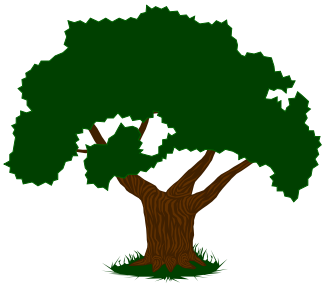
CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

GV : ThS. Trần Văn Thọ
E-mail : thotv@hufi.edu.vn

Đây là gì? Nó có những bộ phận chính nào?



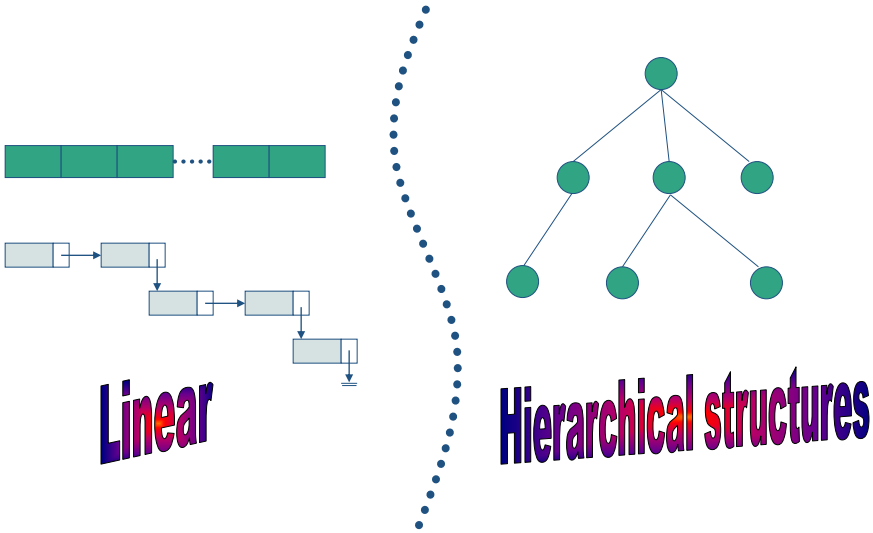
Môn: CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

CHƯƠNG V:
CÂY

Nội dung

- **Cấu trúc cây**
- Cây nhị phân
- Cây nhị phân tìm kiếm
- Cây nhị phân tìm kiếm cân bằng

Cấu trúc dữ liệu

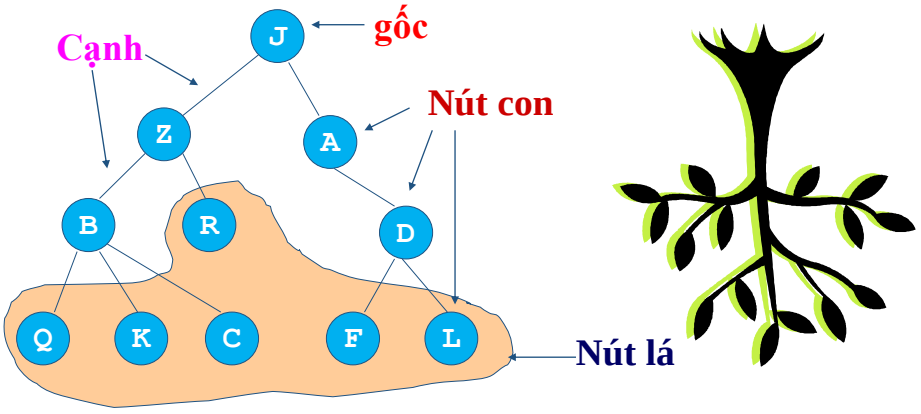


Cấu trúc cây

- **Tập hợp các nút và cạnh nối các nút đó.**
- **Có một nút gọi là *gốc*.**
- **Quan hệ one-to-many giữa các nút.**
- **Có duy nhất một đường đi từ gốc đến nút con.**
- **Các loại cây:**
  - **Nhị phân**: mỗi nút có {0, 1, 2} nút con
  - **Tam phân**: mỗi nút có {0, 1, 2, 3} nút con
  - **n-phân**: mỗi nút có {0, 1, ..., n} nút con

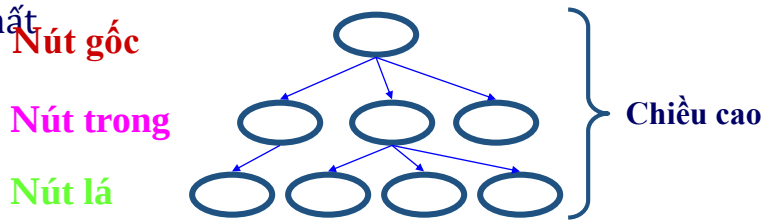
Cấu trúc cây

Khái niệm

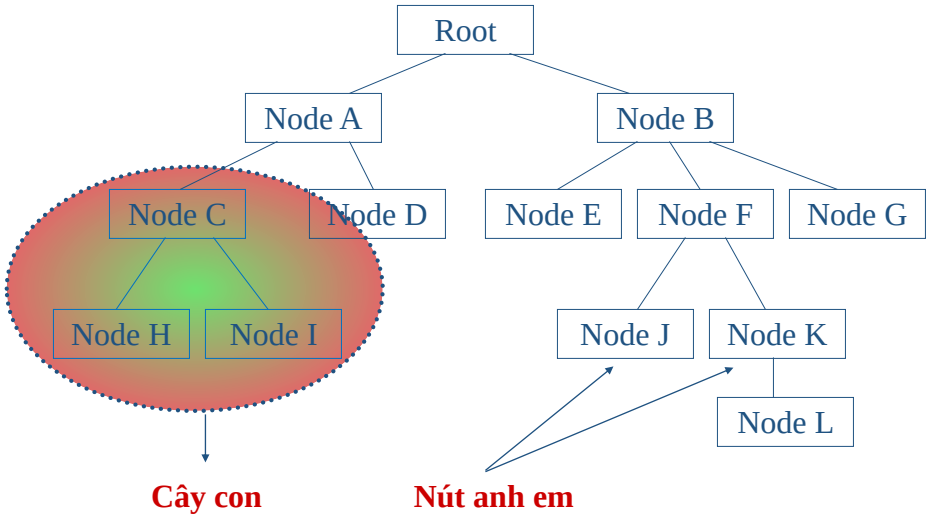


Khái niệm

- **Thuật ngữ**
  - **Nút gốc**: không có nút cha
  - **Nút lá**: không có nút con
  - **Nút trong**: không phải nút lá và nút gốc
  - **Chiều cao**: khoảng cách từ gốc đến lá của nhánh cao nhất



Khái niệm



Nội dung

- Cấu trúc cây
- **Cây nhị phân**
- Cây nhị phân tìm kiếm
- Cây nhị phân tìm kiếm cân bằng

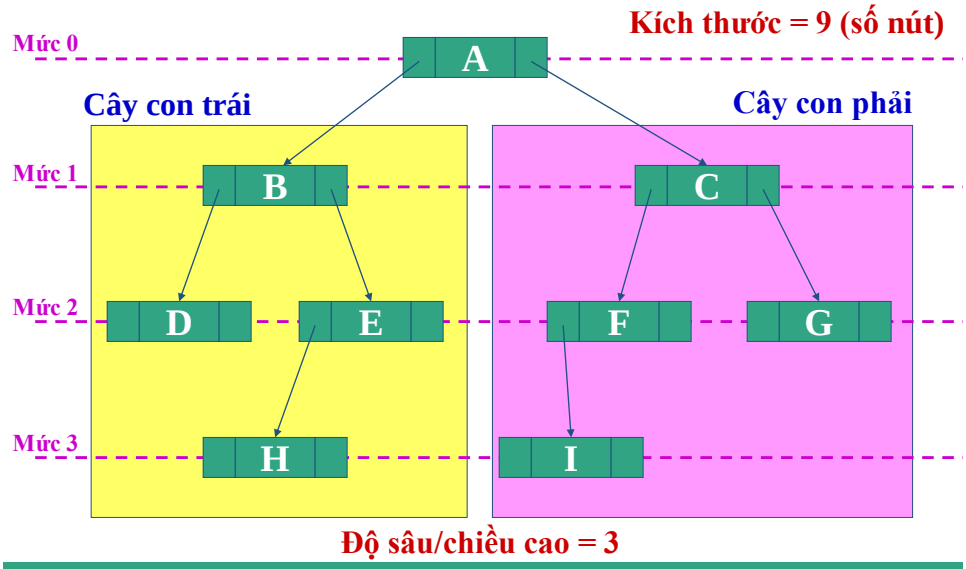
Cây nhị phân

- **Cây nhị phân là một đồ thị có hướng, mỗi nút có tối đa 2 nút con, *hay Cây nhị phân là cây có bậc bằng hai (bậc của mỗi nút tối đa bằng 2)*.**
- **Cấu trúc cây đơn giản nhất**
- **Tại mỗi nút gồm các 3 thành phần**
  - Phần **Data**: chứa giá trị, thông tin của nút
  - Phần **Left**: Liên kết đến nút con trái (nếu có)
  - Phần **Right**: Liên kết đến nút con phải (nếu có)

Cây nhị phân

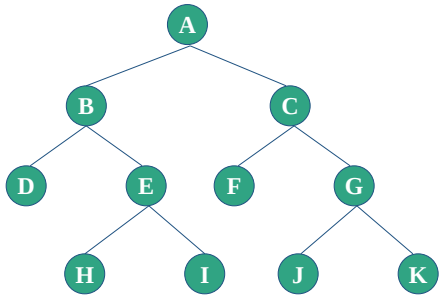
- **Cây nhị phân có thể rỗng (**không có nút nào**)**
- **Cây nhị phân khác rỗng có 1 nút gốc**
  - Có duy nhất 1 đường đi từ gốc đến 1 nút con.
  - Nút không có nút con bên trái và con bên phải là nút lá.

Cây nhị phân



Cây nhị phân

- **Cây nhị phân đúng:**
  - Nút gốc và nút trung gian có đúng 2 con
- **Cây nhị phân đúng có n nút lá thì số nút trên cây 2* n – 1.**



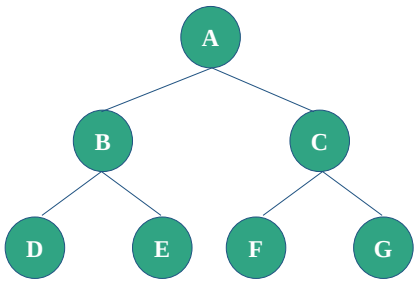
Cây nhị phân

- **Cây nhị phân đầy đủ với chiều sâu d**
  - Phải là cây nhị phân đúng
  - Tất cả nút lá có chiều sâu d

Số nút = $(2^{d+1} -1)$

Số nút trung gian = ?

Biết số nút tính d của cây nhị phân đầy đủ



Cấu trúc cây nhị phân

- **Cấu trúc của cây nhị phân.**

```
typedef int ItemType;
struct TNode
{  //Cấu trúc của một nút
      ItemType Info;
      TNode*   Left;
      TNode*   Right;
};
struct BTree
{  //Cấu trúc của một cây
      TNode* Root;
};
```

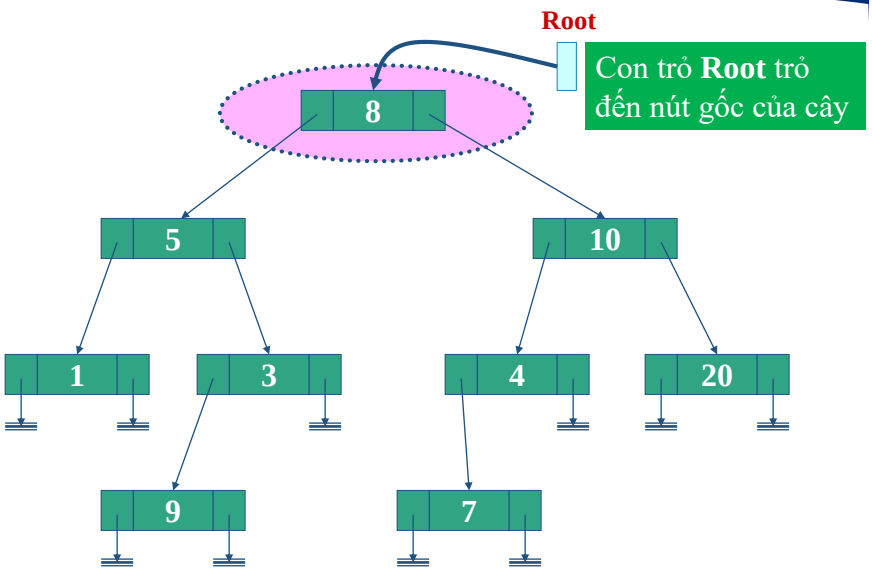
Chứa thông tin của nút

Trỏ đến nút con trái

Trỏ đến nút con phải

Con trỏ đến nút gốc của cây

Sơ đồ cây nhị phân



Các thao tác cơ bản

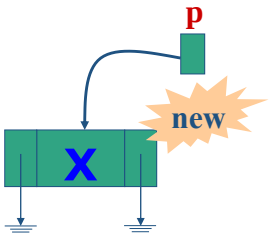
- Khởi tạo cây
- Tạo nút mới.
- Thêm nút con trái T
- Thêm nút con phải T
- Thêm nút có giá trị x
- Duyệt cây
- Xóa con trái T
- Xóa con phải T
- Xóa nút có giá trị x
- Xóa cây.

Khởi tạo cây nhị phân rỗng

```
void initBTree(BTree &bt)
{
      bt.Root = NULL;
}
```

Tạo nút mới

```
TNode* createTNode(ItemType x)
{
        TNode* p = new TNode;
        if(!p) return NULL;
        p→Info = x;
        p→Left = NULL;
        p→Right = NULL;
        return p;
}
```



Thêm nút con bên trái của node T

```
int insertTNodeLeft(TNode* T, ItemType x)
{
    if(T == NULL)
        return 0; //Không tồn tại nút T
    if(T→Left != NULL)
        return 0; //Đã tồn tại nút con trái
    TNode* p = createTNode(x);
    T→Left = p;
    return 1;
}
```

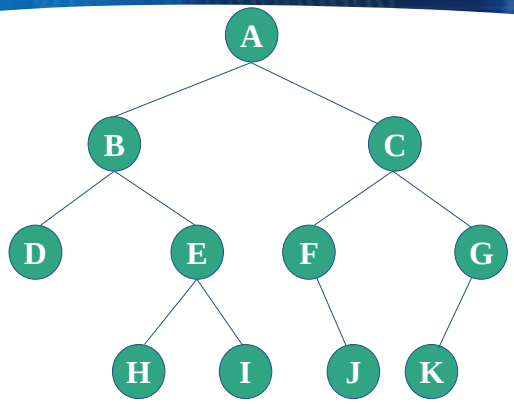
Thêm nút con bên phải của node T

```
int insertTNodeRight(TNode* T, ItemType x)
{
   if(T == NULL)
      return 0; //Không tồn tại nút T
   if(T→Right != NULL)
      return 0; //Đã tồn tại nút con phải
   TNode* p = createTNode(x);
   T→Right = p;
   return 1;
}
```


Duyệt cây

- **Do cây là cấu trúc không tuyến tính**
- **Có 6 cách duyệt cây nhị phân:**
  - Duyệt theo thứ tự trước PreOrder: **traverseNLR**, **traverseNRL**
  - Duyệt theo thứ tự giữa InOrder: **traverseLNR**, **traverseRNL**
  - Duyệt theo thứ tự sau PostOrder: **traverseLRN**, **traverseRLN**

Duyệt cây



PreOrder (traverseNLR, traverseNRL)

InOrder (traverseLNR, traverseRNL)

PostOrder (traverseLRN, traverseRLN)

Duyệt Node Left Right

```
void traverseNLR(TNode* root)
{
    if(root == NULL) return;
    <xử lý gốc>;
    traverseNLR(root→Left);
    traverseNLR(root→Right);
}
```

Duyệt Node Left Right

```
void traverseNRL(TNode* root)
{
    if(root == NULL) return;
    <xử lý gốc>;
    traverseNRL(root→Right);
    traverseNRL(root→Left);
}
```


Duyệt Left Node Right

```
void traverseLNR(TNode* root)
{
    if(root == NULL) return;
    traverseLNR(root→Left);
    <xử lý gốc>;
    traverseLNR(root→Right);
}
```

Duyệt Left Node Right

```
void traverseRNL(TNode* root)
{
    if(root == NULL) return;
    traverseRNL(root→Right);
    <xử lý gốc>;
    traverseRNL(root→Left);
}
```


Duyệt Left Right Node

```
void traverseLRN(TNode* root)
{
    if(root == NULL) return;
    traverseLRN(root→Left);
    traverseLRN(root→Right);
    <xử lý gốc>;
}
```

Duyệt Left Right Node

```
void traverseRLN(TNode* root)
{
    if(root == NULL) return;
    traverseRLN(root→Right);
    traverseRLN(root→Left);
    <xử lý gốc>;
}
```


Xóa nút con trái của node T

```
int deleteTNodeLeft(TNode* T)
{//Nút này phải là nút lá
    if(T == NULL) return 0;
    TNode* p = T → Left;
    if(p == NULL) return 0;
    if(p→Left != NULL || p→Right != NULL)
        return 0;
    delete p;
    return 1;
}
```

Xóa nút con phải của node T

```
int deleteTNodeRight(TNode* T)
{//Nút này phải là nút lá
    if(T == NULL) return 0;
    TNode* p = T→Right;
    if(p == NULL) return 0;
    if(p→Left != NULL || p→Right != NULL)
        return 0;
    delete p;
    return 1;
}
```


Tìm nút có khóa x

```
TNode* findTNode(TNode* root, ItemType x)
{
    if(!root) return NULL;
    if(root→Info == x) return root;
    TNode* p = findTNode(root→Left, x);
    if(p) return p;
    return findTNode(root→Right, x);
}
```

Xóa cây

```
int deleteTree(TNode* &root)
{
    if(!root) return 0;
    deleteTree(root→Left);
    deleteTree(root→Right);
    delete root;
    return 1;
}
```


Các thao tác mở rộng

- Đếm số nút trên cây.
- Đếm số nút lá trên cây.
- Đếm số nút trong.
- Xác định độ sâu/chiều cao của cây.
- Đếm số nút có giá trị bằng x.
- Tìm giá trị nhỏ nhất/lớn nhất trên cây.
- Tính tổng các giá trị trên cây.
- Xuất ra màn hình các nút ở mức thứ k.
- Đếm các nút lá ở mức k

Đếm số nút trên cây

```
int countTNode(TNode* root)
{
    if(!root)  return 0;
    int cnl = countTNode(root→Left);
    int cnr = countTNode(root→Right);
    return (1 + cnl + cnr);
}
```


Đếm số nút lá

```
int countTNodeLeaf(TNode* root)
{
    if(!root)  return 0;
    int cnl = countTNodeLeaf(root→Left);
    int cnr = countTNodeLeaf(root→Right);
    if(!root→Left && !root→Right)
        return (1 + cnl + cnr);
    return (cnl + cnr);
}
```

Đếm số nút không phải nút lá

```
int countTNodeNoLeaf(TNode* root)
{
    if(!root)  return 0;
    int cnl=countTNodeNoLeaf(root→Left);
    int cnr=countTNodeNoLeaf(root→Right);
    if(root→Left || root→Right)
        return (1 + cnl + cnr);
    return (cnl + cnr);
}
```


Đếm số nút trong (*nút trung gian*)

```
int countTNodeMedium(TNode* root)
{
    int cn = countTNode(root);
    if(cn <= 2)  return 0;
    int cnl = countTNodeLeaf(root);
    return (cn – cnl - 1); //trừ nút gốc
}
```

Đếm số nút trong (*nút trung gian*)

```
int countTNodeMedium(TNode* root)
{
    int n = countTNodeNoLeaf(root);
    if(n > 0)
        return (n - 1); //trừ nút gốc
    return 0;
}
```


Tính tổng

```
int sumTNode(TNode* root)
{
    if(!root) return 0;
    int suml = sumTNode(root→Left);
    int sumr = sumTNode(root→Right);
    return (root→Info + suml + sumr);
}
```

Tính chiều cao của cây

```
int highBTree(TNode* root)
{
    if(!root)  return 0;
    int hl = highBTree(root→Left);
    int hr = highBTree(root→Right);
    if(hl > hr)
        return (1 + hl);
    else
        return (1 + hr);
}
```


Đếm số nút có giá trị bằng x

```
int countTNodeX(TNode* root, int x)
{
    if(!root)  return 0;
    int nlx = countTNodeX(root→Left, x);
    int nrx = countTNodeX(root→Right, x);
    if(root→Info == x)
        return (1 + nlx + nrx);
    return (nlx + nrx);
}
```

Tìm giá trị lớn nhất trên cây nhị phân

```
int Max(int a, int b) {return (a>b) ? a : b; }
int maxTNode(TNode* root)
{
    if(!root→Left && !root→Right)
        return (root→Info);
    int maxl = maxTNode(root→Left);
    int maxr = maxTNode(root→Right);
    return  Max(root→Info, Max(maxl, maxr));
}
```


Tìm giá trị lớn nhất trên cây nhị phân

```
int maxTNode(TNode* root) {
    if(!root→Left && !root→Right)
        return (root→Info);
    int maxl = maxTNode(root→Left);
    int maxr = maxTNode(root→Right);
    int max = root→Info;
    if(max < maxl) max = maxl;
    if(max < maxr) max = maxr;
    return max;
}
```

Xuất ra màn hình các nút ở mức k

```
void showTNodeLevelK(TNode* root, int k)
{
    if(!root) return;
    if(k == 0) //đến tầng cần tìm
        printf("%4d", root→Info);
    k--; //mức k giảm dần về 0
    showTNodeLevelK(root→Left, k);
    showTNodeLevelK(root→Right, k);
}
```


Đếm các nút lá ở mức k

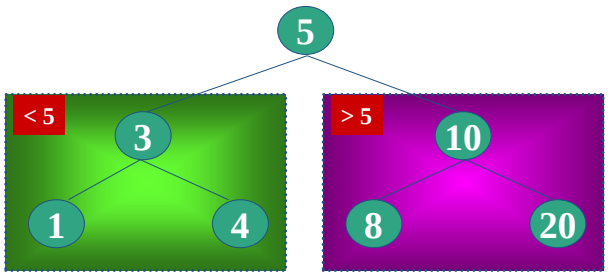
```
int countTNodeLeafLevelK(TNode* root, int k)
{
    if(!root) return 0;
    if(k==0 && !root→Left && !root→Right)
        return (1);
    k--; //mức k giảm dần về 0
    int nl=countTNodeLeafLevelK(root→Left,k);
    int nr=countTNodeLeafLevelK(root→Right,k);
    return (nl + nr);
}
```

Nội dung

- Cấu trúc cây
- Cây nhị phân
- **Cây nhị phân tìm kiếm (BST– Binary Search Tree)**
- Cây nhị phân tìm kiếm cân bằng

Khái niệm

- **BST là cây nhị phân mà mỗi nút thoả:**
  - Giá trị của tất cả nút con trái < nút gốc
  - Giá trị của tất cả nút con phải > nút gốc



CẤU TRÚC CÂY NHỊ PHÂN TÌM KIẾM

- **Cấu trúc dữ liệu của cây nhị phân tìm kiếm.**

```
typedef int ItemType;
struct TNode
{ //Cấu trúc của một nút
      ItemType Info;
      TNode* Left;
      TNode* Right;
};
struct BSTree
{ //Cấu trúc của một cây
      TNode* Root;
};
```

Chứa thông tin của nút

Trỏ đến nút con trái

Trỏ đến nút con phải

Con trỏ đến nút gốc của cây

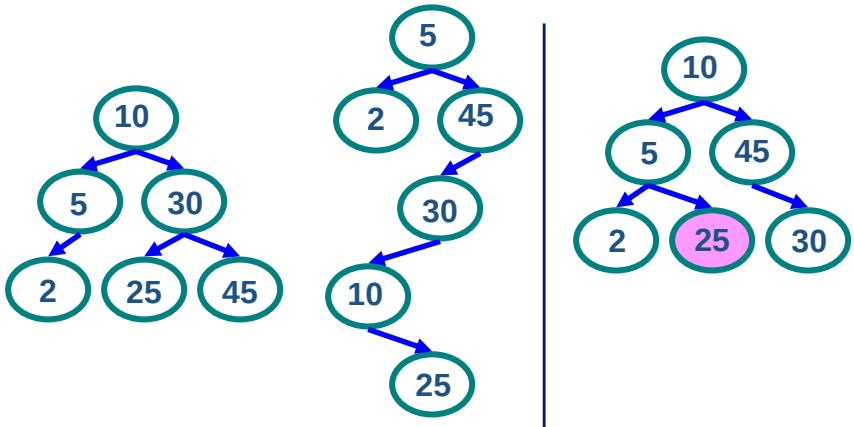
Cây nhị phân tìm kiếm

- **Xây dựng cây BST**
  - Tìm kiếm/ thêm
  - Xóa
- **Luôn duy trì tính chất**
  - Giá trị nhỏ hơn ở bên cây con trái
  - Giá trị lớn hơn ở bên cây con phải

Tìm kiếm

- **Xuất phát từ gốc**
  - Nếu gốc = NULL => không tìm thấy
  - Nếu khóa x = khóa nút gốc => tìm thấy
  - Ngược lại nếu khóa x < khóa nút gốc => Tìm trên cây bên trái
  - Ngược lại => tìm trên cây bên phải

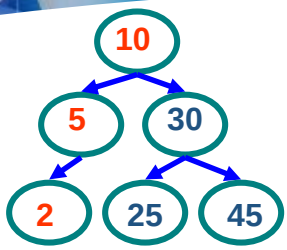
Ví dụ



Binary search trees

Non-binary search tree

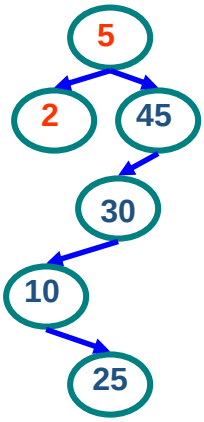
Ví dụ tìm x = 2



10 > 2, Left

5 > 2, Left

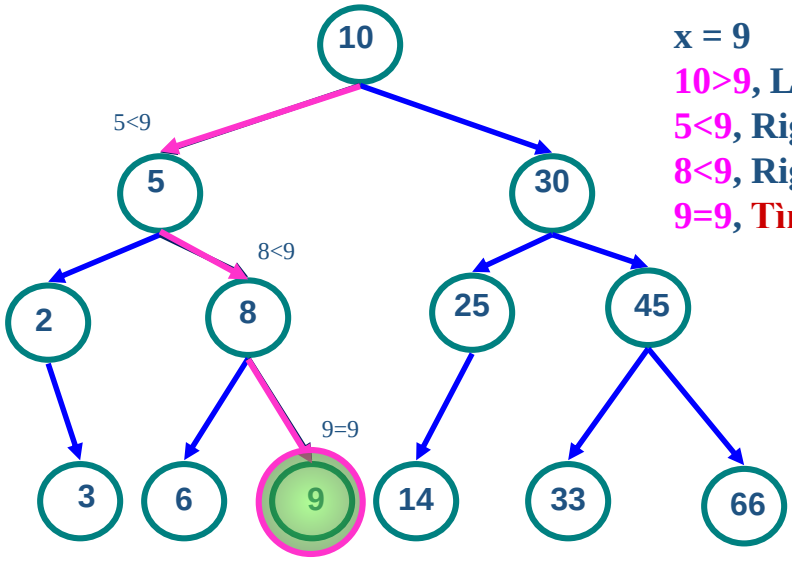
2 = 2, Tìm thấy



5 > 2, Left

2 = 2, Tìm thấy

Ví dụ tìm x= 9



x = 9

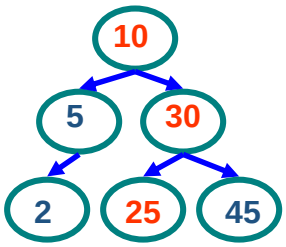
10>9, Left

5<9, Right

8<9, Right

9=9, Tìm thấy

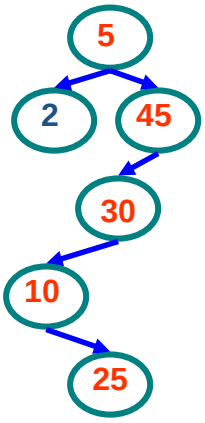
Ví dụ tìm x = 25



10 < 25, Right

30 > 25, Left

25 = 25, **Tìm thấy**



5 < 25, Right

45 > 25, Left

30 > 25, Left

10 < 25, Right

25 = 25, **Tìm thấy**

Tìm giá trị x

```
TNode* findTNodeX(TNode* root, ItemType x)
{ // Dùng đệ qui
    if(!root)  return NULL;
    if(root→Info == x)
        return root;
    if(root→Info > x)
        return findTNodeX(root→Left, x);
    else
        return findTNodeX(root→Right, x);
}
```

Tìm giá trị x

```
TNode* findTNodeX(TNode* root, ItemType x)
{   //Không dùng đệ qui
    TNode* p = root;
    while( p &&  p→Info != x )
    {
        if( p→Info > x )        p = p→Left;
        else                    p = p→Right;
    }
    return p;
}
```


Tìm giá trị lớn nhất trên cây NPTK

```
TNode* maxTNodeBSTree(TNode* root)
{//Hàm tìm nút có giá trị lớn nhất trên cây (là nút bên phải nhất)
    TNode* p=root;
    while(p→Right != NULL)
        p = p→Right;
    return (p);
}
```

Duyệt cây

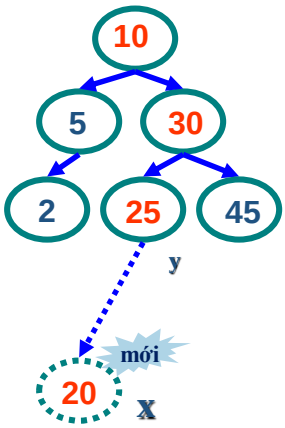
- ***Lưu ý*:** Cây nhị phân tìm kiếm duyệt theo traverseLNR thì thứ tự khóa tăng dần.
- **Ví dụ:** Đếm số nút lớn hơn x

Đếm số phần tử lớn hơn x

```
int countGreaterThanX(TNode* root, ItemType x)
{
   if(!root)   return  0;
   int nlx = countGreaterThanX(root→Left, x);
   int nrx = countGreaterThanX(root→Right, x);
   if( root→Info > x )
      return (1 + nlx + nrx);
   return (nlx + nrx);
}
```

Thêm x vào cây

- Thực hiện tìm kiếm giá trị x
- Tìm đến cuối nút y(nếu x không tồn tại trong cây)
- Nếu **x < y**, thêm nút lá x bên trái của y
- Nếu **x > y**, thêm nút lá x bên phải của y



Thêm x vào cây

```
int insertTNode(TNode* &root, TNode* p)
{//Ham chen 1 nut vao cay NPTK
    if(p == NULL) return 0; //Thêm không thành công
    if(root == NULL)  { //Cây đang rỗng
        root = p;
        return 1; //Thêm thành công
    }
    if(root→Info == p→Info)
        return 0; //Bị trùng khóa nên không thêm nữa
    if(p→Info < root→Info)
        insertTNode(root→Left, p);//thêm vào nhánh trái
    else
        insertTNode(root→Right, p);//thêm vào nhánh phải
    return 1; //Thêm thành công
}
```

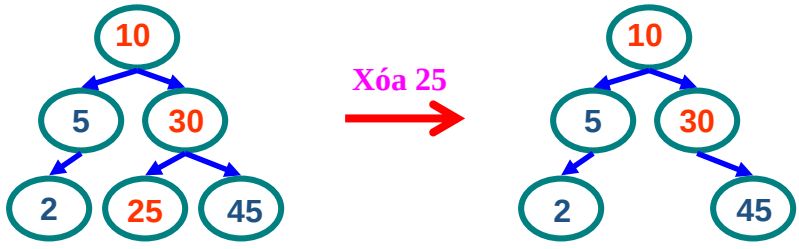
Xóa phần tử khỏi cây

Xóa nhưng phải đảm bảo vẫn là cây BST

- **Thực hiện tìm nút có giá trị x.**
- **Nếu nút là nút lá, xóa nút.**
- **Ngược lại**
  - Thay thế nút bằng một trong hai nút sau.
    - Y là nút lớn nhất của cây con bên trái
    - Z là nút nhỏ nhất của cây con bên phải
  - Chọn nút Y hoặc Z để thế chỗ.
  - Giải phóng nút có giá trị x.

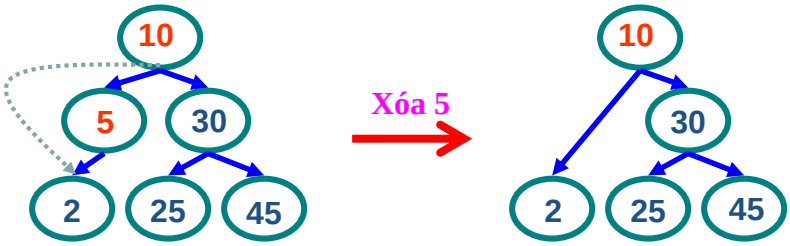
Ví dụ xóa x = 25

- **Trường hợp 1:** nút p là nút lá, xóa bình thường



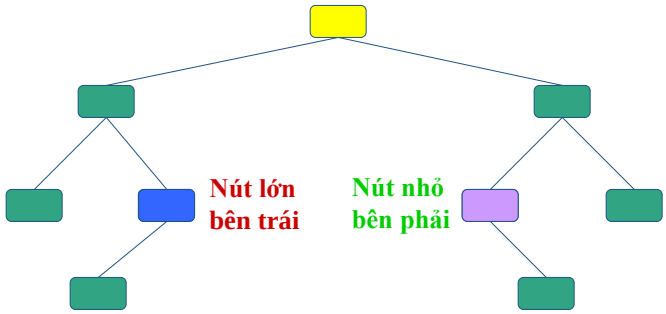
Ví dụ xóa x = 5

- **Trường hợp 2:** p chỉ có 1 cây con, cho nút cha của p trỏ tới nút con duy nhất của nó, rồi hủy p



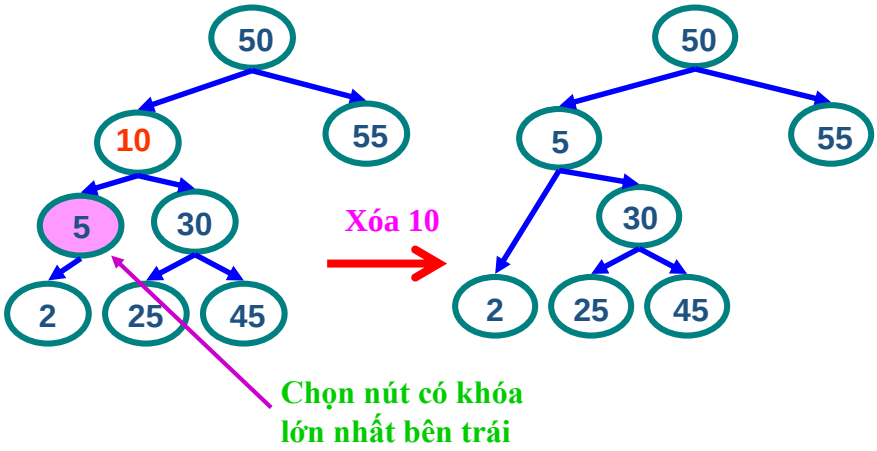
Ví dụ

- **Trường hợp 3:** nút p có 2 cây con, chọn nút thay thế theo 1 trong 2 cách như sau:
  - Nút lớn nhất trong cây con bên trái
  - Nút nhỏ nhất trong cây con bên phải



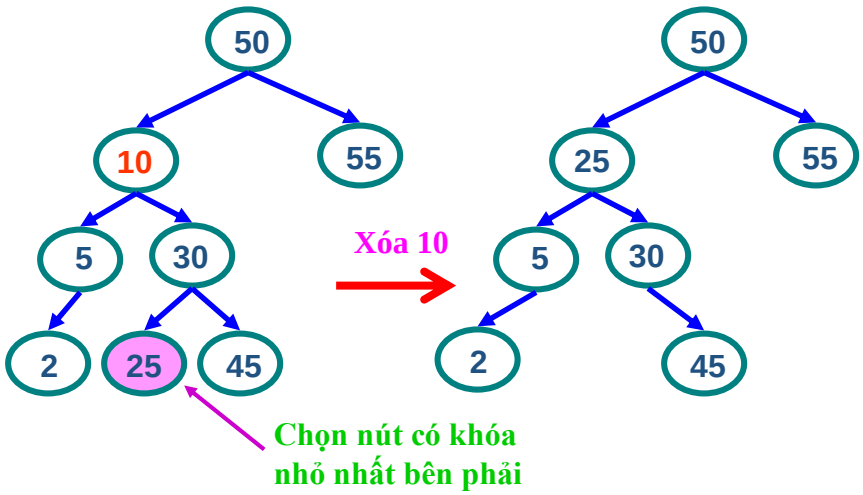
Ví dụ

- **Xóa nút 10:** cách 1

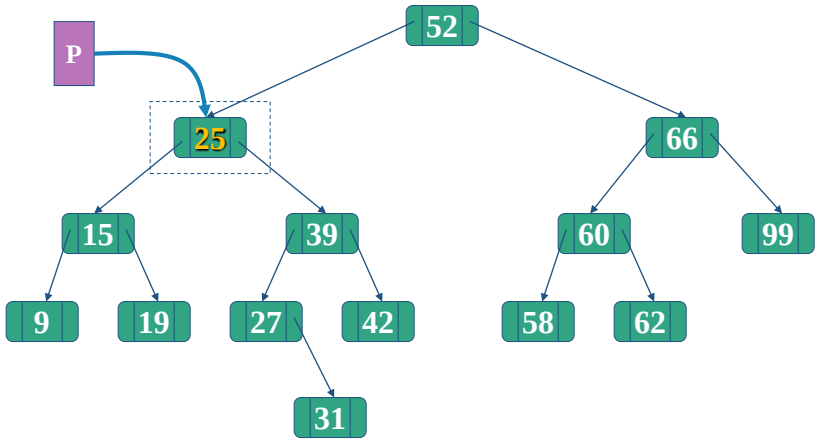


Ví dụ

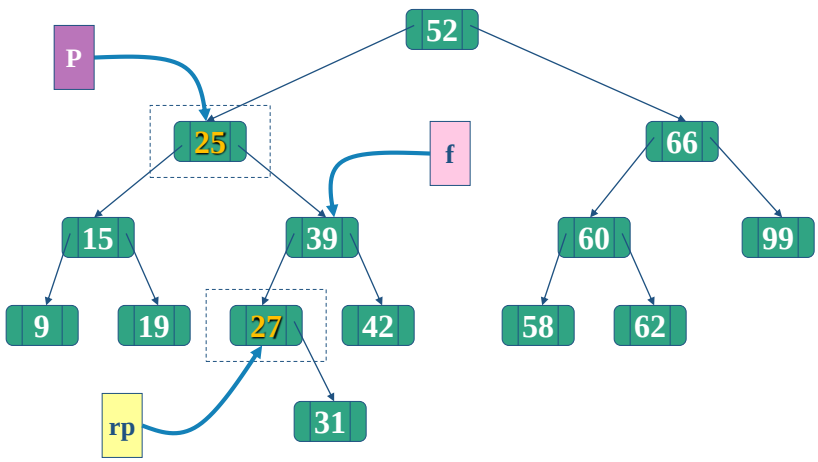
- **Xóa nút 10:** cách 2



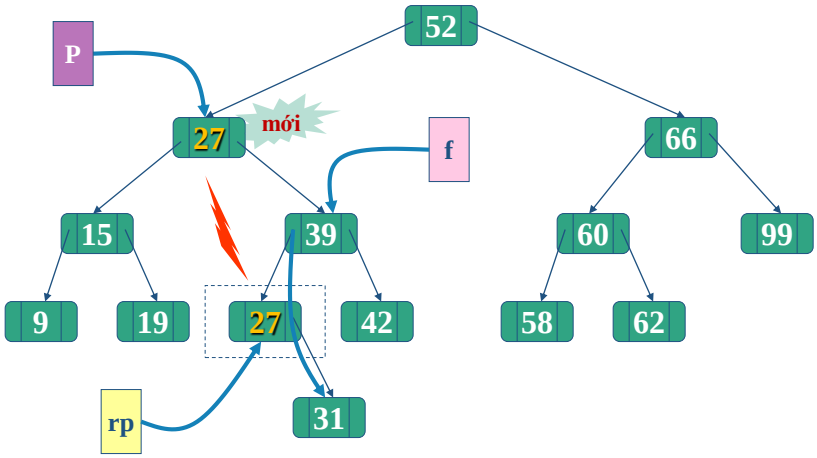
Ví dụ xóa x = 25



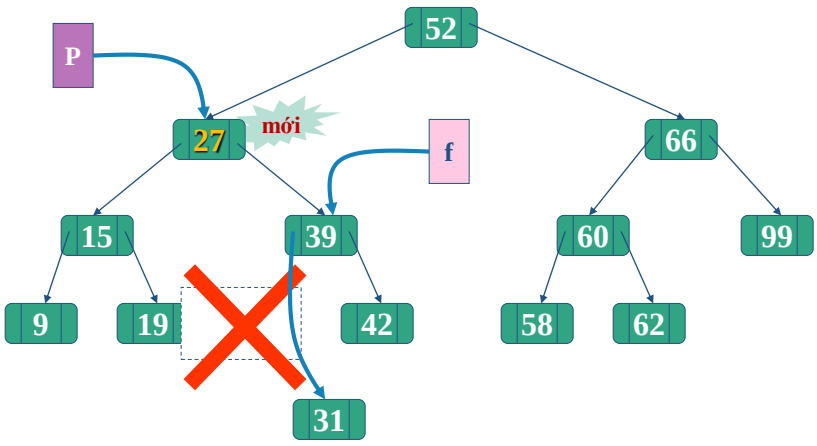
Ví dụ xóa x = 25



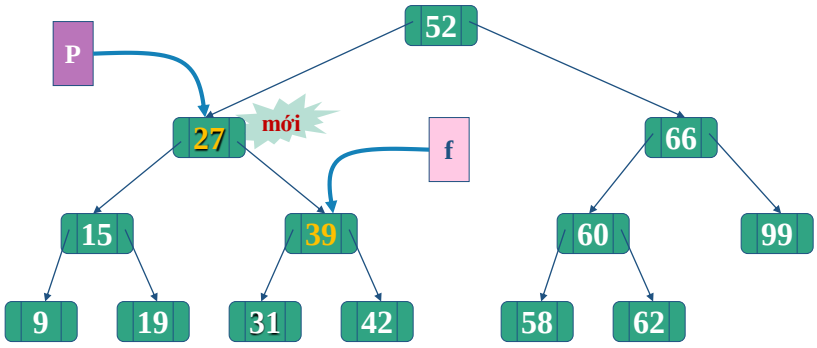
Ví dụ xóa x = 25



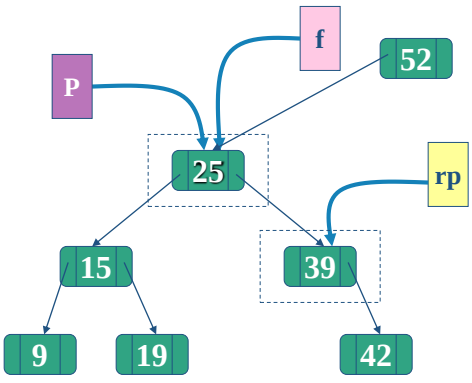
Ví dụ xóa x = 25



Ví dụ xóa x = 25

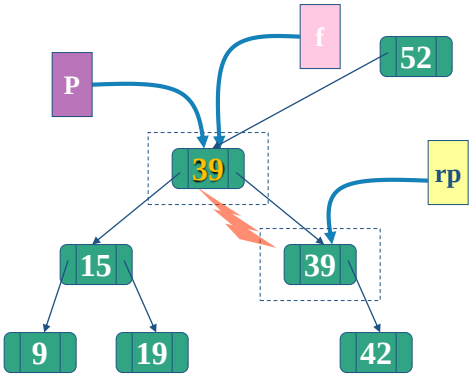


Ví dụ xóa x = 25



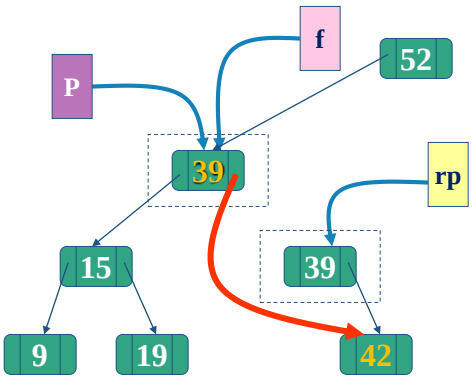
Trường hợp đặc biệt:
f == p
Nút thế mạng rp là nút con phải của nút p cần xóa

Ví dụ xóa x = 25



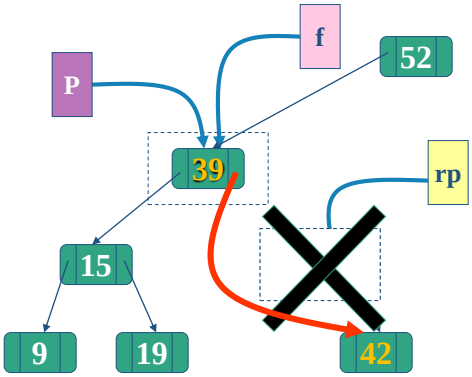
Trường hợp đặc biệt:
f == p
Nút thế mạng rp là nút con phải của nút p cần xóa
- Đưa giá trị của nút rp lên nút p

Ví dụ xóa x = 25



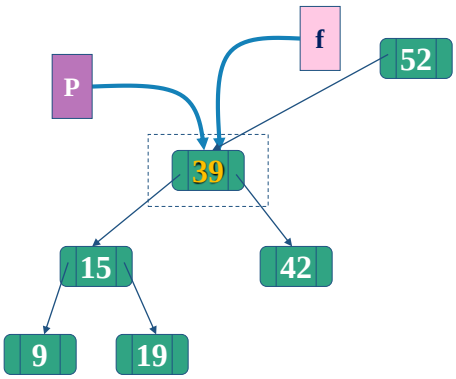
Trường hợp đặc biệt:
f == p
Nút thế mạng rp là nút con phải của nút p cần xóa
- Chuyển liên kết phải của p đến liên kết phải của rp

Ví dụ xóa x = 25



Trường hợp đặc biệt:
f == p
Nút thế mạng rp là nút con phải của nút p cần xóa
- xóa nút rp

Ví dụ xóa x = 25



Trường hợp đặc biệt:
f == p
Nút thế mạng rp là nút con phải của nút p cần xóa
- Sau khi xóa

Giải thuật

Nếu T = NULL ⇒ thoát

Nếu T→Info > x ⇒ Xoa(T→Left, x)

Nếu T→Info <x ⇒ Xoa(T→Right, x)

Nếu T→Info = x

p = T

Nếu T có 1 nút con thì T trỏ đến nút con đó

Giải thuật

Ngược lại có 2 con

Gọi f = p và rp = p→Right;

Tìm nút rp: rp→Left = NULL và nút f là nút cha nút rp

Thay đổi giá trị nội dung của p và rp

Nếu **f = p** (trường hợp đặc biệt) thì:

f→Right = rp→Right;

Ngược lại:

f→Left = rp→Right;

Xóa rp; // xóa nút thế mạng rp

Xóa nút có giá trị là x

```
int deleteTNode(TNode* &root, ItemType x)
{
   if(!root)  return 0;
   if(root→Info > x) //tìm bên trái
      return deleteTNode(root→Left, x);
   else if(root→Info < x) //tìm bên phải
      return  deleteTNode(root→Right, x);
   else
   {
      TNode* p = root;
```


Xóa nút có giá trị là x

```
      if(!root→Left)//khi cay con khong co nhanh trai
         root = root→Right;
      else if(!root→Right)//khi cay con khong co nhanh phai
         root = root→Left;
      else
      {//khi cay con co ca 2 nhanh, chon min cua nhanh phai de the mang
         TNode* p = root;
         TNode* rp = findTNodeReplace(p);
         deleteTNode(rp);
      }
   }
}
```

Tìm nút thế mạng cho nút bị xóa

```
TNode* findTNodeReplace(TNode* &p)
{//Ham tim nut rp nho nhat tren cay con phai de the mang cho nut p
    TNode* f = p,* rp = p→Right;
    while(rp→Left != NULL) {
        f = rp; //Luu cha cua rp
        rp = rp→Left; //rp qua ben trai
    }
    p→Info = rp→Info; //tim duoc phan tu the mang cho p la rp
    if(f == p) //neu cha cua rp la p
        f→Right = rp→Right;
    else
        f→Left = rp→Right;
    return rp; //Tra ve nut rp la nut the mang cho p
}
```


Bài tập

- **Tìm phần tử max âm trên cây.**
- **Đếm có bao nhiêu phần tử chẵn trên cây.**


Thank for you attention!
See you next week!