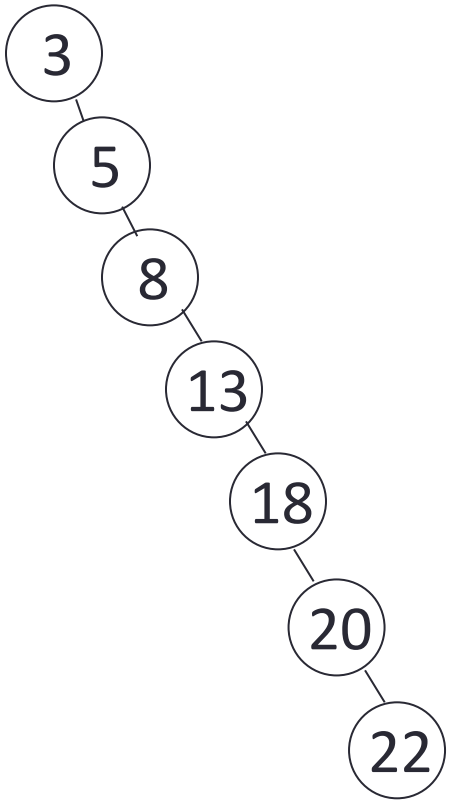
Cây AVL

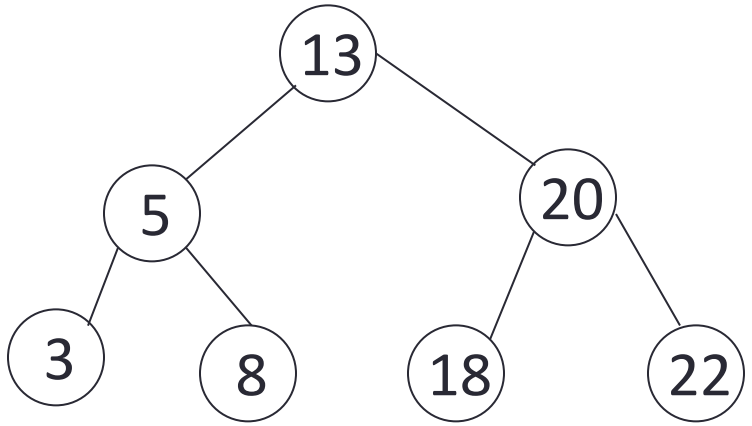
Bài giảng môn Cấu trúc dữ liệu và giải thuật

Khoa Công nghệ thông tin
Trường Đại học Thủy Lợi

Mở đầu

- Khi xây dựng cây nhị phân tìm kiếm, ta muốn có kiểu cây nào hơn?
- Ví dụ: dựng cây từ dãy {3, 5, 8, 20, 18, 13, 22}



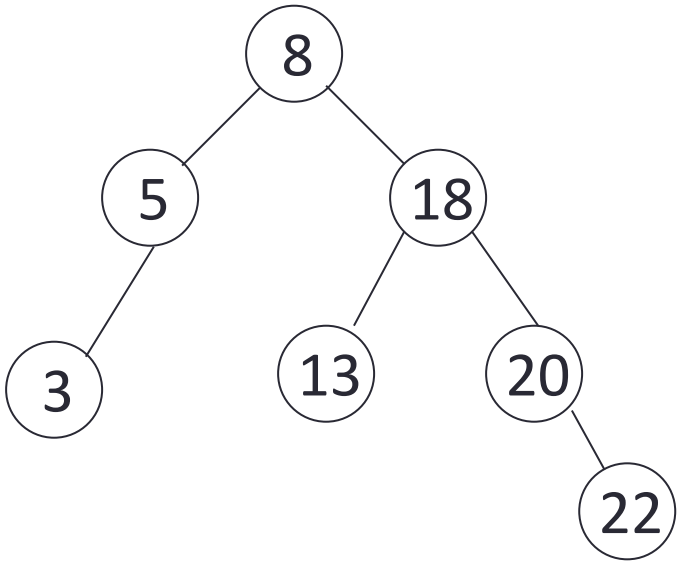


Mở đầu (tiếp)

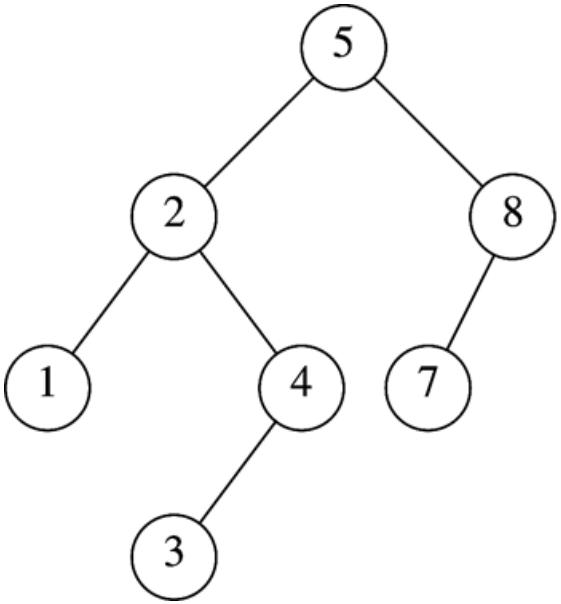
- Ta muốn một cây nhị phân tìm kiếm cân đối:
  - có độ sâu của cây = log n, và do đó
  - cho phép chèn và xóa với thời gian chạy O(log n) trong mọi trường hợp.
- Cây AVL là một kiểu cây như vậy!

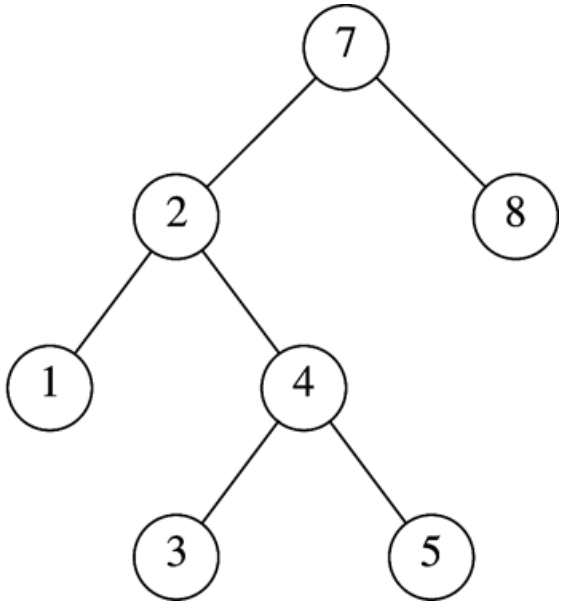
Cây AVL (Adelson-Velskii & Landis)

- Cây AVL là một cây nhị phân tìm kiếm thỏa mãn **điều kiện cân bằng**:
  - Với mọi nút X, chiều cao của hai cây con trái và phải của X sai khác không quá 1.
  - Quy ước cây rỗng có chiều cao -1.



Cây nào là cây AVL ?

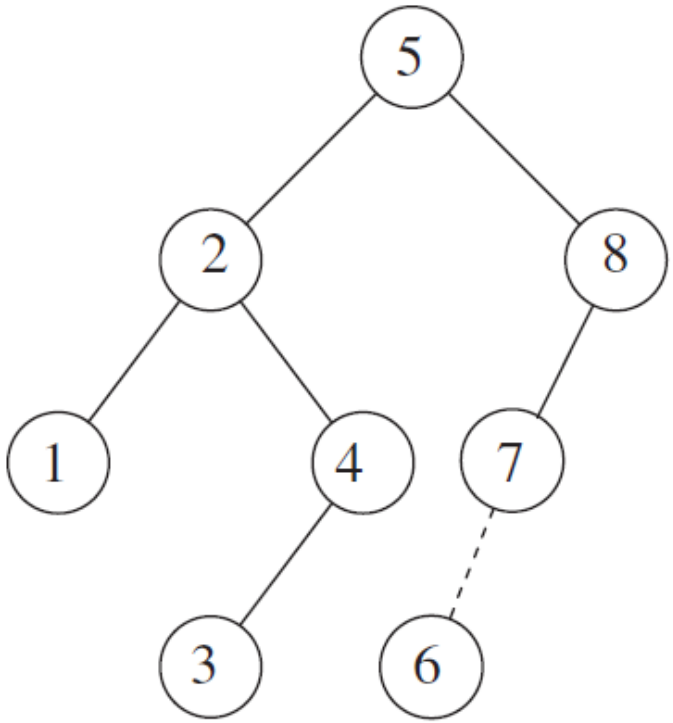




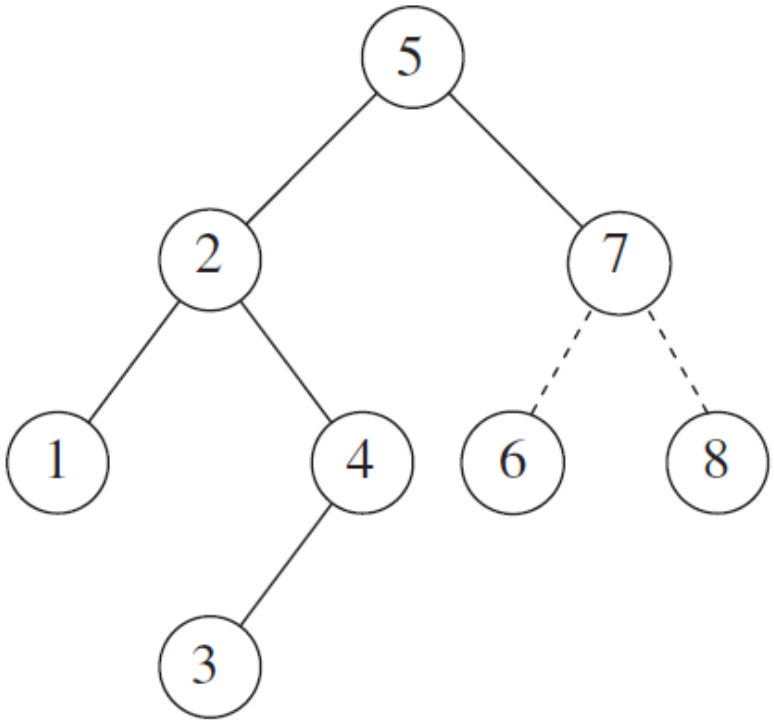
Chèn và xóa trên cây AVL

- Thực hiện chèn/xóa như trong cây nhị phân tìm kiếm thông thường.
- Sau khi chèn/xóa, điều kiện cân bằng có thể bị vi phạm:
  - Sửa bằng **phép xoay**.
  - Sau phép xoay, cây trở lại cân bằng.

Ví dụ phép chèn



Chèn 6 làm điều kiện cân bằng bị vi phạm tại nút 8.



Sửa bằng phép xoay quanh nút 8.

Vi phạm điều kiện cân bằng

- Nếu điều kiện cân bằng bị vi phạm, những nút nào cần được xoay?
  - Chỉ những nút trên đường đi từ điểm chèn ngược về gốc có thể bị ảnh hưởng.
- Chỉ cần tái cân bằng dùng phép xoay tại nút sâu nhất có điều kiện cân bằng bị vi phạm.
  - Toàn bộ cây sẽ được tái cân bằng.

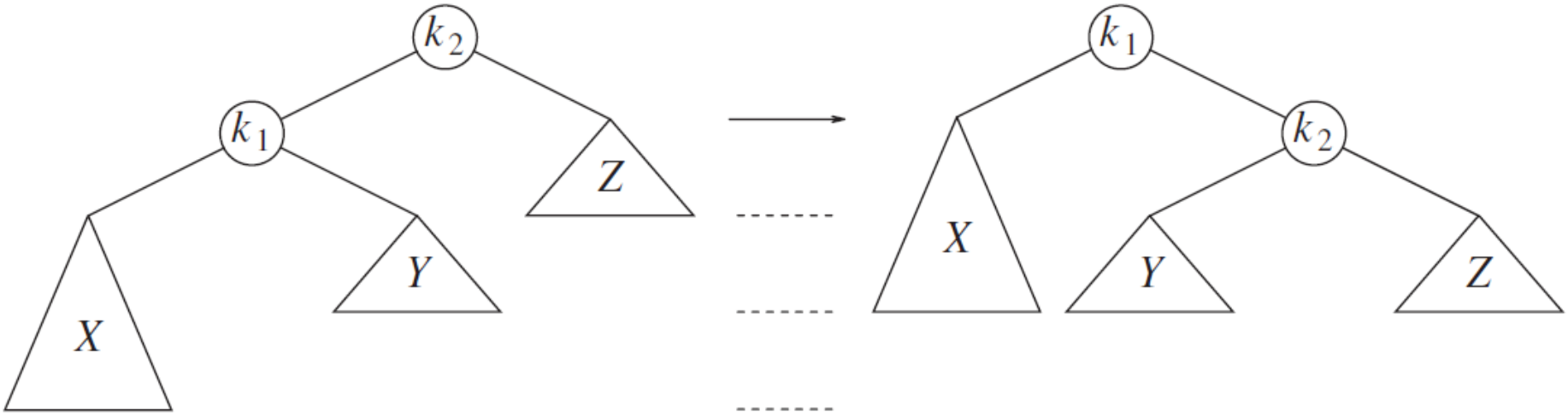
Các trường hợp vi phạm

- Giả sử nút k là nơi xảy ra vi phạm. Có 4 trường hợp:
  1. trái-trái: chèn vào cây con trái của con trái của k
  2. trái-phải: chèn vào cây con phải của con trái của k
  3. phải-trái: chèn vào cây con trái của con phải của k
  4. phải-phải: chèn vào cây con phải của con phải của k
- Hai trường hợp 1 và 4 (chèn ngoài) tương tự nhau:
  - **Phép xoay đơn** để tái cân bằng.
- Hai trường hợp 2 và 3 (chèn trong) tương tự nhau:
  - **Phép xoay kép** để tái cân bằng.

Kiểu dữ liệu của các nút

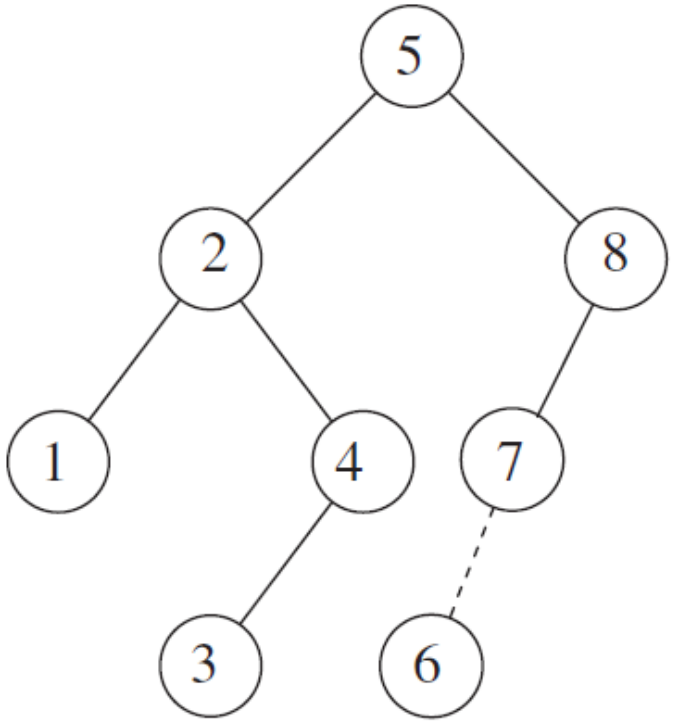
```
// Kiểu phần tử
typedef int T;

// Kiểu của các nút trên cây AVL
struct AvlNode {
    T elem;           // Phần tử
    AvlNode * left;   // Liên kết tới con trái
    AvlNode * right; // Liên kết tới con phải
    int height;       // Chiều cao của nút
};

// Hàm trả về chiều cao của nút
int height(AvlNode * t) {
    return t == NULL ? -1 : t->height;
}
```

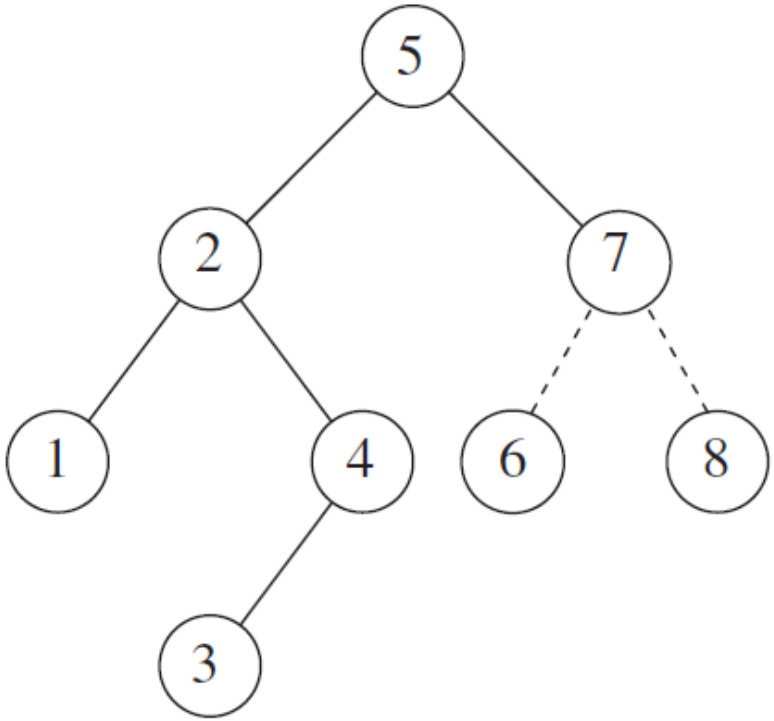
Phép xoay đơn (trường hợp 1)



- Thay nút k_2 bằng nút k_1.
- Cho nút k_2 trở thành con phải của nút k_1.
- Cho cây con Y trở thành cây con trái của nút k_2.
- Với trường hợp 4, cách làm tương tự (xoay theo chiều ngược lại).

Ví dụ phép xoay đơn

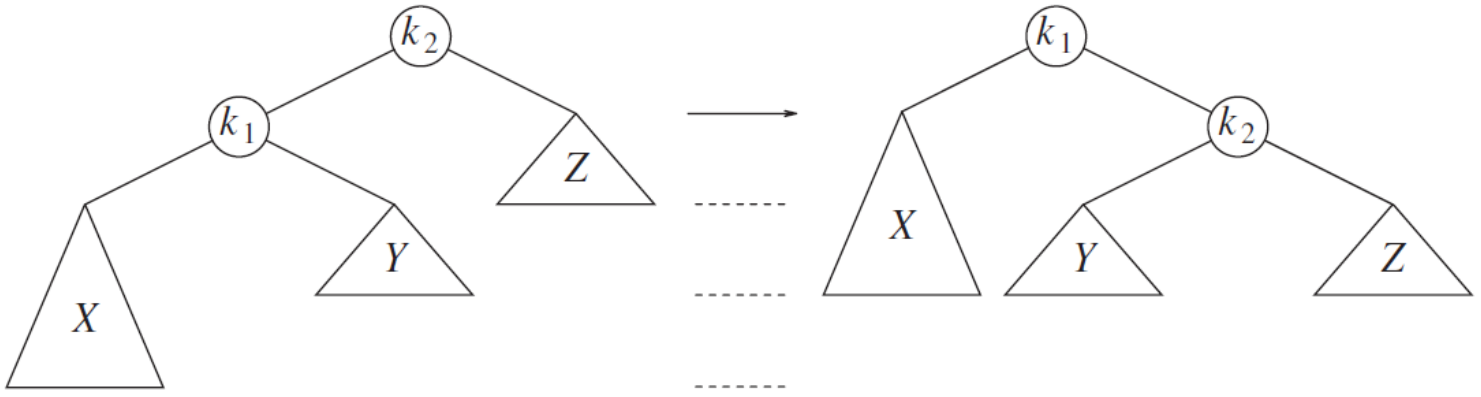




Sau khi chèn 6, điều kiện cân bằng ở nút 8 bị vi phạm.

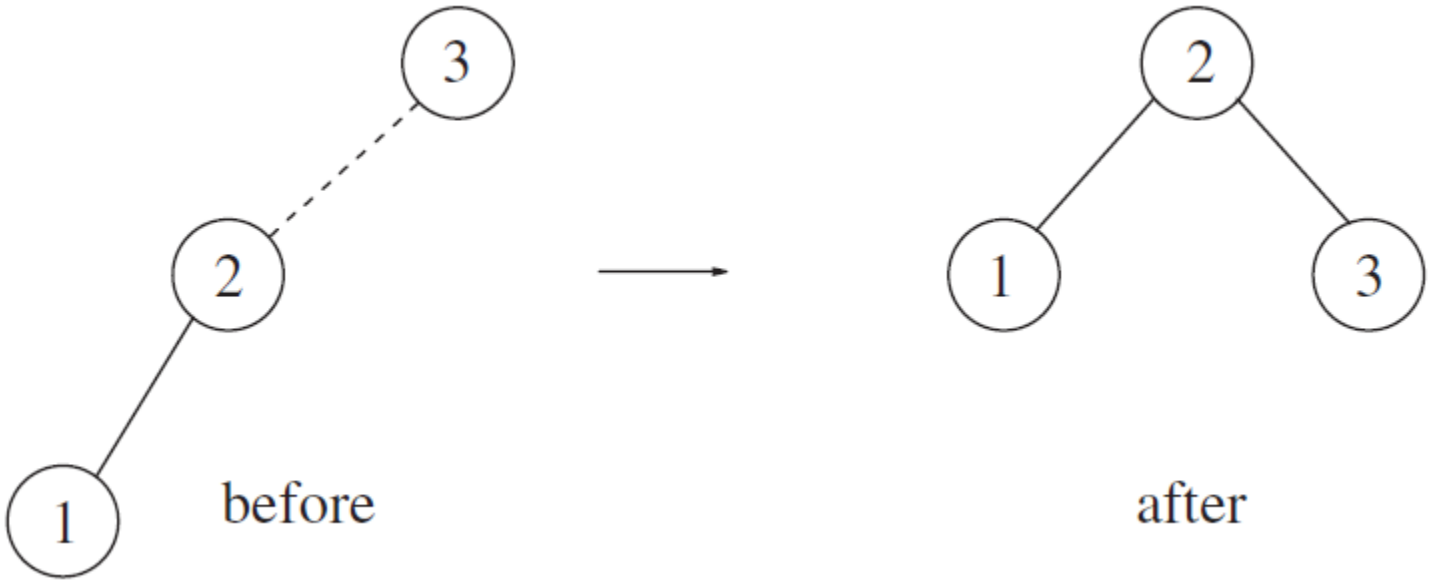
Phép xoay đơn (trường hợp 1)

```
void rotateWithLeftChild(AvlNode * & k2)
{
    AvlNode * k1 = k2->left;
    k2->left = k1->right;
    k1->right = k2;
    k2->height = max(height(k2->left), height(k2->right))
                 + 1; // Hàm max trả về giá trị lớn hơn
    k1->height = max(height(k1->left), k2->height) + 1;
    k2 = k1;
}
```



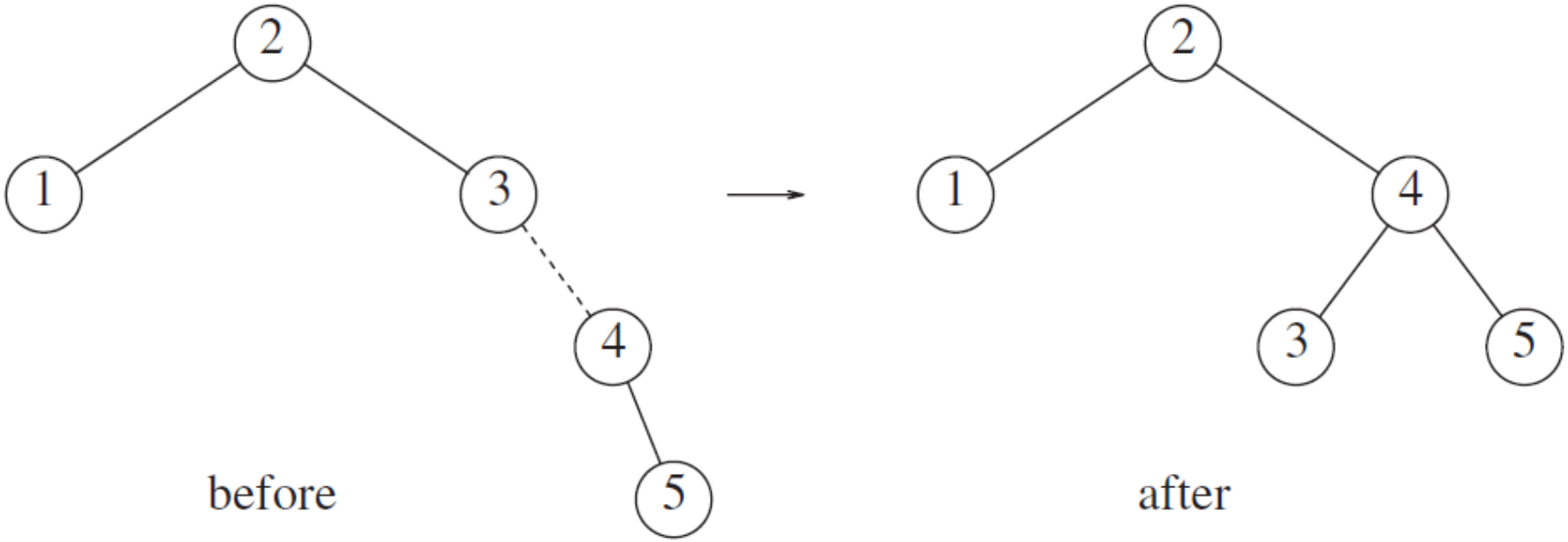
Ví dụ phép xoay đơn (1)

- Chèn tuần tự 3, 2, 1 và sau đó 4, 5, 6, 7 vào một cây AVL đang rỗng



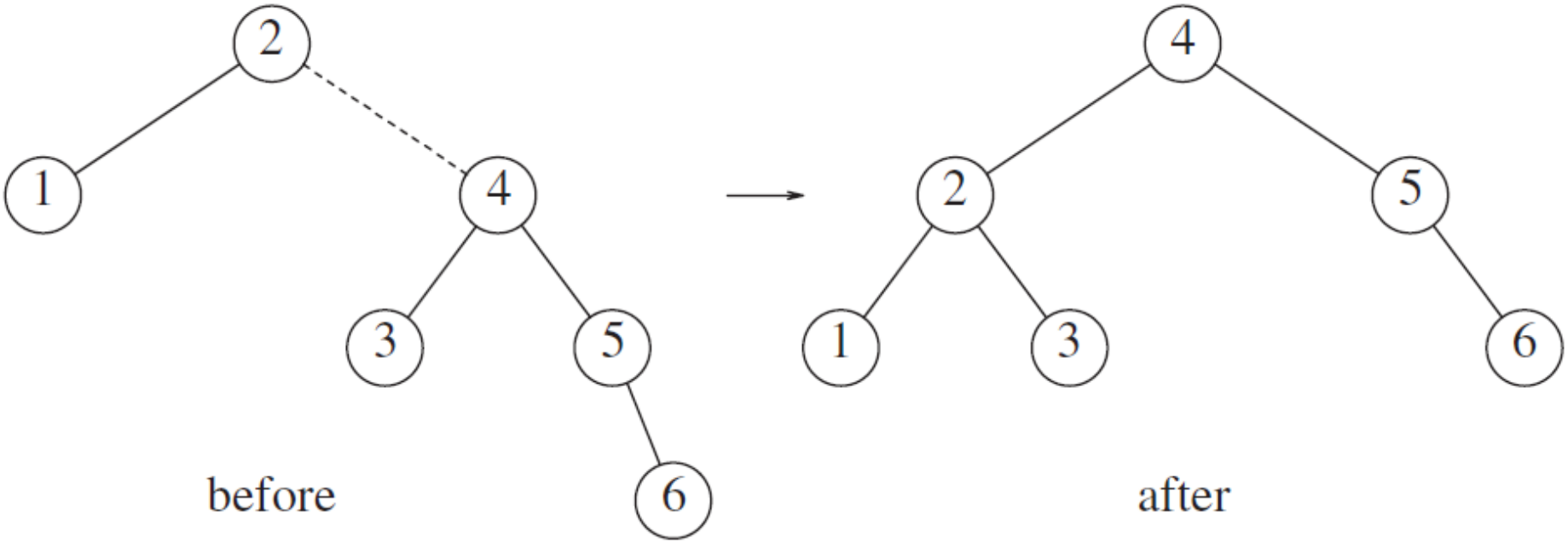
Ví dụ phép xoay đơn (2)

- Chèn 4, 5:



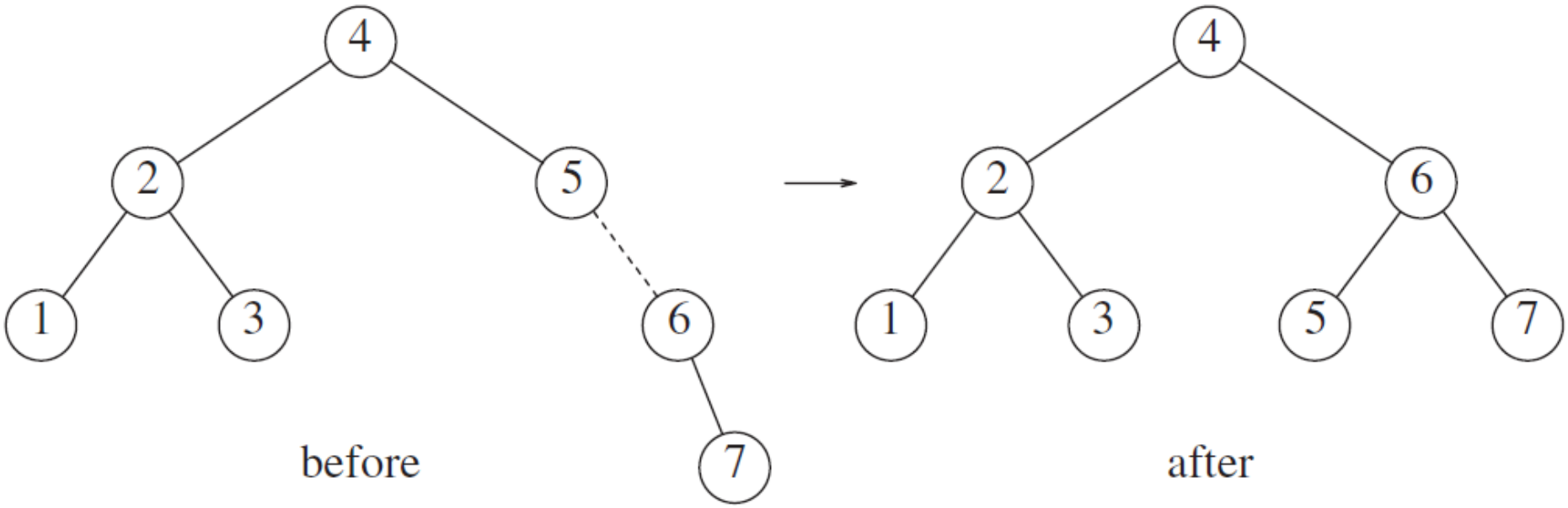
Ví dụ phép xoay đơn (3)

- Chèn 6:

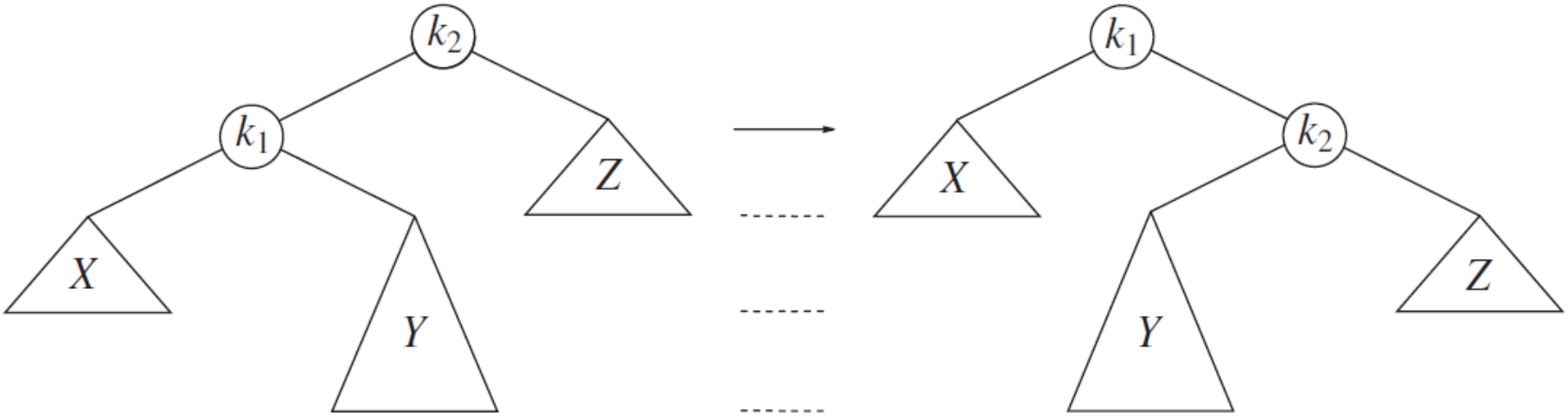


Ví dụ phép xoay đơn (4)

- Chèn 7:

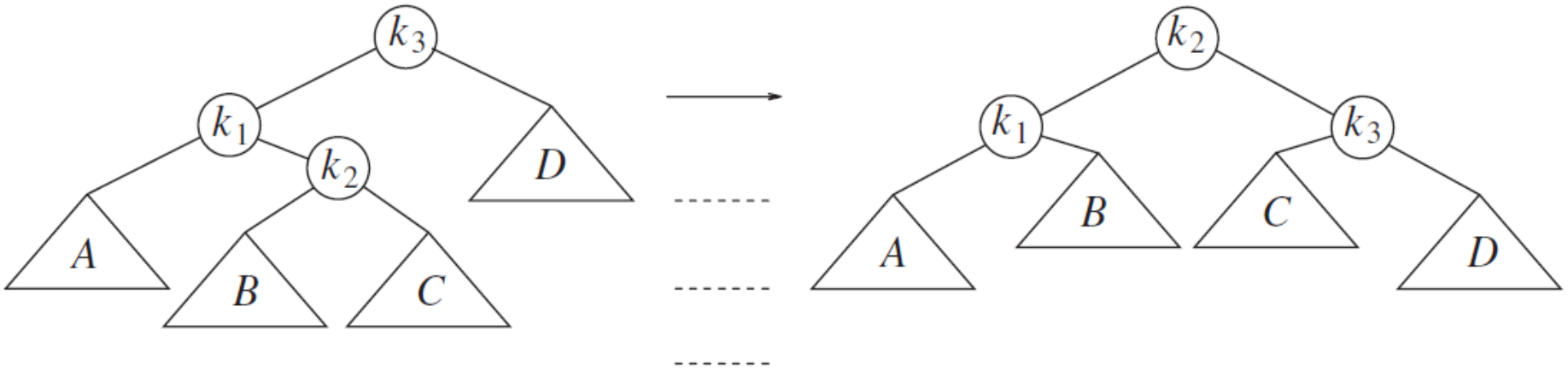


Phép xoay đơn không giải quyết được các trường hợp khác



- Đối với trường hợp 2:
  - Sau phép xoay đơn, nút k_1 không cân bằng.
- Ta cần dùng phép xoay kép cho hai trường hợp 2 và 3.

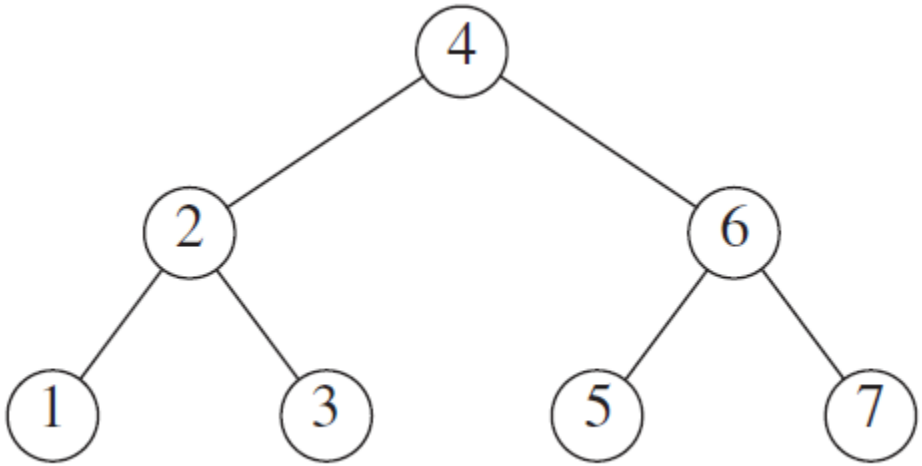
Phép xoay kép (trường hợp 2)



- Phép xoay kép trái-phải để sửa trường hợp 2:
  - Đầu tiên xoay giữa nút k_1 và nút k_2.
  - Sau đó xoay giữa nút k_2 và nút k_3.
- Với trường hợp 3, cách làm tương tự (xoay theo chiều ngược lại).

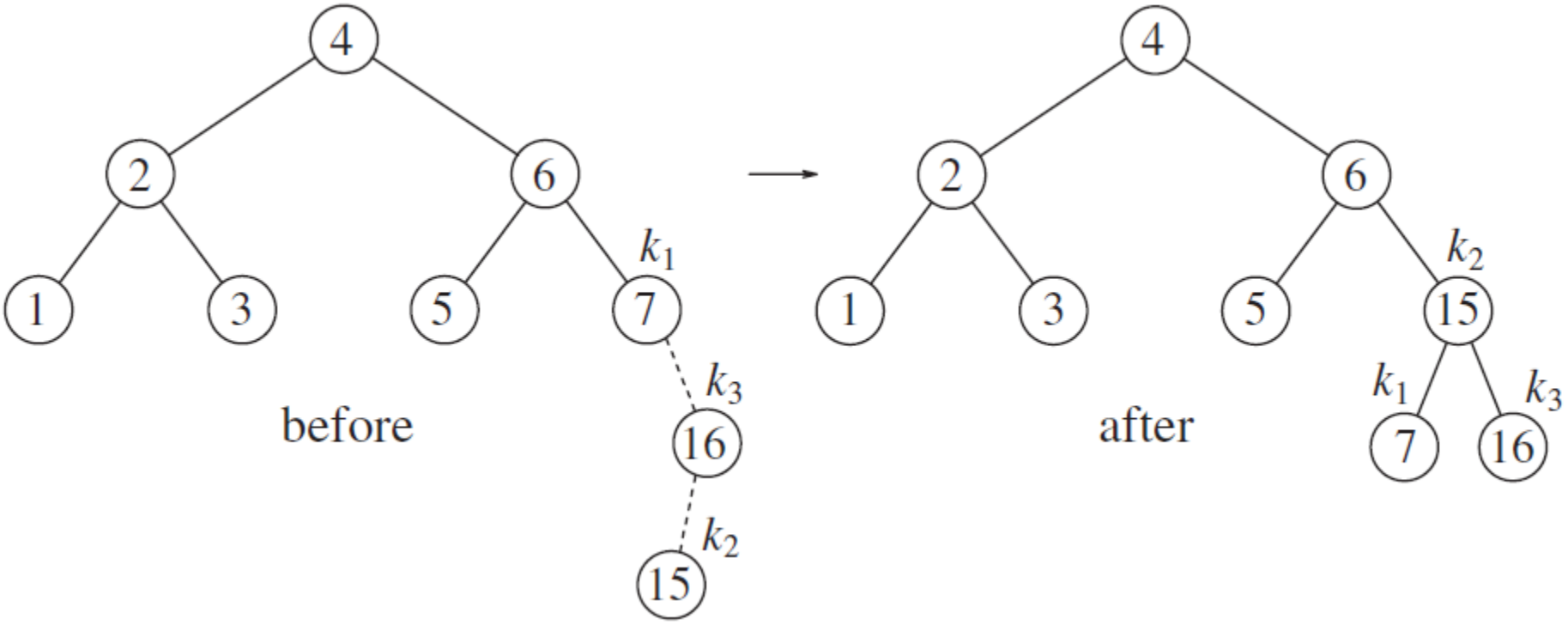
Ví dụ phép xoay kép (1)

- Chèn tuần tự 16, 15 và 14 vào cây AVL sau đây:



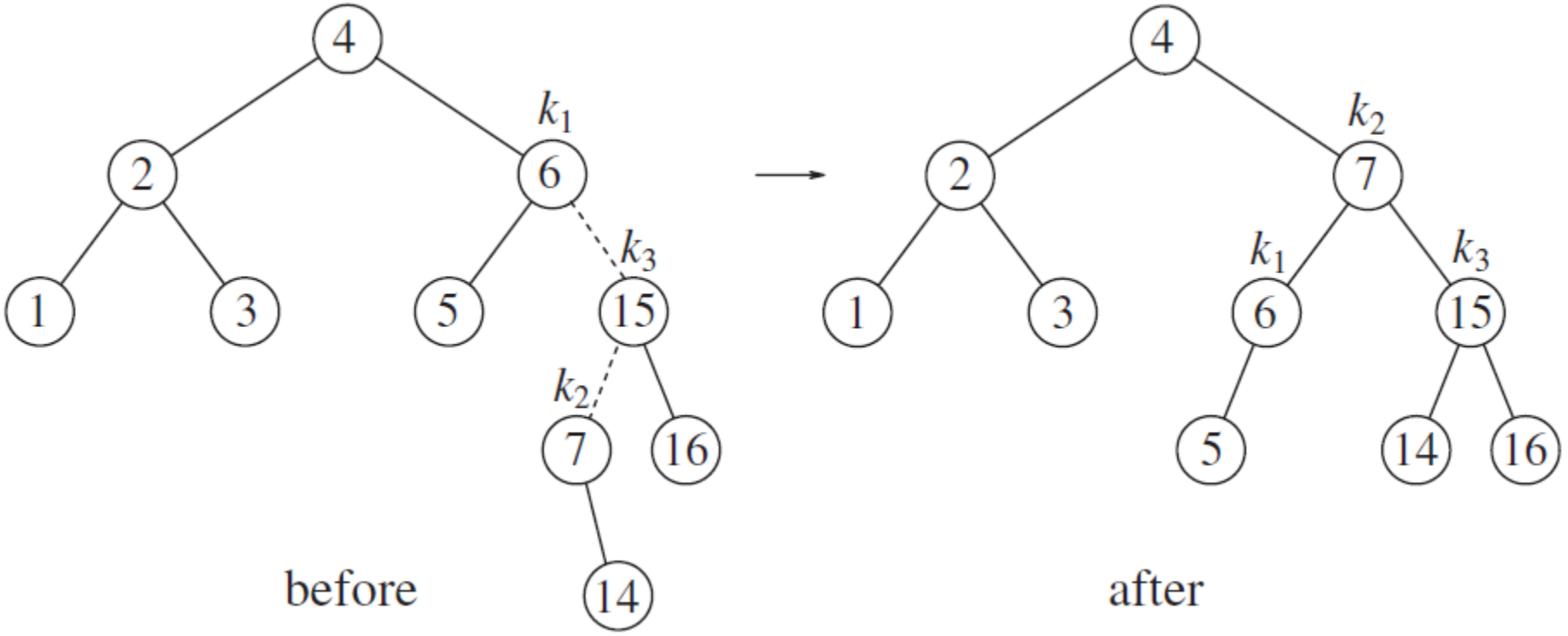
Ví dụ phép xoay kép (2)

- Chèn 16, 15:



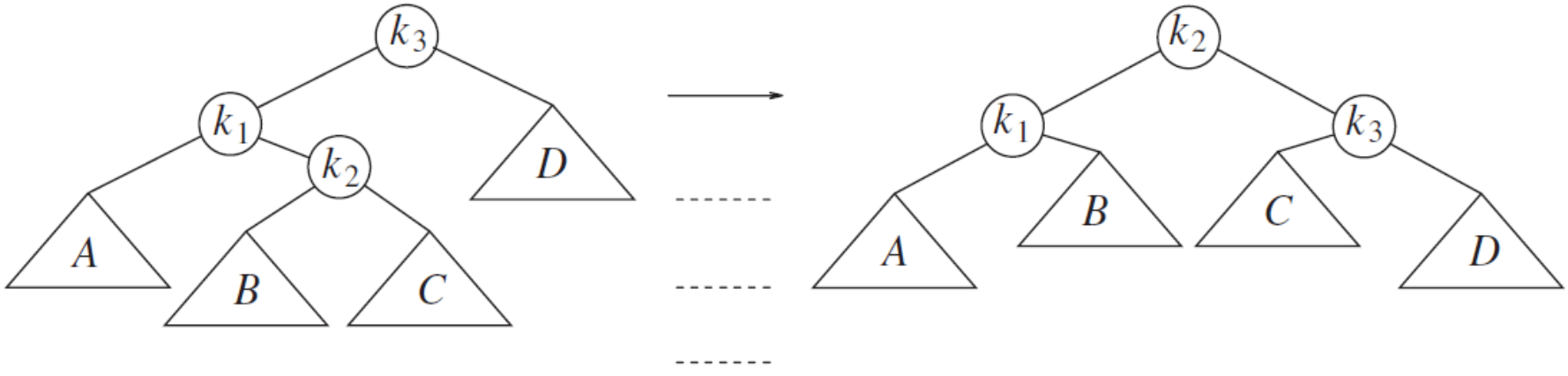
Ví dụ phép xoay kép (3)

- Chèn 14:



Phép xoay kép (trường hợp 2)

```
void doubleWithLeftChild(AvlNode * & k3) {
    rotateWithRightChild(k3->left);
    rotateWithLeftChild(k3);
}
```



Chèn vào cây AVL

```
// Chèn x vào cây có gốc t
void insert(T x, AvlNode * & t) {
    if (t == NULL) {
        t = new AvlNode;
        t->elem = x;
        t->left = NULL;
        t->right = NULL;
        t->height = 0;
    }
    else if (x < t->elem)
        insert(x, t->left);
    else if (x > t->elem)
        insert(x, t->right);

    balance(t); // Cân bằng cây sau khi chèn
}
```

Xóa khỏi cây AVL

```
// Xóa phần tử x khỏi cây có gốc t
void remove(T x, AvlNode * & t) {
    if (t == NULL) return; // Thoát ra nếu cây rỗng
    if (x < t->elem)       remove(x, t->left);
    else if (x > t->elem)  remove(x, t->right);
    else if (t->left != NULL && t->right != NULL) { // Nút 2 con
        t->elem = findMin(t->right)->elem;
        remove(t->elem, t->right);
    }
    else { // Nút 1 con hoặc lá
        AvlNode * old = t;
        t = (t->left != NULL) ? t->left : t->right;
        delete old;
    }
    balance(t); // Cân bằng cây sau khi xóa
}
```

Cân bằng cây AVL sau khi chèn/xóa

```
void balance(AvlNode * & t) {
    if (t == NULL)
        return;

    if (height(t->left) - height(t->right) > 1)
        if (height(t->left->left) >= height(t->left->right))
            rotateWithLeftChild(t);
        else
            doubleWithLeftChild(t);
    else if (height(t->right) - height(t->left) > 1)
        if (height(t->right->right) >= height(t->right->left))
            rotateWithRightChild(t);
        else
            doubleWithRightChild(t);

    t->height = max(height(t->left), height(t->right)) + 1;
}
```

Hàm **balance** được gọi tại mỗi nút trong quá trình đi ngược từ chỗ chèn/xóa về nút gốc. Phải cập nhật chiều cao của nút t vì chiều cao của nút t phụ thuộc vào các nút phía dưới, trong khi phía dưới có thể bị xáo trộn do đã xảy ra một phép xoay nào đó.

Bài tập

Vẽ hình mô tả quá trình biến đổi của một cây AVL đang rỗng khi chèn lần lượt vào cây các giá trị sau vào cây:

{ 2, 1, 4, 5, 9, 3, 6, 7 }