HCMUTE

CHƯƠNG 6. CÂY (TREE)

HCMUTE

Nội dung

1. Khái niệm cây_ Cây Nhị phân
2. Cây nhị phân tìm kiếm

HCMUTE

1 Khái niệm cây_ Cây Nhị phân

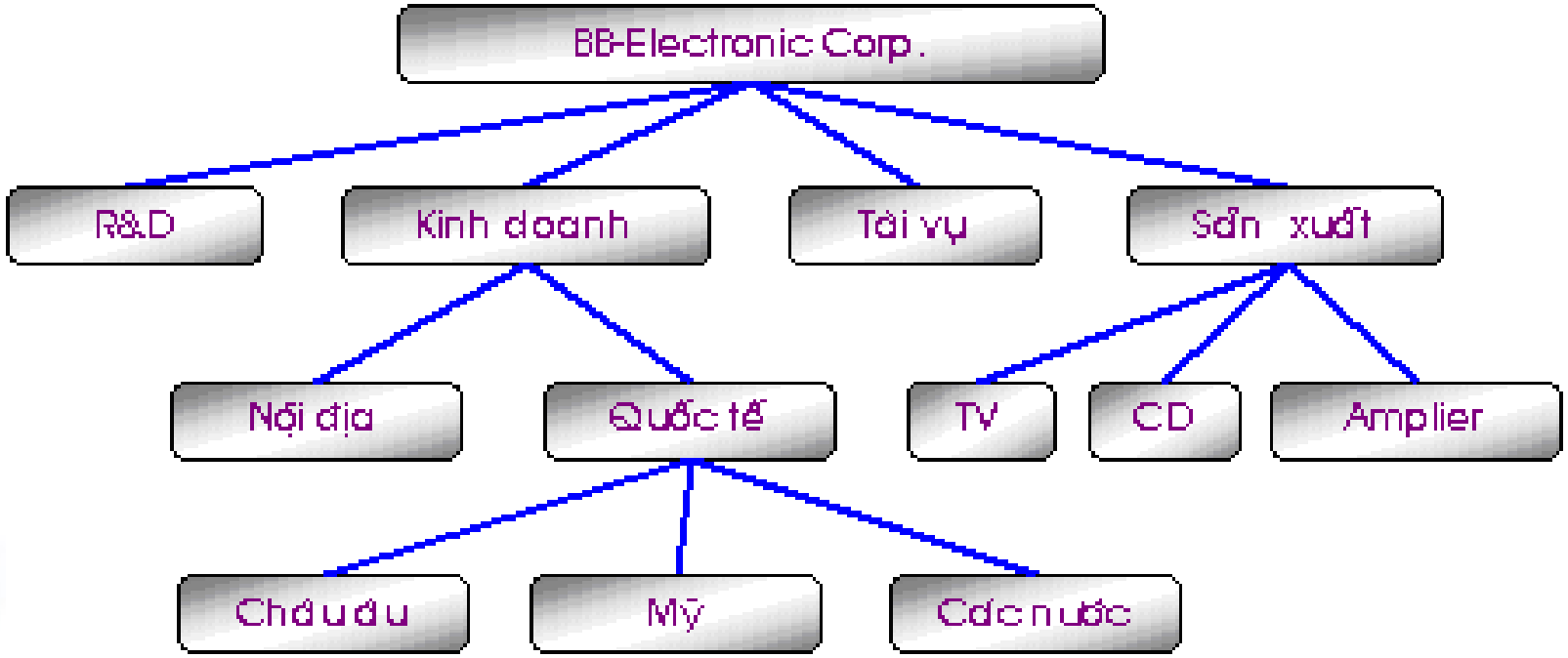
HCMUTE

Định nghĩa cây

- Cây là một tập hợp T các phần tử (gọi là nút của cây), trong đó có một nút đặc biệt gọi là nút gốc, các nút còn lại được chia thành những tập rời nhau T_1, T_2, …,T_n theo quan hệ phân cấp, trong đó T_i cũng là 1 cây. Mỗi nút ở cấp i sẽ quản lý một số nút ở cấp i+1. Quan hệ này người ta gọi là quan hệ cha – con.

Định nghĩa cây

- VD cây thể hiện sơ đồ tổ chức của một công ty



HCMUTE

Một số khái niệm

- Bậc của một nút: là số cây con của nút đó .
- Bậc của một cây: là bậc lớn nhất của các nút trong cây
- Nút gốc: là nút không có nút cha.
- Nút lá: là nút có bậc bằng 0.
- Mức của một nút:
  - Mức (gốc (T)) = 1.
  - Gọi T1, T2, T3, ... , Tn là các cây con của T0 :

    Mức (T1) = Mức (T2) = . . . = Mức (Tn) = Mức (T0) + 1.
- Độ dài đường đi từ gốc đến nút x: là số nhánh cần đi qua kể từ gốc đến x.

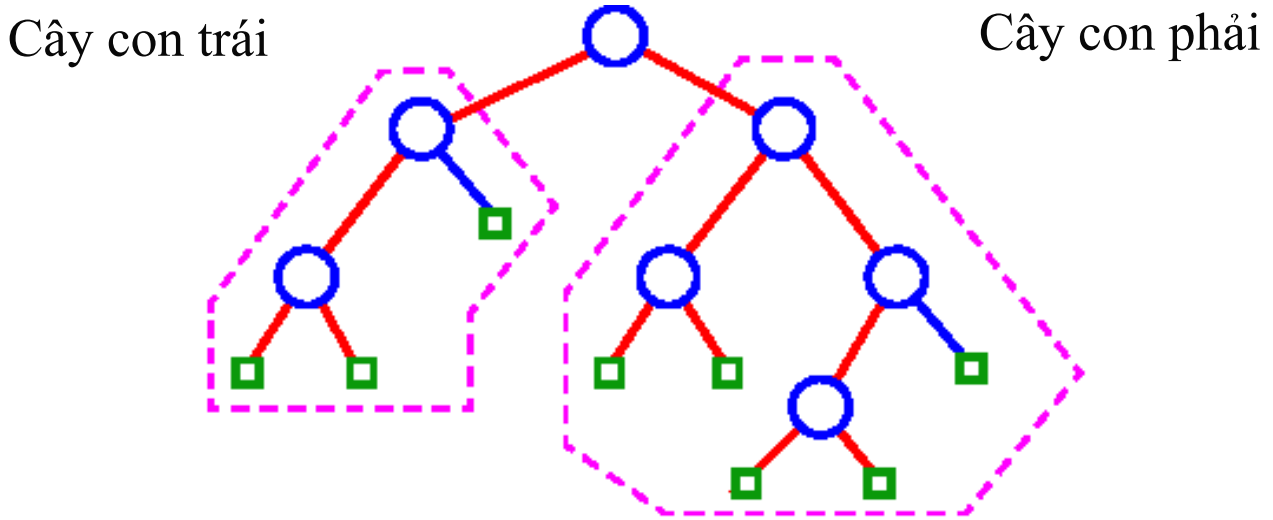
HCMUTE

Khái niệm Cây

- **Nhận xét:**
  - Trong cấu trúc cây không tồn tại chu trình.
  - Tổ chức 1 cấu trúc cây cho phép truy cập nhanh đến các phần tử của nó.

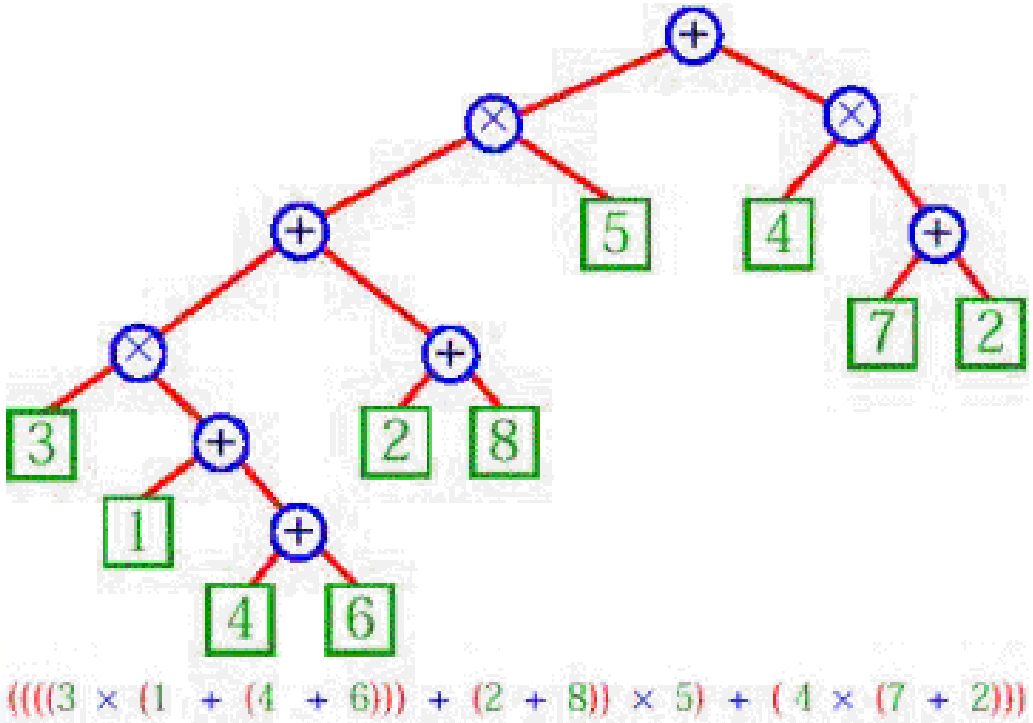
Cây nhị phân

- **Mỗi nút có tối đa 2 cây con**



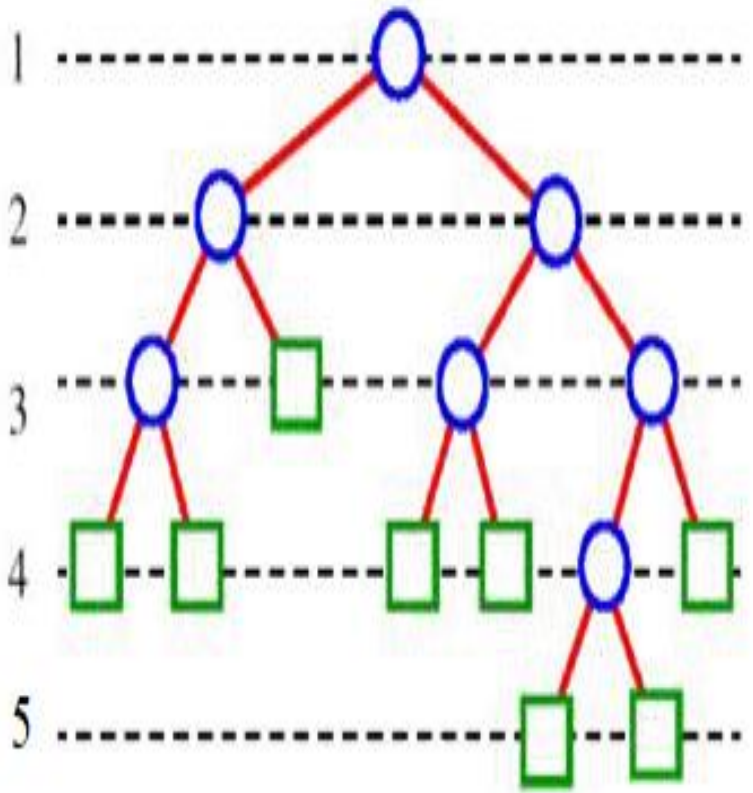
Cây nhị phân

- VD dùng cây NP biểu diễn biểu thức toán học



Một số tính chất trên cây nhị phân

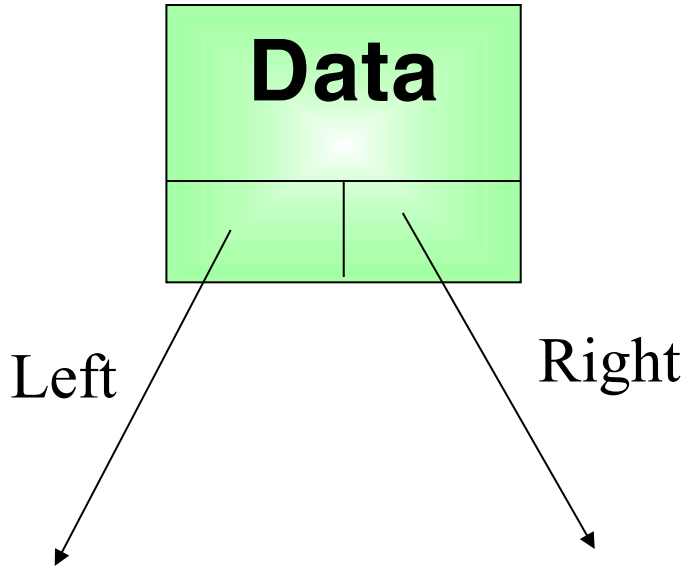
- Số nút nằm ở mức $i \leq 2^{i-1}$.
- Số nút lá $\leq 2^{h-1}$, với h là chiều cao của cây.
- Chiều cao của cây $h \geq \log_2 N$. (Với N = số nút trong cây)



Cấu trúc dữ liệu

```
//Khai báo Node
class Node
{
    public Node LeftNode { get; set; }
    public Node RightNode { get; set; }
    public int Data { get; set; }
    //Tạo nút mới có nội dung là theData
    public Node(int theData)
    {
        Data = theData;
        LeftNode = null;
        RightNode = null;
    }

}
```

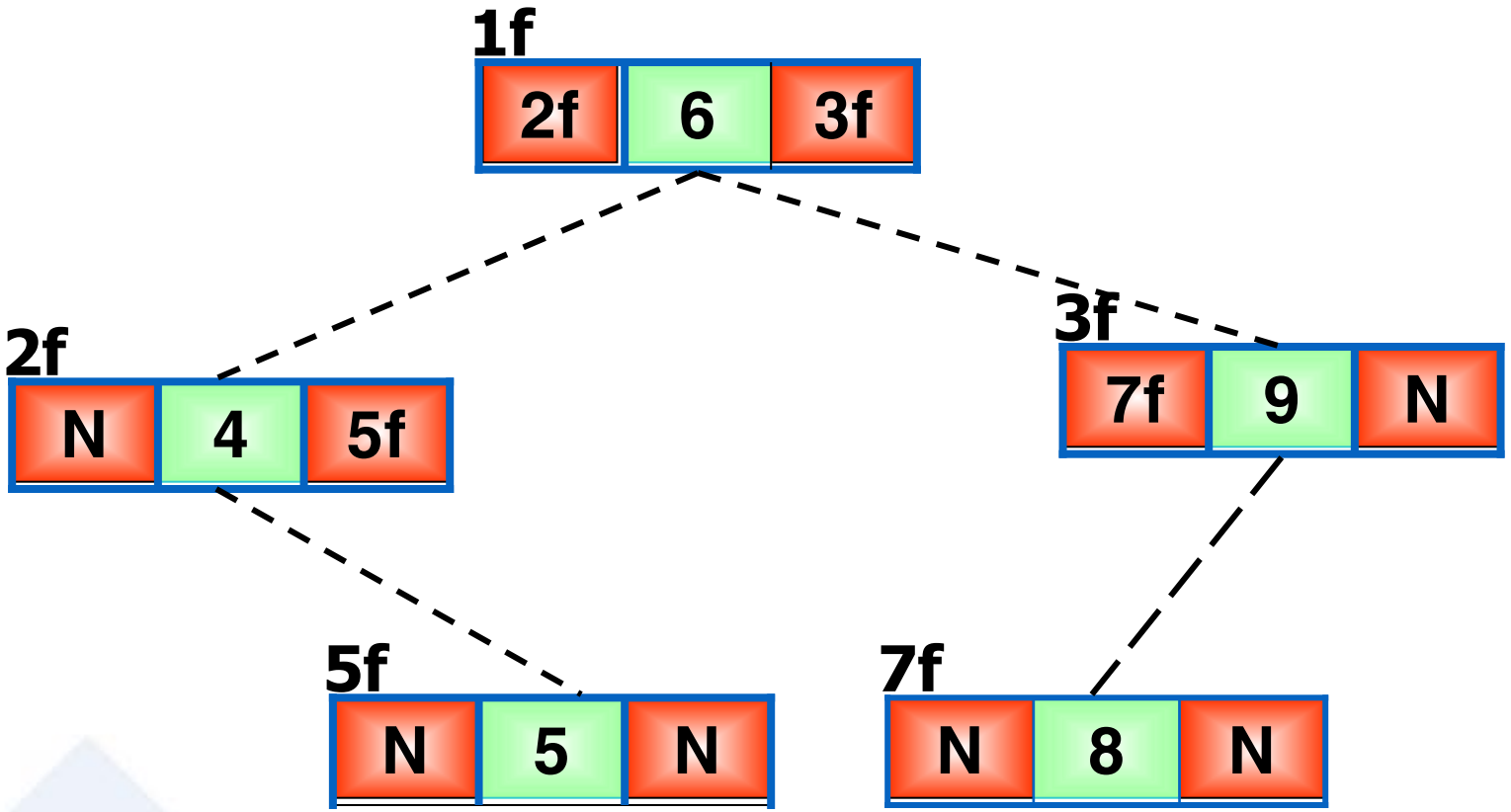


Cấu trúc dữ liệu

- Cấu trúc cây nhị phân có một nút rỗng, hàm kiểm tra cây rỗng.

```
//Khai báo lớp cây nhị phân có nút gốc rỗng
class BinaryTree
{
    private Node Root { get; set; }
    //Hàm tạo cây có một nút rỗng
    public BinaryTree()
    {
        Root = null;
    }
    //Kiểm tra cây rỗng
    public bool IsEmpty()
    {
        bool result = (Root == null);
        return result;
    }
}
```

Ví dụ Cây được tổ chức trong bộ nhớ

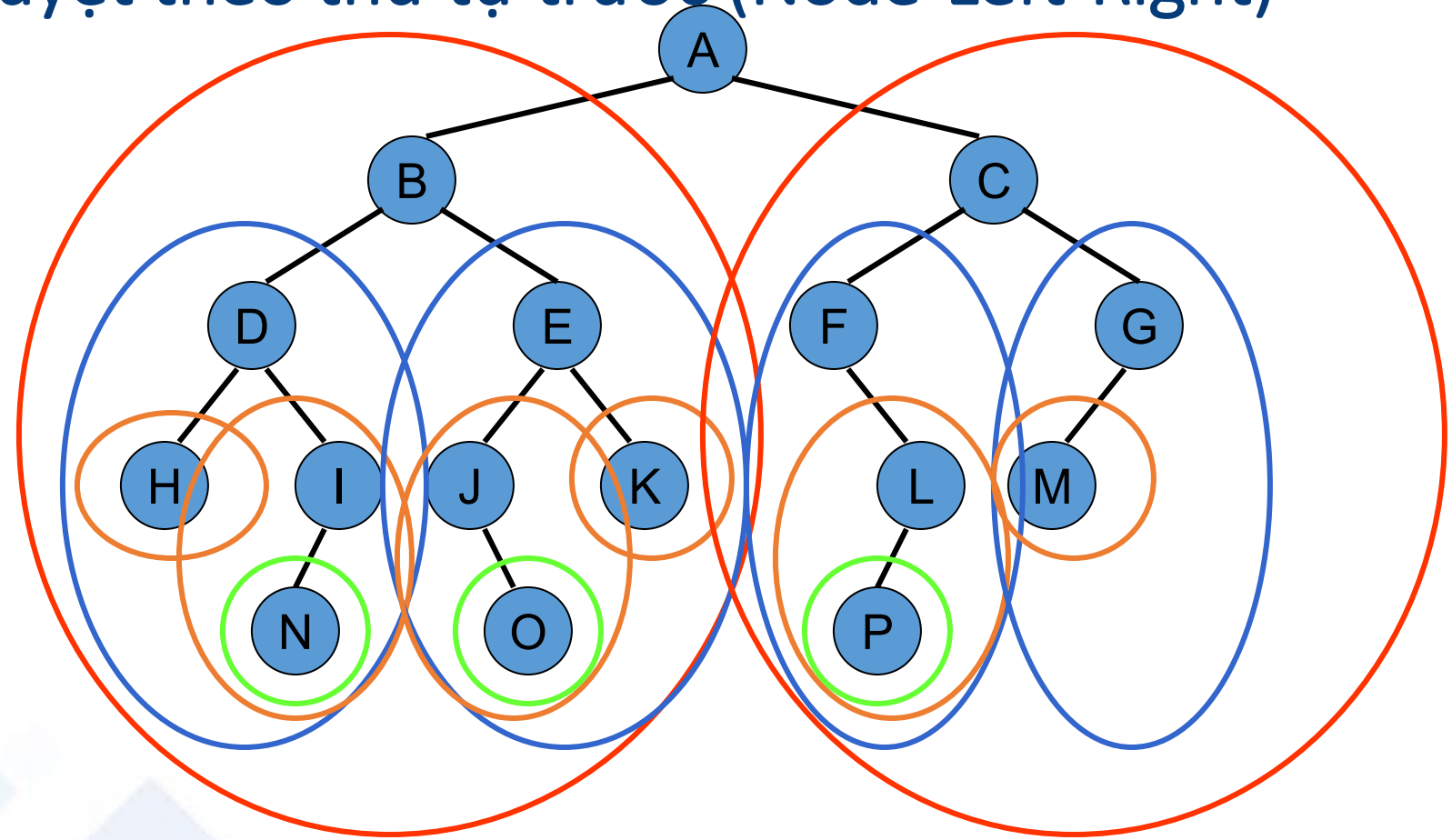


HCMUTE

Duyệt cây nhị phân

- PreOrder Traversal - duyệt theo thứ tự trước NLR.
- InOrder Traversal - duyệt theo thứ tự giữa LNR.
- PostOrder Traversal- duyệt theo thứ tự cuối LRN.
- Độ phức tạp O ($\log_2(h)$). Trong đó h là chiều cao cây.

Duyệt theo thứ tự trước (Node-Left-Right)



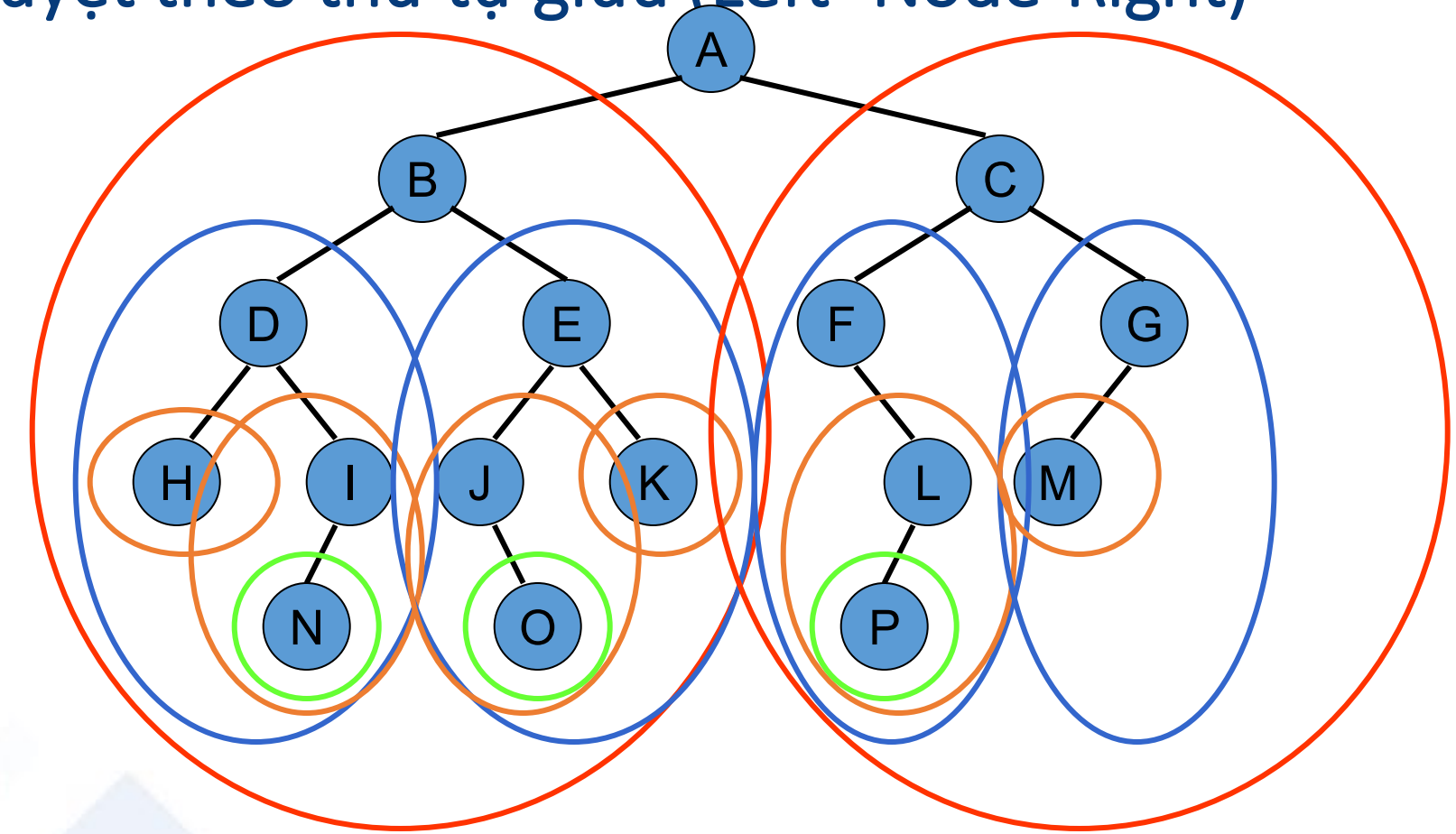
Kết quả: A B D H I N E J O K C F L P G M

HCMUTE

Duyệt trước - NLR

```
void PreOrderTraversal(BinaryTreeNode node)
{
      if (node != null)
      {
            Console.Write(node.Data + " ");
            PreOrderTraversal(node.LeftNode);
            PreOrderTraversal(node.RightNode);
      }
}
```

Duyệt theo thứ tự giữa (Left- Node-Right)



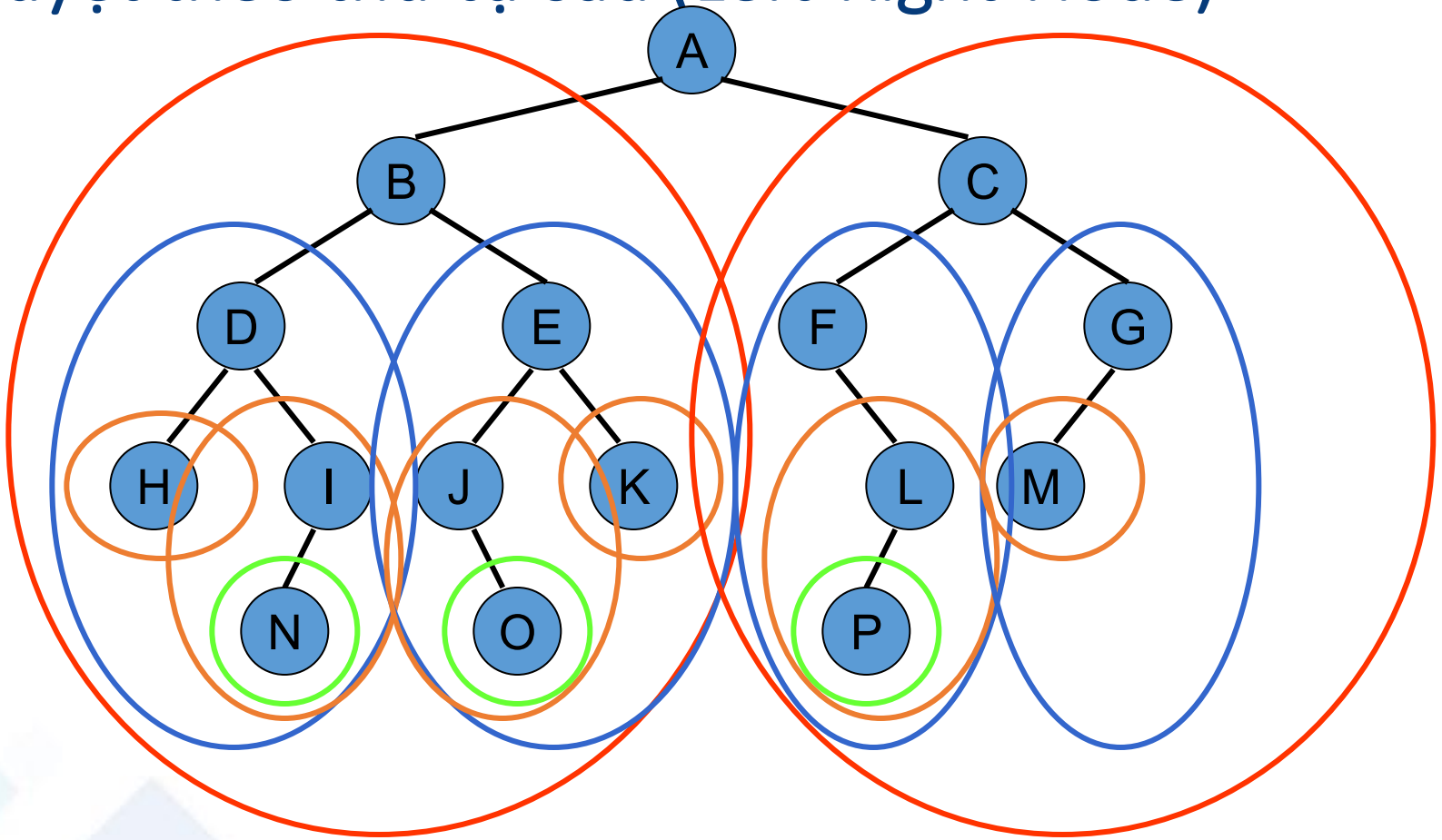
Kết quả: H D N I B J O E K A F P L C M G

Duyệt giữa - LNR

```
void InorderTraversal(BinaryTreeNode node)
{
      if (node != null)
      {
          InorderTraversal(node.LeftNode);
          Console.Write(node.Data + " ");
          InorderTraversal(node.RightNode);
      }

}
```

Duyệt theo thứ tự sau (Left-Right-Node)



Kết quả: H N I D O J K E B P L F M G C A

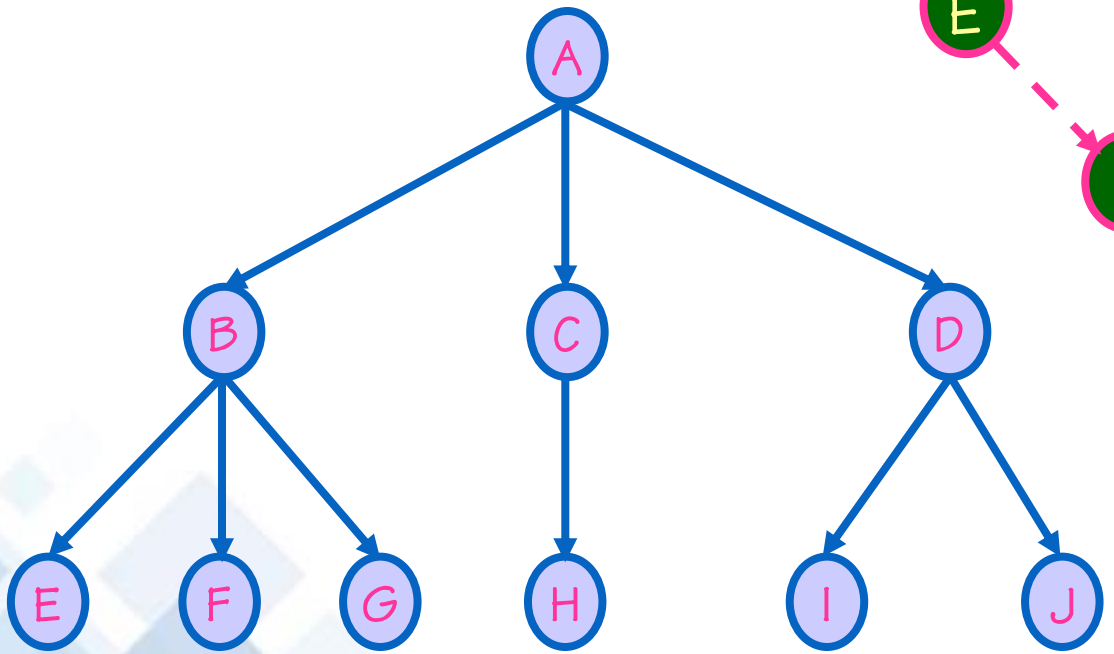
Duyệt sau (LRN)

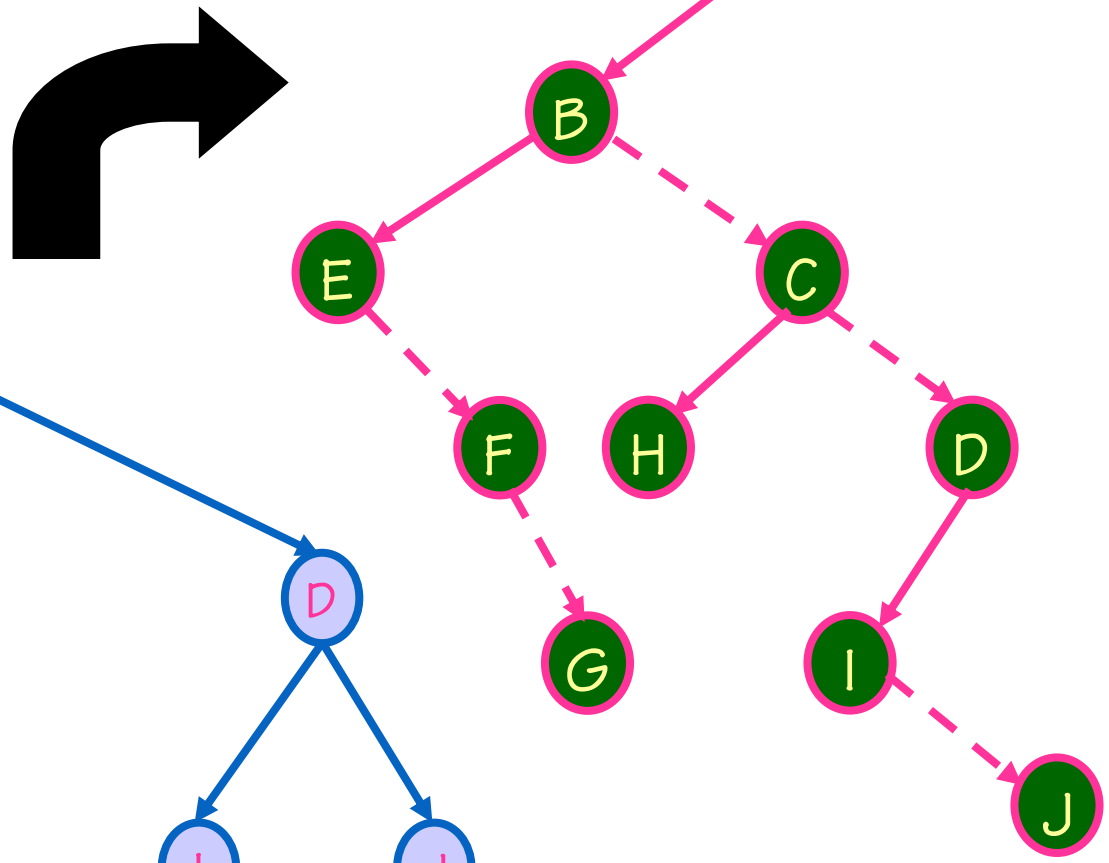
```
void PostOrderTraversal(BinaryTreeNode node)
{
        if (node != null)
        {
            PostOrderTraversal(node.LeftNode);
            PostOrderTraversal(node.RightNode);
            Console.Write(node.Data + " ");
        }

}
```

HCMUTE

Biểu Diễn Cây Tổng Quát Bằng Cây Nhị Phân





Biểu Diễn Cây Tổng Quát Bằng Cây Nhị Phân

- Quy tắc:
  - Giữ lại nút con trái nhất làm nút con trái.
  - Các nút con còn lại chuyển thành nút con phải.
- Như vậy, trong cây nhị phân mới, con trái thể hiện quan hệ cha con và con phải thể hiện quan hệ anh em trong cây tổng quát ban đầu.

HCMUTE

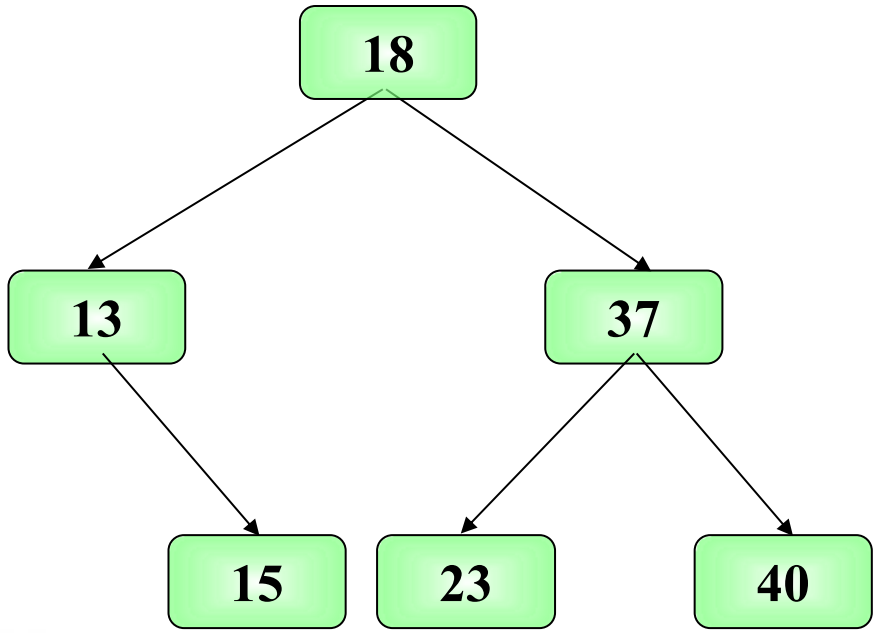
2 Cây nhị phân tìm kiếm (Binary Search Tree – BST)

Định nghĩa cây nhị phân tìm kiếm

Cây nhị phân BST phải bảo đảm nguyên tắc bố trí khoá tại mỗi nút:

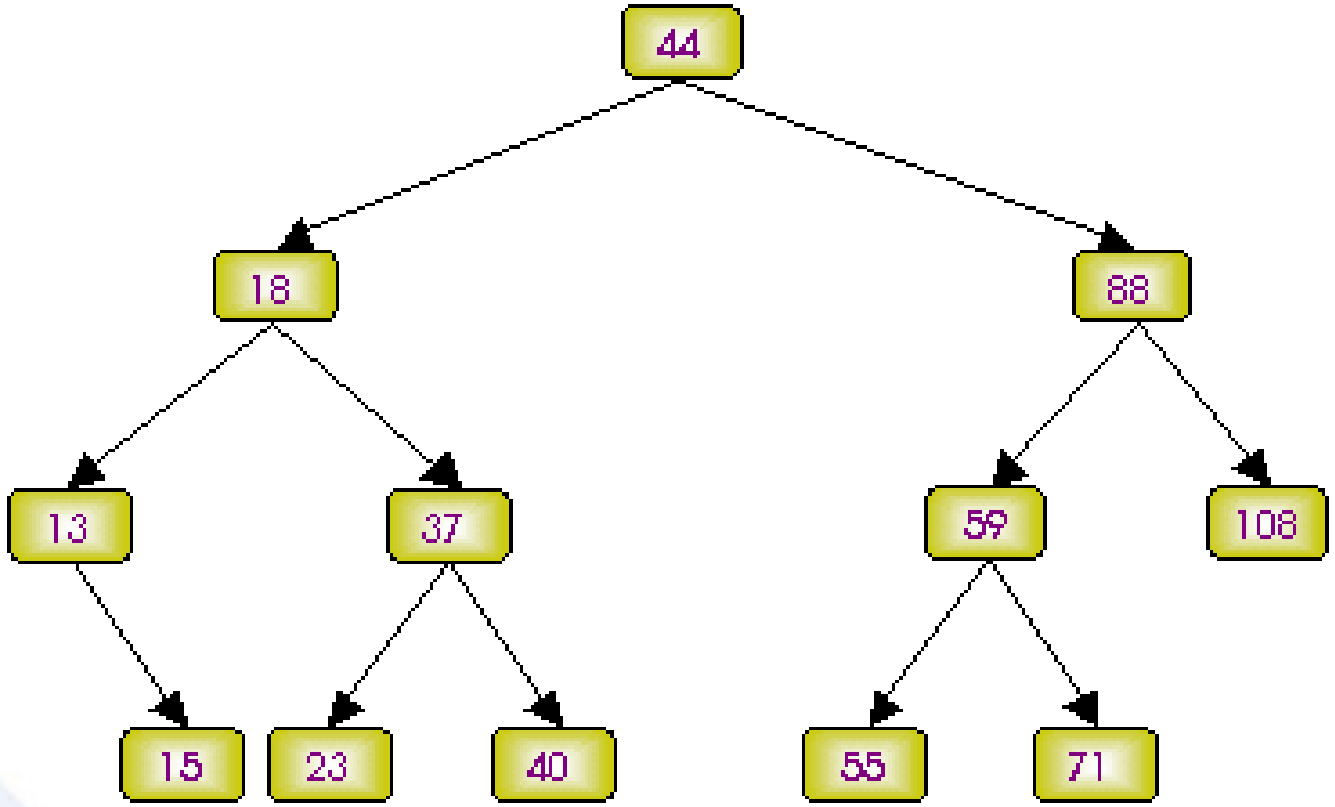
- Các nút trong **cây trái nhỏ** hơn nút hiện hành.
- Các nút trong **cây phải lớn** hơn nút hiện hành.

Ví dụ:



HCMUTE

Ví dụ



Ưu điểm của cây BST

Nhờ trật tự bố trí khóa trên cây :

- Định hướng được khi tìm kiếm

Cây gồm N phần tử :

- Chi phí tìm kiếm trung bình chỉ khoảng $\log_2 N$.

Cấu trúc dữ liệu của cây BST

- Cấu trúc một Node:

```
public class Node
{
    public int Data { get; set; }
    public Node RightNode { get; set; }
    public Node LeftNode { get; set; }
    public void Display()
    {
        Console.Write(Data + " ");
    }
}
```

Cấu trúc dữ liệu của cây BST

- Cấu trúc dữ liệu của cây BST rỗng

```
public class Node
{
    public int Data { get; set; }
    public Node RightNode { get; set; }
    public Node LeftNode { get; set; }
    public Node()
    {
        //Assign data to the new node
        //set left and right children to null
        Data = 0;
        LeftNode = null;
        RightNode = null;
    }
}
```

HCMUTE

Các thao tác trên cây BST

- Tạo một cây rỗng
- Tạo một nút có trường Key bằng x
- Tìm một nút có khoá bằng x trên cây
- Thêm một nút vào cây nhị phân tìm kiếm
- Xoá một nút có Key bằng x trên cây

Tạo cây rỗng

- Cây rỗng -> địa chỉ nút gốc bằng NULL

```
class BinarySearchTree
{
    public Node Root;
    //Khởi tạo cây BST rỗng
    public BinarySearchTree(){Root = null;}
}
```

```
class Program
{
    static void Main(string[] args)
    {
        BinarySearchTree myBST = new BinarySearchTree();
    }
}
```

HCMUTE

Tạo cây có một nút x

```
public class BinarySearchTree
{
        public Node Root;
        public BinarySearchTree(int x)
        {
             Root.Data = x;
        }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        int x = 5;
        BinarySearchTree myBST = new BinarySearchTree(x);
    }
}
```

HCMUTE

Tìm nút có khoá bằng key

Ta có quy trình như sau:

- Nếu Node hiện tại có giá trị = giá trị cần tìm, trả về true và kết thúc.
- Nếu Node hiện tại có giá trị > giá trị cần tìm, ➔gọi đệ quy tìm ở cây con bên trái.
- Nếu Node hiện tại có giá trị < giá trị cần tìm ➔gọi đệ quy tìm ở cây con bên phải
- Nếu tìm đến hết cây(Node đó = NULL) mà không tìm thấy ➔ trả về false và kết thúc.

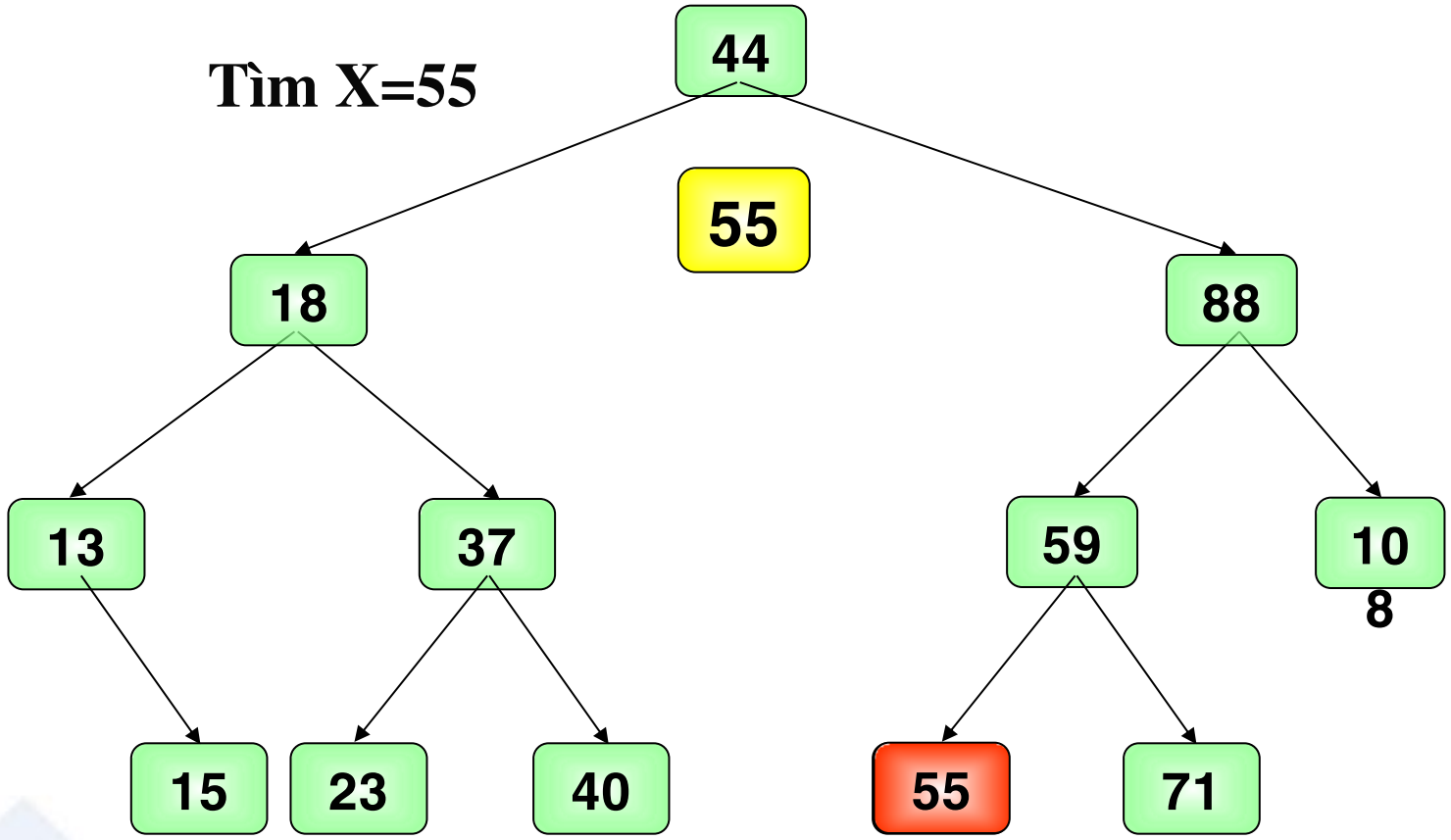
Tìm nút có khoá bằng key (không dùng đệ quy)

```
//Tìm phần tử có giá trị bằng key, trả về node
public Node Find(int key)
{
    Node current = Root;
    while (current.Data != key)
    {
        if (key < current.Data)
            current = current.LeftNode;
        else
            current = current.RightNode;
        if (current == null) //đã duyệt hết phần tử
            break;
    }
    return current;
}
```

Tìm nút có khoá bằng key (dùng đệ quy)

```
public Node Find(Node root, int key)
{
    Node p = null;
    if (root!=null)
    {
        if (root.Data == key)
        {
            p = root;
        }
        //Search in left subtree
        else if (root.Data > key)
        {
            p = Find(root.LeftNode, key);
        }
        //Search in right subtree
        else
        {
            p = Find(root.RightNode, key);
        }
    }
    return p;
}
```

Minh họa tìm một nút



Tìm thấy X=55

HCMUTE

Thêm một Node vào cây BST

❖Ta có quy trình như sau:

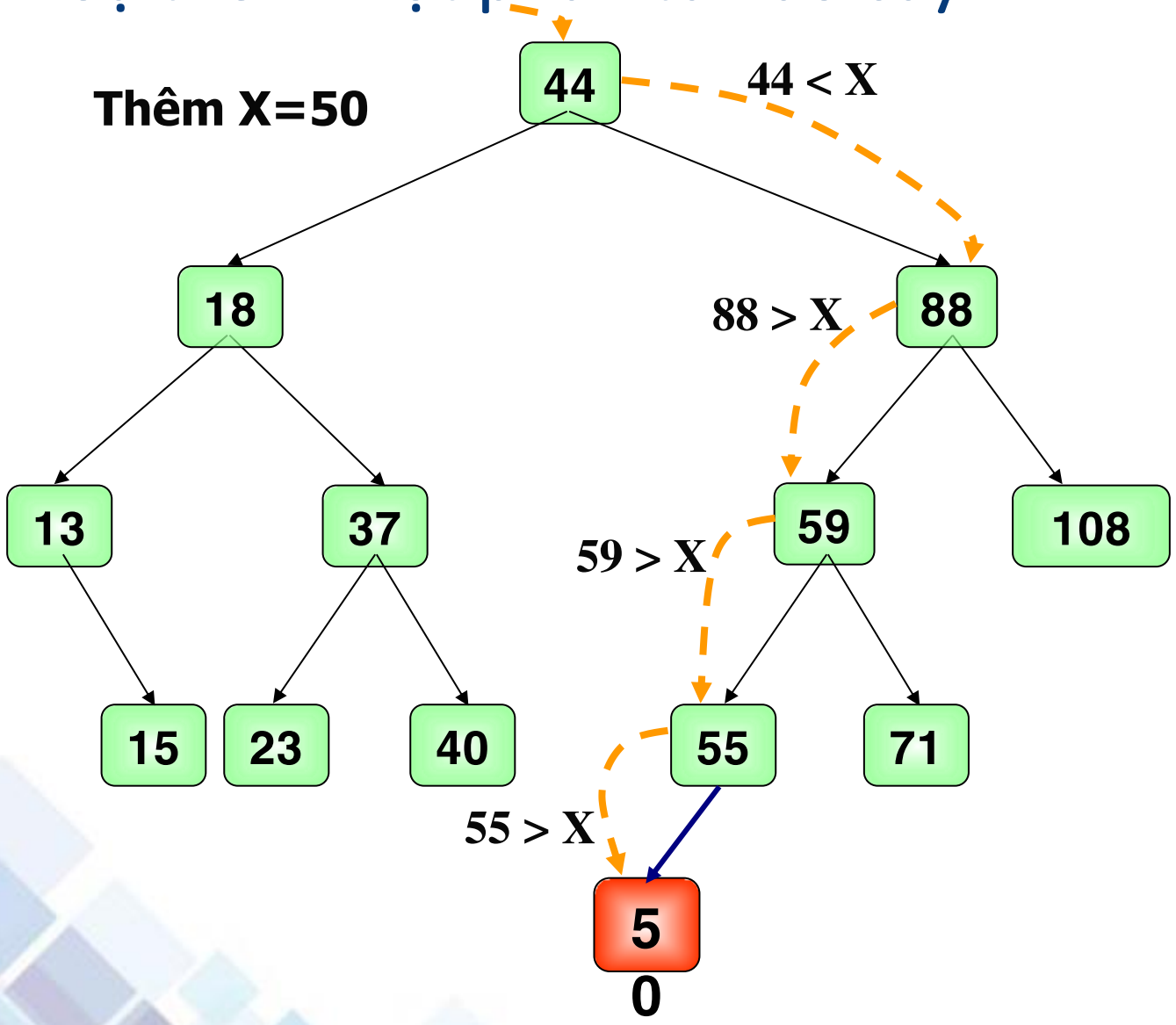
- Nếu Node hiện tại = NULL, đó là vị trí cần thêm. Thêm vào BST và kết thúc.
- Nếu giá trị cần thêm < giá trị root hiện tại ➔gọi đệ quy Insert vào cây con bên trái.
- Nếu giá trị cần thêm > giá trị root hiện tại ➔gọi đệ quy Insert vào cây con bên phải.

Thêm một nút

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

```
public void Insert(int item){// Thêm một nút mới cho BST
    Node newNode = new Node(); //Tạo nút mới
    newNode.Data = item;
    if (Root == null){//TH cây rỗng
        Root = newNode;
    }
    else{
        Node current = Root;
        Node parrent;
        while (true){
            parrent = current;
            if (item < current.Data){
                current = current.LeftNode;
                if (current == null){
                    parrent.LeftNode = newNode;
                    break;
                }
            }
            else{
                current = current.RightNode;
                if (current == null){
                    parrent.RightNode = newNode;
                    break;
                }
            }
        }
    }
}
```

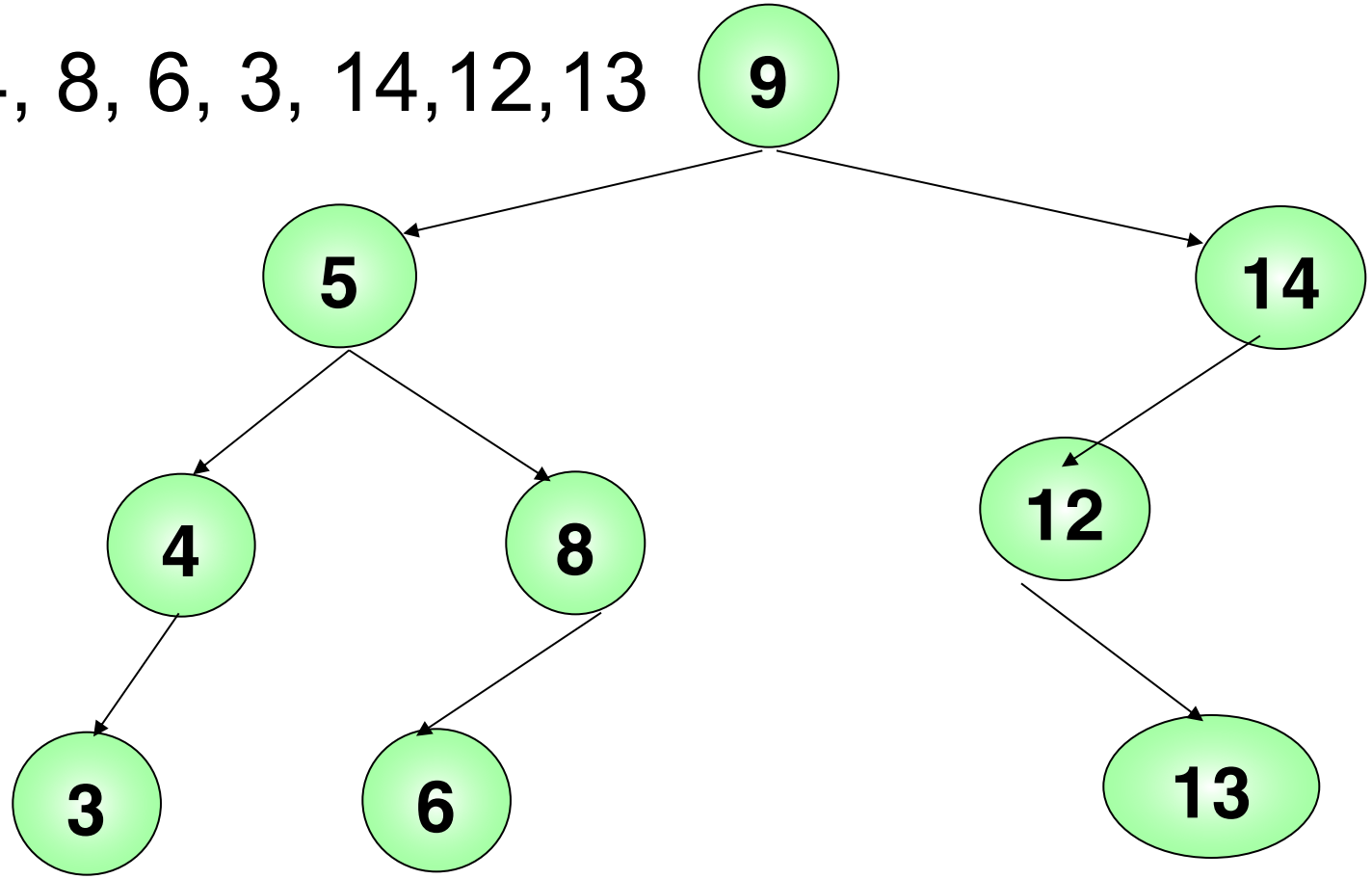
Minh hoạ thêm một phần tử vào cây



HCMUTE

Minh hoạ thành lập 1 cây từ dãy số

9, 5, 4, 8, 6, 3, 14,12,13



Xóa một nút 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 nút trên cây
  1. X là nút lá
  2. X chỉ có 1 cây con (cây con trái hoặc cây con phải)
  3. X có đầy đủ 2 cây con

Hủy một nút có khoá bằng X trên cây BST

- TH1: Ta xoá nút lá mà không ảnh hưởng đến các nút khác trên cây
- TH2: Trước khi xoá x ta liên kết cha của X với con duy nhất của X.

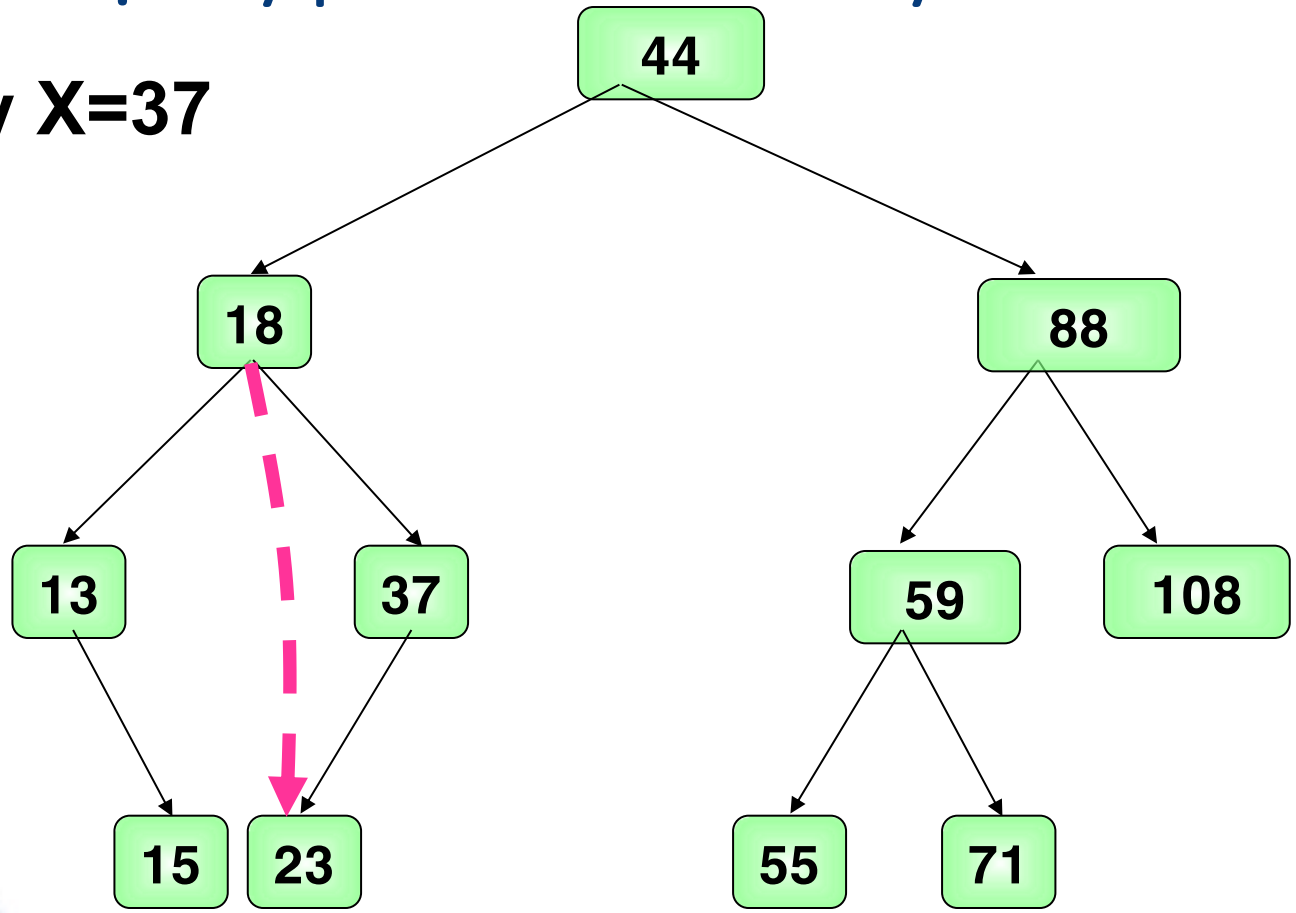
Hủy một nút có khoá bằng X trên cây BST

❖TH3: Ta dùng cách xoá gián tiếp như sau:

- Tìm Node của con trái nhất(giả sử nó là *leftmost)* của cây con bên phải của Node cần xóa. (Node thế mạng)
- Cập nhật giá trị của Node cần xóa = giá trị của Node *leftmost.*
- Gọi đệ quy hàm Delete xóa Node leftmost khỏi BST.

Minh hoạ hủy phần tử x có 1 cây con

Hủy X=37



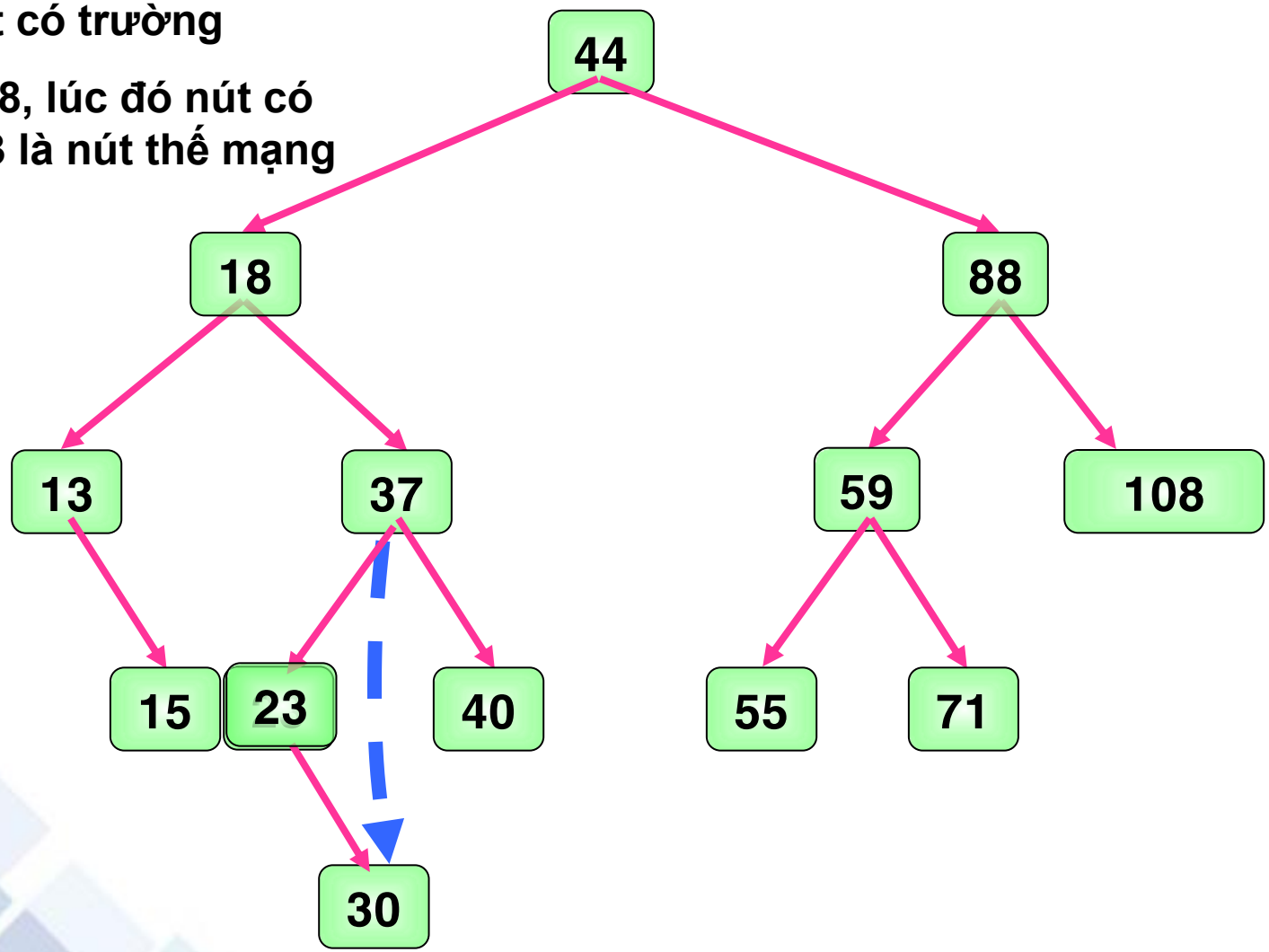
Hủy một nút có 2 cây con

- Cách tìm nút thế mạng Y cho X: Có 2 cách
  1. Nút Y là nút có khoá nhỏ nhất (trái nhất) bên cây con phải X
  2. Nút Y là nút có khoá lớn nhất (phải nhất) bên cây con trái của X

Minh họa hủy phần tử X có 2 cây con

Xoá nút có trường

Key = 18, lúc đó nút có khoá 23 là nút thế mạng



Cài đặt thao tác xoá nút có trường Key = x

- Tìm giá trị khoá trái nhỏ nhất:

```
public int MinValue(Node root)
{
    int minValue = root.Data;
    while (root.LeftNode != null)
    {
        minValue = root.LeftNode.Data;
        root = root.LeftNode;
    }
    return minValue;
}
```

Cài đặt thao tác xoá nút có trường Key = x

```
public Node DeleteNode(Node root, int key)
{
    if (root == null) return root; //Cây rỗng
    if (key < root.Data)
        root.LeftNode = DeleteNode(root.LeftNode, key);
    else if (key > root.Data)
        root.RightNode = DeleteNode(root.RightNode, key);
    else  // Nếu key = x thì xoá nút
    {
        if (root.LeftNode == null) //Nút lá hoặc nút có một nút con
            return root.RightNode;
        else if (root.RightNode == null)
            return root.LeftNode;
        //Tìm khoá nhỏ nhất cây con bên phải
        root.Data = MinValue(root.RightNode);
        // Xoá inorder successor
        root.RightNode = DeleteNode(root.RightNode, root.Data);
    }
    return root;
}
```

HCMUTE

Cài đặt thao tác xoá nút có trường Key = x

```
// Gọi hàm xoá nút có data = key
public void DeleteKey(int key)
{
    Root = DeleteNode(Root, key);
}
```