CÂY NHỊ PHÂN TÌM KIẾM (BINARY SEARCH TREE)

DATA STRUCTURES AND ALGORITHMS

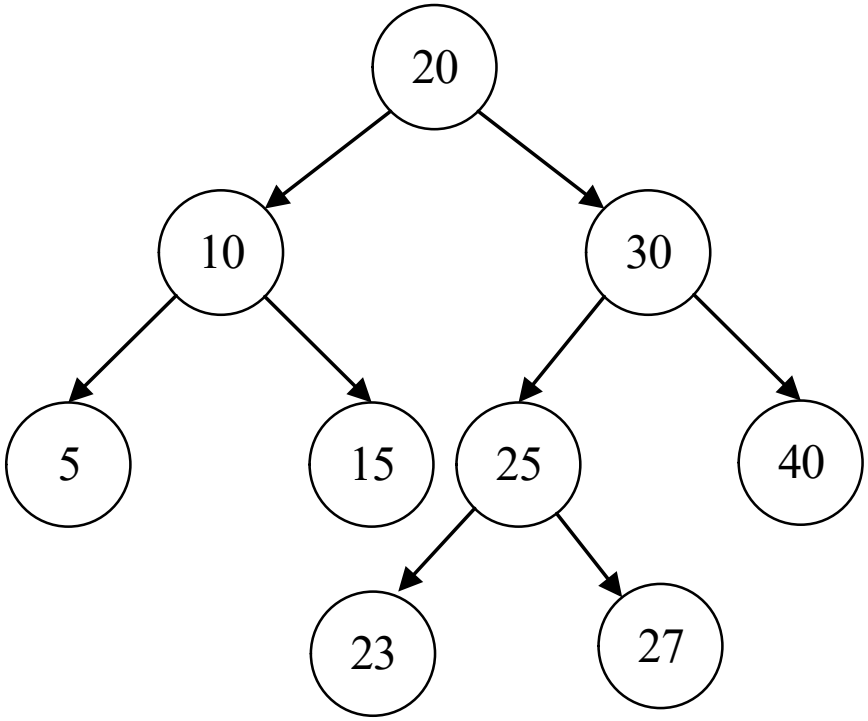
Nội dung

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

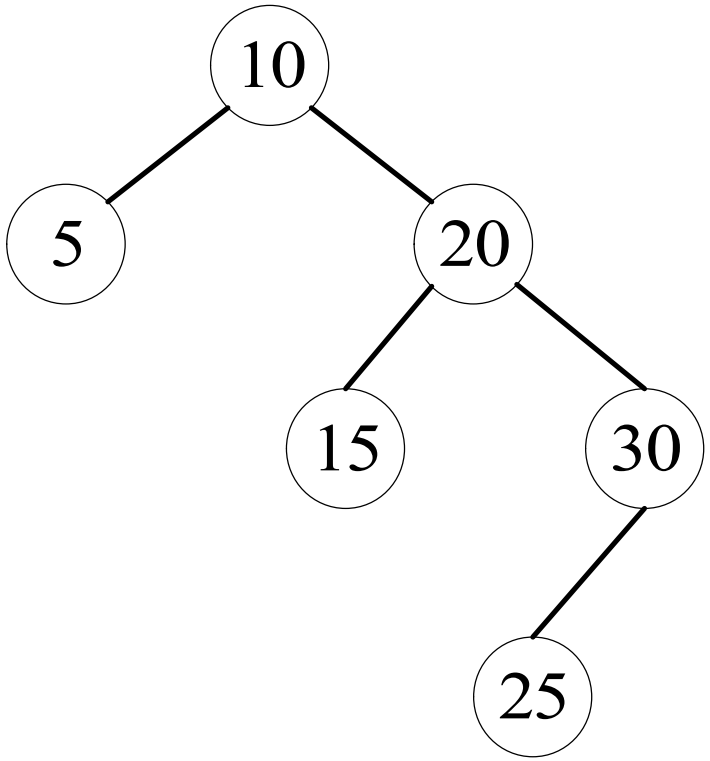
Định nghĩa cây nhị phân tìm kiếm

Cây phị phân tìm kiếm:

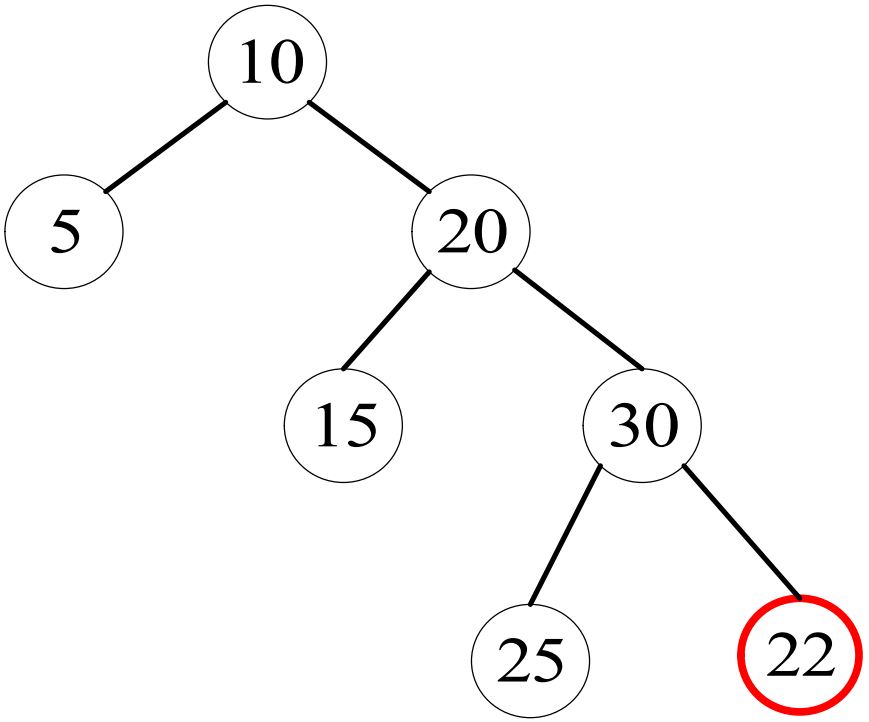
- Là cây nhị phân
- Bảo đảm nguyên tắc bố trí khoá tại mỗi node:
  - Các node trong cây trái nhỏ hơn node hiện hành
  - Các node trong cây phải lớn hơn node hiện hành
- Ví dụ:



Định nghĩa cây nhị phân tìm kiếm



- Là cây nhị phân tìm kiếm



- Không là cây nhị phân tìm kiếm

Ưu điểm của cây nhị phân tìm kiếm

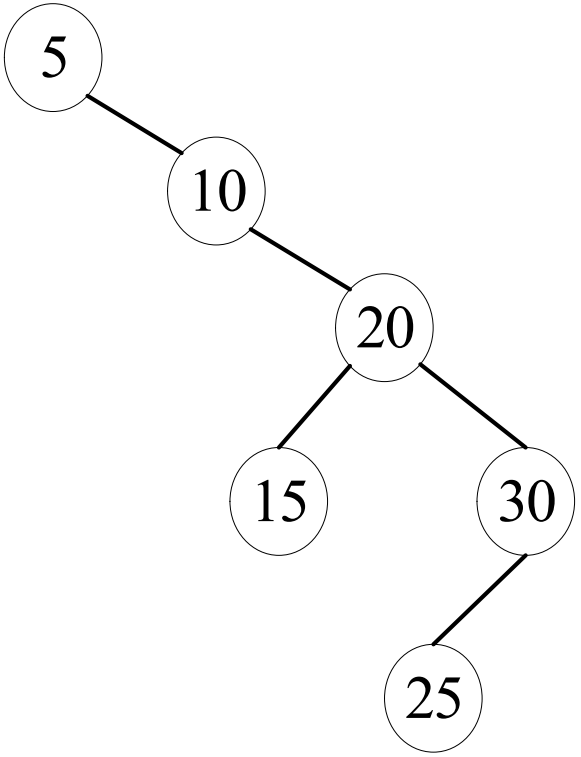
- Nhờ trật tự bố trí khóa trên cây :
  - Định hướng được khi tìm kiếm

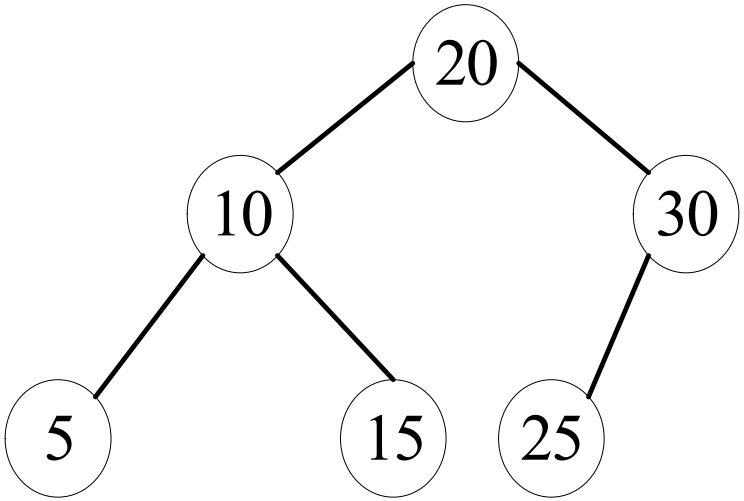
Cây gồm N phần tử :

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

Ưu điểm của cây nhị phân tìm kiếm

- Cho 2 cây nhị phân tìm kiếm như ví dụ bên dưới: gồm 6 phần tử như nhau, nhưng được bố trí khác nhau, nên chiều cao của cây cũng khác nhau.





Cấu trúc dữ liệu của cây nhị phân tìm kiếm

- *Cấu trúc dữ liệu của 1 node*

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

- *Cấu trúc dữ liệu của cây*

```
typedef TNODE* TREE;
```

Các thao tác trên cây nhị phân tìm kiếm

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

Tìm kiếm x trên cây nhị phân tìm kiếm

- Phép tìm kiếm bắt đầu từ node gốc
- Nếu x bằng khóa của gốc thì kết luận tìm thấy
- Nếu x nhỏ hơn khóa ở gốc, ta phải **tìm x bên cây con trái**. Nếu x lớn hơn khóa ở gốc, ta phải **tìm x bên cây con phải**.
- Nếu cây con (trái hoặc phải) là rỗng thì kết luận không tìm thấy x.

- Time complexity: O(h) với h là chiều cao của cây

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

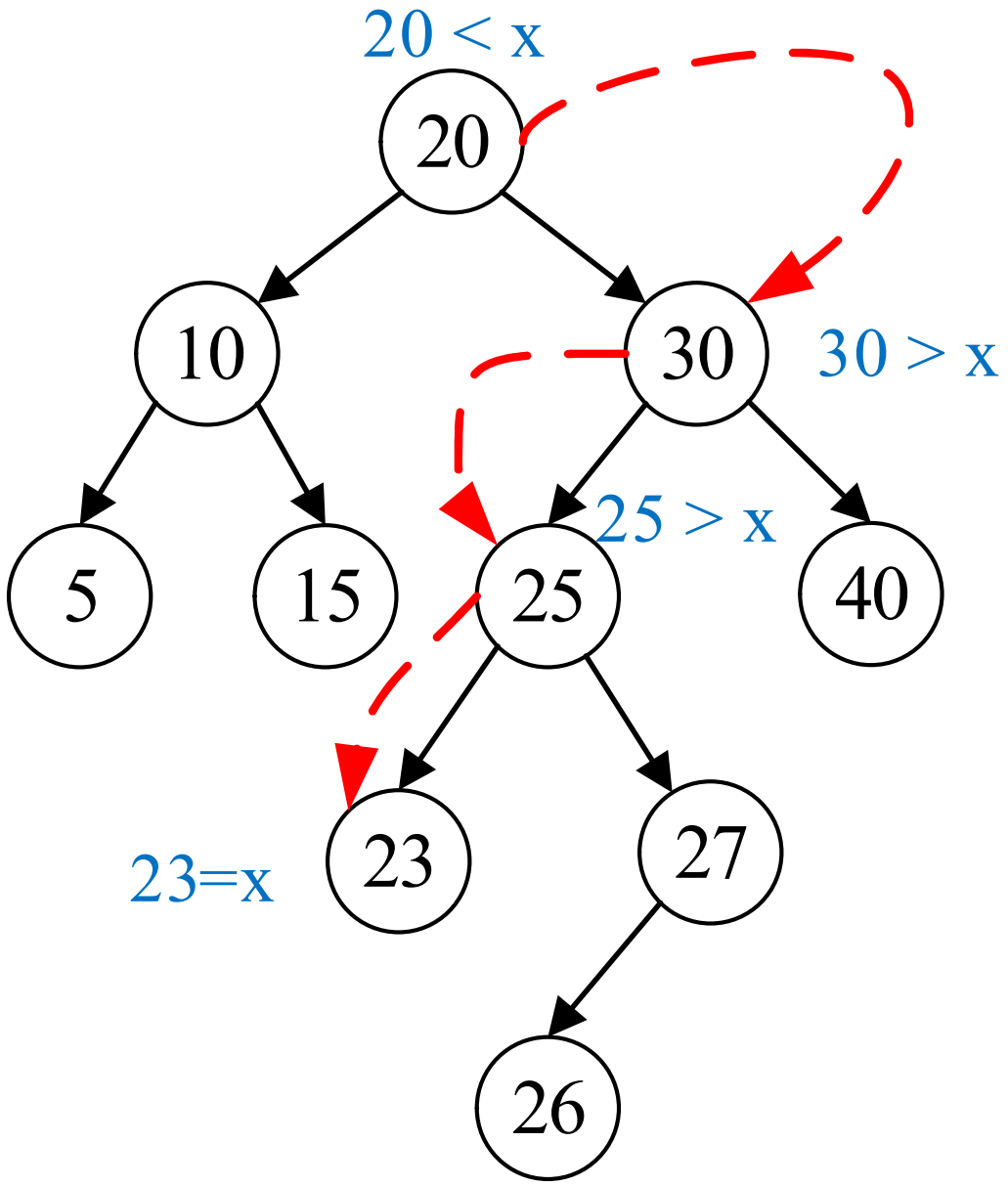
```
TNODE* searchNode(TREE Root, int x) {
    TNODE *p = Root;
    while (p != NULL) {
        if (x == p->key) return p;
        if (x < p->key) p = p->pLeft;
        else p = p->pRight;
    }
    return NULL;
}
```

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



Tạo cây rỗng

- Cây rỗng -> địa chỉ node gốc bằng NULL

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

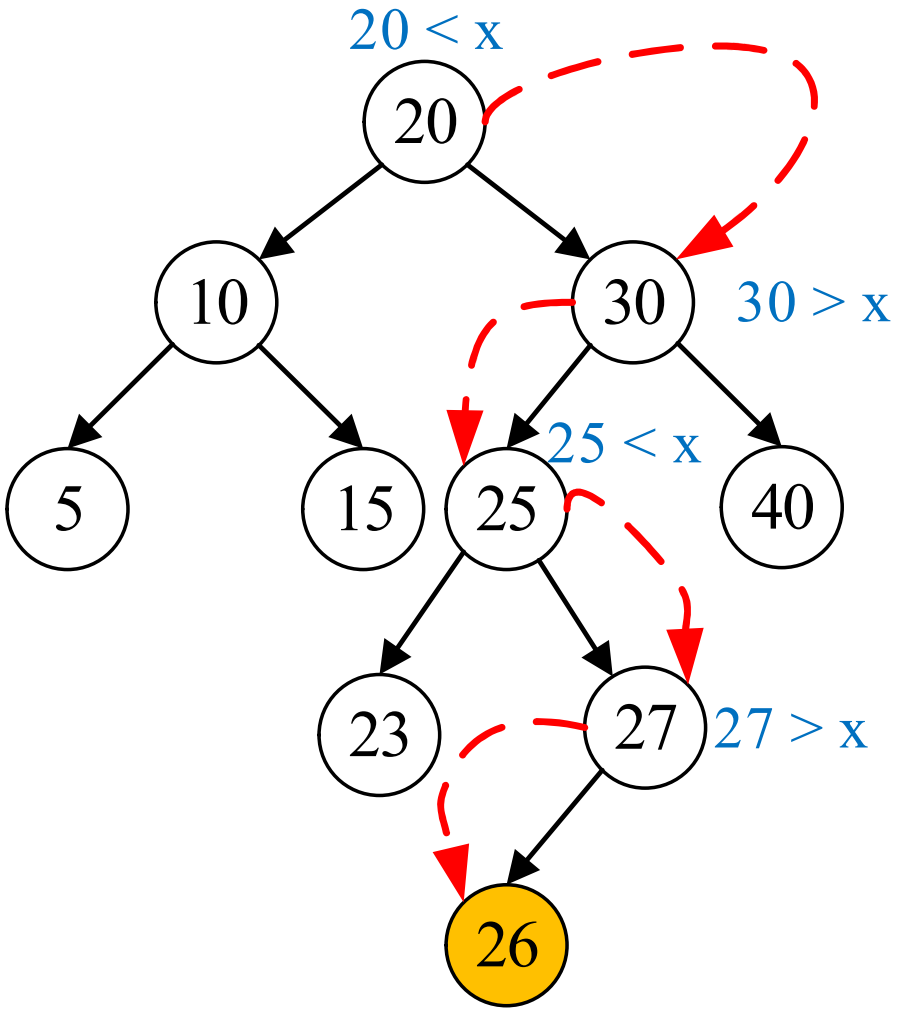
Thêm giá trị x vào cây

- Bắt đầu từ gốc của cây
- Nếu khóa của gốc bằng x thì không làm gì hết (hoặc tùy yêu cầu của bài toán mà có xử lý phù hợp)
- Ngược lại thì **tìm x trong cây con trái hoặc phải**. Nếu con trái hoặc con phải là rỗng (không tìm thấy) thì thêm 1 node có giá trị x cần chèn.

- Time complexity: O(h) với h là chiều cao của cây

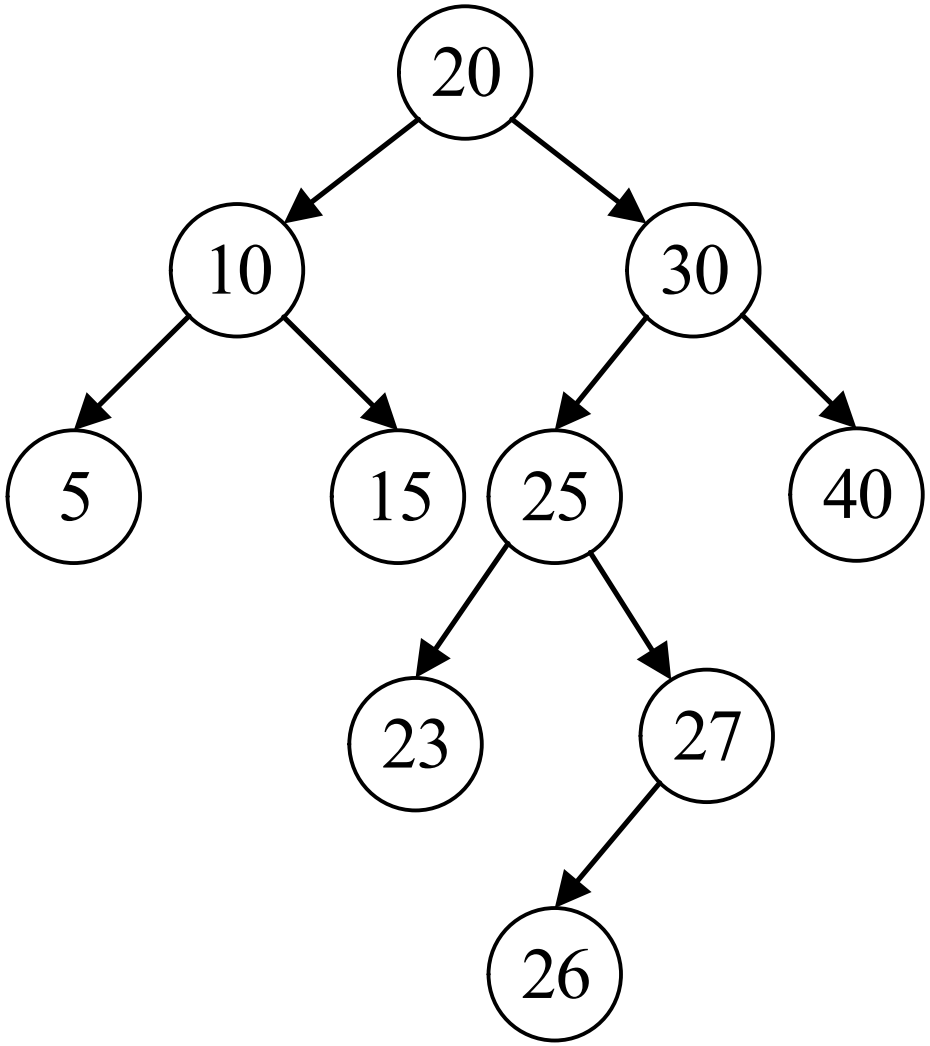
Thêm giá trị x vào cây: Minh họa

- Minh họa thêm x=26:



Minh hoạ thành lập 1 cây từ dãy số

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



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 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

- Depth First Traversals: có 3 cách cơ bản để duyệt cây
  - Duyệt trước (preorder): NLR
  - Duyệt giữa (inorder): LNR
  - Duyệt sau (postorder): LRN
- Breadth First or Level Order Traversal

Time Complexity: O(n)

Duyệt giữa - inorder (LNR)

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

Tìm giá trị Min, Max

Tìm node chứa giá trị min: (Tương tự cho tìm max)

- Trả về node chứa giá trị nhỏ nhất trên cây.
- Bắt đầu từ gốc và đi qua bên trái chừng nào node đang xét vẫn còn con trái. Điểm dừng chính là giá trị cần tìm.

```
NODE* finMin(TREE &T) {
    if (T==NULL) return NULL;
    if (T->pLeft == NULL)
        return T;
    return findMin (T->pLeft);
}
```

- **Time Complexity:** O(h) với h là chiều cao của cây

Hủy 1 node có khóa 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
  - Trường hợp 1: x là node lá
  - Trường hợp 2: x chỉ có 1 cây con (cây con trái hoặc cây con phải)
  - Trường hợp 3: x có đầy đủ 2 cây con

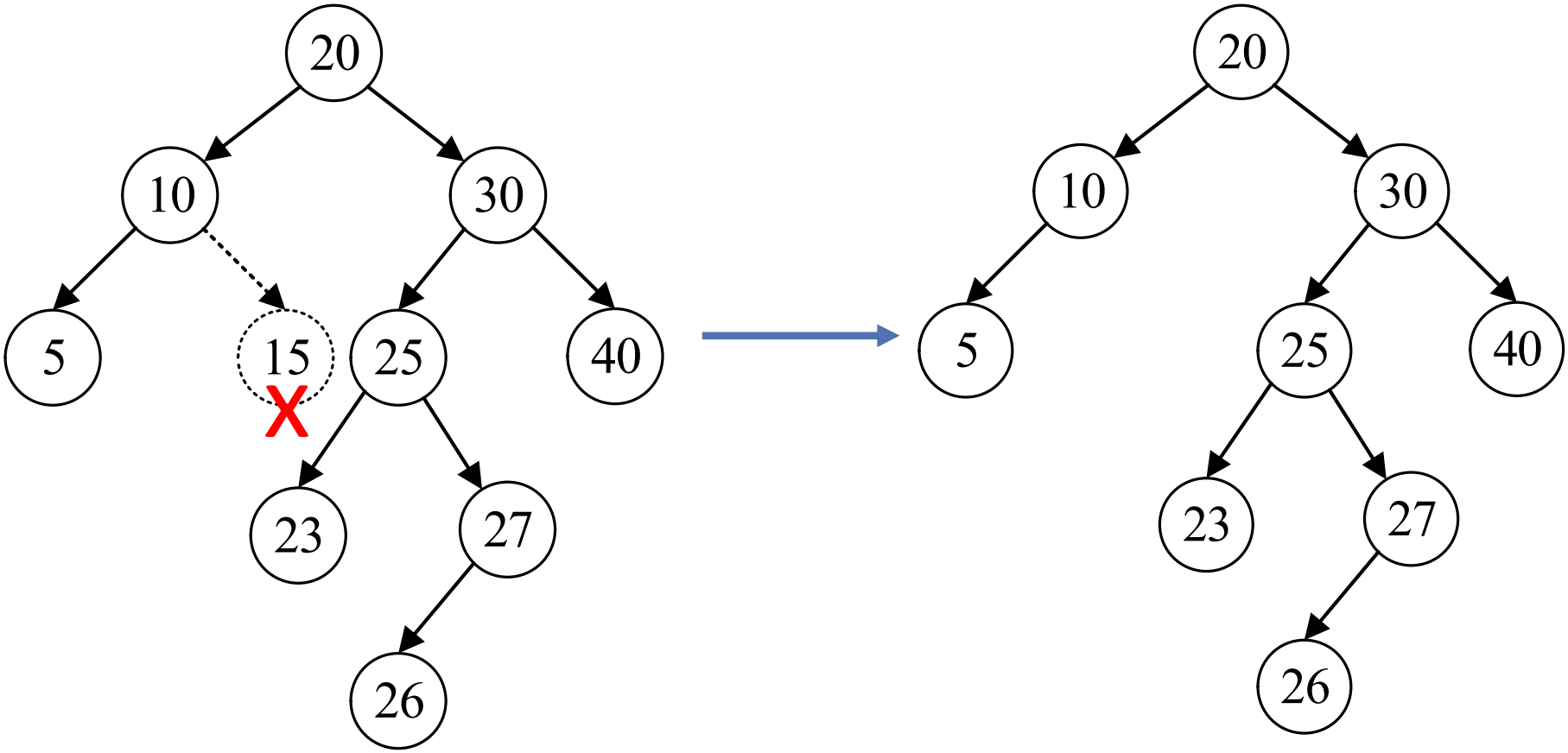
Hủy 1 node có khóa bằng X trên cây

- **Trường hợp 1**: Xóa node lá
  - Xóa trực tiếp node X khỏi cây, Node cha của node lá X sẽ trỏ đến NULL tương ứng với vị trí của X là con trái hay con phải.
- **Trường hợp 2**: Xóa node có 1 con
  - Trước khi xoá X ta móc nối cha của X với con duy nhất của X.
- **Trường hợp 3**: X có đầy đủ 2 cây con, ta thực hiện các bước sau:
  - **Bước 1:** Tìm node thay thế Y: Node Y sẽ là node có giá trị nhỏ nhất của cây con bên phải X, hoặc là node có giá trị lớn nhất của cây con bên trái X.
  - **Bước 2:** Thế chỗ giá trị node X cần xóa bằng giá trị của node Y
  - **Bước 3:** Thực hiện xóa node Y: Khi này node Y sẽ là node lá hoặc là node có 1 con, ta lại quay về **Trường hợp 1** hoặc **Trường hợp 2**

Time complexity = O(chiều cao của cây)

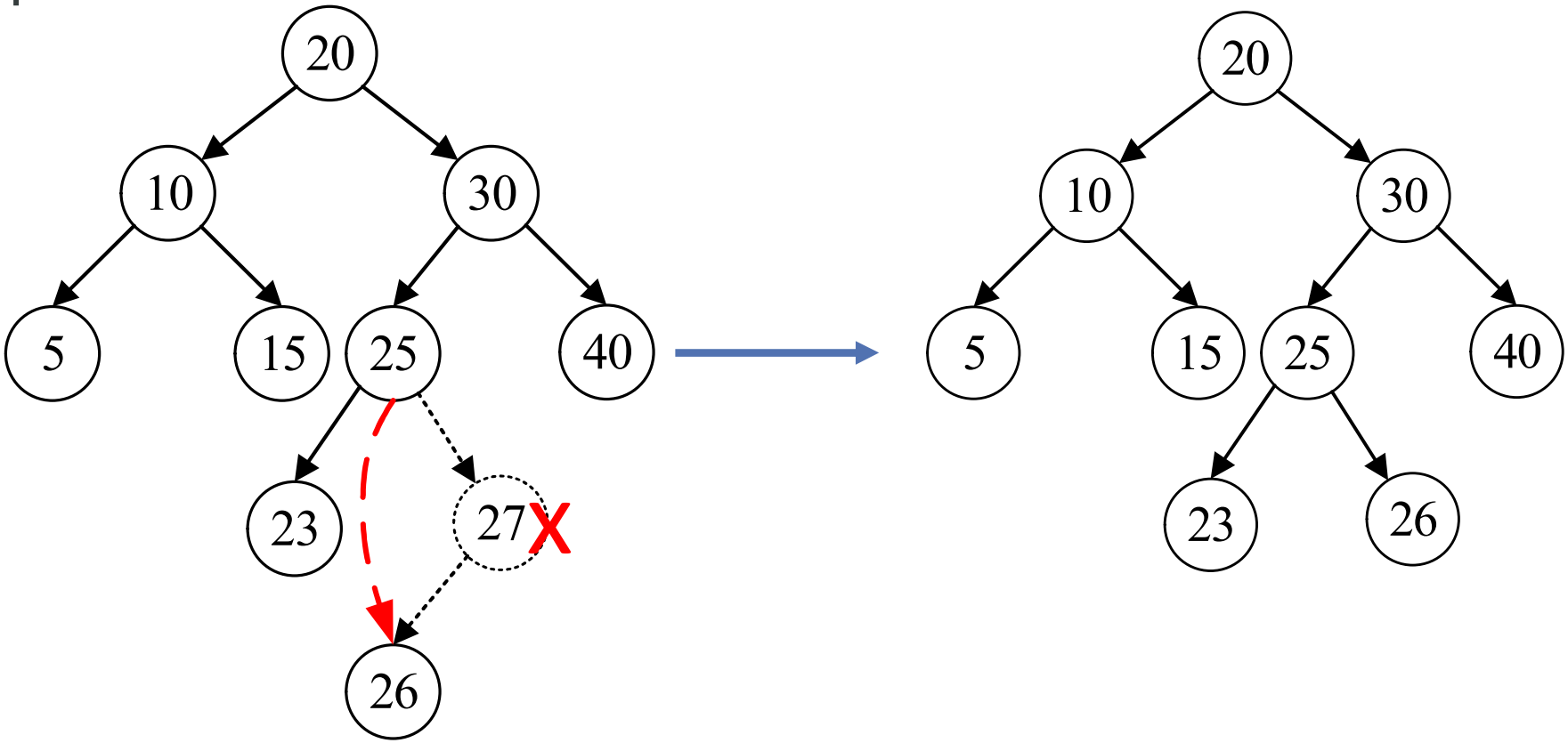
Minh họa hủy phần tử X là node lá

- Hủy X=15 trong cây:
  - Node chứa 15 là node lá
  - Node cha của 15 là node 10
  - Khi xóa node 15 thì con phải của 10 sẽ rỗng.



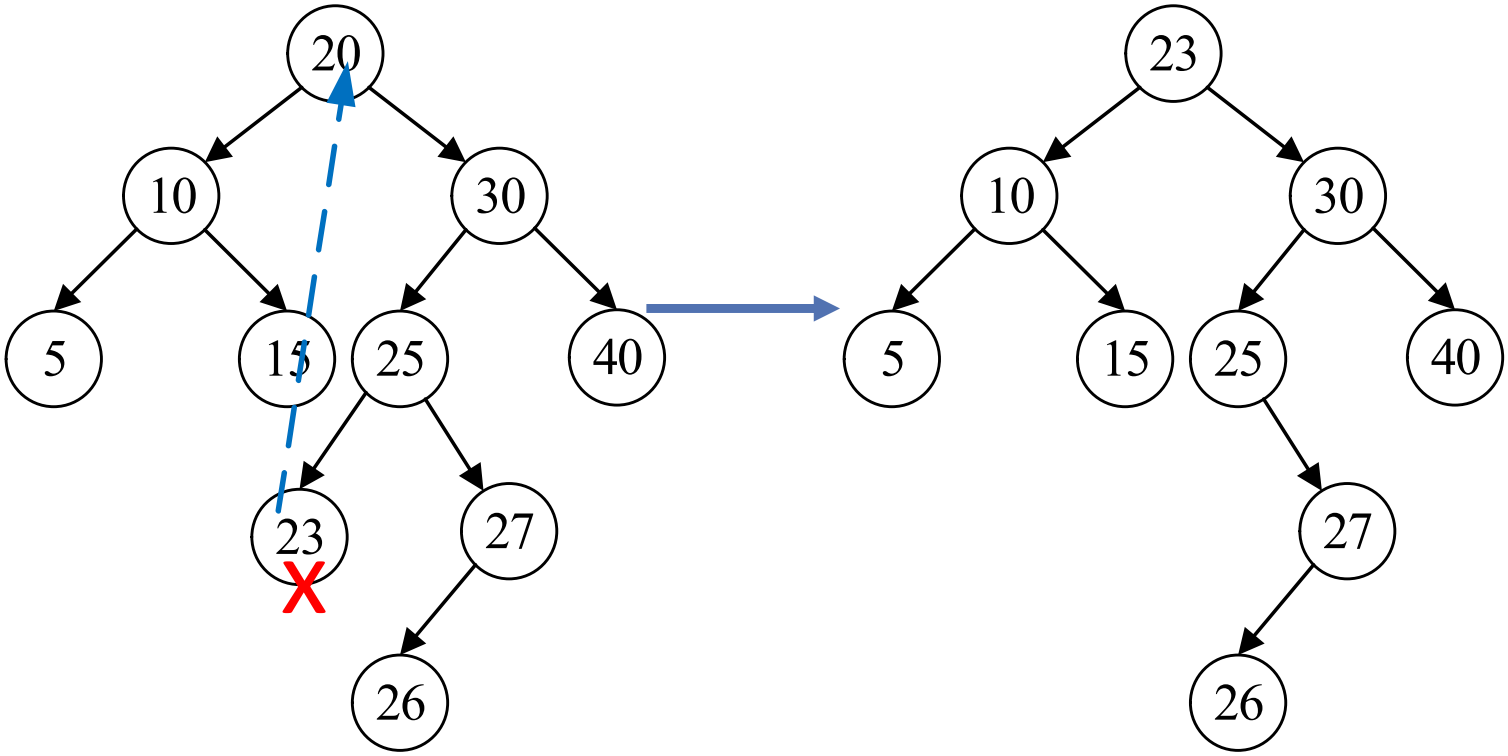
Minh hoạ hủy phần tử X có 1 cây con

- Hủy X=27 trong cây:
  - 27 có 1 cây con trái
  - 27 là con **phải** của 25
  - Xóa 27: con trái của 27 sẽ đưa lên làm con phải của 25



Minh hoạ hủy phần tử X có 2 cây con

- Hủy X=20 trong cây sau:
  - X có 2 cây con
  - Tìm node thế mạng Y là node chứa giá trị 23 (hoặc có thể chọn node 15)
  - Đưa giá trị 23 lên thay cho giá trị node 20
  - Xóa node thế mạng: trường hợp này xóa node Y là node lá.



Cài đặt thao tác xoá 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);

}
```

Ứng dụng

Bài tập

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.

Bài tập (tt)

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

- 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
- Kiểm tra 2 node có phải anh em
- 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

Bài tập (tt)

4. Cho mảng 1 chiều các số nguyên, dung cây nhị phân tìm kiểm để:

- Đếm có bao nhiêu số phân biệt
- Đếm số lần xuất hiện của từng con số trong dãy.

5. Cài đặt chương trình mô phỏng trực quan các thao tác trên cây nhị phân tìm kiếm.

Chúc các em học tốt!