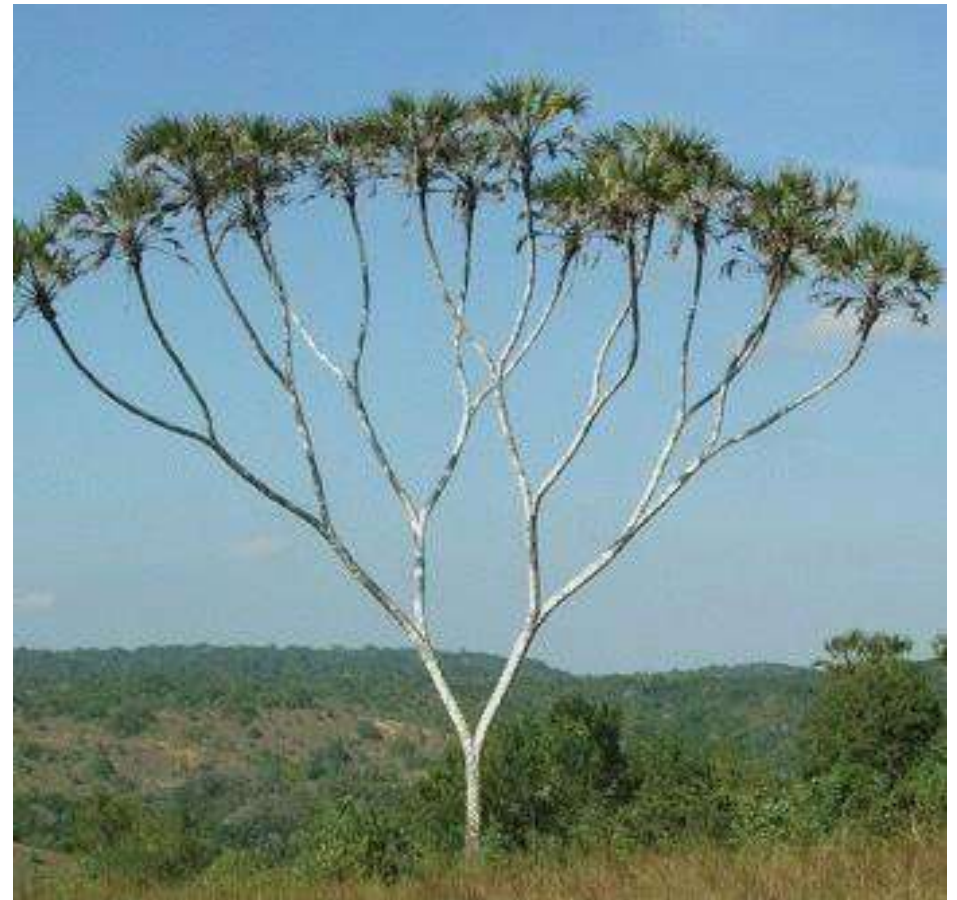




Cây tìm kiếm nhị phân

Nội dung

- Cấu trúc cây
- Cây tìm kiếm nhị phân
- Cân bằng cây
- Cây AVL



Mảng và danh sách liên kết

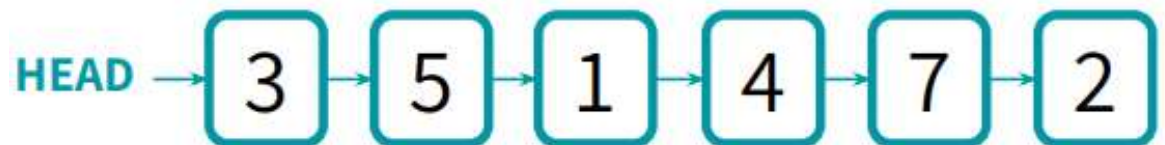
■ Lưu trữ dữ liệu dạng tuyến tính

- ☐ Tìm kiếm
- ☐ Thêm phần tử
- ☐ Xóa phần tử

Sorted Arrays

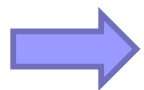


Linked Lists



Mảng và danh sách liên kết

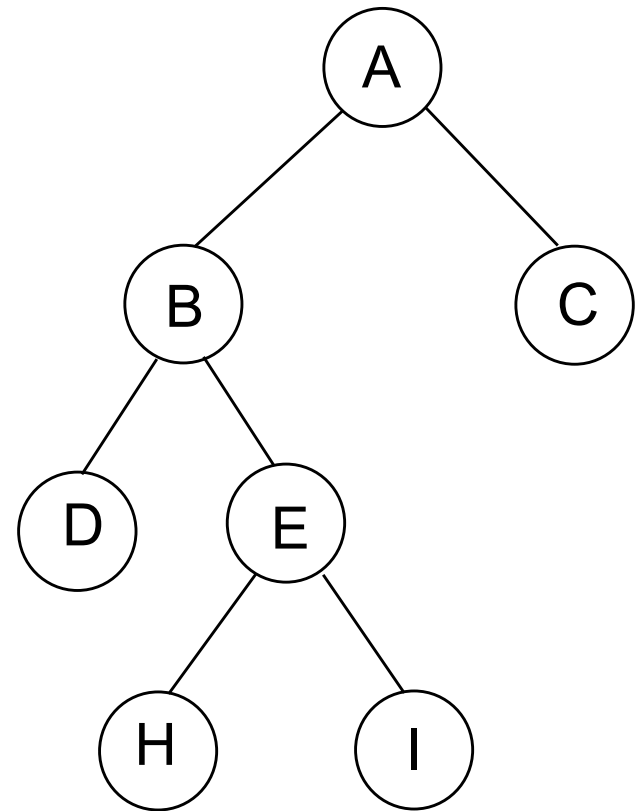
OPERATION	SORTED ARRAY	UNSORTED LINKED LIST
SEARCH	$O(\log(n))$	$O(n)$
DELETE	$O(n)$	$O(n)$
INSERT	$O(n)$	$O(1)$



Làm thế nào để kết hợp lợi thế của 2 cấu trúc này

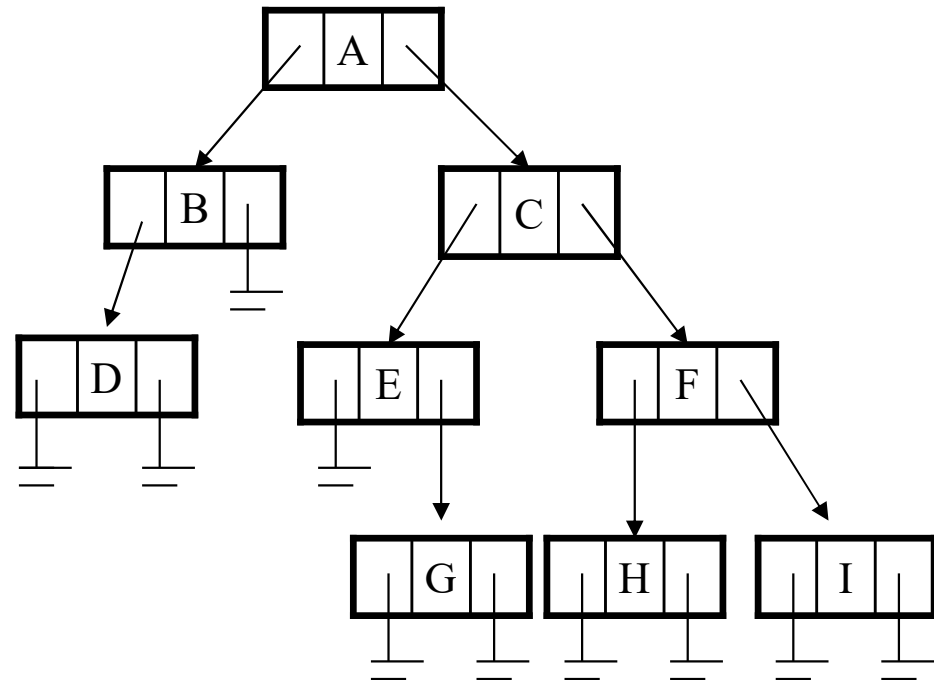
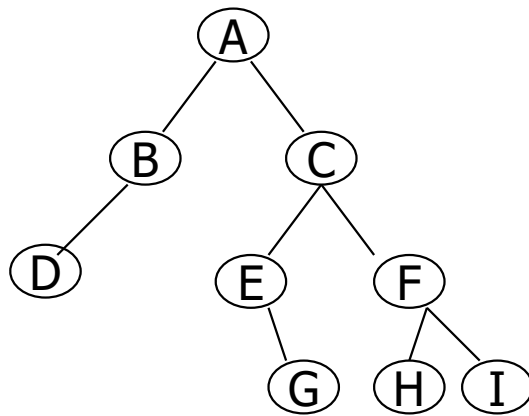
Cấu trúc cây và cây nhị phân

- Là cấu trúc dữ liệu dựa trên liên kết các node
- Mỗi node có thể có liên kết với các nút con
 - Không có vòng lặp
 - Nút gốc: không có nút cha; Nút lá: không có nút con
 - Cây con: bộ phận của cây có đỉnh không phải là nút gốc
- Cây nhị phân
 - Chỉ có tối đa 2 nút con
 - Là cấu trúc cây phổ biến



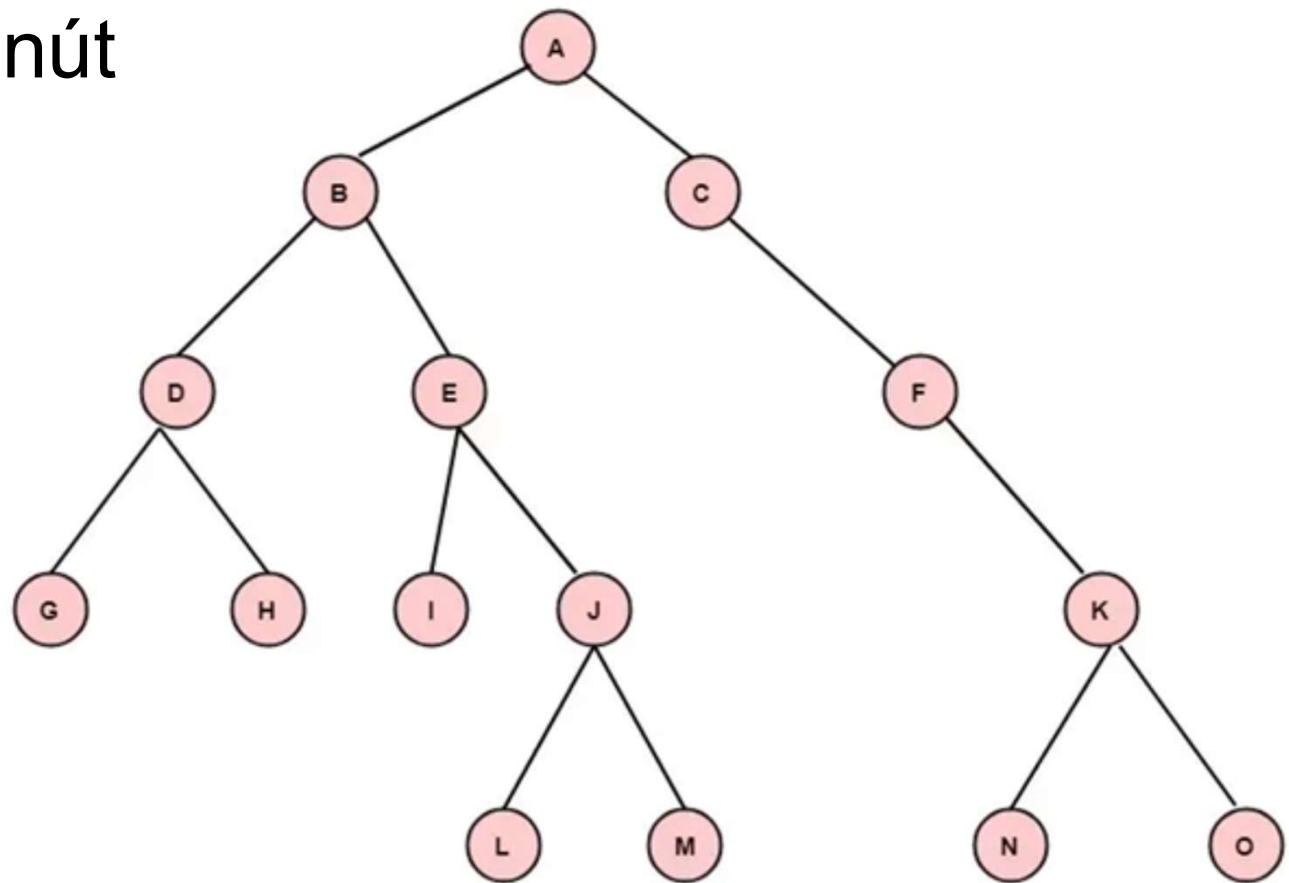
Cây nhị phân

```
struct node {  
    long data;  
    node* left_child,* right_child;  
};
```



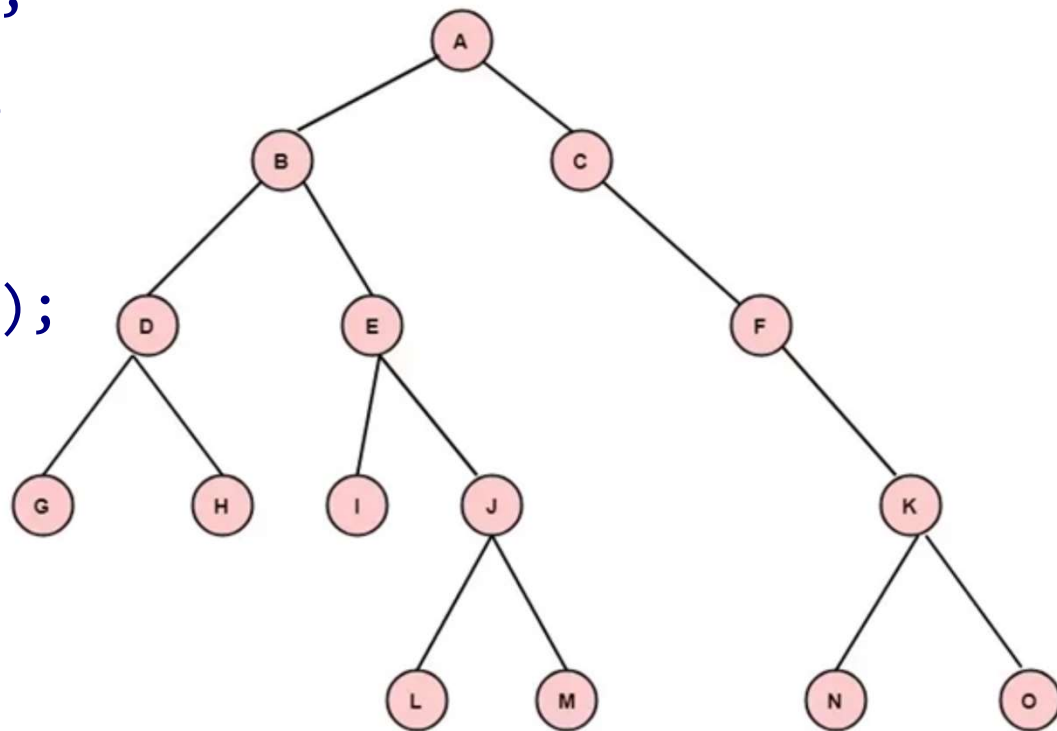
Các thao tác trên cây

- Tạo cây
- Thêm nút / xóa nút
- Duyệt cây
- Tìm kiếm



Duyệt cây

```
void print(node* t) {  
    if (t->left_child)  
        print(t->left_child);  
    cout << t->data << endl;  
    if (t->right_child)  
        print(t->right_child);  
}
```

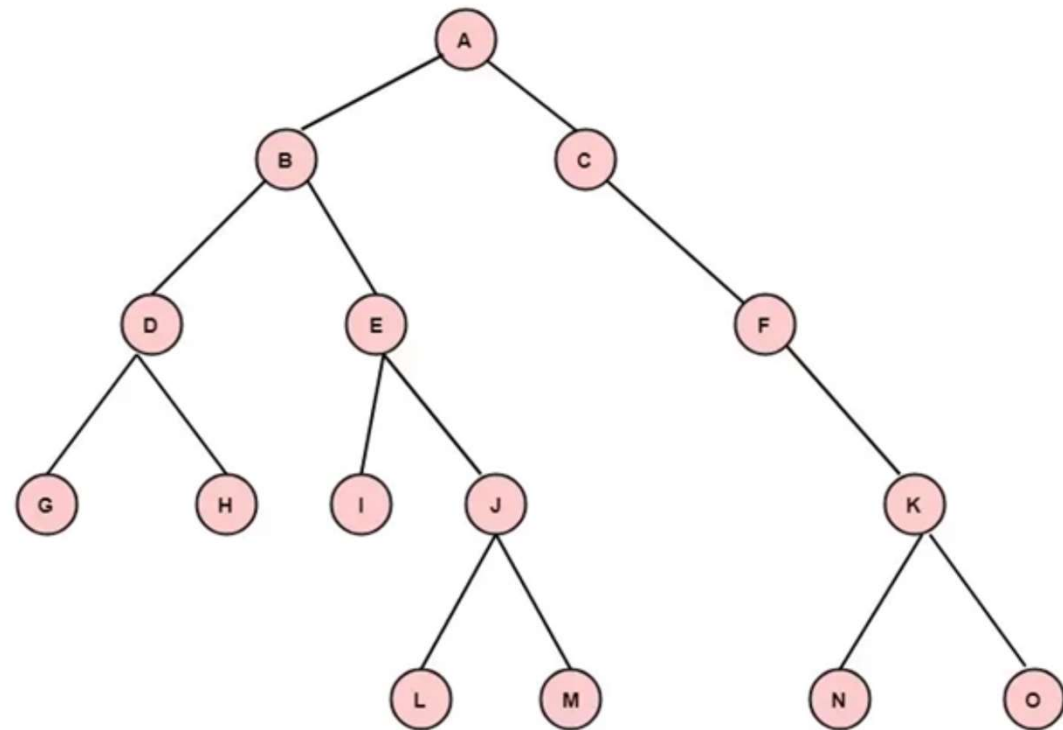


Tìm kiếm

```
node* search(node* t, long key) {
```

???

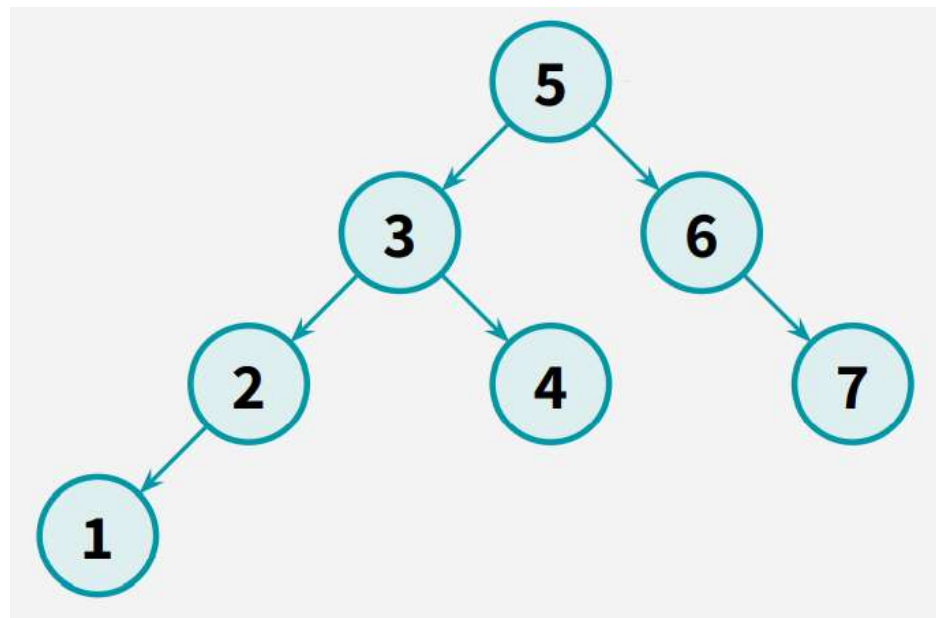
}



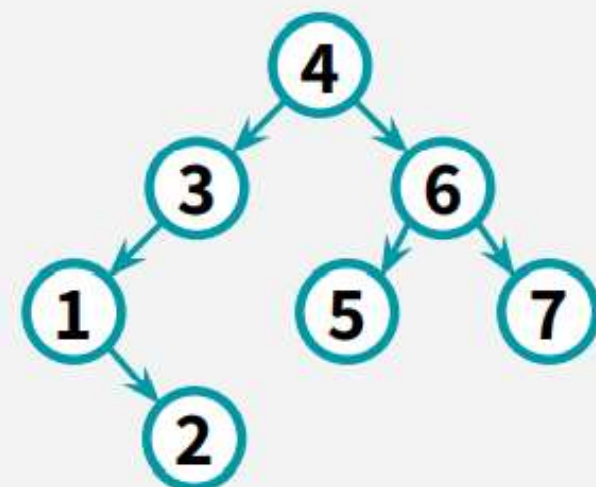
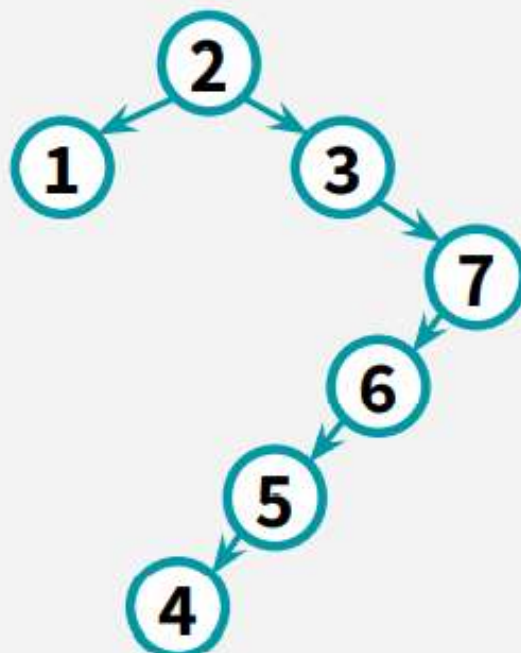
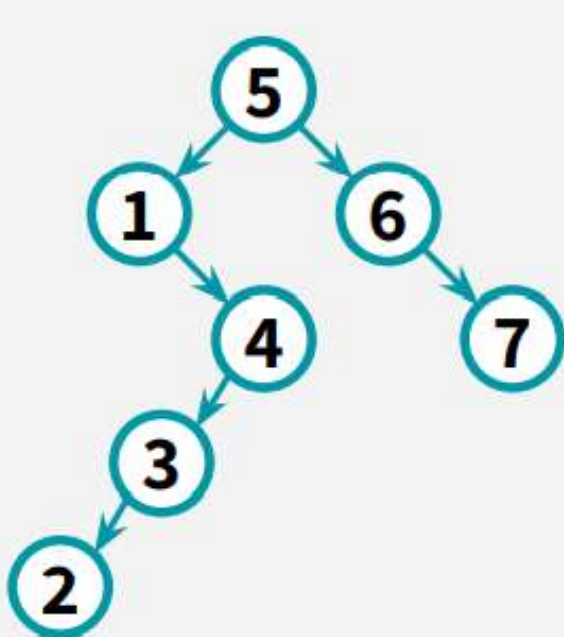
Cây tìm kiếm nhị phân

Binary Search Tree - BST

- Là cây nhị phân được sắp xếp
- Cây con bên trái chứa các nút nhỏ hơn (hoặc bằng) nút cha, cây con bên phải chứa các nút lớn hơn nút cha

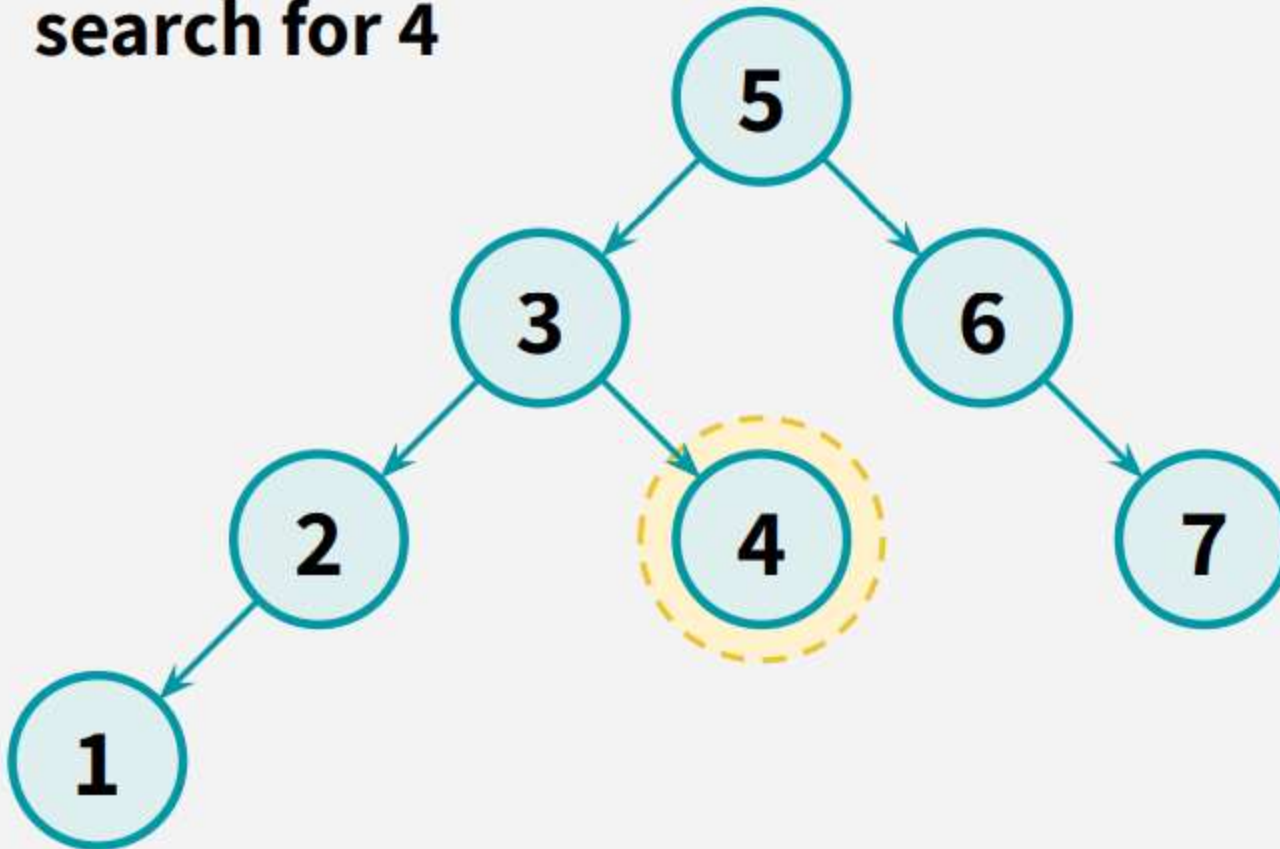


BST: nhiều cây cho cùng một tập dữ liệu



BST: tìm kiếm

search for 4



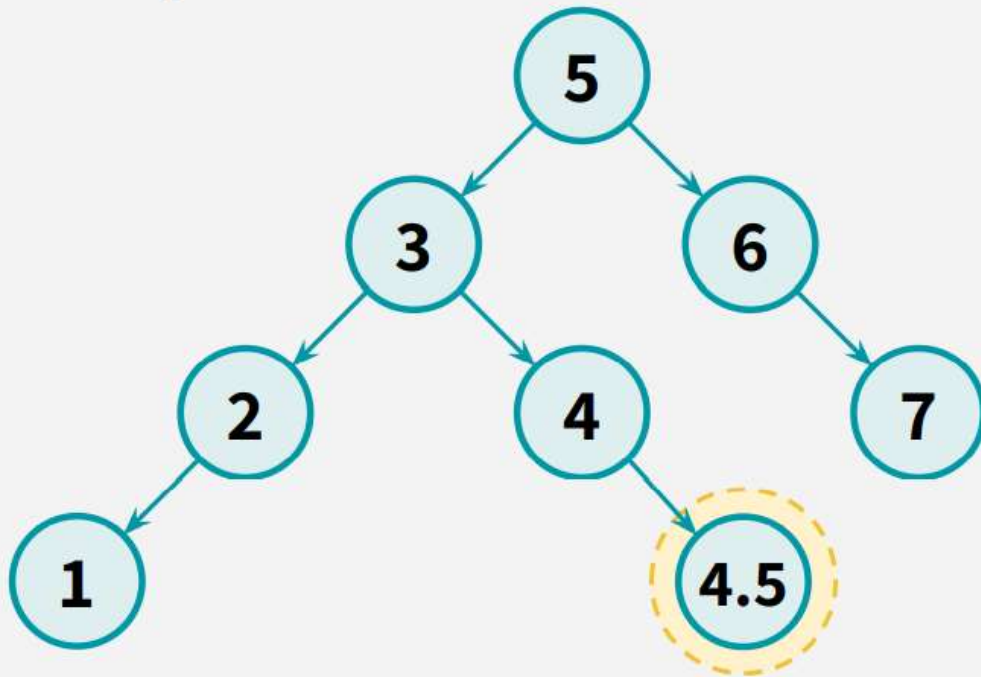
Compare **4** with **root**:
4 is smaller → go left!

Compare **4** with **3**:
4 is larger → go right!

Compare **4** with **4**:
 $4 = 4 \rightarrow$ We found it!

BST: Thêm nút mới

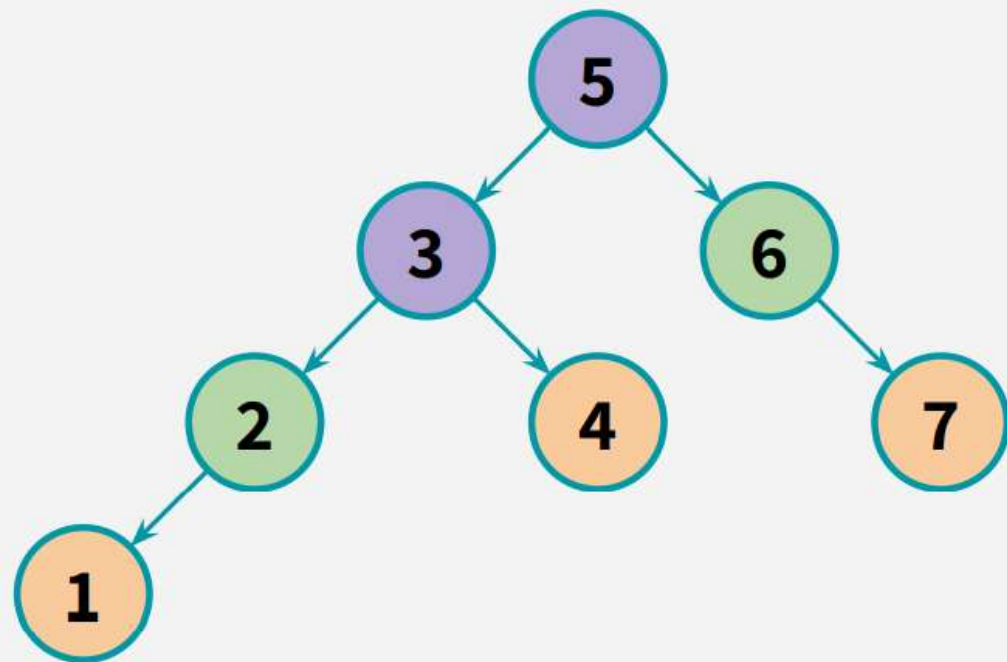
Example: **Insert 4.5**



```
INSERT(root, key):  
    x = SEARCH(root, key)  
    node = new node with key  
    if key < x.key:  
        x.left = node  
    if key > x.key:  
        x.right = node  
    if key = x.key:  
        return
```

BST: Xóa nút

- 3 trường hợp: nút lá, nút có 1 nút con, có 2 nút con

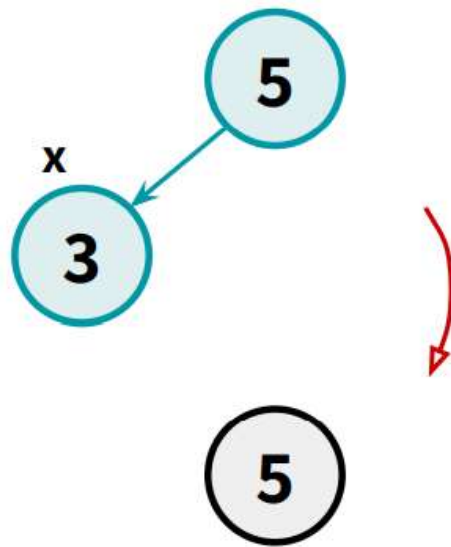


```
DELETE(root, key):  
  x = SEARCH(root, key)  
  if key = x.key:  
    CASE 1: x is a leaf  
    CASE 2: x has 1 child  
    CASE 3: x has 2 children
```


BST: Xóa nút

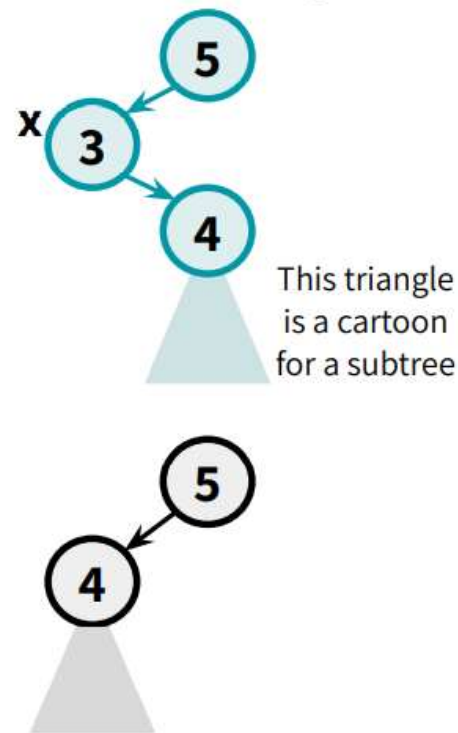
CASE 1: x is a leaf

Just delete x!



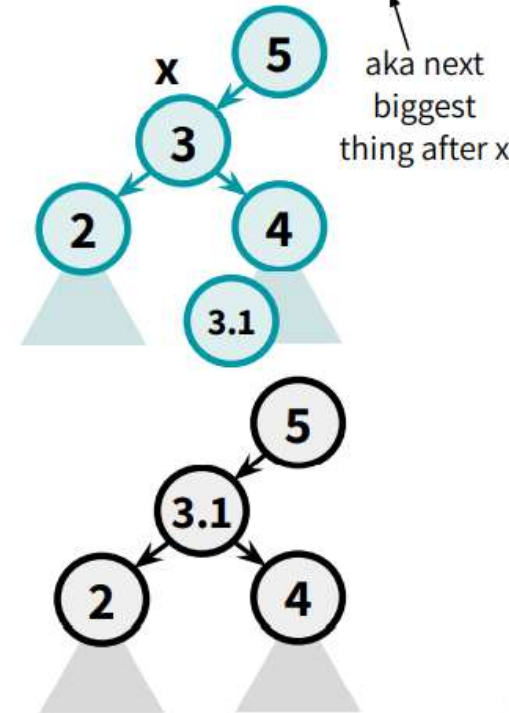
CASE 2: x has 1 child

Move its child up!



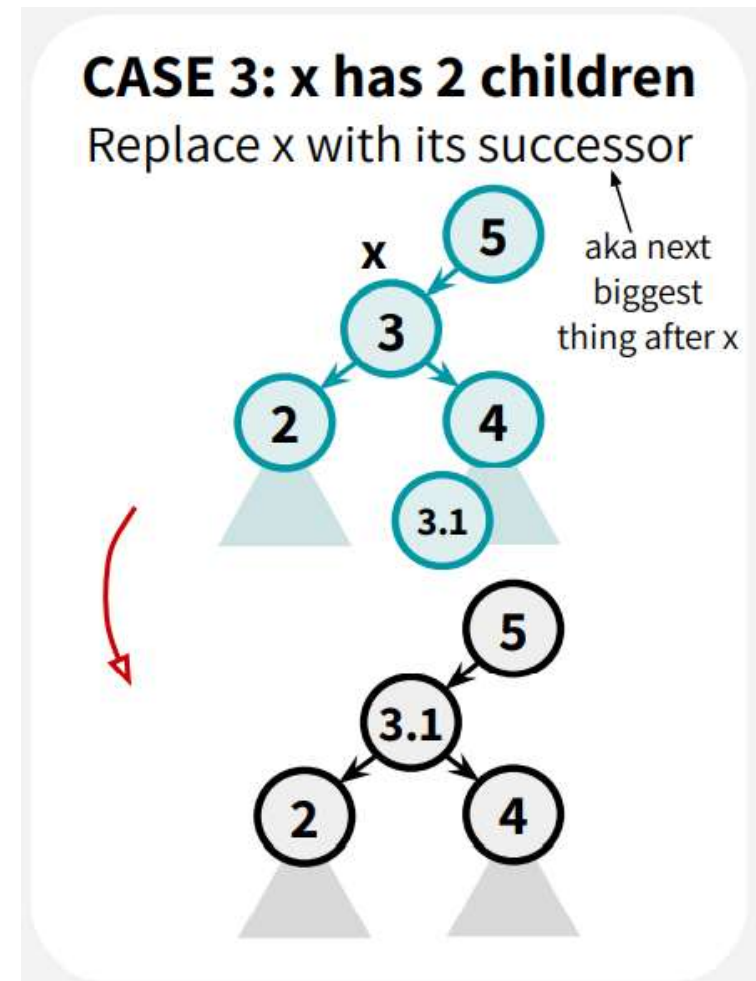
CASE 3: x has 2 children

Replace x with its successor



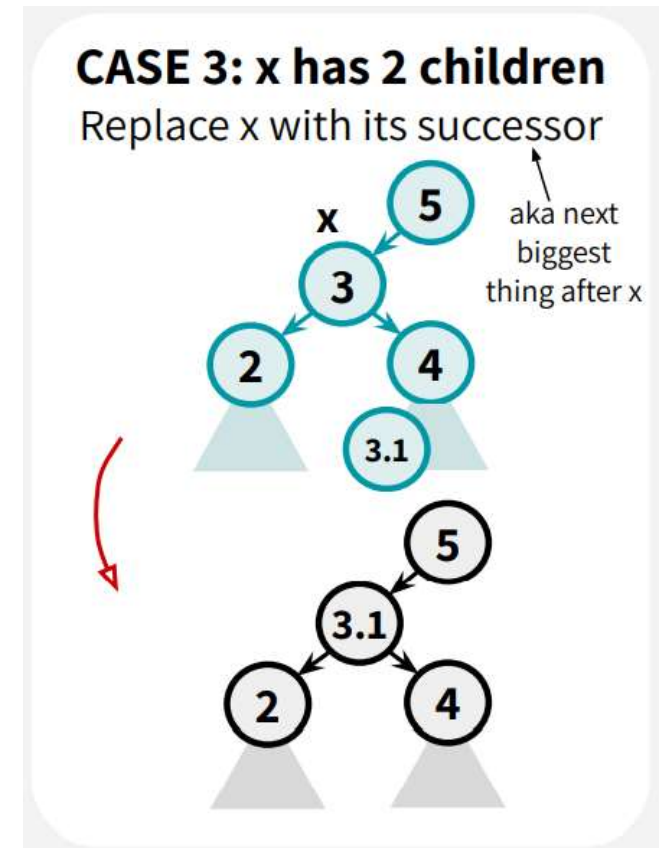
BST: Xóa nút

- Trường hợp nút xóa có 2 nút con
 - Nút thay thế là nút lá
 - Nút thay thế có 1 nút con
 - **Nút thay thế có 2 nút con không?**



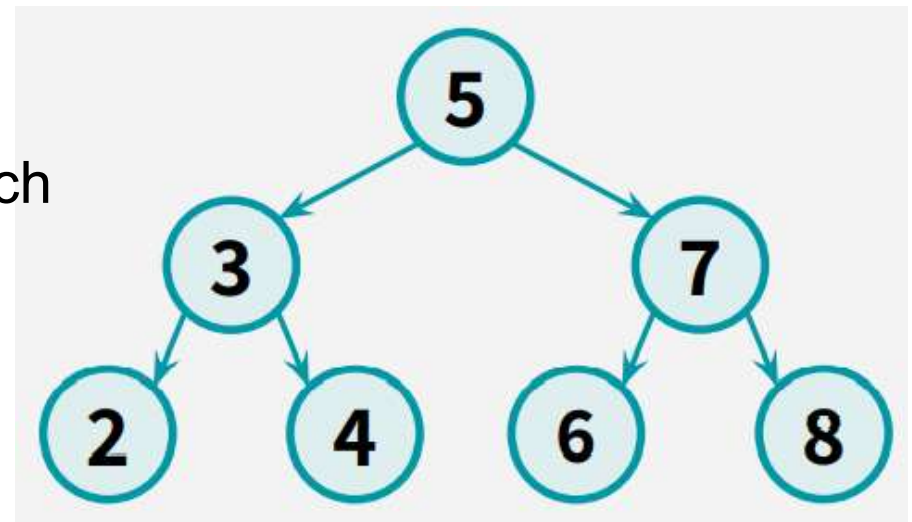
Xóa nút

- Cần tìm nút có giá trị lớn nhất/nhỏ nhất



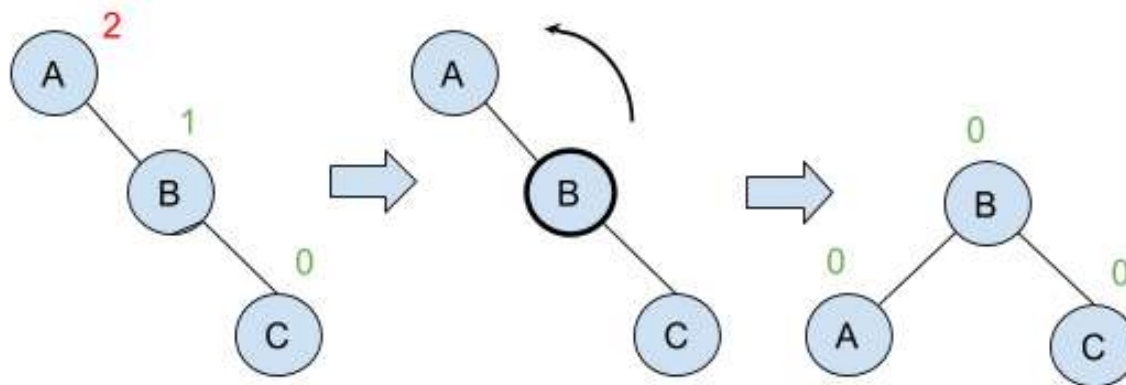
BST: Độ phức tạp

- Độ phức tạp của các thuật toán
 - Tìm kiếm?
 - Thêm nút?
 - Xóa nút?
- Để hiệu quả, cần cân bằng độ cao của cây BST
- Cây BST cân bằng
 - độ cao cây con không chênh lệch



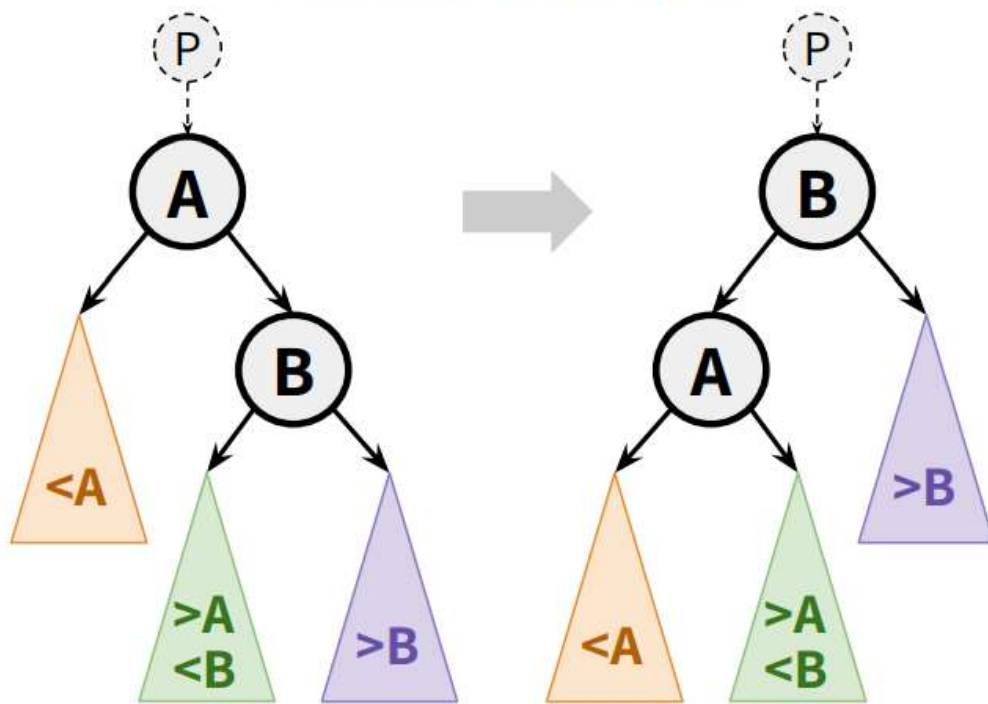
BST: cân bằng cây

- Điều chỉnh độ cao để cây cân bằng
- Điều chỉnh thông qua phép “xoay cây”
 - Di chuyển cây con từ nhánh trái qua nhánh phải hoặc ngược lại
 - Thay đổi nút đỉnh để đảm bảo tính chất của cây

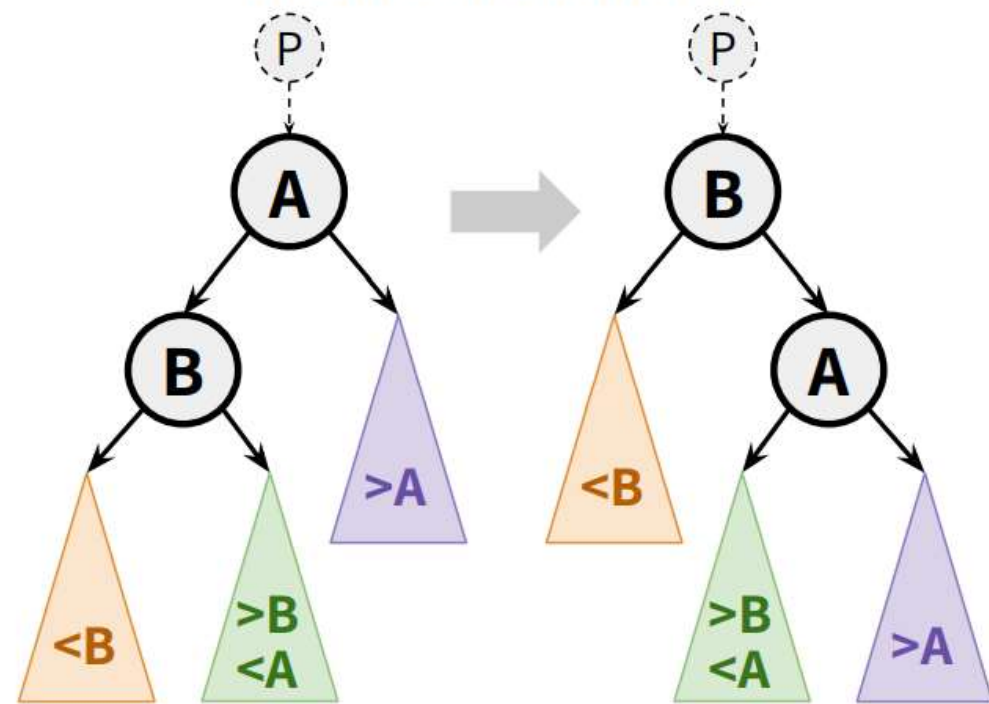


BST: xoay cây

LEFT ROTATION



RIGHT ROTATION

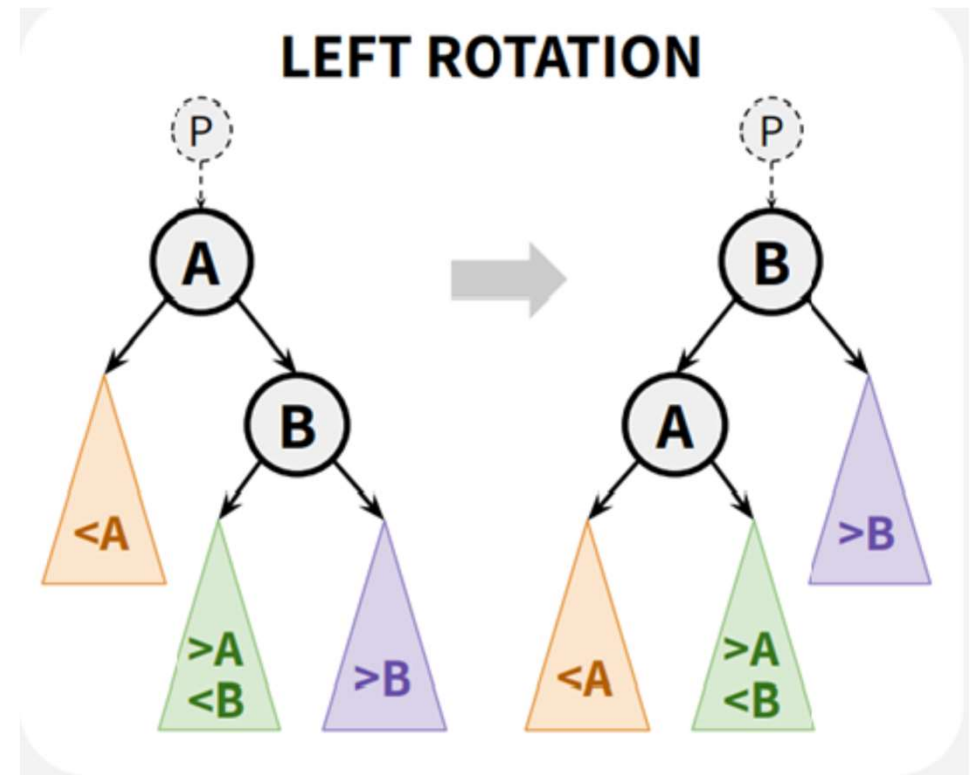


Xoay cây

```
node* left_rotate(node* root) {
```

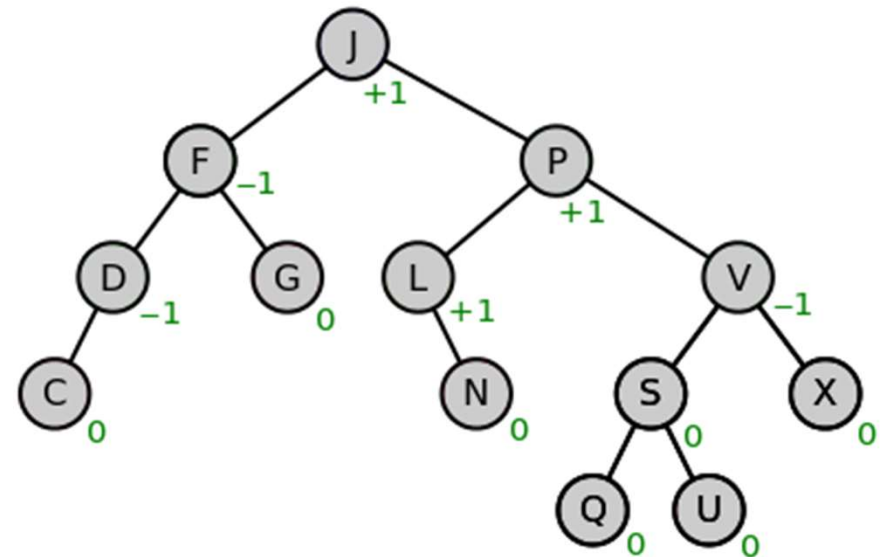
???

}



Cây AVL (**A**delson-**V**elsky and **L**andis)

- Là cây BST cân bằng cao
 - Độ lệch tối đa 1
 - Mỗi nút chứa thêm thông tin cân bằng
- Với một cây AVL
 - Việc thêm, bớt nút trước tiên thực hiện như với cây BST
 - Nếu cây mới mất cân bằng (độ lệch 2) thì thực hiện phép xoay để cân bằng cây

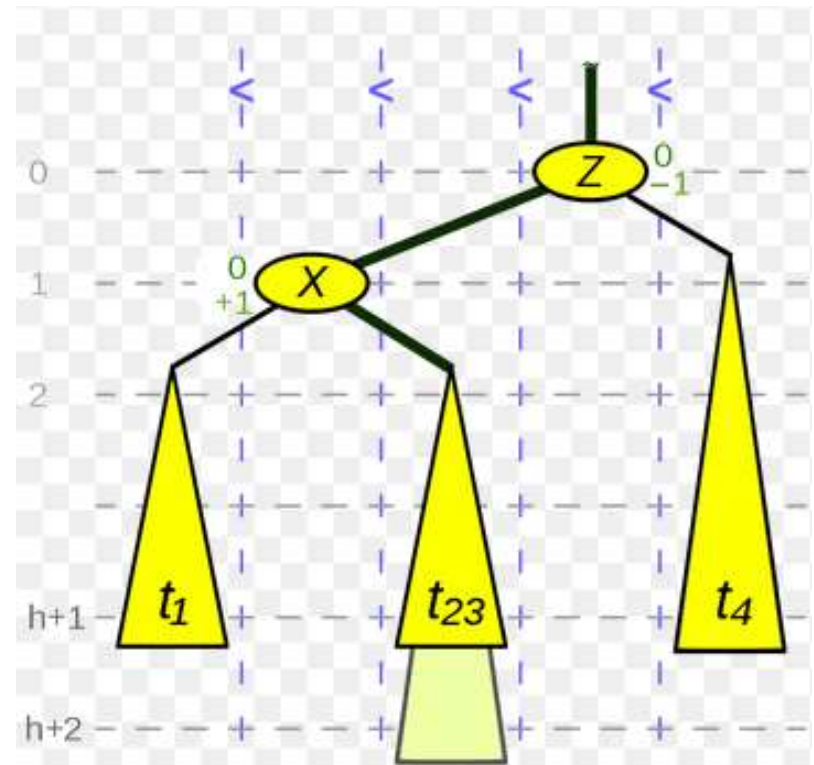
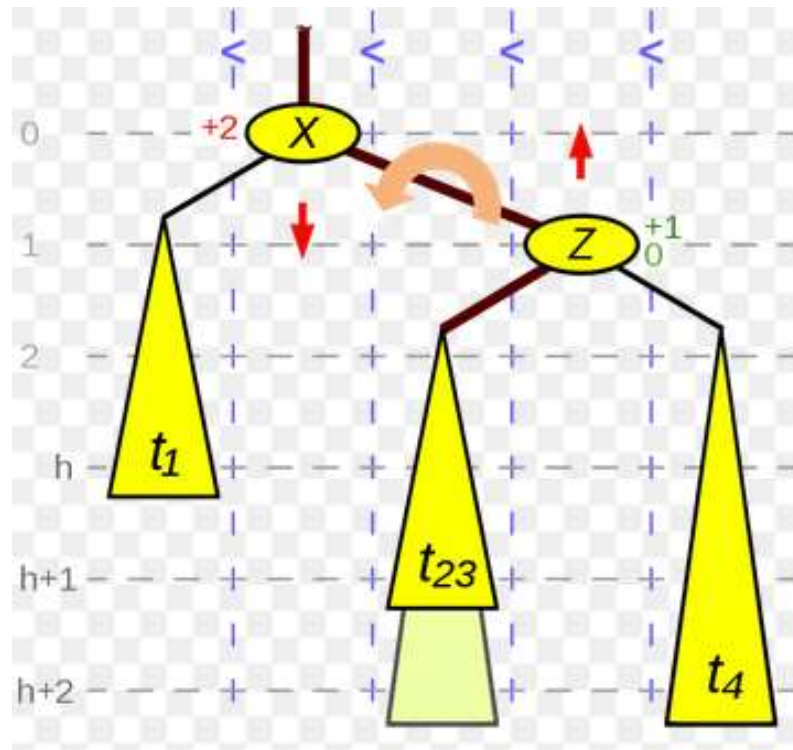


Độ lệch: -1, 0, 1

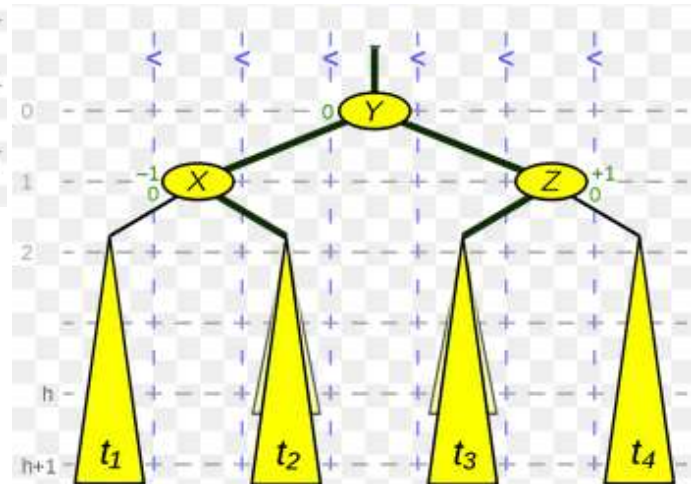
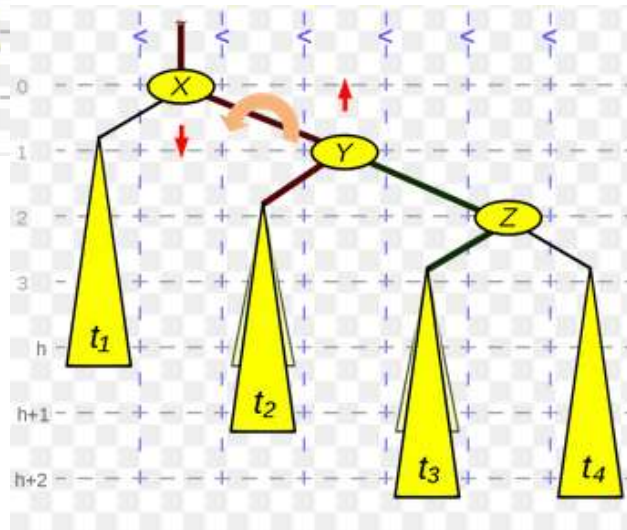
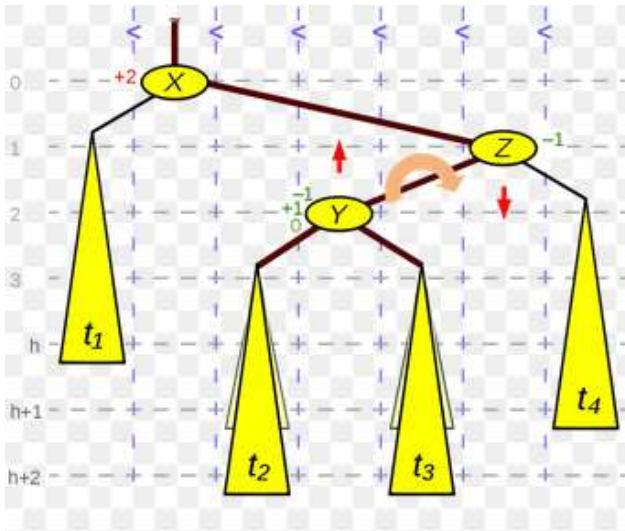
Cân bằng cây AVL

- 4 trường hợp cần cân bằng
- Trường hợp cần 1 phép xoay
 - Cây con phải lệch phải
 - Cây con trái lệch trái
- Trường hợp cần 2 phép xoay
 - Cây con phải lệch trái
 - Cây con trái lệch phải

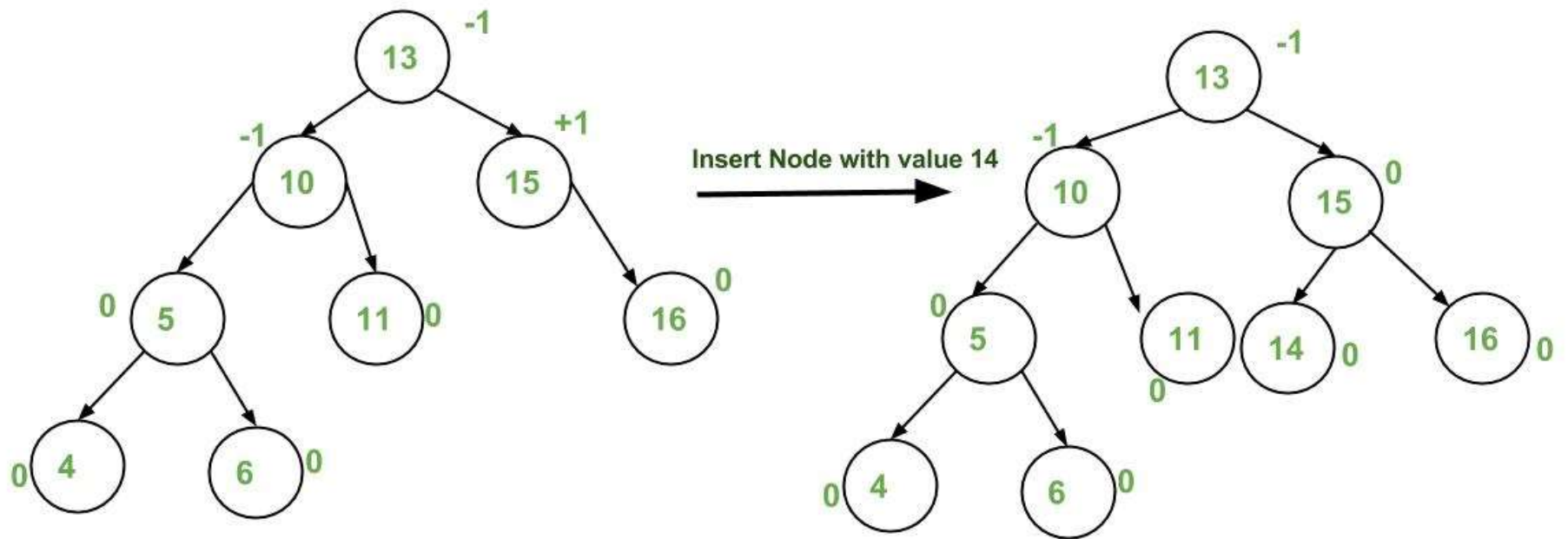
AVL: Right - Right



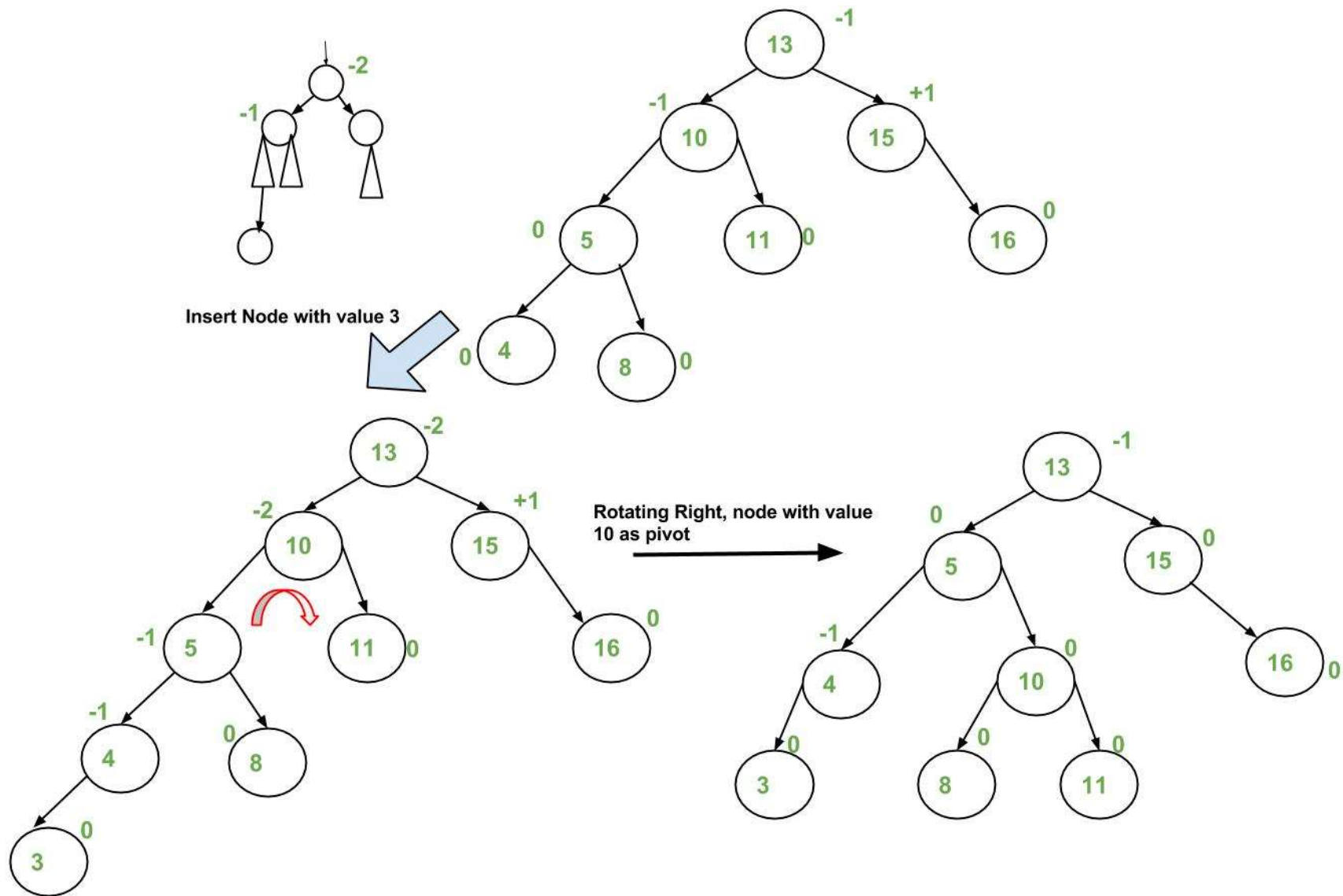
AVL: Right - Left



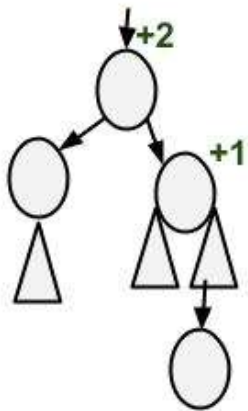
Ví dụ: thêm nút



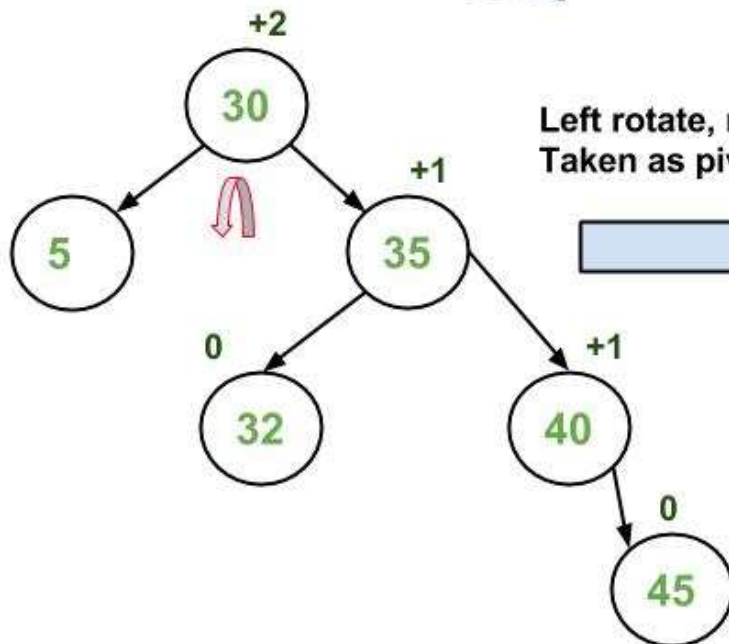
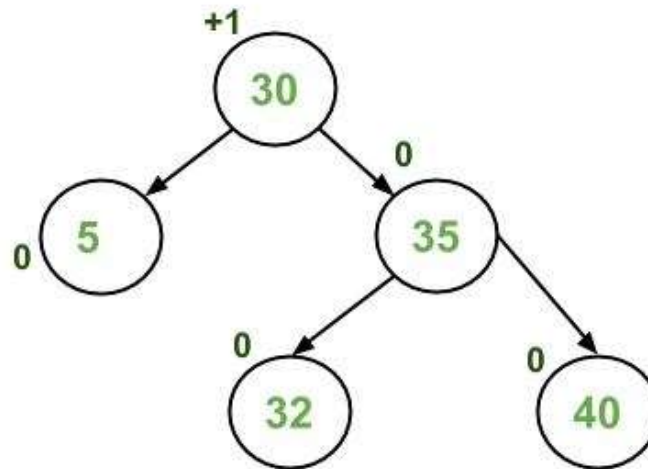
Ví dụ: thêm nút



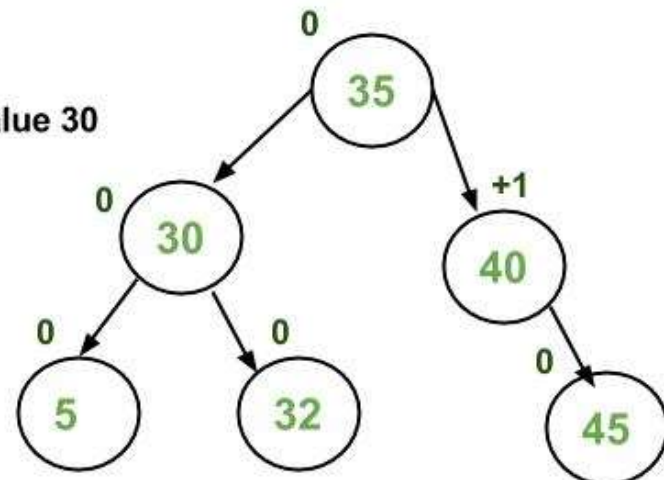
Ví dụ: thêm nút

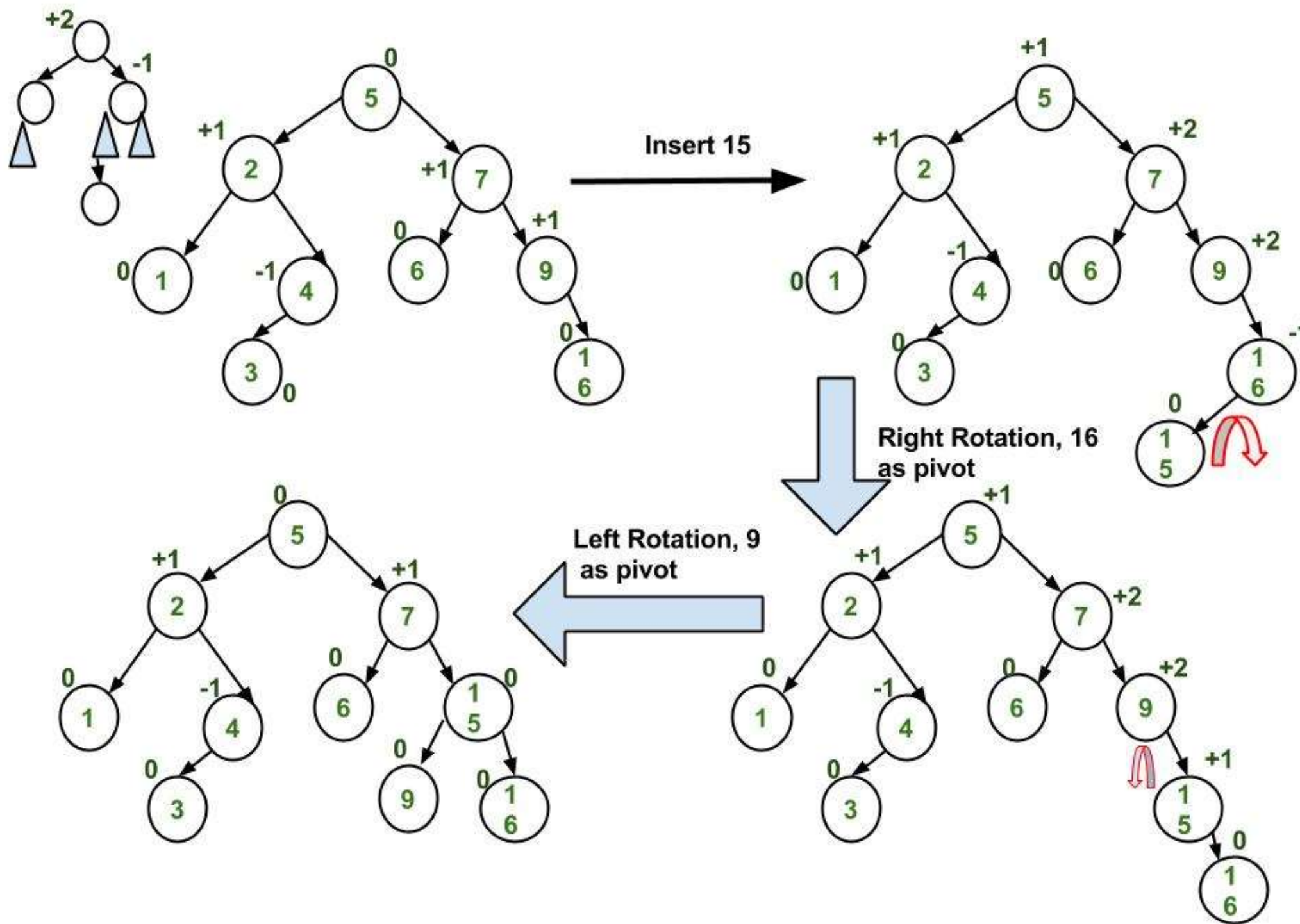


Insert 45



Left rotate, node with value 30
Taken as pivot





Mảng, danh sách và cây BST

OPERATION	SORTED ARRAY	UNSORTED LINKED LIST	BST (WORST CASE)	BST (BALANCED)
SEARCH	$O(\log(n))$	$O(n)$	$O(n)$	$O(\log(n))$
DELETE	$O(n)$	$O(n)$	$O(n)$	$O(\log(n))$
INSERT	$O(n)$	$O(1)$	$O(n)$	$O(\log(n))$



Bài tập/thực hành


- Cài đặt cây tìm kiếm nhị phân, với các phép toán tìm kiếm, thêm, bớt nút
- Cài đặt phép xoay cây
- Cài đặt cây AVL

Tìm hiểu thêm

- Cây tìm kiếm nhị phân “đỏ đen” (Red Black Trees)

Tự học

- Tìm hiểu giải thuật duyệt cây theo chiều sâu (DFS), duyệt cây theo chiều rộng (BFS)
- Cài đặt DFS bằng đệ qui và không dùng đệ qui



Chuẩn bị

- Cây heap, hàng đợi ưu tiên
- Thuật toán sắp xếp heapsort