CÂY NHỊ PHÂN TÌM KIẾM

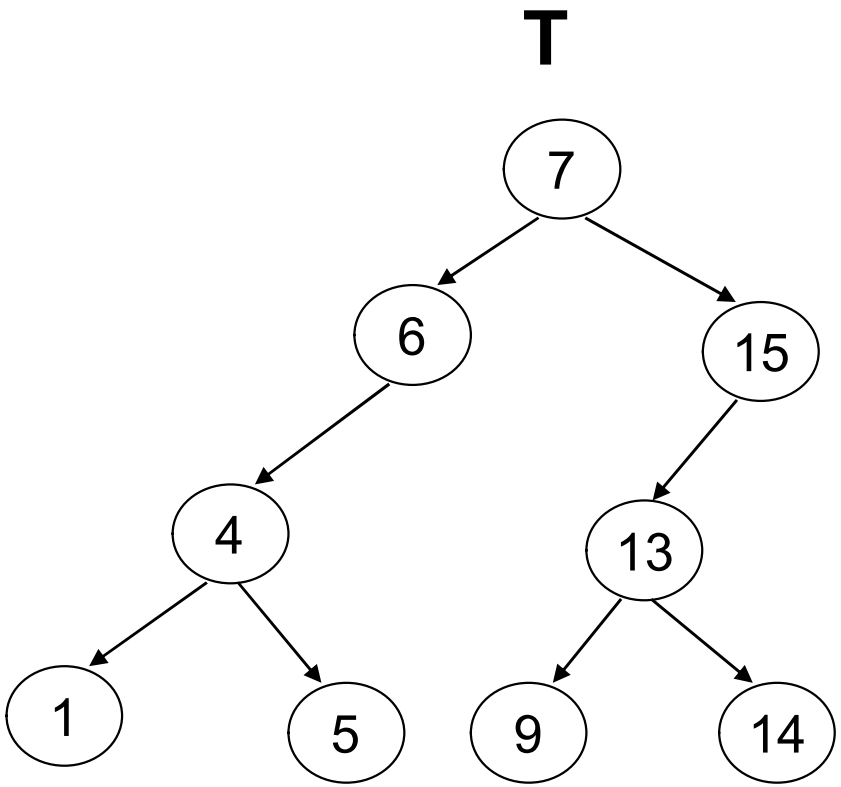
ĐỊNH NGHĨA

Cây nhị phân tìm kiếm (Binary Search Tree - BST): là cây nhị phân thỏa tính chất: tại mỗi nút k, khóa của k lớn hơn khóa của tất cả các nút nằm trên cây con bên trái (nếu có) của k và nhỏ hơn khóa của tất cả các nút nằm trên cây con bên phải (nếu có) của k.

Mỗi nút trên cây nhị phân tìm kiếm đều là gốc của một cây nhị phân tìm kiếm.

CÂY NHỊ PHÂN TÌM KIẾM

- ĐỊNH NGHĨA



CÂY NHỊ PHÂN TÌM KIẾM

❖ ỨNG DỤNG

Cây nhị phân tìm kiếm được sử dụng để tăng hiệu quả tìm kiếm trên cấu trúc liên kết động nhờ vào thứ tự của các nút trên cây.

Chí phí tìm kiếm trên cây nhị phân tìm kiếm trong trường hợp xấu nhất là h, với h là chiều cao của cây.

Cho cây T có n phần tử, chiều cao của cây T là:

- Trong trường hợp tốt nhất h = log2(n)
- Trong trường hợp xấu nhất h = n

CÂY NHỊ PHÂN TÌM KIẾM

❖CÁC THAO TÁC

- Thêm một nút có khóa x
- Tìm nút có giá trị khóa là x
- Xóa nút có khóa x
- Hủy toàn bộ cây

CÂY NHỊ PHÂN TÌM KIẾM

❖CÁC THAO TÁC

- **Thêm một nút**

Thuật toán:

Đầu vào: cây với gốc root, giá trị khóa cần thêm x

Đầu ra: cây với gốc root đã được nút có khóa x

B1: Nếu root = NULL thì p←CreateNode(x), root←p,qua B4

B2: Nếu root->key = x thì qua B4

B3: Nếu root->key < x thì thực hiện thêm nút vào cây có gốc là root->pRight. Ngược lại thực hiện thêm nút vào cây có gốc là root->pLeft

B4: Kết thúc

CÂY NHỊ PHÂN TÌM KIẾM

❖CÁC THAO TÁC

- **Thêm một nút**

```
int Compare(TenDulieu x, TenDulieu y); // trả về 0 nếu x=y,
    -1 nếu x < y và 1 nếu x > y
int AddNode(TREE &root, TenDulieu x) {
    Node * p;
    if (root == NULL) {
    p = CreateNode(x);
    if (p == NULL) return -1;
    root = p; return 1;
    }
```

CÂY NHỊ PHÂN TÌM KIẾM

```
    if (Compare(root->key, x) == 0) return 0;
    if (Compare(root->key, x) == 1)
        return AddNode(root->pLeft, x);
    else
        return AddNode(root->pRight, x);
}
```

CÂY NHỊ PHÂN TÌM KIẾM

❖CÁC THAO TÁC

- Ví dụ: Viết chương trình nhập vào một dãy số nguyên và tạo cây nhị phân tìm kiếm từ dãy số nguyên đó theo thứ tự nhập, cho biết quá trình tạo cây nhị phân với dãy số nguyên

4 7 5 9 8 1 3 2 6 5.

CÂY NHỊ PHÂN TÌM KIẾM

```
struct Node {
    int key;
    Node *pLeft, *pRight;
};
typedef Node *TREE;

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

CÂY NHỊ PHÂN TÌM KIẾM

```
Node * CreateNode(int x) {
    Node *p = new Node;
    if (p != NULL)  {
        p->key = x; p->pLeft = NULL; p->pRight = NULL;
    }
    return p;
}
```

CÂY NHỊ PHÂN TÌM KIẾM

```
int AddNode(TREE &root, int x) {
    Node *p;
    if (root == NULL) {
        p = CreateNode(x);
        if (p == NULL) return -1;
        root = p; return 1;
    }
    if (root->key == x) return 0;
    if (root->key > x) return AddNode(root->pLeft, x);
    else return AddNode(root->pRight, x);
}
```

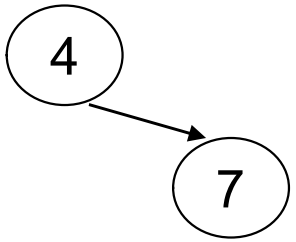
CÂY NHỊ PHÂN TÌM KIẾM

```
int main() {
    int n, i, x;
    TREE root;
    CreateTree(root);
    cout << "Kich thuoc day so nguyen "; cin >> n;
    for (i = 0; i < n; i++) {
        cin >> x;
        AddNode(root, x);
    }
    return 0;
}
```

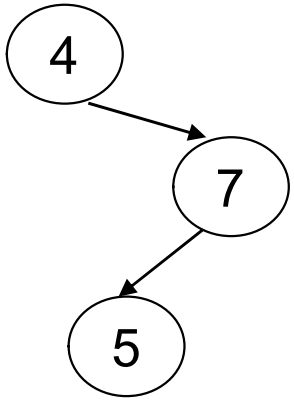
CÂY NHỊ PHÂN TÌM KIẾM

4 7 5 9 8 1 3 2 6 5

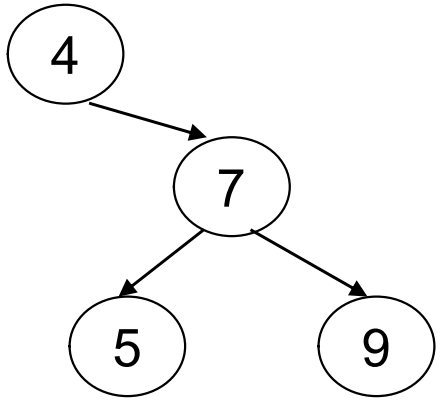
7 5 9 8 1 3 2 6 5



5 9 8 1 3 2 6 5

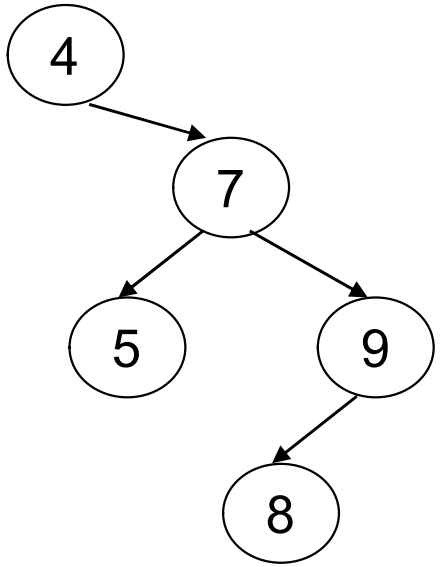


9 8 1 3 2 6 5

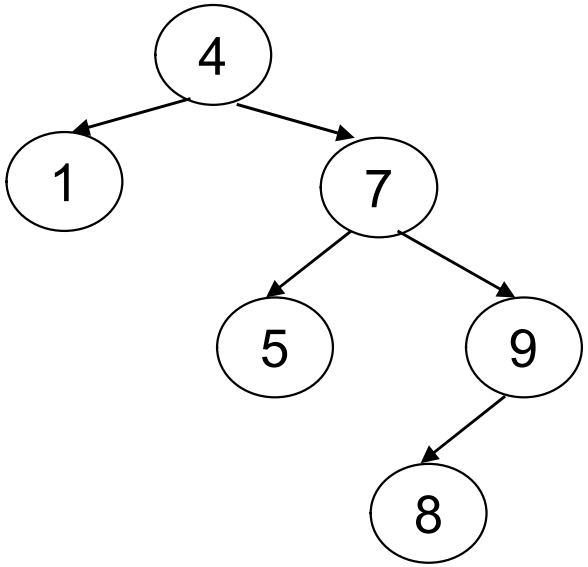


CÂY NHỊ PHÂN TÌM KIẾM

8 1 3 2 6 5

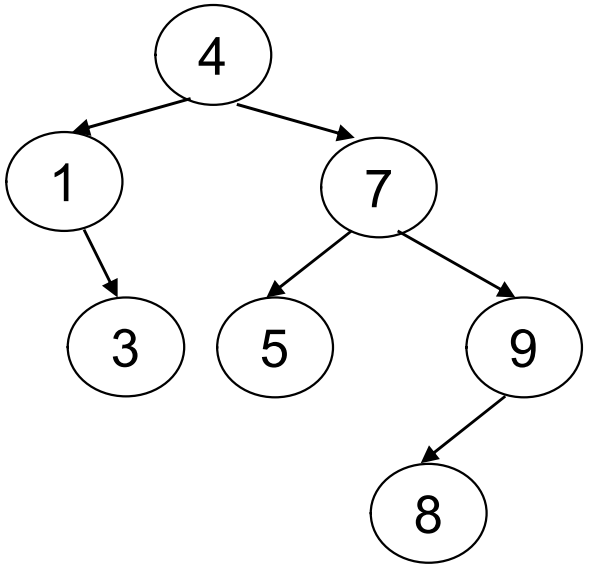


1 3 2 6 5

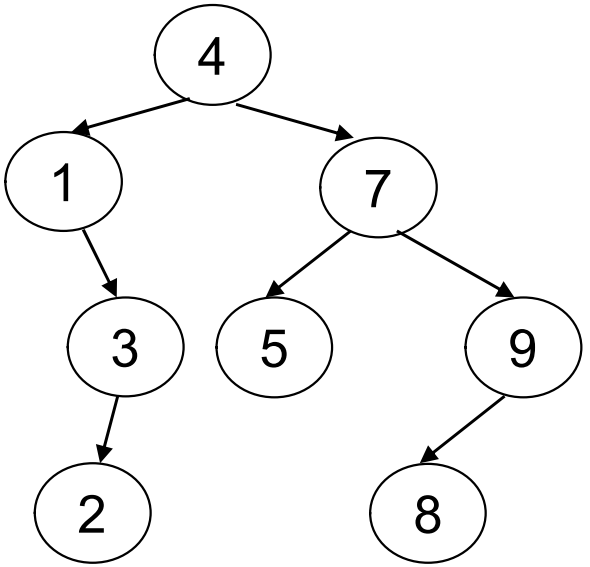


CÂY NHỊ PHÂN TÌM KIẾM

3 2 6 5

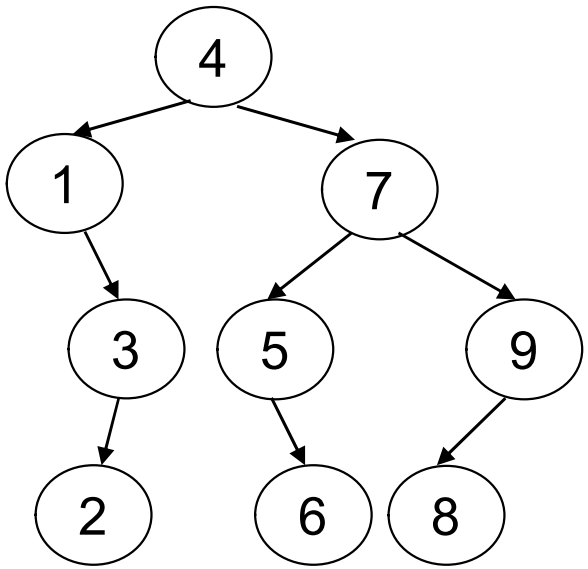


2 6 5

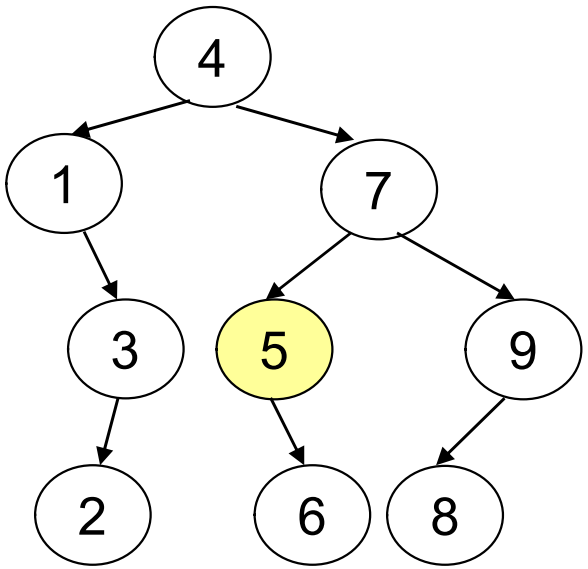


CÂY NHỊ PHÂN TÌM KIẾM

6 5



5



CÂY NHỊ PHÂN TÌM KIẾM

❖CÁC THAO TÁC

- **Tìm nút có giá trị khóa là x**

Thuật toán:

Đầu vào: cây với gốc root, giá trị khóa cần tìm x

Đầu ra: nút có khóa x

B1: Nếu root = NULL thì trả về NULL, qua B5

B2: Nếu root->key = x thì trả về root, qua B5

B3: Nếu root->key < x thì thực hiện tìm nút p có khóa x trên cây có gốc là root->pRight. Ngược lại thực hiện tìm nút p có khóa x trên cây có gốc là root->pLeft.

B4: Trả về p

B5: Kết thúc.

CÂY NHỊ PHÂN TÌM KIẾM

❖CÁC THAO TÁC

- **Tìm nút có giá trị khóa là x**

```
Node * Search(TREE root, TenDulieu x) {
    Node *p;
    if (root == NULL) return NULL;
    if (Compare(root->key, x) == 0) return root;
    if (Compare(root->key, x) > 0)
    p = Search(root->pLeft, x);
    else
    p = Search(root->pRight, x);
    return p;
}
```

CÂY NHỊ PHÂN TÌM KIẾM

❖CÁC THAO TÁC

- **Xóa nút có khóa x**

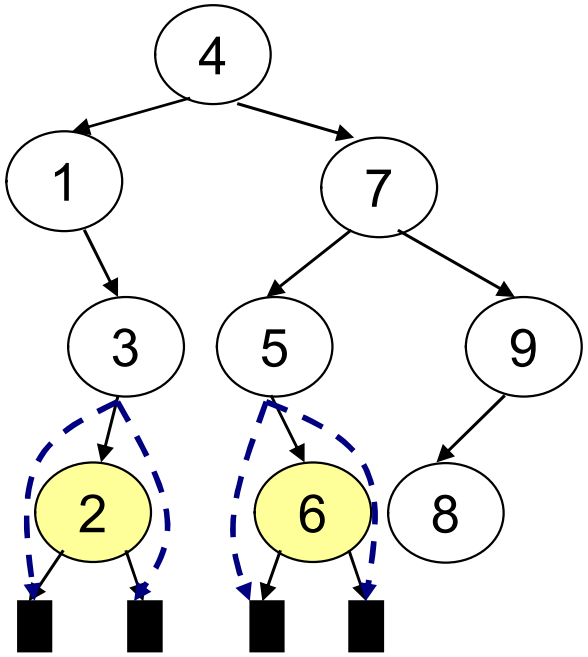
Có 3 trường hợp cần xử lý:

- TH1: Nút cần xóa không có cây con
- TH2: Nút cần xóa có 1 cây con
- TH3: Nút cần xóa có 2 cây con

CÂY NHỊ PHÂN TÌM KIẾM

❖CÁC THAO TÁC

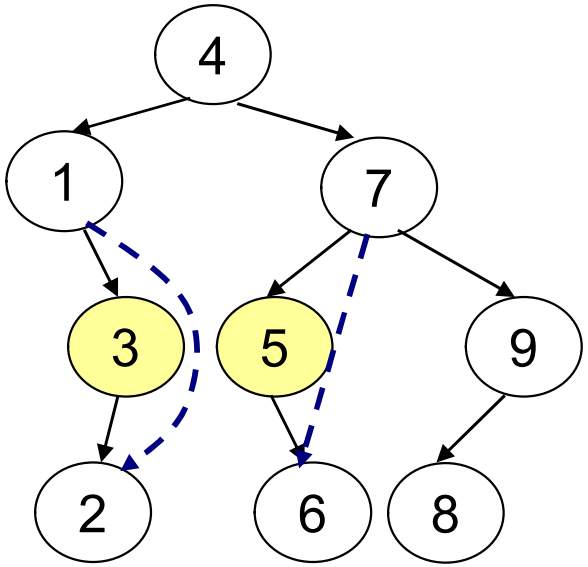
- **Xóa nút có khóa x: TH1**
- Gán lại địa chỉ trỏ đến p của nút cha là NULL
- Giải phóng vùng nhớ tại nút p



CÂY NHỊ PHÂN TÌM KIẾM

❖CÁC THAO TÁC

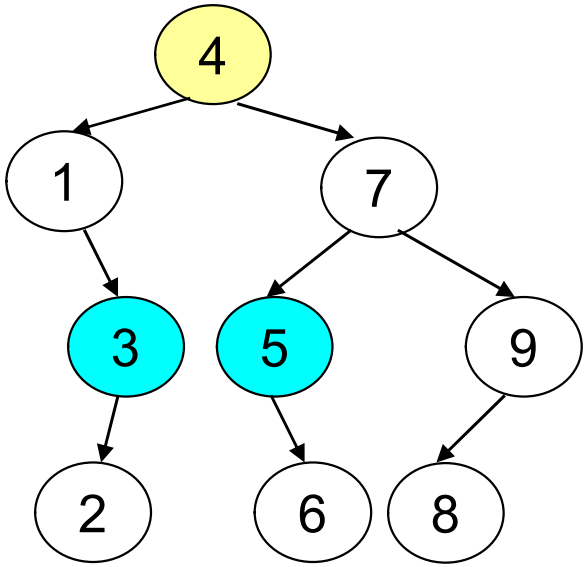
- **Xóa nút có khóa x: TH2**
- Gán lại địa chỉ trỏ đến p của nút cha là nút con duy nhất của p
- Giải phóng vùng nhớ tại nút cần xóa p



CÂY NHỊ PHÂN TÌM KIẾM

❖CÁC THAO TÁC

- **Xóa nút có khóa x: TH3**
- Tìm phần tử thay thế q (standby) cho p. q là phần tử trái nhất của cây con bên phải hoặc là phần tử phải nhất của cây con bên trái của p
- Gán p->key ← q->key
- Xóa q



CÂY NHỊ PHÂN TÌM KIẾM

❖CÁC THAO TÁC

- **Xóa nút có khóa x**

Đầu vào: cây với nút gốc root, giá trị khóa x cần xóa

Đầu ra: cây với nút gốc root đã xóa phần tử có khóa x

B1: Nếu root = NULL thì qua B8.

B2: Nếu root->key < x thì thực hiện xóa nút có khóa x cho cây root->pRight, qua B8

B3: Nếu root->key > x thì thực hiện xóa nút có khóa x cho cây root->pLeft, qua B8

B4: q = root

B5: Nếu root->pLeft = NULL thì root = root->pRight, qua B7

CÂY NHỊ PHÂN TÌM KIẾM

B6: Nếu root->pRight = NULL thì root = root->pLeft,

Ngược lại tìm phần tử thay thế q cho root trên cây con root->pRight

B7: Giải phóng q.

B8: Kết thúc.

CÂY NHỊ PHÂN TÌM KIẾM

❖CÁC THAO TÁC

- **Xóa nút có khóa x**

```
void SearchStandFor(TREE &p, TREE &q);
void DeleteNode(TREE &root, TenDulieu x) {
    Node *q;
    if (root == NULL) return;
    if (Compare(root->key, x) < 0)
    { DeleteNode(root->pRight, x); return; }
    if (Compare(root->key, x) > 0)
    { DeleteNode(root->pLeft, x); return; }
```

CÂY NHỊ PHÂN TÌM KIẾM

```
    q = root;
    if (root->pLeft == NULL) root = root->pRight;
    else
        if (root->pRight == NULL) root = root->pLeft;
        else SearchStandFor(q, root->pRight);
    delete q;
}
```

CÂY NHỊ PHÂN TÌM KIẾM

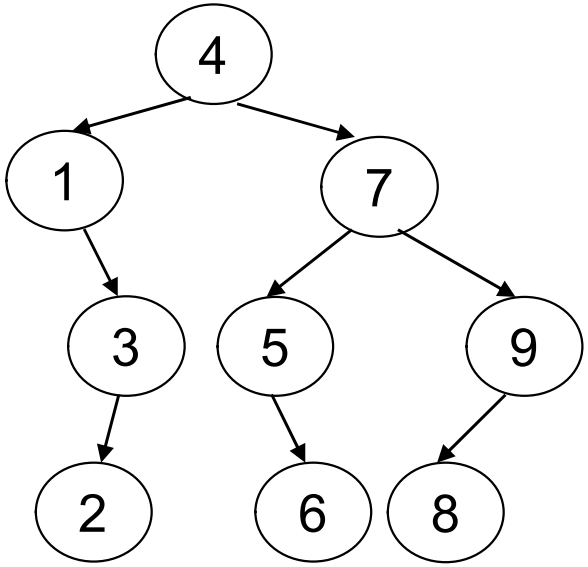
```
void SearchStandFor(TREE &q, TREE &p) {
    if (p->pLeft != NULL)
        SearchStandFor(q, p->pLeft);
    else {
        q->key = p->key;
        q = p;
        p = p->pRight;
    }
}
```

CÂY NHỊ PHÂN TÌM KIẾM

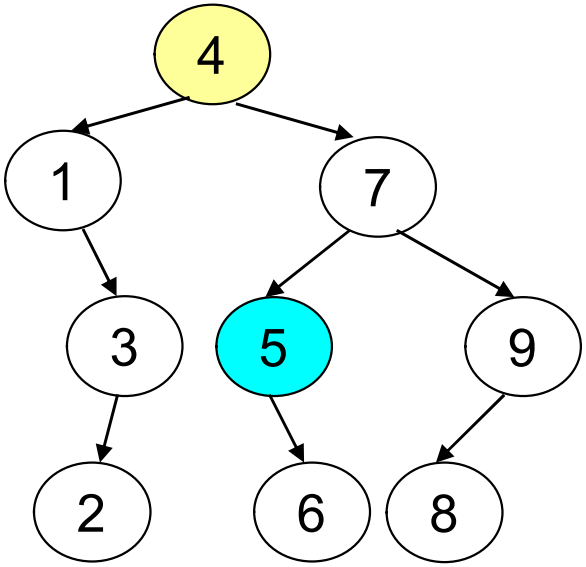
❖CÁC THAO TÁC

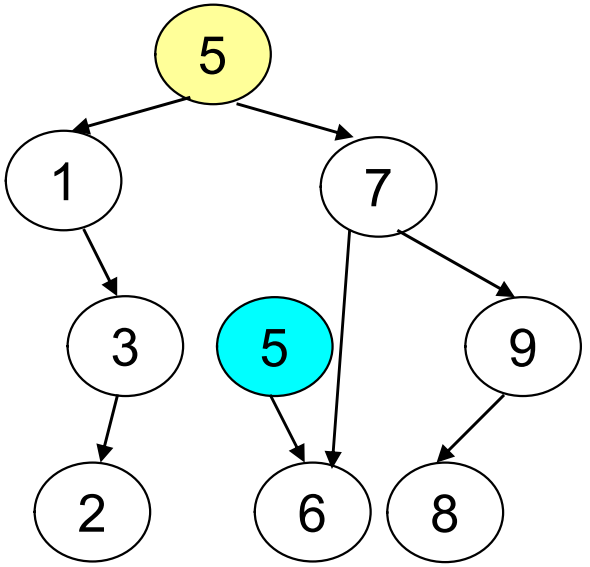
- **Xóa nút có khóa x**

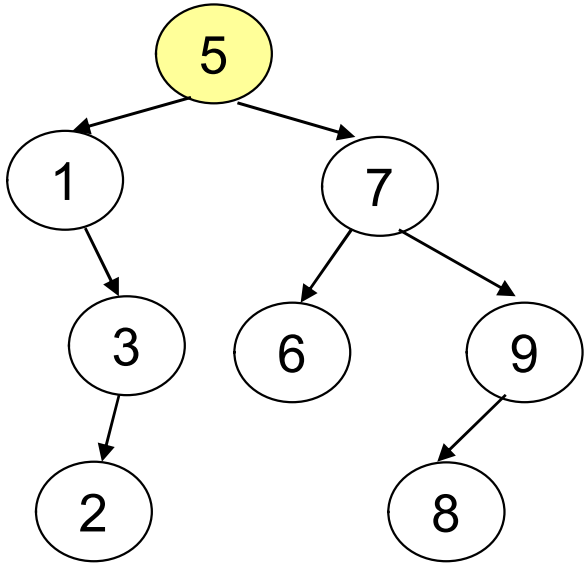
Ví dụ: xóa nút có khóa là 4



CÂY NHỊ PHÂN TÌM KIẾM







CÂY NHỊ PHÂN TÌM KIẾM

❖CÁC THAO TÁC

- **Hủy toàn bộ cây**

Đầu vào: cây có nút gốc root

Đầu ra: cây đã được hủy có nút gốc root

B1: nếu root == NULL, qua B5.

B2: Hủy toàn bộ cây có gốc là root->pLeft.

B3: Hủy toàn bộ cây có gốc là root->pRight

B4: Giải phóng vùng nhớ của root, root = NULL.

B5: Kết thúc

CÂY NHỊ PHÂN TÌM KIẾM

❖CÁC THAO TÁC

- **Hủy toàn bộ cây**

```
void DeleteTree(TREE &root) {
    if (root == NULL) return;
    DeleteTree(root->pLeft);
    DeleteTree(root->pRight);
    delete root; root = NULL;
}
```