

# CÂY NHỊ PHÂN TÌM KIẾM (BINARY SEARCH TREE)

---

DATA STRUCTURES AND ALGORITHMS

TS. Nguyễn Đình Hiền



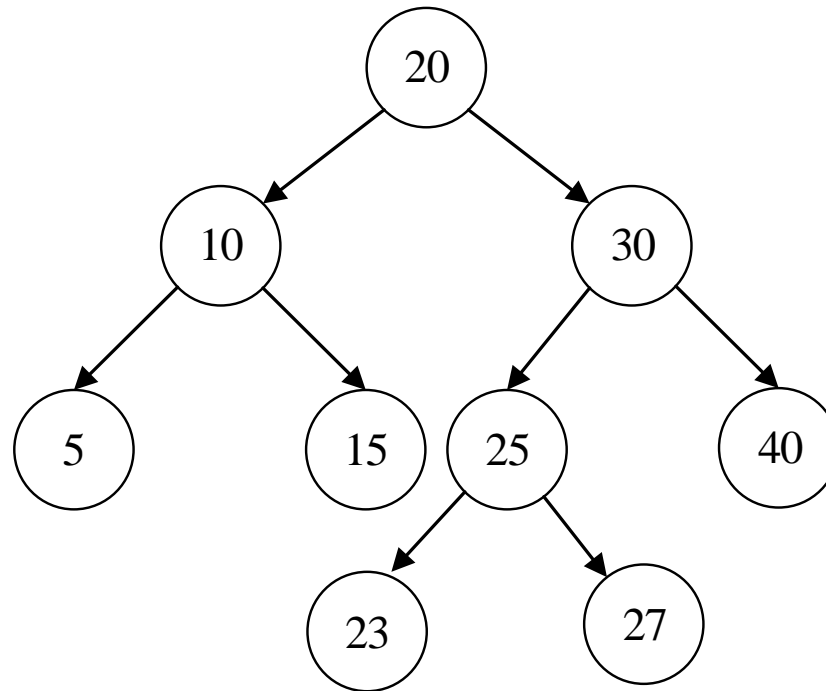
- Định nghĩa
- Tổ chức lưu trữ
- Các thao tác
- Ứng dụng
- Bài tập



# Định nghĩa cây nhị phân tìm kiếm

- Cây nhị phân
- Bảo đảm nguyên tắc bố trí khoá tại mỗi node:
  - Các node trong cây trái nhỏ hơn node hiện hành
  - Các node trong cây phải lớn hơn node hiện hành

• Ví dụ:





# Ưu điểm của cây nhị phân tìm kiếm

- Nhờ trật tự bố trí khóa trên cây :
  - Định hướng được khi tìm kiếm

Cây gồm N phần tử :

- Trường hợp tốt nhất  $h = \log_2 N$
- Trường hợp xấu nhất  $h = n-1$
- Tình huống xảy ra trường hợp xấu nhất ?



- *Cấu trúc dữ liệu của 1 node*

```
struct TNode {  
    int key;  
    TNode* pLeft;  
    TNode* pRight;  
};
```

- *Cấu trúc dữ liệu của cây*

```
typedef TNode* TREE;
```



- Tạo 1 cây rỗng
- Tạo 1 node có trường key bằng x
- Thêm 1 node vào cây nhị phân tìm kiếm
- In danh sách node trong cây
- Tìm 1 node có khoá bằng x trên cây
- Xoá 1 node có key bằng x trên cây



# Tạo cây rỗng

- Cây rỗng -> địa chỉ node gốc bằng NULL

```
void CreateTree(TREE &T) {  
    T = NULL;  
}
```



## Tạo 1 node có key bằng x

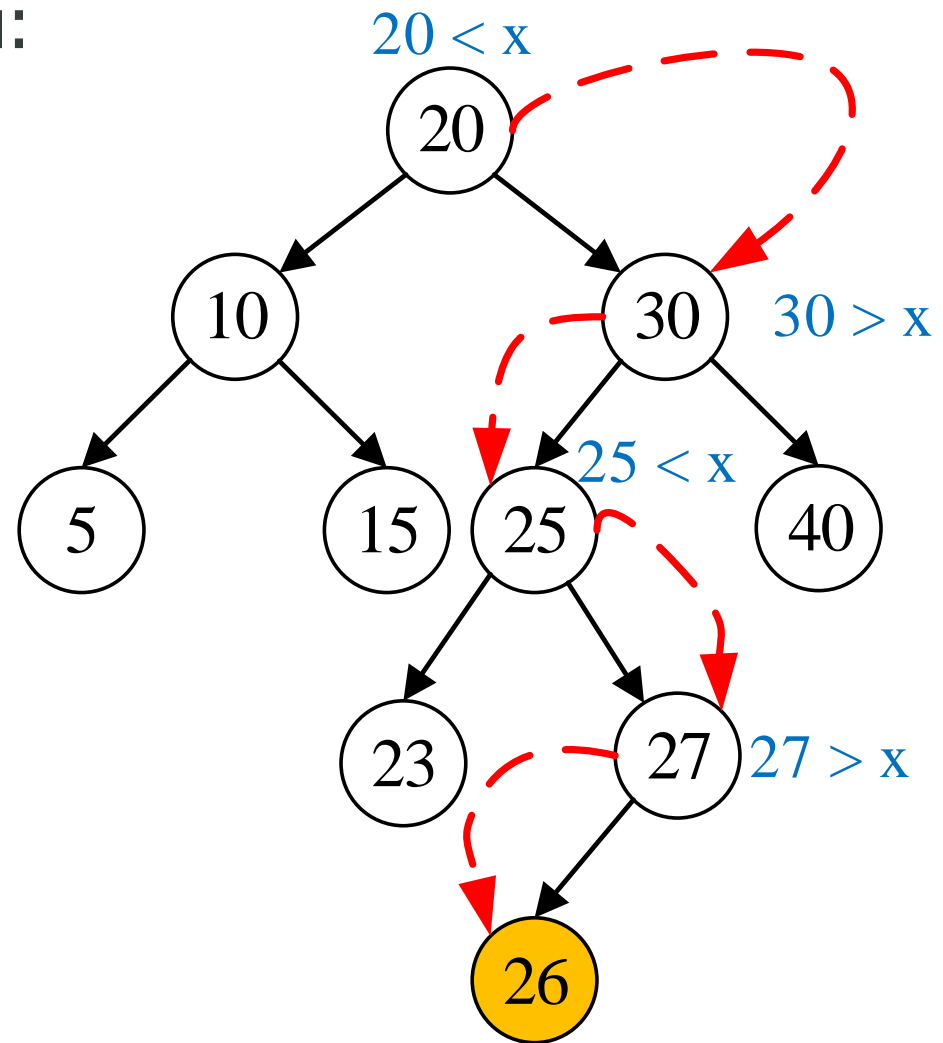
```
TNODE* CreateTNode(int x) {  
    TNODE *p;  
    p = new TNODE; //cấp phát vùng nhớ động  
    if (p == NULL)  
        exit(1); // thoát  
    p->key = x; //gán trường dữ liệu của node = x  
    p->pLeft = NULL;  
    p->pRight = NULL;  
    return p;  
}
```



# Thêm giá trị x vào cây: Minh họa



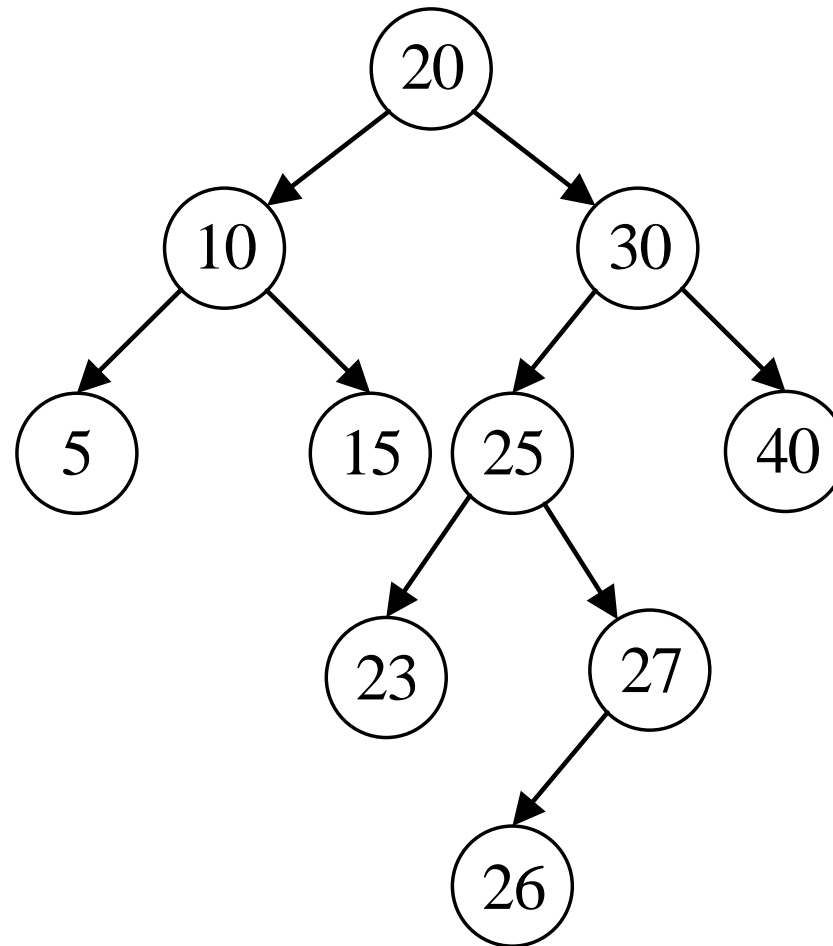
- Thêm  $x=26$  vào cây sau:





# Minh hoạ thành lập 1 cây từ dãy số

- Tạo cây từ dãy sau: 20, 10, 5, 30, 15, 25, 40, 27, 23, 26





## Thêm một node x vào cây

- **Rằng buộc**: Sau khi thêm cây đảm bảo là cây nhị phân tìm kiếm.

```
int Insert(TREE &T, int x) {  
    if (T) {  
        if (T->key == x) return 0;  
        if (T->key > x)  
            return Insert(T->pLeft, x);  
        return Insert(T->pRight, x);  
    }  
    T = CreateTNode(x);  
    return 1;  
}
```



# Thêm một node x vào cây (Không đệ quy)

```
int Insert(TREE &Root, int x) {  
    if (Root==NULL) Root = CreateTNode(x);  
    TREE T=Root;  
    while (T) {  
        if (T->key == x) return 0;  
        if (T->key > x) {  
            if (T->pLeft == NULL)  
                T->pLeft = CreateTNode(x);  
            else T = T->pLeft;  
        }  
        else {  
            if (T->pRight == NULL)  
                T->pRight = CreateTNode(x);  
            else T = T->pRight;  
        }  
    }  
    return 1;  
}
```



# Duyệt cây Nhị phân

- Có 3 trình tự thăm gốc: (3 cách cơ bản để duyệt cây)
  - Duyệt trước (preorder): NLR
  - Duyệt giữa (inorder): LNR
  - Duyệt sau (postorder): LRN



# Duyệt trước – preorder (NLR)

```
void preorder(TREE Root) {  
    if (Root != NULL) {  
        cout << Root->key << "\\t";  
        preorder(Root->pLeft);  
        preorder(Root->pRight);  
    }  
}
```



# Duyệt giữa - inorder (LNR)

```
void inorder(TREE Root) {  
    if (Root != NULL) {  
        inorder(Root->pLeft);  
        cout << T->key << "\\t";  
        inorder(Root->pRight);  
    }  
}
```



## Duyệt sau - postorder (LRN)

```
void postorder(TREE Root) {  
    if (Root != NULL) {  
        postorder(Root->pLeft);  
        postorder(Root->pRight);  
        cout << T->key << "\\t";  
    }  
}
```



# Tìm node có khoá bằng x (không dùng đệ quy)



```
TNODE* searchNode(TREE Root, int x) {  
    TNODE *p = Root;  
    while (p != NULL) {  
        if (x == p->key) return p;  
        if (x < p->key) p = p->pLeft;  
        else p = p->pRight;  
    }  
    return NULL;  
}
```



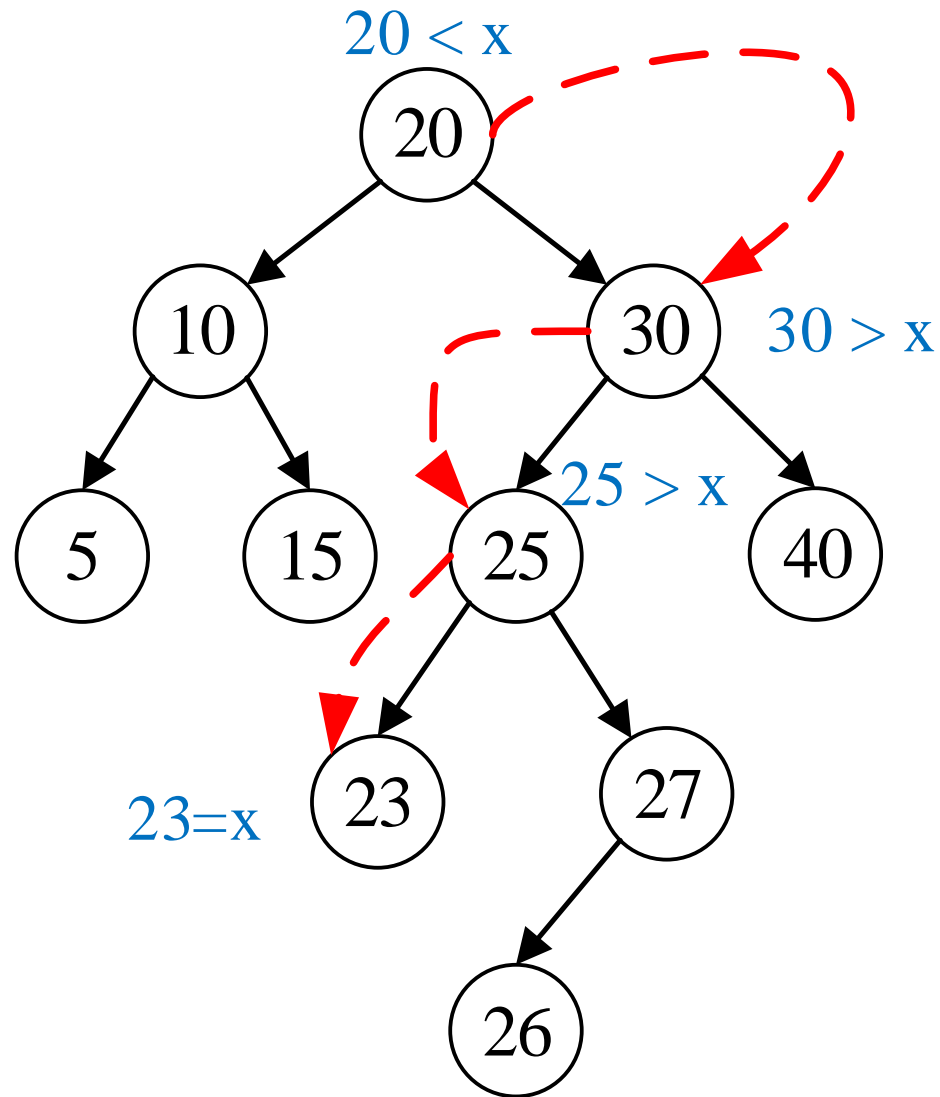
## Tìm node có khoá bằng x (dùng đệ quy)

```
TNODE* searchNode(TREE T, int x) {  
    if (T != NULL) {  
        if (T->key == x)  
            return T;  
        if (T->key > x)  
            return search(T->pLeft, x);  
        return search(T->pRight, x);  
    }  
    return NULL;  
}
```

# Minh hoạ tìm một node



- Tìm  $x=23$





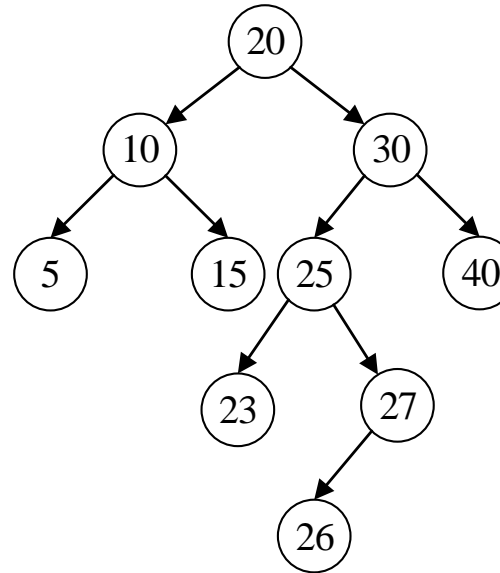
# Hủy 1 node có khoá bằng X trên cây

- Hủy 1 phần tử trên cây phải đảm bảo điều kiện ràng buộc của Cây nhị phân tìm kiếm
- Có 3 trường hợp khi hủy 1 node trên cây
  - TH1: X là node lá
  - TH2: X chỉ có 1 cây con (cây con trái hoặc cây con phải)
  - TH3: X có đầy đủ 2 cây con
- TH1: Ta xoá node lá mà không ảnh hưởng đến các node khác trên cây
- TH2: Trước khi xoá x ta móc nối cha của X với con duy nhất của X.
- TH3: Ta dùng cách xoá gián tiếp

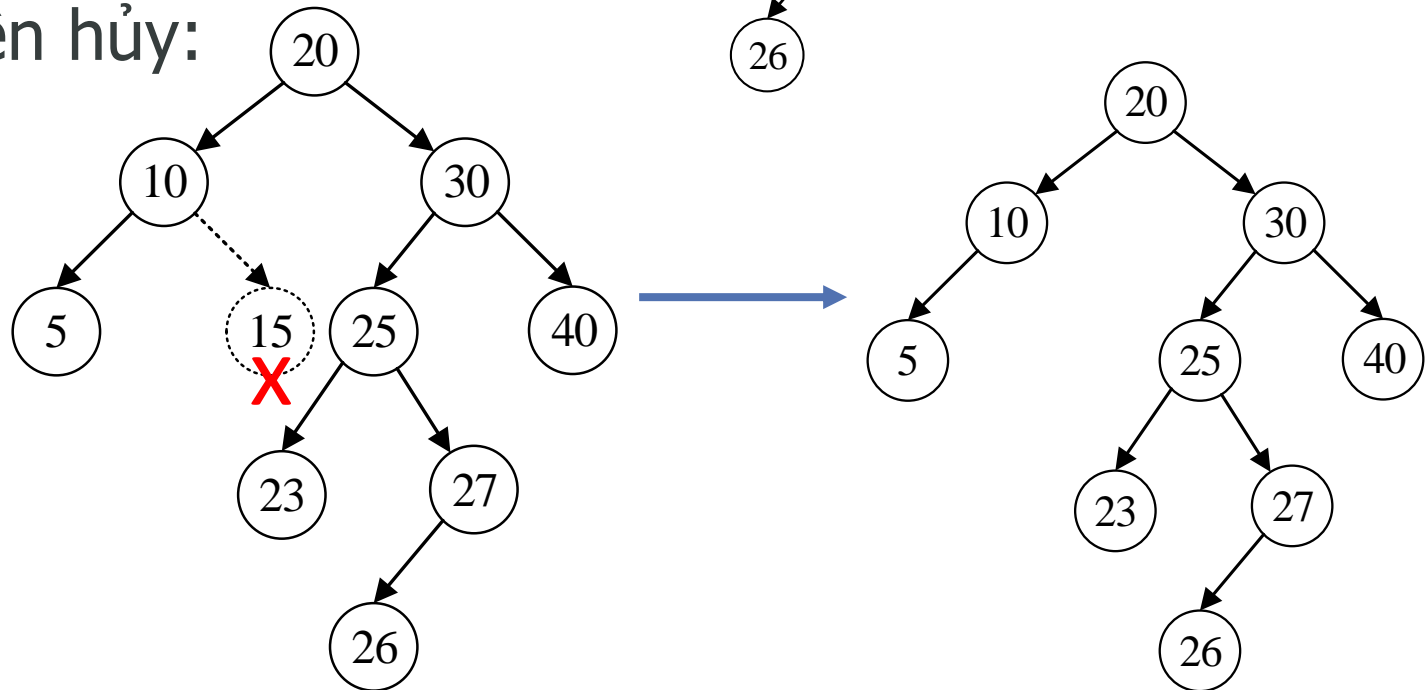


# Minh hoạ hủy phần tử x là node lá

- Hủy  $x=15$  trong cây sau:



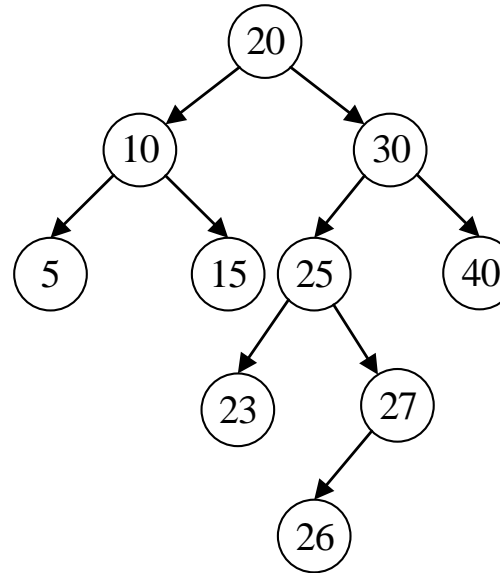
=> Thực hiện hủy:



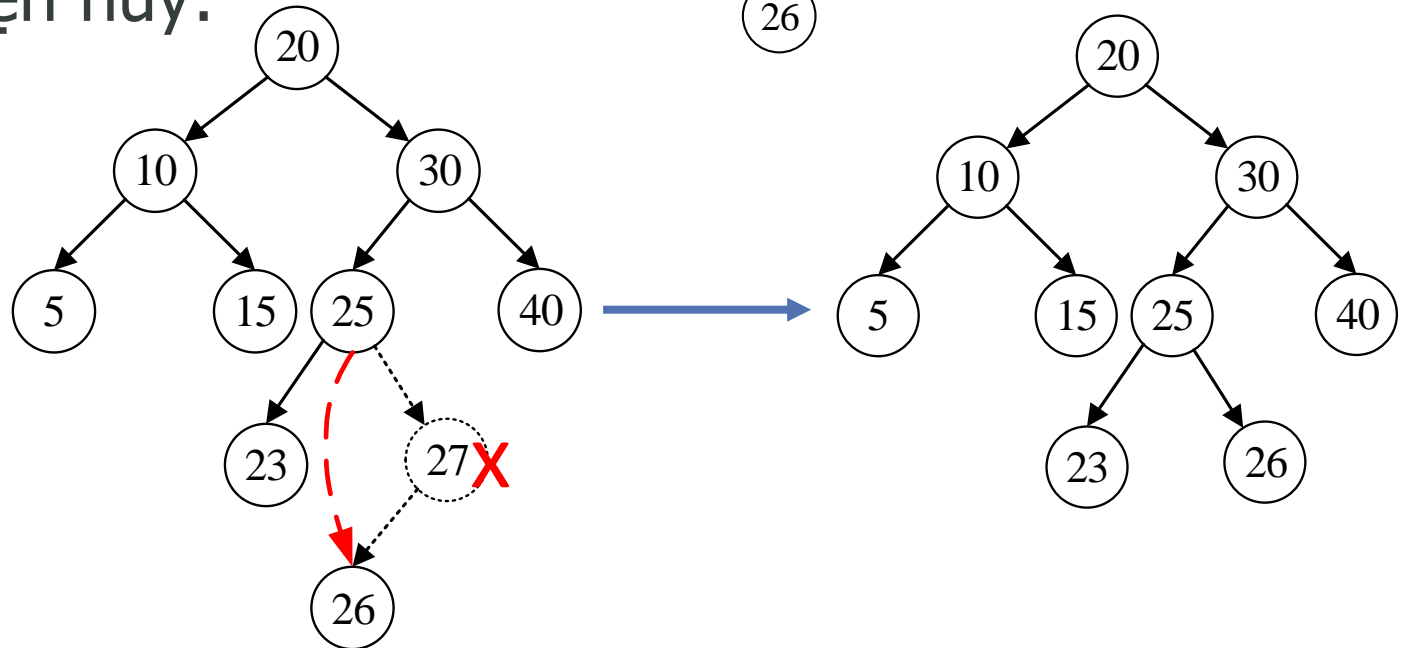


# Minh hoạ hủy phần tử x có 1 cây con

- Hủy  $x=27$  trong cây sau:



=> Thực hiện hủy:





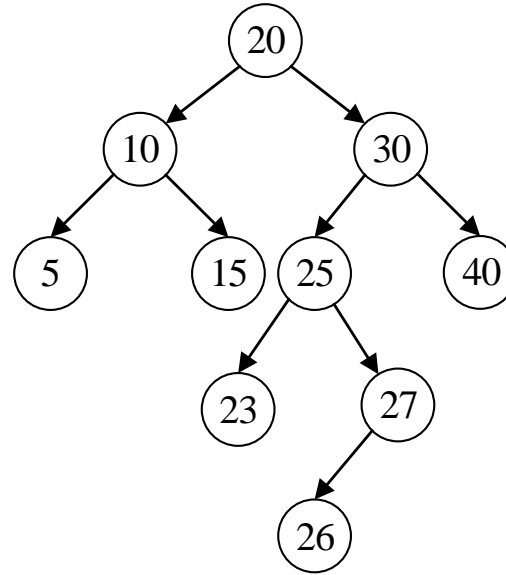
# Hủy 1 node có 2 cây con

- Ta dùng cách hủy gián tiếp, do X có 2 cây con
- Thay vì hủy X ta tìm phần tử thế mạng Y. Nút Y có tối đa 1 cây con.
- Thông tin lưu tại node Y sẽ được chuyển lên lưu tại X.
- Ta tiến hành xoá hủy node Y (xoá Y giống 2 trường hợp đầu)
- Cách tìm node thế mạng Y cho X: Có 2 cách
  - C1: Nút Y là node có khoá nhỏ nhất (trái nhất) bên cây con phải X
  - C2: Nút Y là node có khoá lớn nhất (phải nhất) bên cây con trái của X



# Minh hoạ hủy phần tử x có 2 cây con

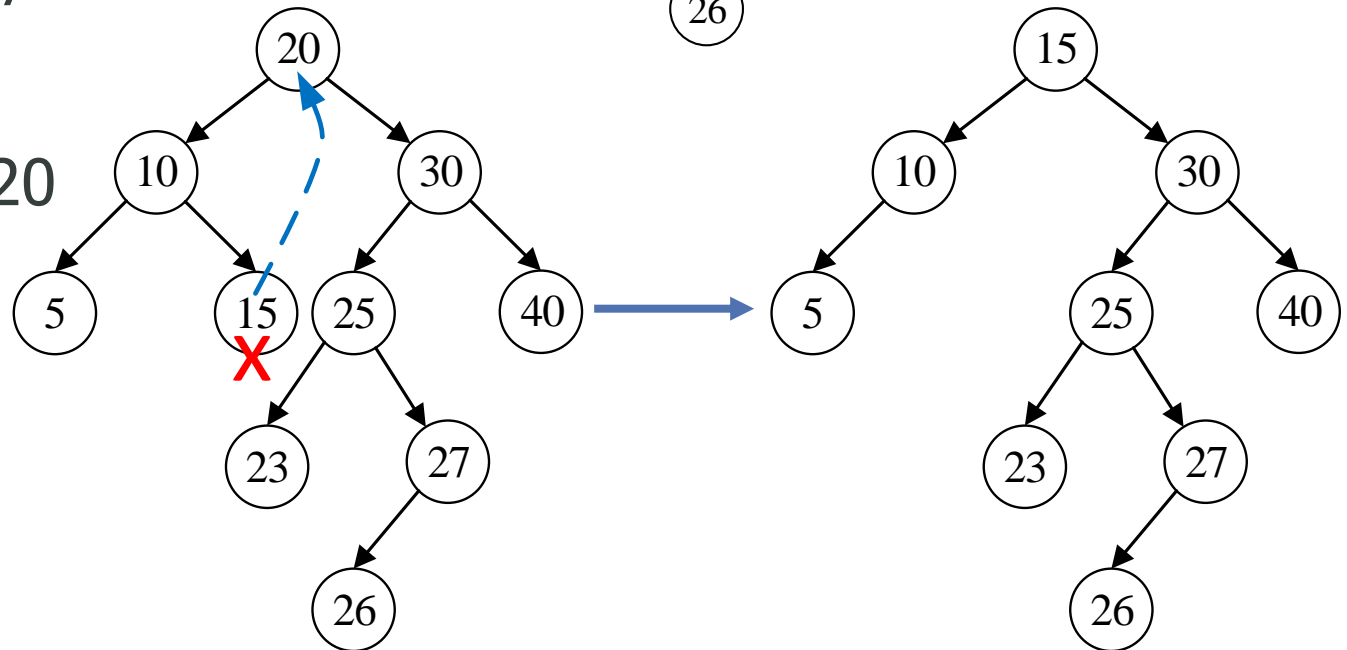
- Hủy  $x=20$  trong cây sau:



⇒ Thực hiện hủy:

Dùng node 15

thay thế node 20

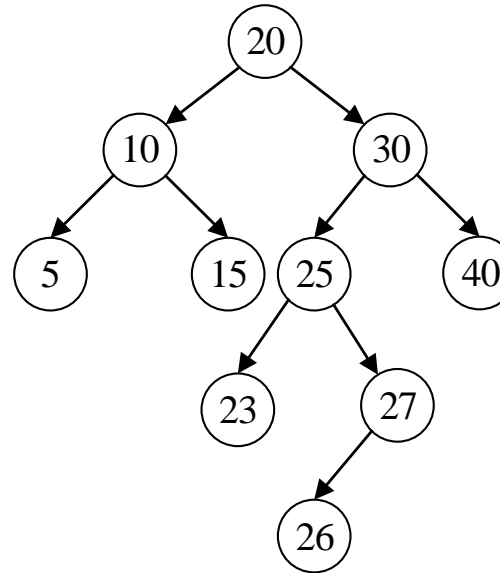




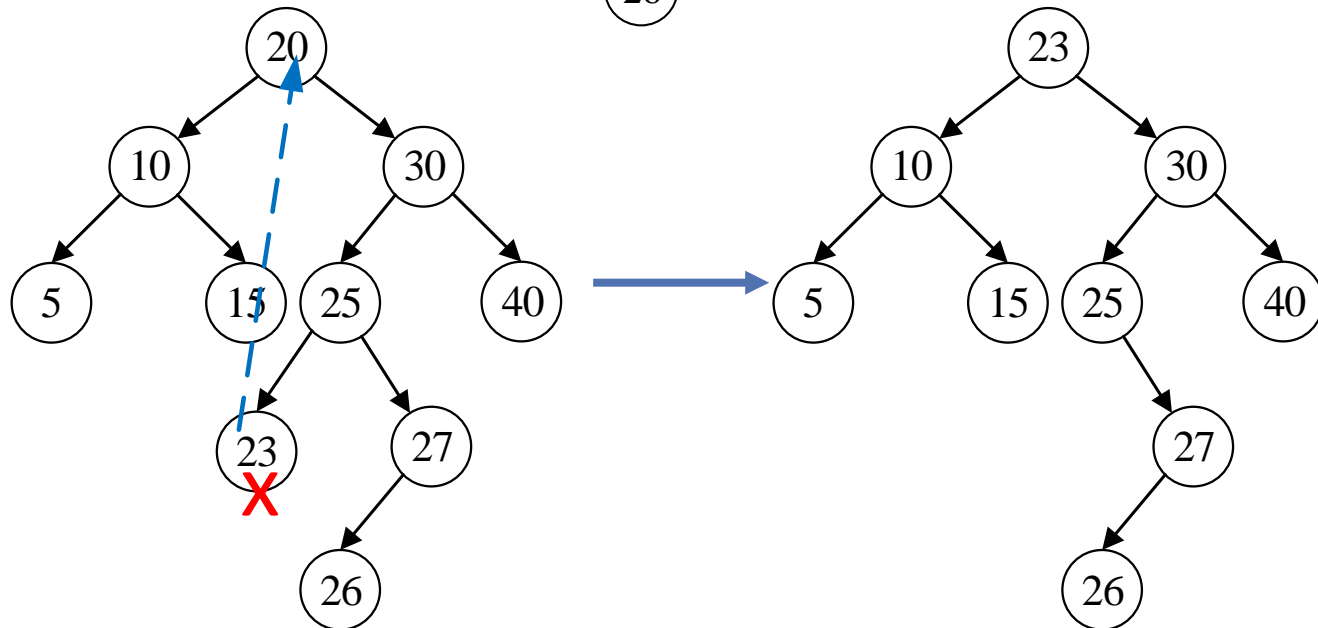


# Minh hoạ hủy phần tử x có 2 cây con

- Hủy  $x=20$  trong cây sau:



⇒ Thực hiện hủy:  
Dùng node 23  
thay thế node 20



# Cài đặt thao tác xóa node có trường key = x



```
void DeleteNodeX(TREE &T, int x) {
    if (T != NULL) {
        if (T->key < x) DeleteNodeX(T->pRight, x);
        else {
            if (T->key > x) DeleteNodeX(T->pLeft, x);
            else { //tim thấy Node có trường dữ liệu = x
                TNODE *p;
                p = T;
                if (T->pLeft == NULL) T = T->pRight;
                else {
                    if (T->pRight == NULL) T = T->pLeft;
                    else ThayThe(p, T->pRight); // tìm bên cây con phải
                }
                delete p;
            }
        }
    }
}
```



# Hàm tìm phần tử thế mạng

```
void ThayThe(TREE &p, TREE &T) {  
    if (T->pLeft != NULL)  
        ThayThe(p, T->pLeft);  
    else {  
        p->key = T->key;  
        p = T;  
        T = T->pRight;  
    }  
}
```



# Tính chiều cao của cây

```
int Height(TNODE* T) {  
    if (!T) return -1;  
    int a = Height(T->pLeft);  
    int b = Height(T->pRight);  
    return (a > b ? a : b) + 1;  
}
```



# Đếm số lượng node có trong cây

// Cách 1: Dùng không đệ quy

```
int DemNode(TREE t) {  
    if (t == NULL)  
        return 0;  
    int a = DemNode(t->pLeft);  
    int b = DemNode(t->pRight);  
    return (a + b + 1);  
}
```

// Cách 2: Dùng đệ quy

```
void DemNode(TREE t, int &count) {  
    if (t == NULL)  
        return;  
    DemNode(t->pRight, count);  
    count++;  
    DemNode(t->pLeft, count);  
}
```





1. Hãy trình bày định nghĩa, đặc điểm và hạn chế của cây nhị phân tìm kiếm.
2. Xét thuật giải tạo cây nhị phân tìm kiếm. Nếu thứ tự các khóa nhập vào như sau:

3 5 2 20 11 30 9 18 4

thì hình ảnh cây tạo được như thế nào?

Sau đó, nếu hủy lần lượt các node 5, 20 thì cây sẽ thay đổi như thế nào trong từng bước hủy, vẽ sơ đồ.



3. Tạo một cây nhị phân tìm kiếm lưu các số nguyên.

Viết các hàm tương ứng sau:

- In cây nhị phân tìm kiếm nói trên theo các thứ tự:

LNR, LRN, NLR, NRL, RNL, RLN.

- Tìm một nút có khoá bằng X trên cây.
- Viết hàm Đếm số nút có trong cây, số nút lá, số nút có đúng 1 cây con, nút có đầy đủ 2 cây con, số nút chẵn, số nút lá chẵn.
- Tính tổng các nút trong cây, tổng các nút có đúng một con, tổng các nút có đúng 1 con mà thông tin tại nút đó là số nguyên tố
- Tính chiều cao của cây
- Xoá 1 nút có khoá bằng X trên cây, nếu không có thì thông báo không có
- Viết hàm kiểm tra 2 cây nhị phân giống nhau





4. Cài đặt chương trình mô phỏng trực quan các thao tác trên cây nhị phân tìm kiếm.
5. Cho một dãy số nguyên (các con số có thể trùng nhau). Hãy viết hàm đếm số lần xuất hiện của từng con số trong dãy.
6. Viết chương trình cho phép tạo, tra cứu và sửa chữa từ điển Anh-Việt.



# Chúc các em học tốt!

