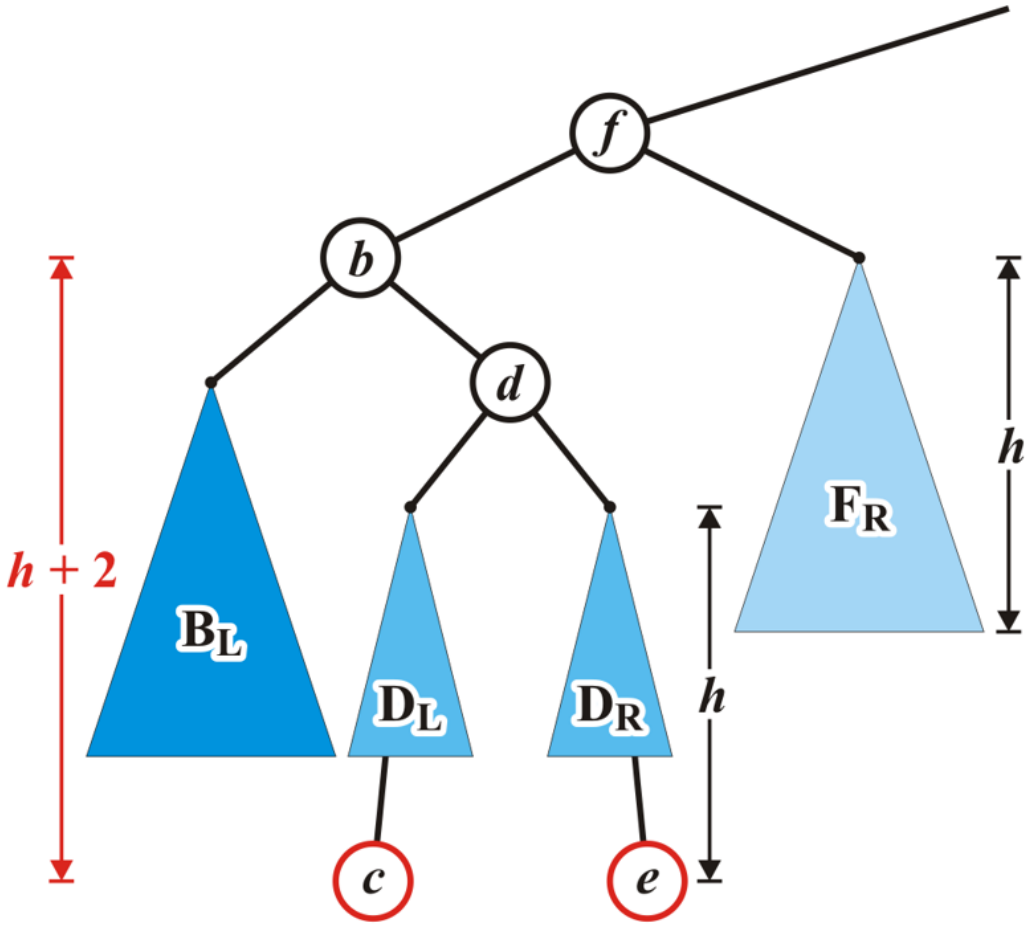
Gán lại nhãn cho cây bằng cách chia cây con phải của cây con trái của cây f thành cây có gốc tại d và với chiều cao $h - 1$



Duy trì sự cân bằng của cây

Trường hợp 2: LR

Chèn vào cây sẽ gây ra mất cân bằng tại f

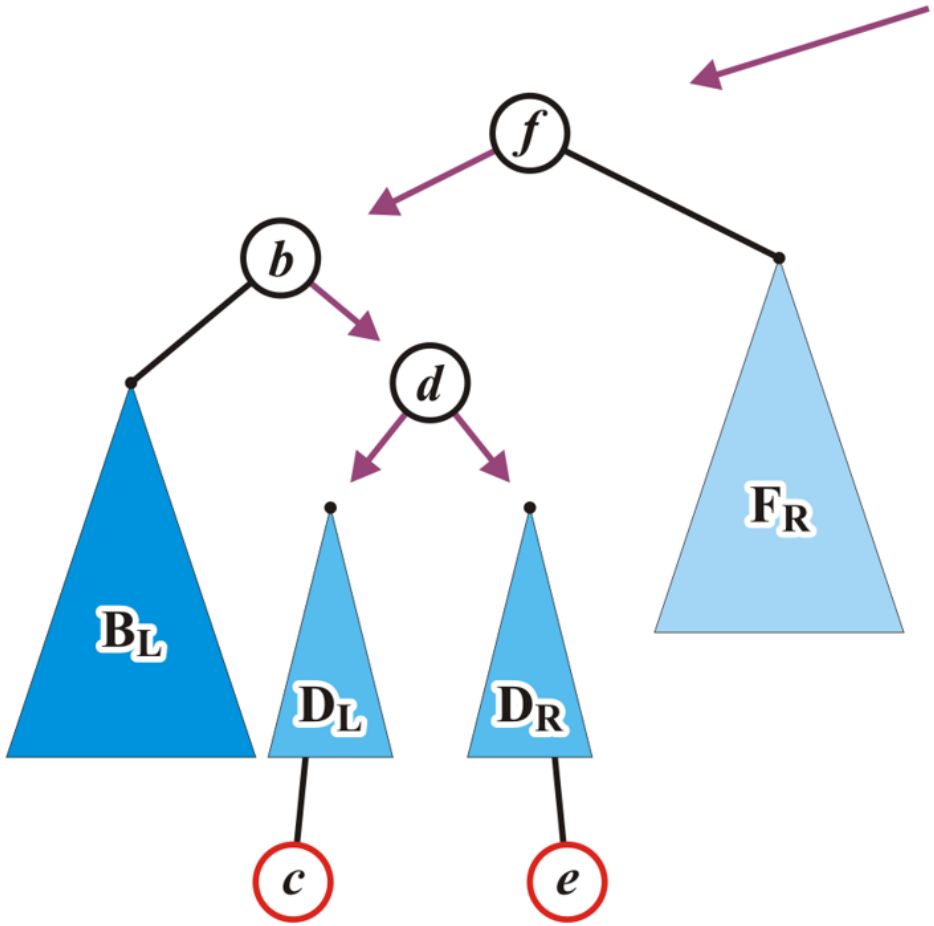


Duy trì sự cân bằng của cây

Trường hợp 2: LR

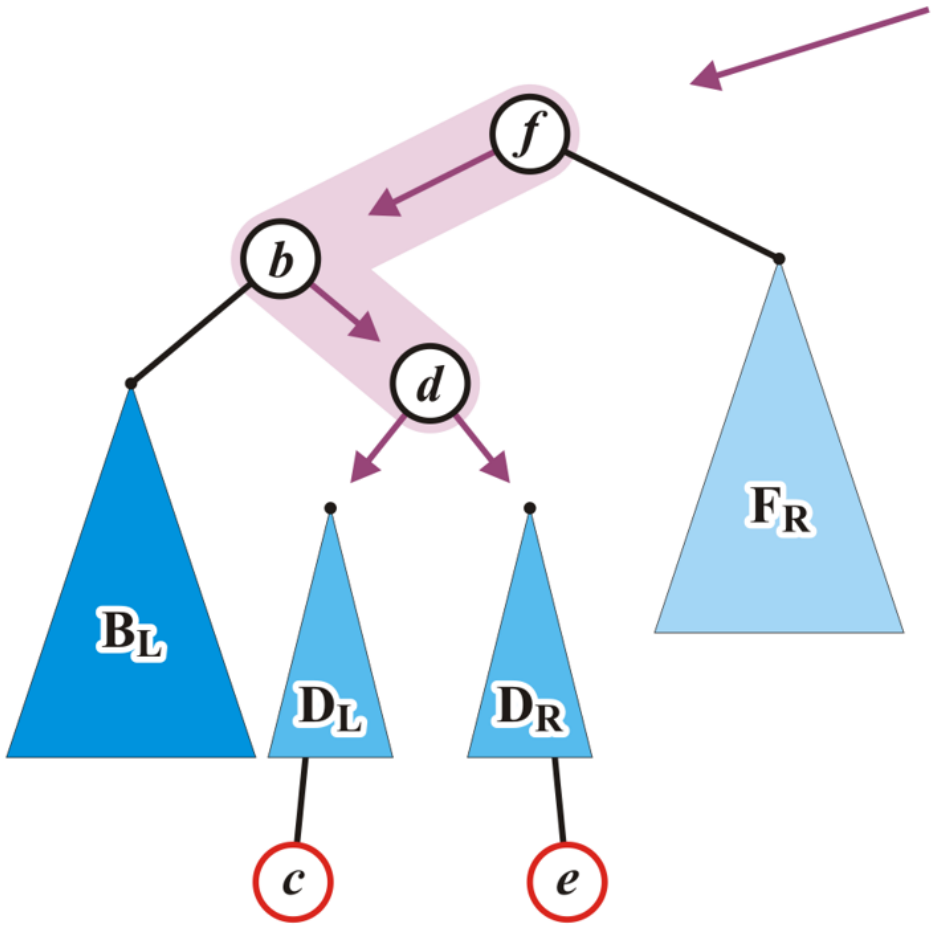
Cân bằng cây bằng cách thực hiện hai phép quay:

- Quay trái tại b
- Quay phải tại f



Duy trì sự cân bằng của cây

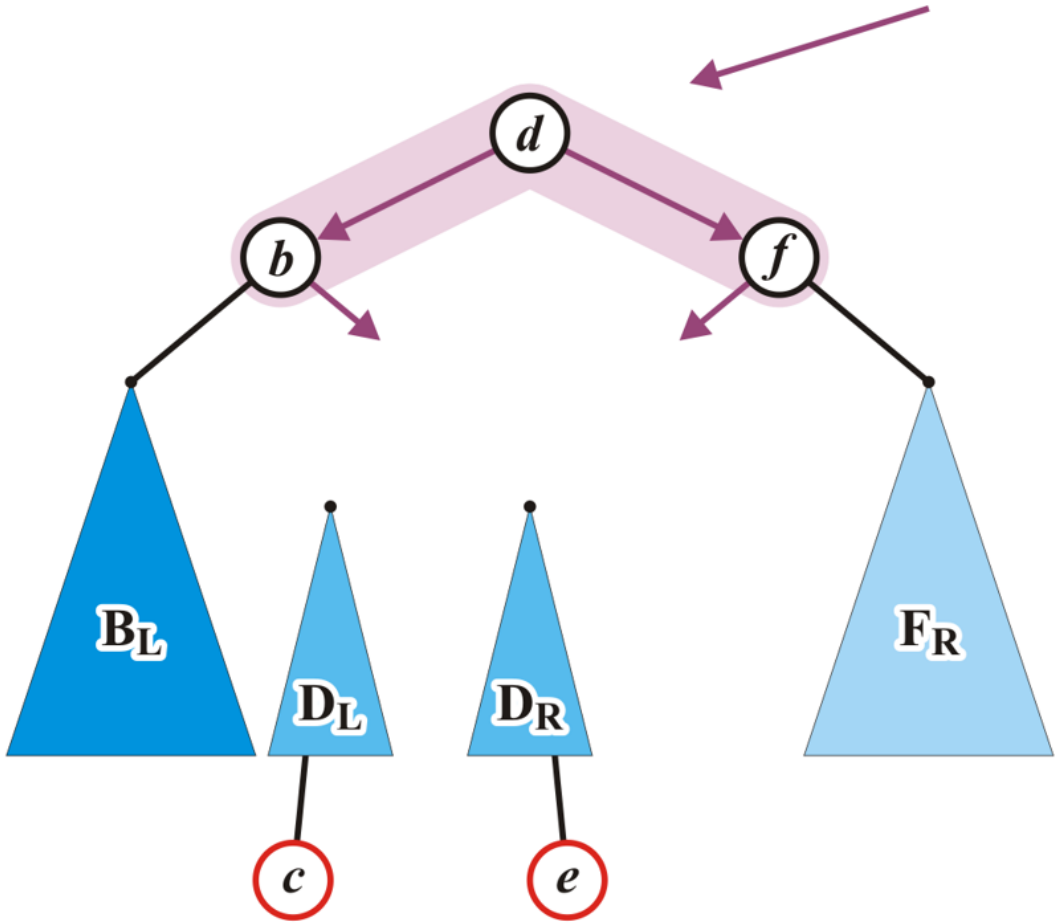
Trường hợp 2: LR



Duy trì sự cân bằng của cây

Trường hợp 2: LR

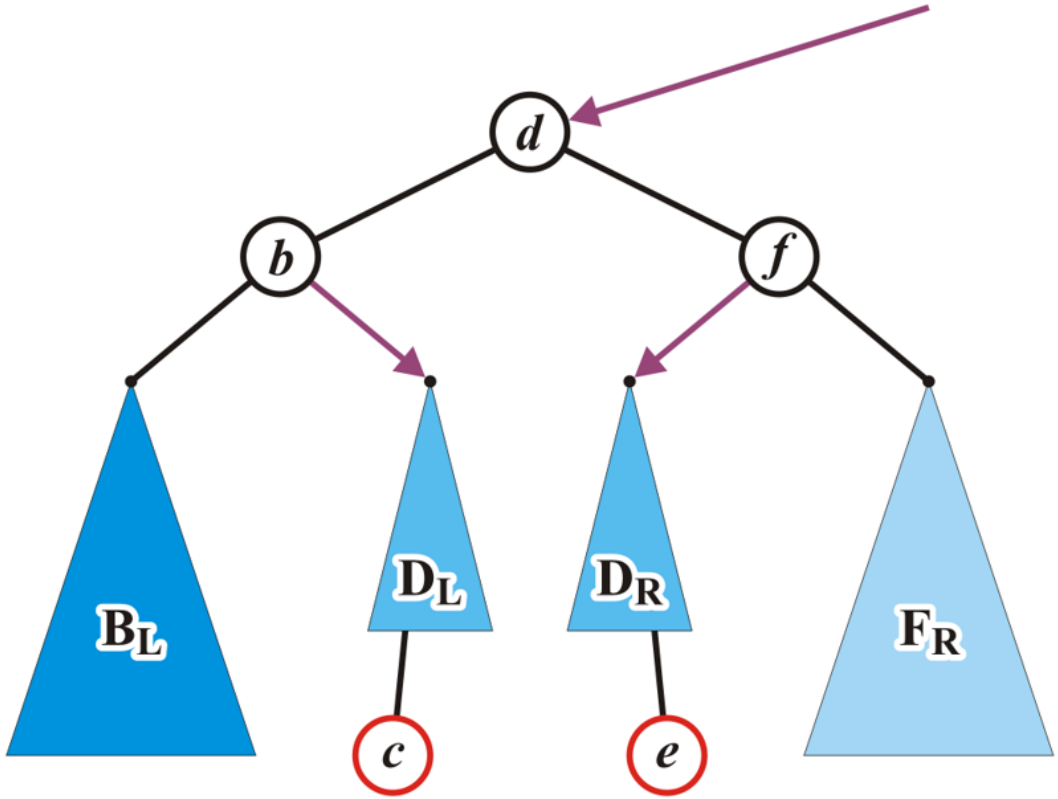
Để d làm gốc Lấy con trái và con phải của d



Duy trì sự cân bằng của cây

Trường hợp 2: LR

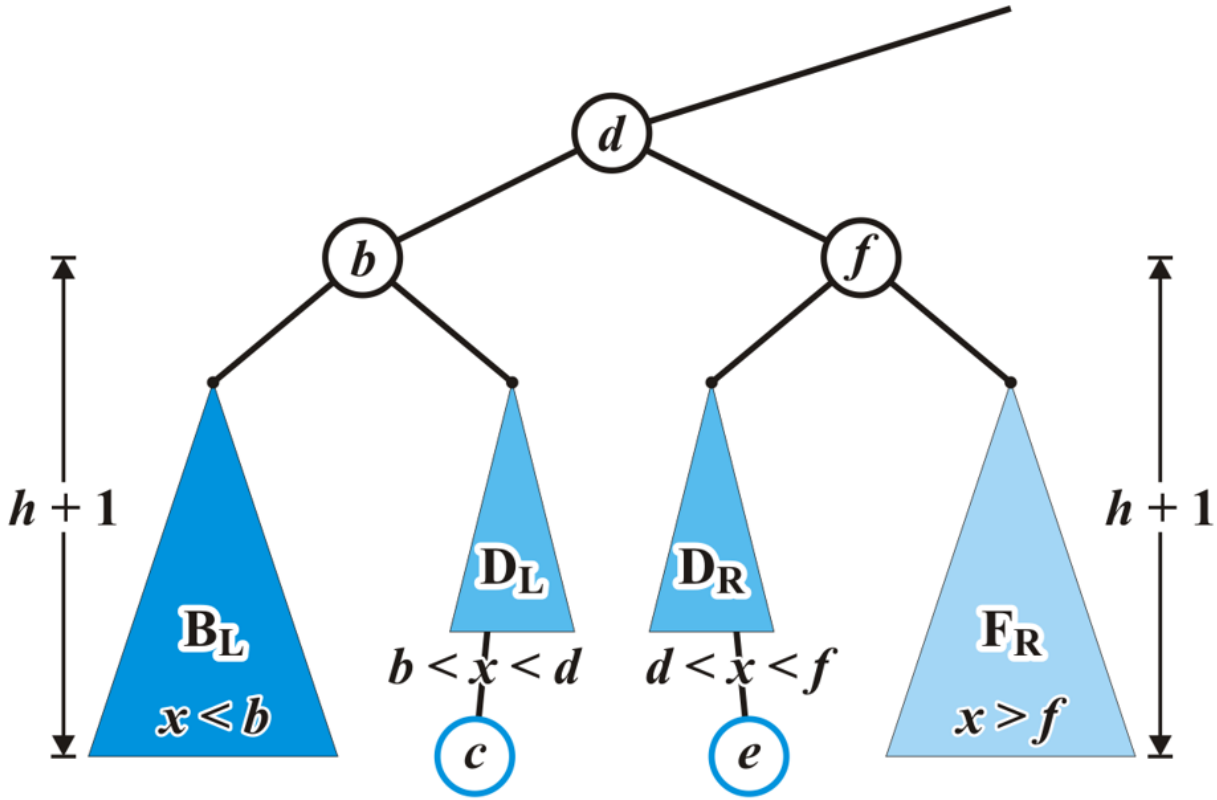
Để con trái của d làm con phải của b, để con phải của d làm con trái của f



Duy trì sự cân bằng của cây

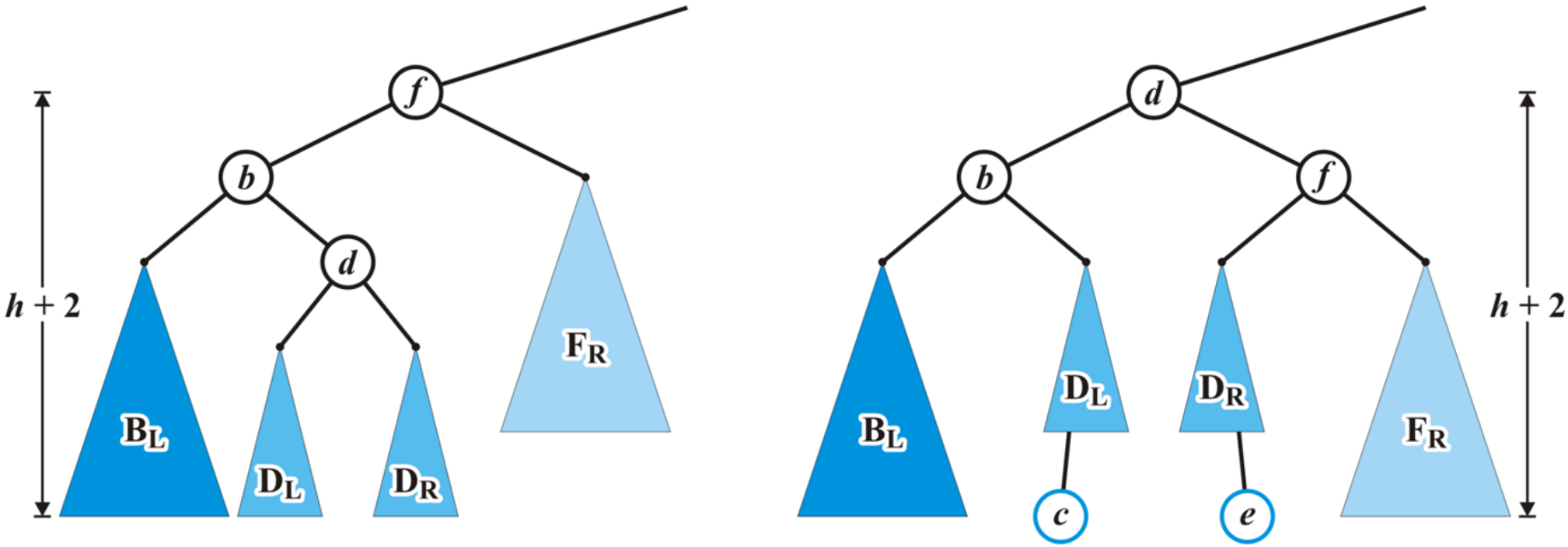
Trường hợp 2: LR

Cây có gốc tại d đã cân bằng



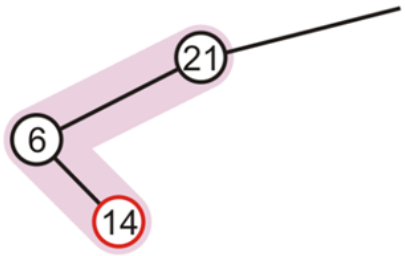
Duy trì sự cân bằng của cây

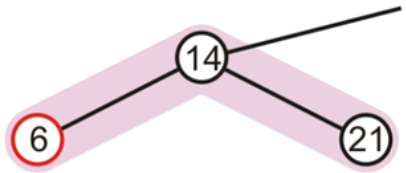
Trường hợp 2: LR

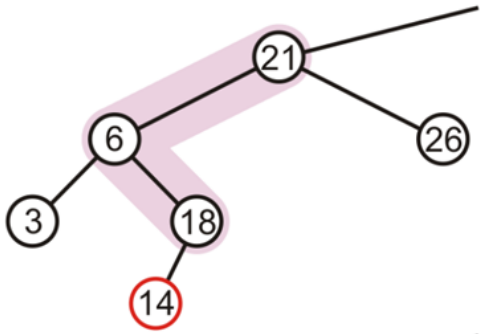


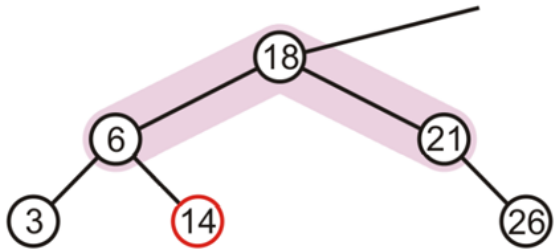
Duy trì sự cân bằng của cây

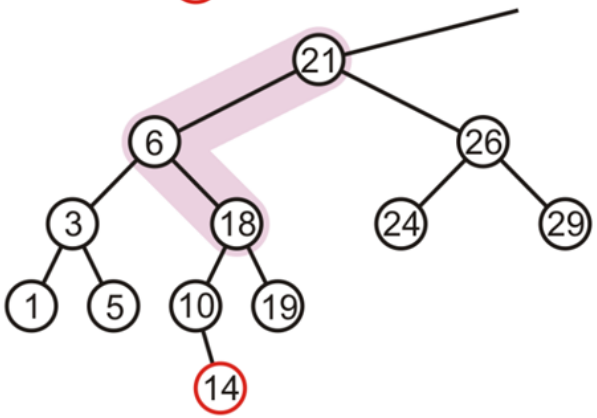
Ví dụ trường hợp 2

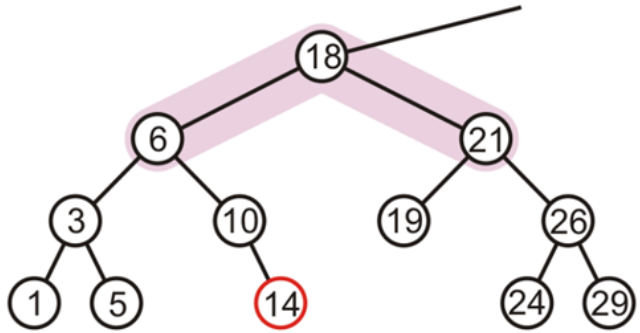






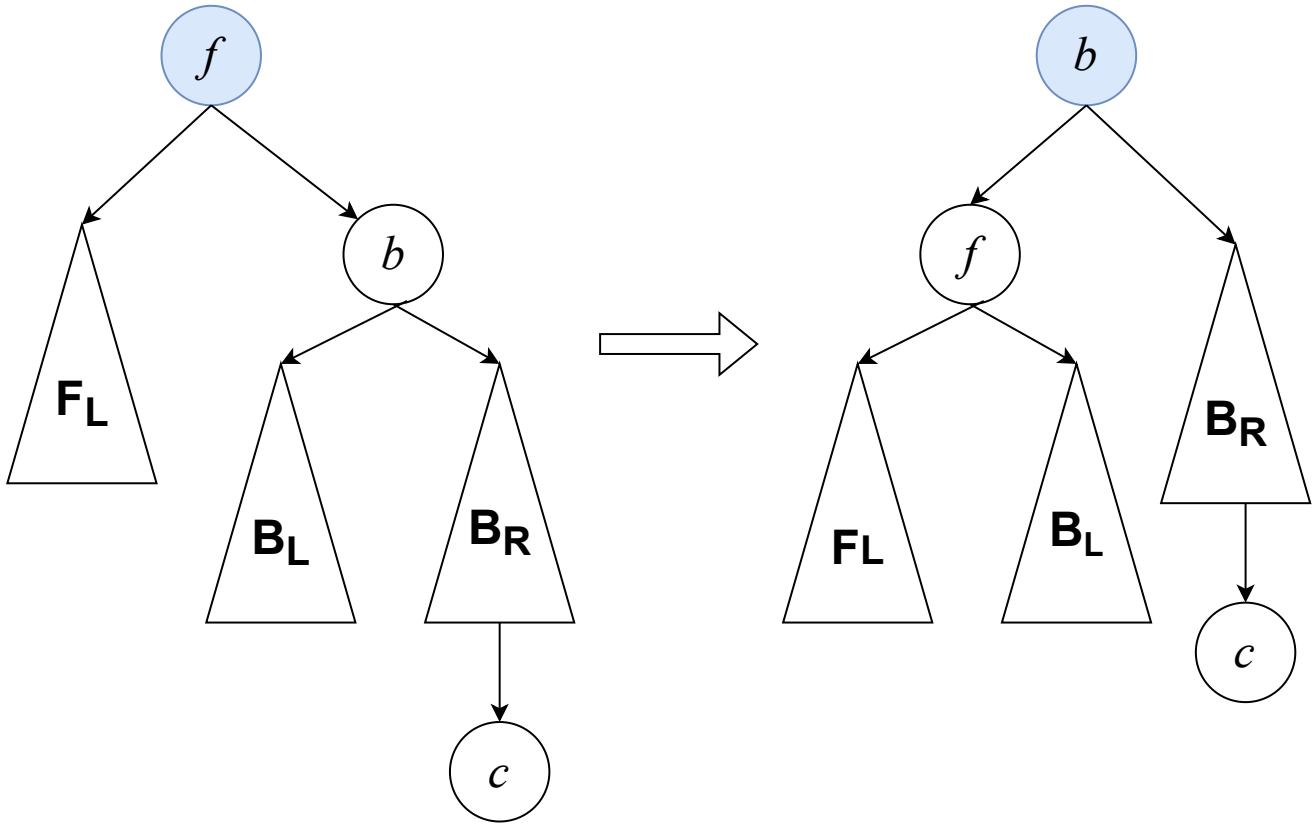






Duy trì sự cân bằng của cây

Trường hợp 4: RR

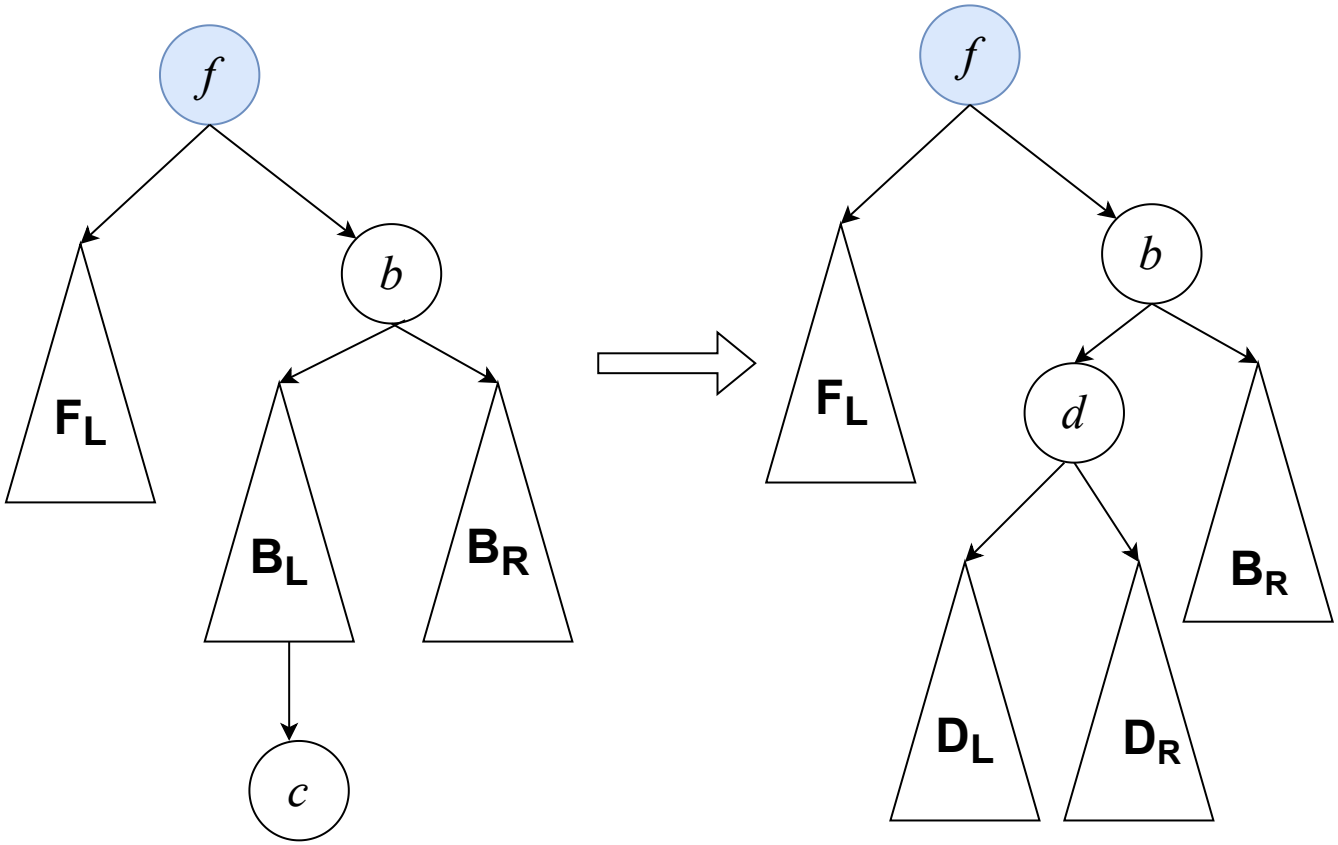


Hình 2: Quay trái quanh nút f

Duy trì sự cân bằng của cây

Trường hợp 4: RL

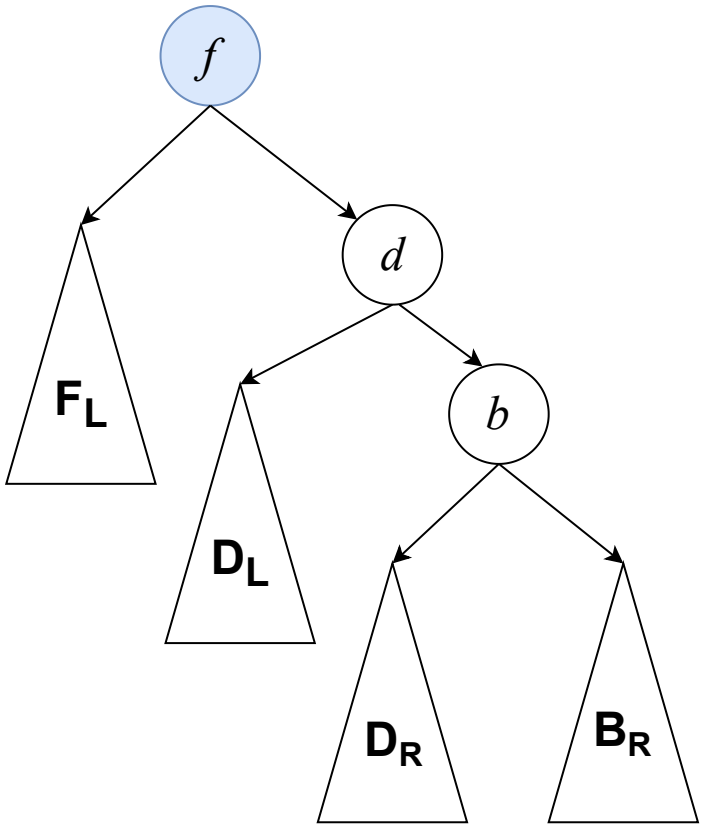
Cây ban đầu được vẽ lại như sau



Duy trì sự cân bằng của cây

Trường hợp 4: RL

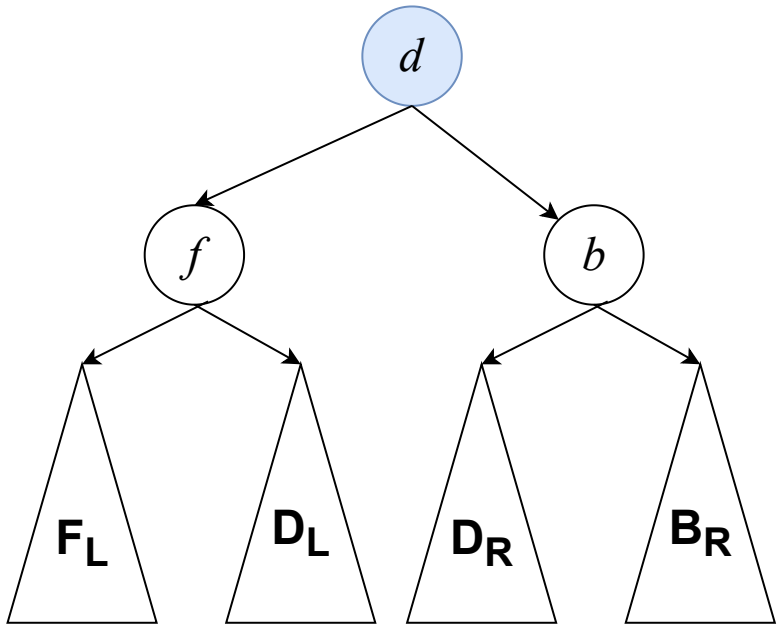
Quay phải quanh cây b



Duy trì sự cân bằng của cây

Trường hợp 4: RL

Quay trái quanh cây f



Ý tưởng thực hiện lập trình cài đặt cây AVL

- Dùng trường dữ liệu chiều cao (height) để xác định hệ số cân bằng cho nút cha
- Nút mới luôn có chiều cao là 1
- Khi chèn cũng như xoá một nút cần cập nhật giá trị chiều cao để phát hiện sự mất cân bằng

Các bước chèn

- Thực hiện chèn như cây BST bình thường
- Nút hiện tại sẽ là nút tổ tiên của nút mới k đang chèn, cập nhật lại chiều cao của nó
- Tính hệ số cân bằng mới của nút hiện tại (chiều cao cây con trái - chiều cao cây con phải)
- Nếu hệ số lớn hơn 1 sẽ có thể xảy ra hai tình huống
  - Tình huống 1
  - Tình huống 2
- Nếu hệ số nhỏ hơn -1 sẽ có thể xảy ra hai tình huống
  - Tình huống 3
  - Tình huống 4

Các bước chèn

```
public BSTNode insertToAVL(BSTNode root, int data) {
    /*1. chen*/
    if (root == null) {
        return new BSTNode(data);
    }
    if (data < root.data) {
        root.left = insertToAVL(root.left, data);
    } else if (data > root.data) {
        root.right = insertToAVL(root.right, data);
    } else {
        return root;
    }
    /* 2. cap nhat chieu cao */
    root.height = 1 + Math.max(height(root.left),
        height(root.right));
    /* 3. tinh toan he so can bang */
    int balance = calculateBalanceFactor(root);
```

Các bước chèn

```
        /* 4. kiem tra cac truong hop quay */
        if (balance > 1) {
            if (data > root.left.data) {
                root.left = rotateLeft(root.left);
            }
            return rotateRight(root);
        }
        if (balance < -1) {
            if (data < root.right.data) {
                root.right = rotateRight(root.right);
            }
            return rotateLeft(root);
        }

        return root;
    }
```