CẤU TRÚC DỮ LIỆU & GIẢI THUẬT 1

CÂY (TREE)

Tree

- Cây là một tập hữu hạn các node có cùng kiểu dữ liệu, trong đó có 1 node đặc biệt gọi là node gốc (root).
- Giữa các node có một quan hệ phân cấp gọi là “quan hệ cha con”.

Định nghĩa cây dưới dạng đệ quy:

- Một node là 1 cây, node đó cũng là gốc (root) của cây đó.
- Cây gồm 1 node kết hợp với một số cây con bên dưới.

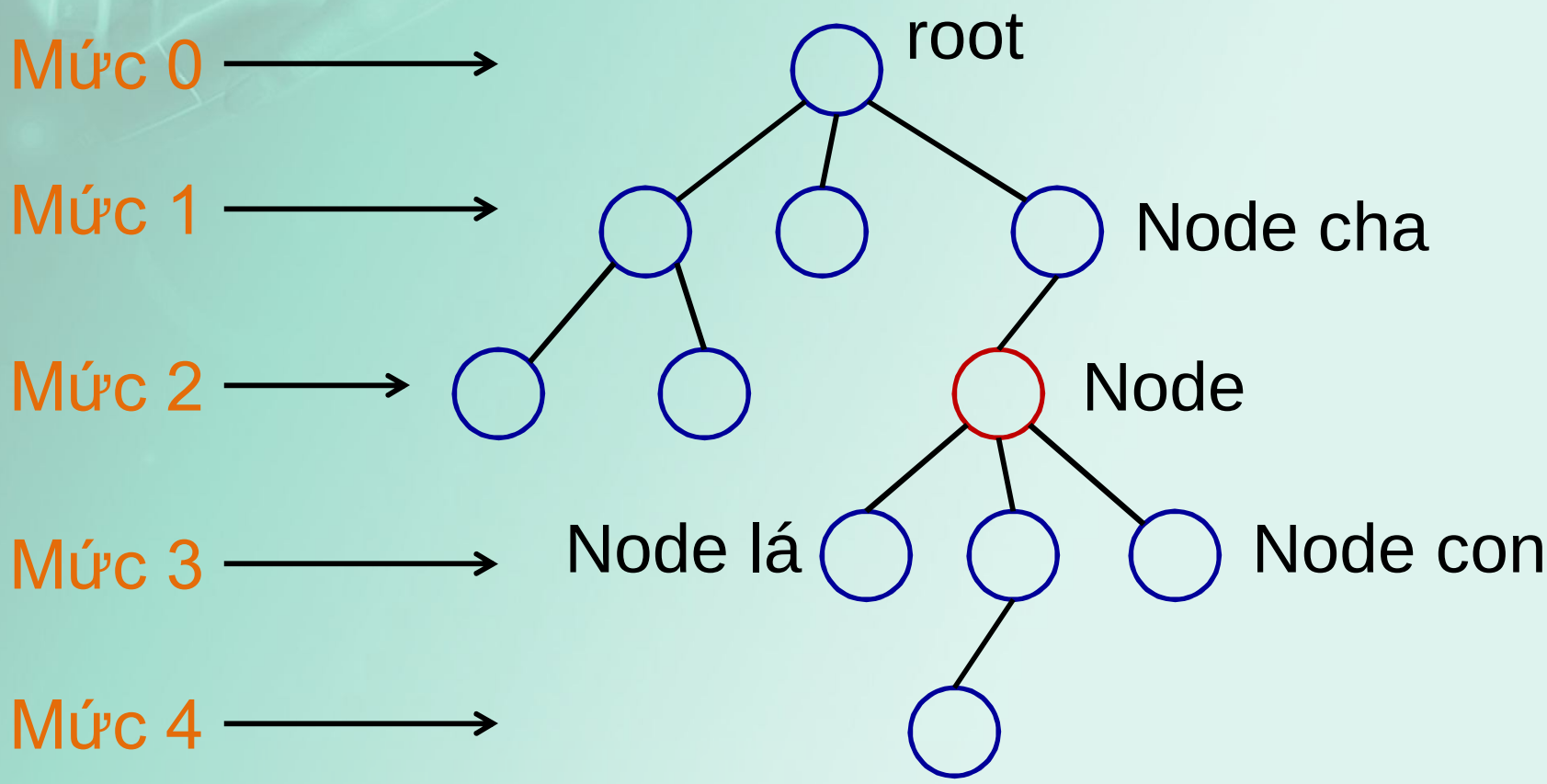
Tree

- Cây rỗng: Cây không có bất kỳ một node nào.
- Các node được gọi là cùng 1 cây khi có đường đi giữa các node này.
- Mỗi node trong cây (trừ node gốc) có duy nhất 1 node nằm trên nó, gọi là node cha.
- Node con: Các node nằm ngay dưới 1 node.
- Node lá: Các node không có con.
- Mức của node: node gốc có mức là 0. Nếu node cha có mức là i thì node con có mức là i+1.
- Chiều cao (sâu) của cây: Số mức lớn nhất của node có trên cây.

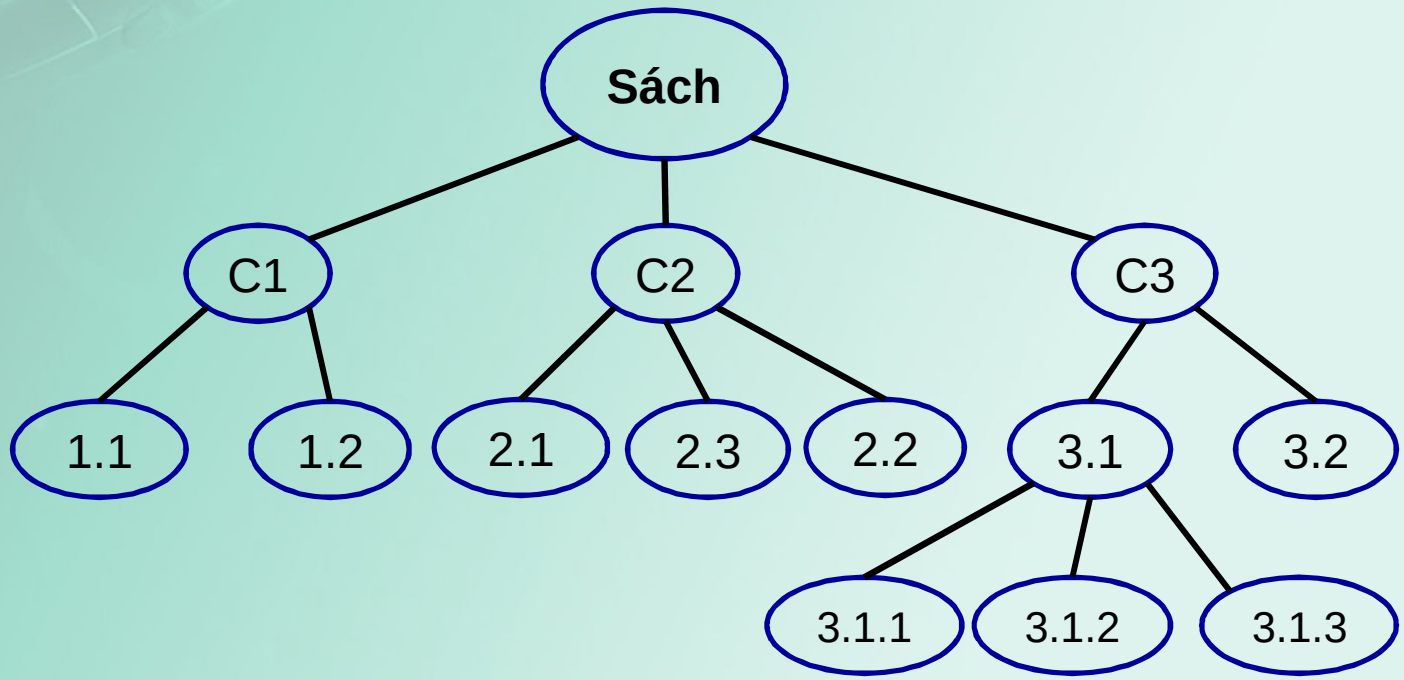
Tree

- Độ sâu của một node: Độ dài đường đi duy nhất giữa node gốc và node đó. Một node tại mức i có độ dài là i.
- Bậc của node: Số các cây con của 1 node. Node có bậc = 0 gọi là node lá.
- Bậc của cây: Bậc lớn nhất của các node trong cây. Cây có bậc n gọi là cây n-phân.
- Node nhánh: Node có bậc khác 0 và không phải là node gốc.
- Các node có cùng cha gọi là anh em.
- Tập các cây con riêng biệt gọi là rừng.
- Một cây được gọi là có thứ tự nếu các node con của 1 node được bố trí theo thứ tự nào đó, ngược lại gọi là cây không có thứ tự.

Tree

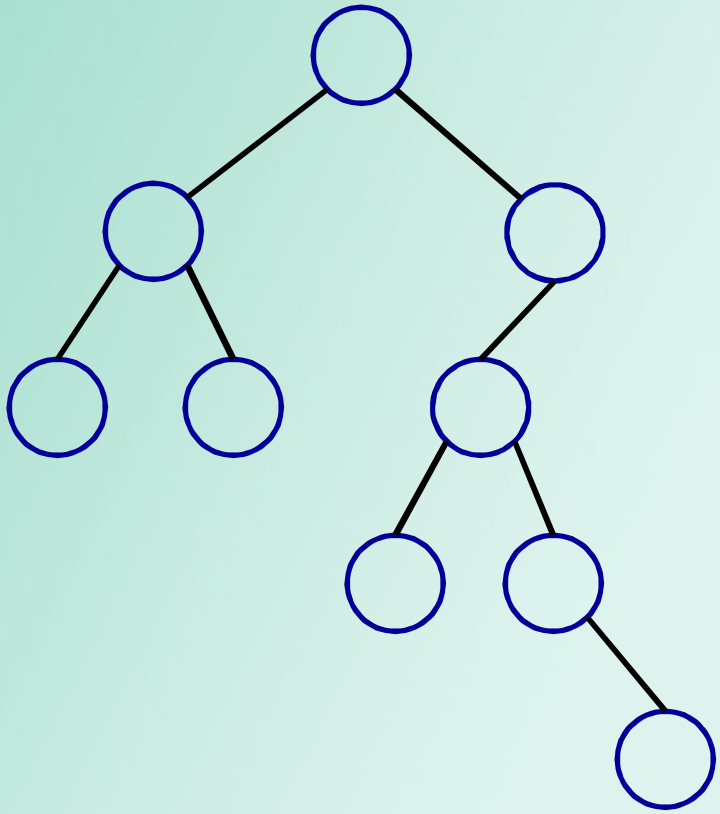


Ví dụ: Cây mục lục của 1 quyển sách



Cây nhị phân

- Cây nhị phân là cây rỗng hoặc là cây mà mỗi node có tối đa 2 node con.



Biểu diễn cây nhị phân

- Sử dụng danh sách liên kết

```
struct  node
{
        int data;
        node *left, *right;
};
typedef node * Pnode;
```

- Để khai báo biến cho 1 cây, khai báo biến con trỏ và cho con trỏ này chỉ vào phần tử gốc.

Ví dụ: Pnode tree;

Một số thao tác trên cây nhị phân

- Khởi tạo cây rỗng

  tree = NULL;

- Duyệt cây
  - Duyệt tiền tự (NLR): duyệt nút gốc, duyệt tiền tự con trái rồi duyệt tiền tự con phải.
  - Duyệt trung tự (LNR): duyệt trung tự con trái, duyệt nút gốc, rồi duyệt trung tự con phải.
  - Duyệt hậu tự (LRN): duyệt hậu tự con trái, duyệt hậu tự con phải, sau đó duyệt nút gốc.

Duyệt tiền tự

- Nếu cây rỗng => không làm gì.
- Nếu cây khác rỗng:
  - Thăm node gốc.
  - Duyệt cây con bên trái.
  - Duyệt cây con bên phải.
- Hàm:

```
void duyet_NLR (Pnode tree)
{
        if (tree!=NULL)
        {
                <duyệt node gốc>;
                duyet_NLR(tree->left);
                duyet_NLR(tree->right);
        }
}
```

Duyệt trung tự

- Nếu cây rỗng => không làm gì.
- Nếu cây khác rỗng:
  - Duyệt cây con bên trái.
  - Thăm node gốc.
  - Duyệt cây con bên phải.
- Hàm:

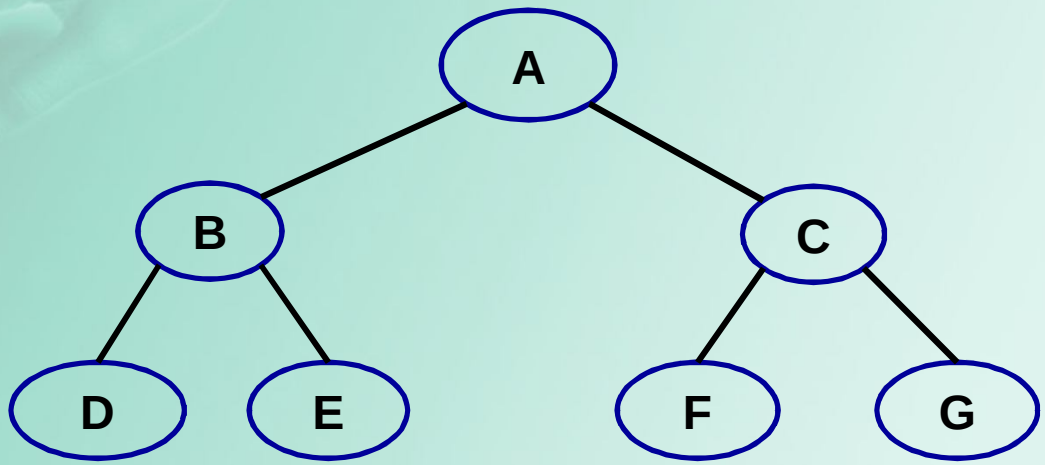
```
void duyet_LNR (Pnode tree)
{
        if (tree!=NULL)
        {
                duyet_LNR(tree->left);
                <duyệt node gốc>;
                duyet_LNR(tree->right);
        }
}
```

Duyệt hậu tự

- Nếu cây rỗng => không làm gì.
- Nếu cây khác rỗng:
  - Duyệt cây con bên trái.
  - Duyệt cây con bên phải.
  - Thăm node gốc.
- Hàm:

```
void duyet_LRN (Pnode tree)
{
        if (tree!=NULL)
        {
                duyet_LRN (tree->left);
                duyet_LRN (tree->right);
                <duyệt node gốc>;
        }
}
```

Duyệt cây



- Duyệt tiền tự: A -> B -> D -> E -> C -> F -> G
- Duyệt trung tự: D -> B -> E -> A -> F -> C -> G
- Duyệt hậu tự: D -> E -> B -> F -> G -> C -> A

Tìm node có nội dung là x trên cây nhị phân

- Nếu node gốc có nội dung là x => node gốc chính là node cần tìm.
- Nếu node gốc = NULL => không có.
- Nếu nội dung node gốc khác x và node gốc khác NULL:
  - Tìm node theo nhánh cây con bên trái.
  - Tìm node theo nhánh cây con bên phải.

Tìm kiếm

```
Pnode search (Pnode tree, int x)
{
        Pnode p;
        if (tree->data == x)
                return tree;
        if (tree == NULL)
                return NULL;
        p = search(tree->left, x);
        if (p == NULL)
                search(tree->right, x);
}
```

Cây nhị phân tìm kiếm

- Cây nhị phân tìm kiếm (Binay Search Tree) là một cây nhị phân mà tại mỗi node:
  - Phần tử ở node đó lớn hơn các phần tử ở cây con bên trái.
  - Nhỏ hơn các phần tử ở cây con bên phải.
- **Cài đặt cây nhị phân tìm kiếm**: Sử dụng danh sách liên kết.

```
struct  node
{
        int data;
        node *left, *right;
};
typedef node * Pnode;
```

Tìm kiếm 1 node trên cây NPTK

- Để tìm kiếm 1 node có giá trị x trên cây NPTK, so sánh giá trị của node gốc với x:
  - Nếu node gốc = NULL $\Rightarrow$ không có x trên cây.
  - Ngược lại, nếu x bằng giá trị node gốc $\Rightarrow$ dừng, tìm được node chứa x.
  - Nếu x > giá trị node gốc $\Rightarrow$ tìm x trên cây con bên phải.
  - Nếu x < giá trị node gốc $\Rightarrow$ tìm x trên cây con bên trái.

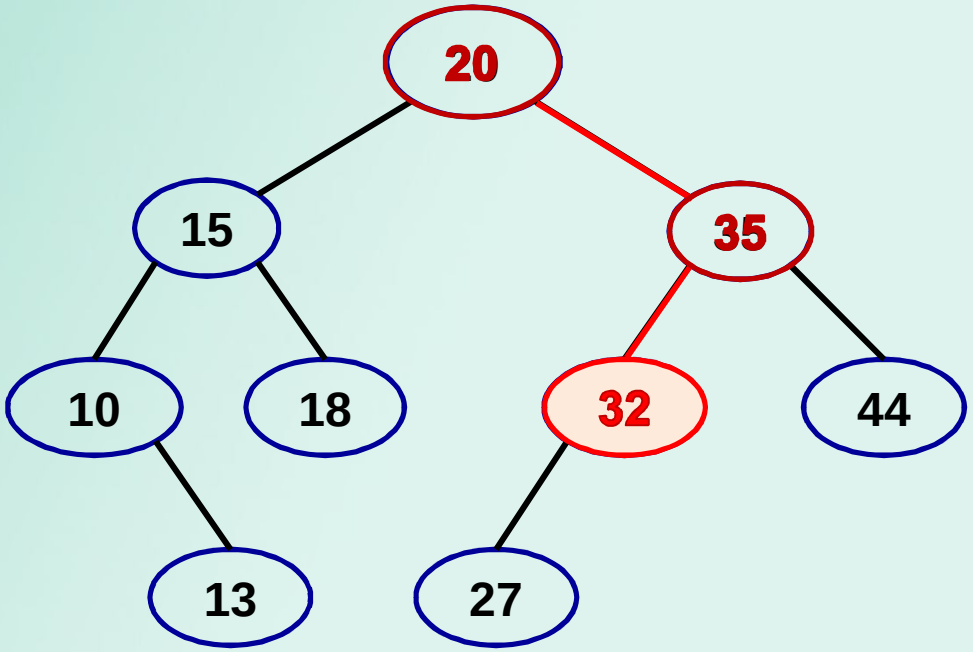
Tìm kiếm 1 node trên cây NPTK

```
Pnode search (Pnode tree,int x)
{
        if(tree==NULL)
                return NULL;
        if(tree->data==x)
                return tree;
        if(x<tree->data)
                search(tree->left,x);
        else
                search(tree->right,x);
}
```

Ví dụ

Tìm node có giá trị 32 trong cây ở hình bên.

- So sánh 32 với node gốc là 20, do 32>20 $\Rightarrow$ tìm tiếp trên cây con bên phải là cây có node gốc là 35.
- So sánh 32 với node gốc là 35, do 32<35 $\Rightarrow$ tìm tiếp trên cây con bên trái là cây có node gốc 32.
- So sánh 32 với node gốc là 32, 32=32 $\Rightarrow$ dừng, tìm được node chứa giá trị cần tìm.



Thêm 1 node vào cây NPTK

- Thuật toán thêm 1 node có giá trị x vào cây NPTK:
  - Nếu node gốc = NULL: x chưa có trên cây $\Rightarrow$ thêm node mới chứa x.
  - Nếu x trùng với node gốc $\Rightarrow$ không thể thêm.
  - Nếu x < node gốc $\Rightarrow$ thêm x ở cây con bên trái.
  - Nếu x > node gốc $\Rightarrow$ thêm x ở cây con bên phải.

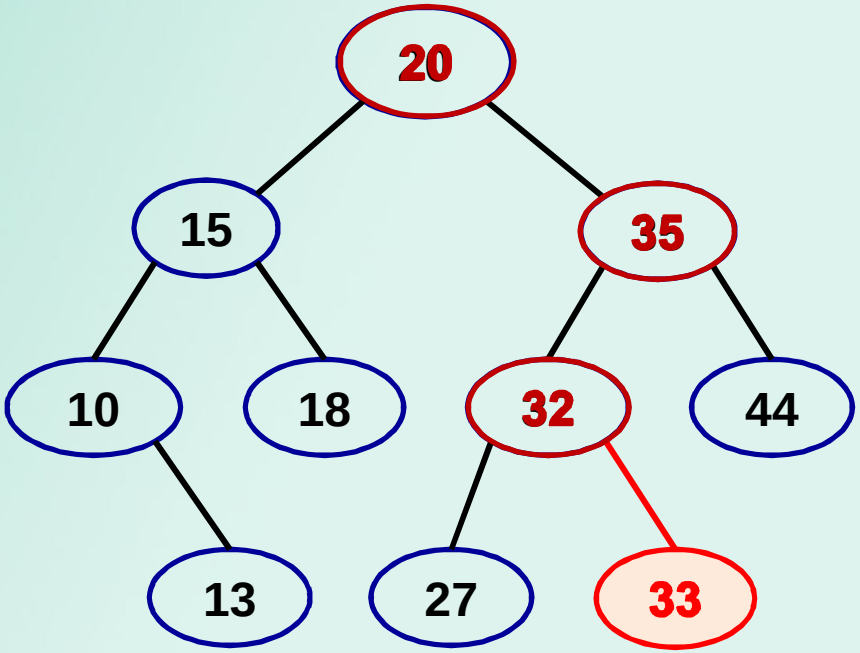
Thêm 1 node vào cây NPTK

```
void insert (Pnode &tree, int x)
{
        if(tree==NULL)
        {
                tree=new node;
                tree->data=x;
                tree->left=tree->right=NULL;
                return;
        }
        if(x==tree->data)
                return;
        else if(x < tree->data)
                 insert (tree->left,x);
        else
                 insert (tree->right,x);
}
```

Ví dụ

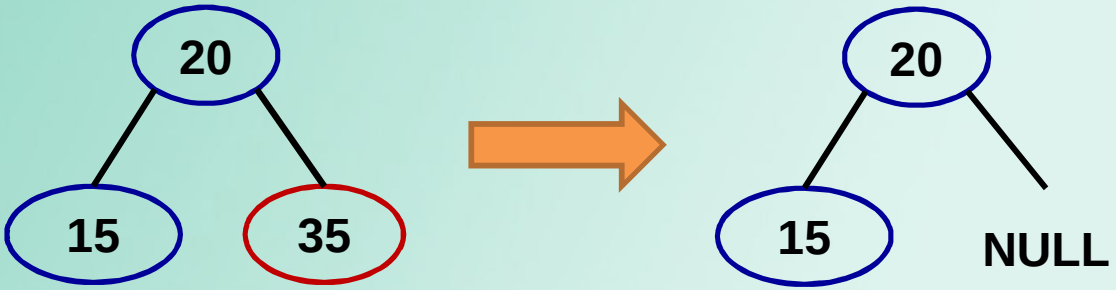
Thêm 1 node có giá trị 33 vào cây.

- So sánh 33 với node gốc là 20, do 33>20 $\Rightarrow$ xét tiếp cây con bên phải là cây có node gốc 35.
- So sánh 33 với node gốc là 35, do 33<35 $\Rightarrow$ xét tiếp cây con bên trái là cây có node gốc 32.
- So sánh 33 với node gốc là 32, 33>32 $\Rightarrow$ xét tiếp cây con bên phải. Vì node con bên phải = NULL, thêm node mới chứa 33 vào bên phải node chứa giá trị 32.



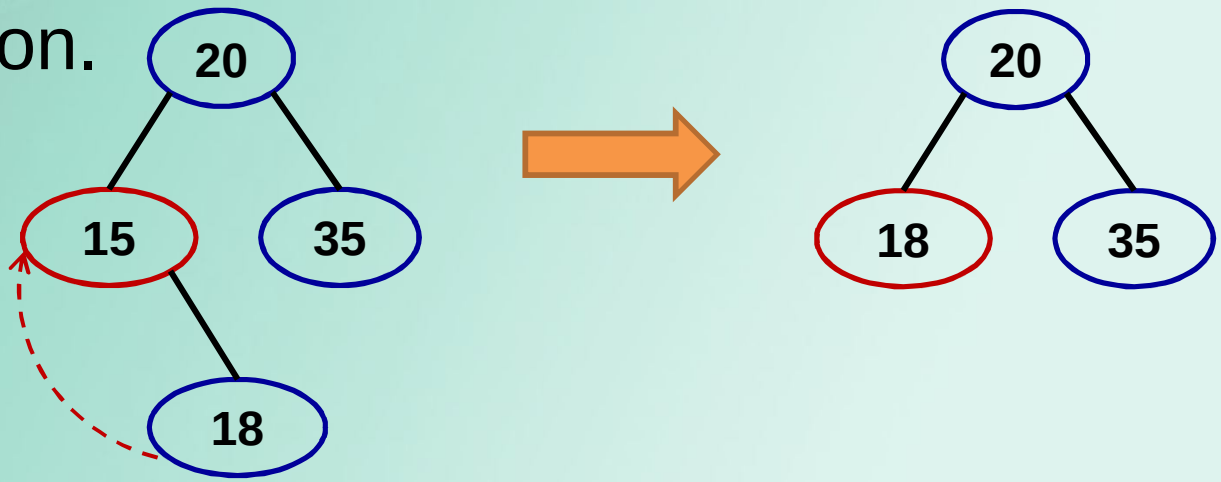
Xóa 1 phần tử x trên cây NPTK

- Khi xóa 1 node, cần phải điều chỉnh lại cây để nó vẫn là cây nhị phân tìm kiếm.
- **Các trường hợp khi xóa:**
- Xóa node có giá trị 35 là node lá.

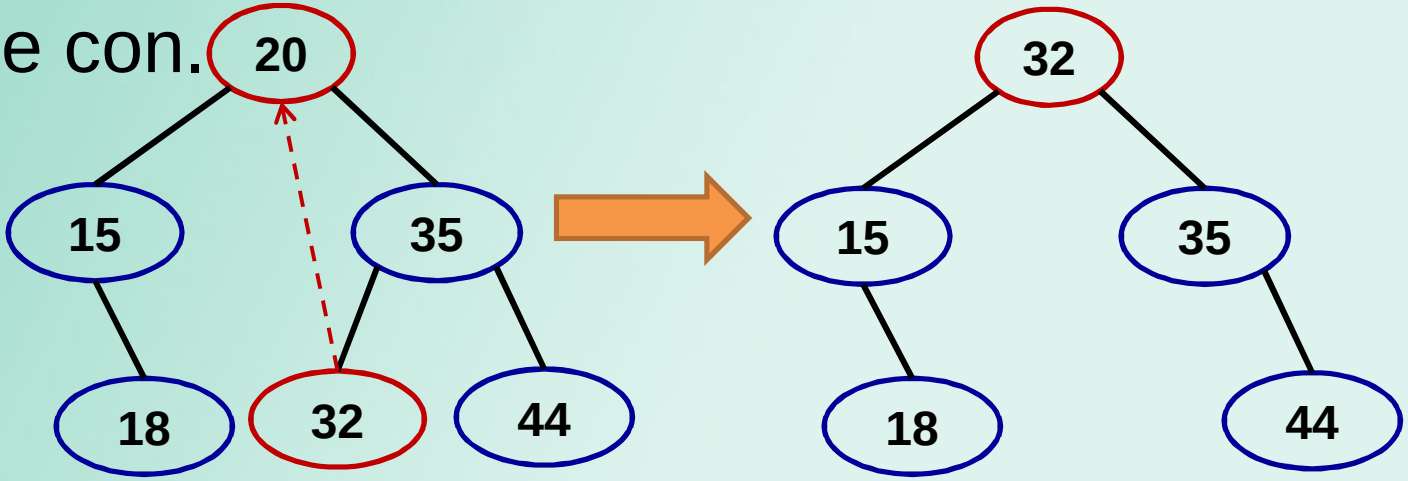


Xóa 1 phần tử x trên cây NPTK

- Xóa node có giá trị 15 là node trung gian có 1 node con.



- Xóa node có giá trị 20 là node trung gian có 2 node con.



Thuật toán xóa 1 node

- Nếu không tìm thấy node chứa giá trị x $\Rightarrow$ dừng.
- Nếu gặp node p chứa x:
  - TH1: Nếu p là node lá $\Rightarrow$ hủy node p.
  - TH2: Nếu p chỉ có 1 node con $\Rightarrow$ cho tree chỉ đến node con, hủy p.
  - TH3: Nếu p có đủ 2 node con, tìm node nhỏ nhất ở cây con bên phải (hoặc node lớn nhất ở cây con bên trái), thay thế giá trị tại node p bằng giá trị của node nhỏ nhất ở cây con bên phải, sau đó xóa node nhỏ nhất.

Xóa 1 phần tử x trên cây NPTK

```
void deletenode (Pnode &tree,int x)
{
    Pnode p,f;
    p=tree;
    if(tree==NULL)
        return;
    if(x < tree->data)
        deletenode(tree->left,x);
    else if(x > tree->data)
        deletenode(tree->right,x);
    else if(tree->left==NULL && tree->right
==NULL)
    {
        delete tree;
        tree=NULL;
    }
    else if(tree->left==NULL)
    {
        tree=tree->right;
        delete p;
    }
    else if(tree->right==NULL)
    {
        tree=tree->left;
        delete p;
    }
    else
    {
        p=min(tree->right);
        f=parent(tree,p);
        tree->data=p->data;
        if(f!=NULL)
        {
            if(f->left==p)
              f->left=NULL;
            else if(f->right==p)
              f->right=p->right;
            delete p;
        }
    }
}
```

Xóa 1 phần tử x trên cây NPTK

```
Pnode min (Pnode tree)
{
        if(tree->left==NULL)
                return tree;
        return min(tree->left);
}
Pnode parent (Pnode tree, Pnode p)
{
        if(tree==NULL || tree==p)
                return NULL;
        if(tree->left==p||tree->right==p)
                return tree;
        if(p->data > tree->data)
                return parent(tree->right,p);
        else return parent(tree->left,p);
}
```

Bài tập

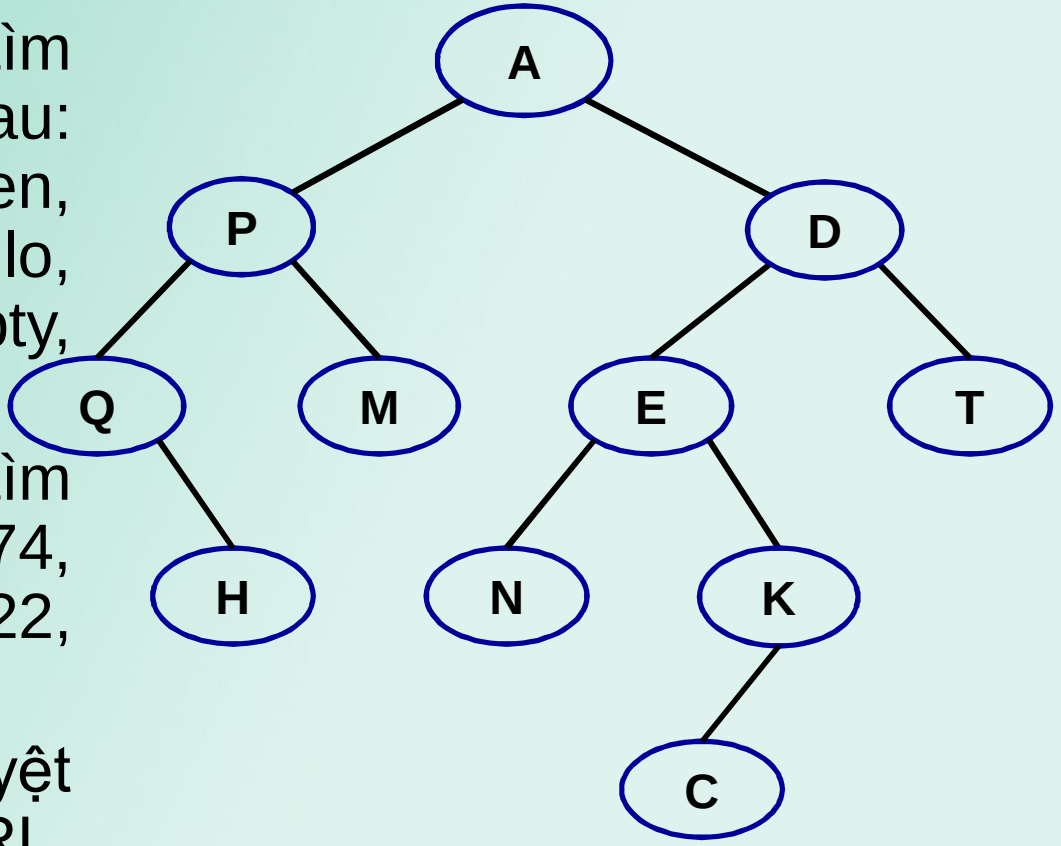
1. Viết chương trình thực hiện các thao tác trên cây nhị phân tìm kiếm:
   a. Thêm phần tử.
   b. Tìm kiếm phần tử.
   c. Hủy bỏ phần tử.
   d. Duyệt cây theo NLR, LNR, LRN.
   e. Đếm số node lá trên cây.
   f. Tính chiều cao của cây.
   g. Đếm số node có: bậc 1, bậc 2.
   h. Tìm node con bên trái, bên phải của 1 node.
   i. Tính tổng các phần tử trên cây.

Bài tập

2. Hãy vẽ cây nhị phân tìm kiếm cho các giá trị sau: paper, hero, talent, green, paint, time, potato, hello, tomato, talkative, empty, storage.

3. Hãy vẽ cây nhị phân tìm kiếm cho dãy số: 40, 25, 74, 57, 65, 80, 16, 20, 43, 22, 36, 78.

4. Cho biết thứ tự duyệt LNR, NLR, LRN, RNL, NRL, RLN của cây ở hình bên:



Bài tập

5. Một sinh viên có các thông tin: mã số, họ tên, năm sinh, điểm trung bình.

Viết chương trình tạo cây nhị phân tìm kiếm kiểu sinh viên, sinh viên nhập đầu tiên làm node gốc, dựa vào mã số sinh viên để lưu vào cây con bên trái hoặc cây con bên phải của cây nhị phân tìm kiếm, viết một số thao tác sau:

a. Thêm 1 sinh viên.
b. Tìm kiếm sinh viên theo mã số.
c. Hủy bỏ sinh viên theo mã số.
d. Xuất danh sách sinh viên.
e. Đếm số sinh viên có điểm trung bình >=5.