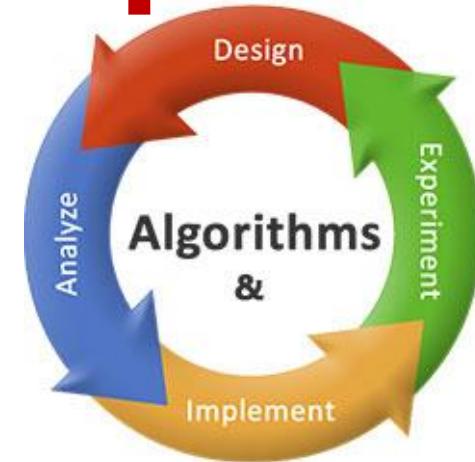




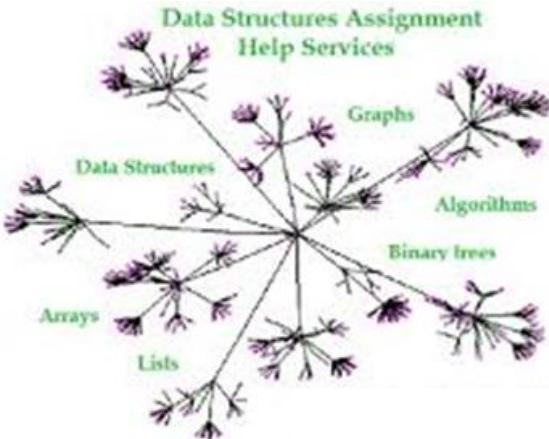
# CẤU TRÚC DỮ LIỆU

&

# GIẢI THUẬT



**Datastructures**



Lê Văn Hạnh

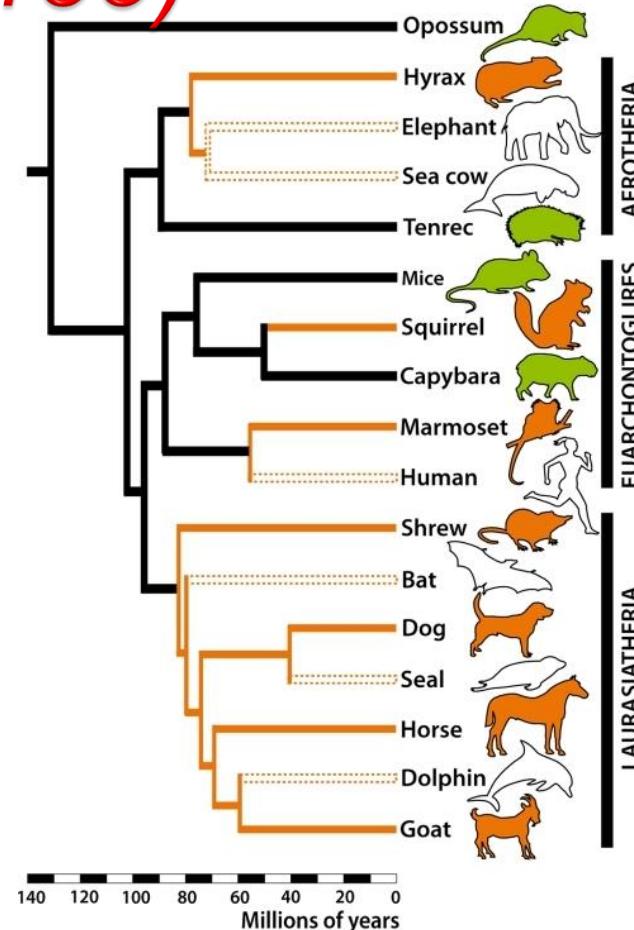
levanhanhvn@gmail.com

# NỘI DUNG MÔN HỌC

- Chương 1: Ôn tập ngôn ngữ lập trình C
- Chương 2: Kiểu dữ liệu con trỏ
- Chương 3: Tổng quan về cấu trúc dữ liệu và giải thuật
- Chương 4: Danh sách kề (Danh sách tuyến tính)
- Chương 5: Các giải thuật tìm kiếm trên danh sách kề
- Chương 6: Các giải thuật sắp xếp trên danh sách kề
- Chương 7: Danh sách liên kết động (Linked List)
- Chương 8: Ngăn xếp (*Stack*)
- Chương 9: Hàng đợi (*Queue*)
- **Chương 10: Cây nhị phân tìm kiếm (*Binary Search Tree*)**
- Chương 11: Cây cân bằng (*Balanced binary search tree – AVL tree*)
- Chương 12: Bảng băm (*Hash Table*)

# Chương 10

# CÂY NHỊ PHÂN TÌM KIẾM (*Binary Search Tree*)



# NỘI DUNG

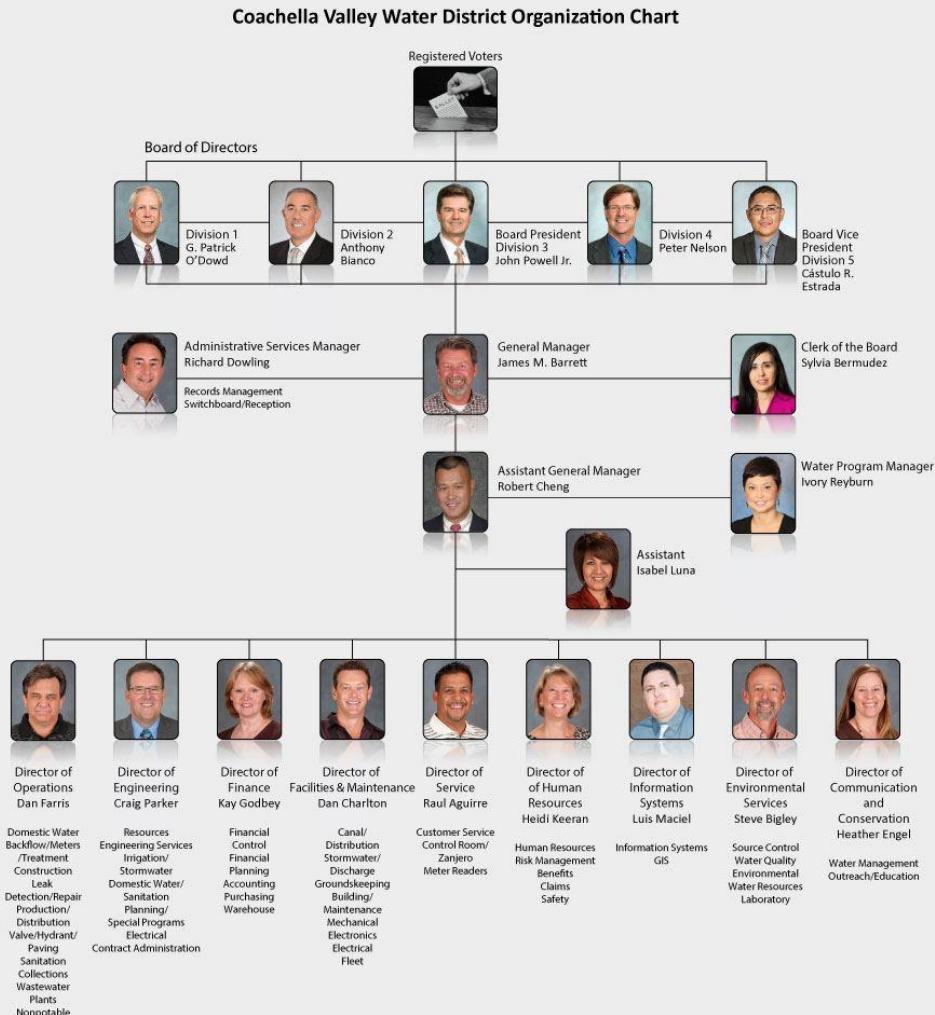
1. Giới thiệu
2. Một số tính chất
3. Một số thuật ngữ
4. Cây nhị phân (*Binary Tree*)
5. Cây nhị phân tìm kiếm (*Binary Search Tree*)
6. Cây nhị phân tìm kiếm cân bằng (*Balanced Binary Search Tree*)

# 1. GIỚI THIỆU

## 1.1. Một số ứng dụng của cây

### - *Organization Chart*

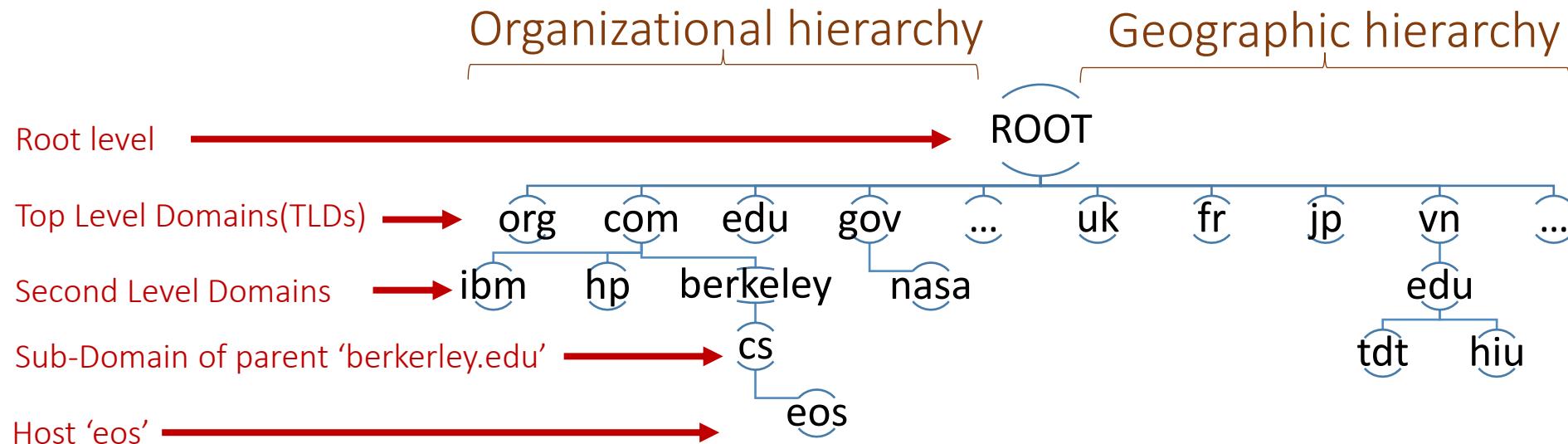
Coachella Valley Water District Organization Chart



# 1. GIỚI THIỆU

## 1.1. Một số ứng dụng của cây

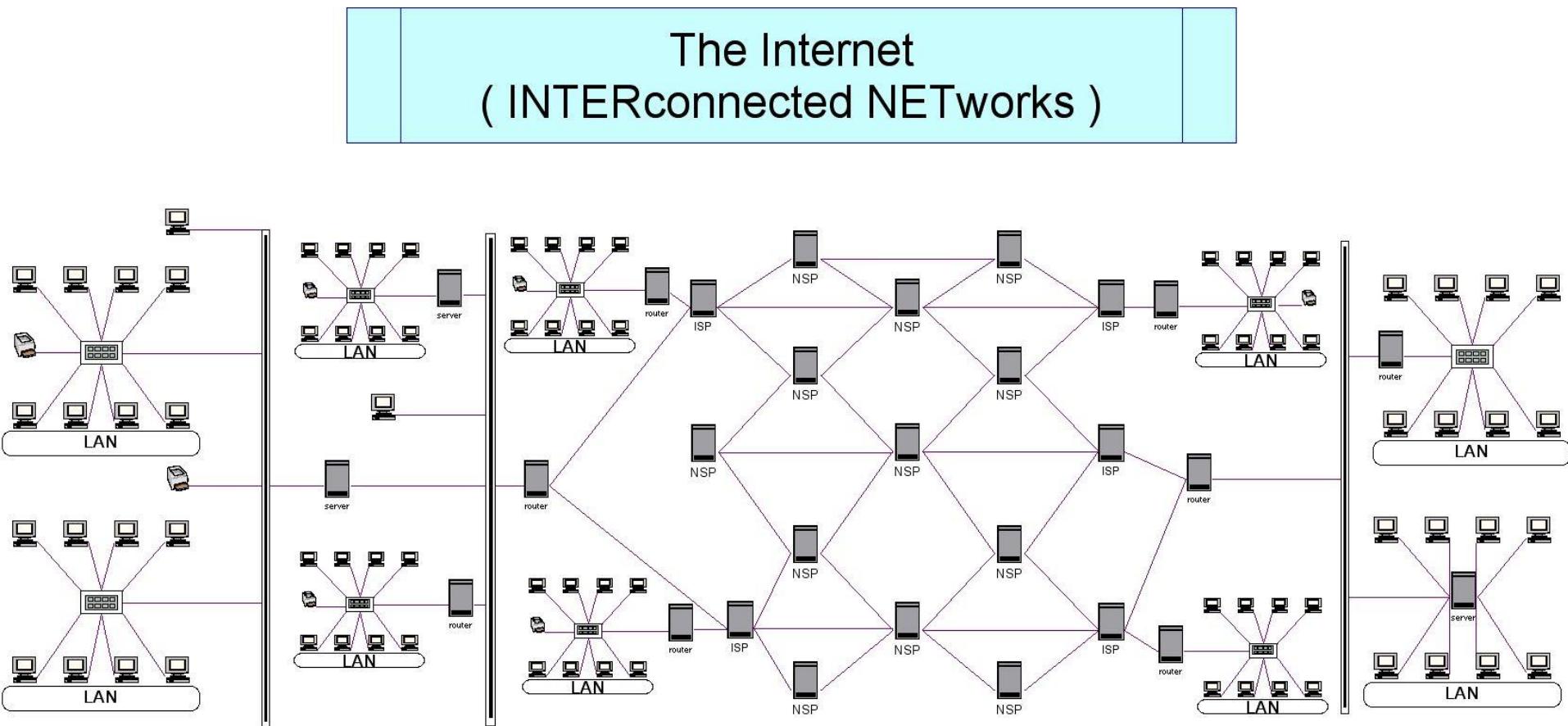
- *internet*



# 1. GIỚI THIỆU

## 1.1. Một số ứng dụng của cây

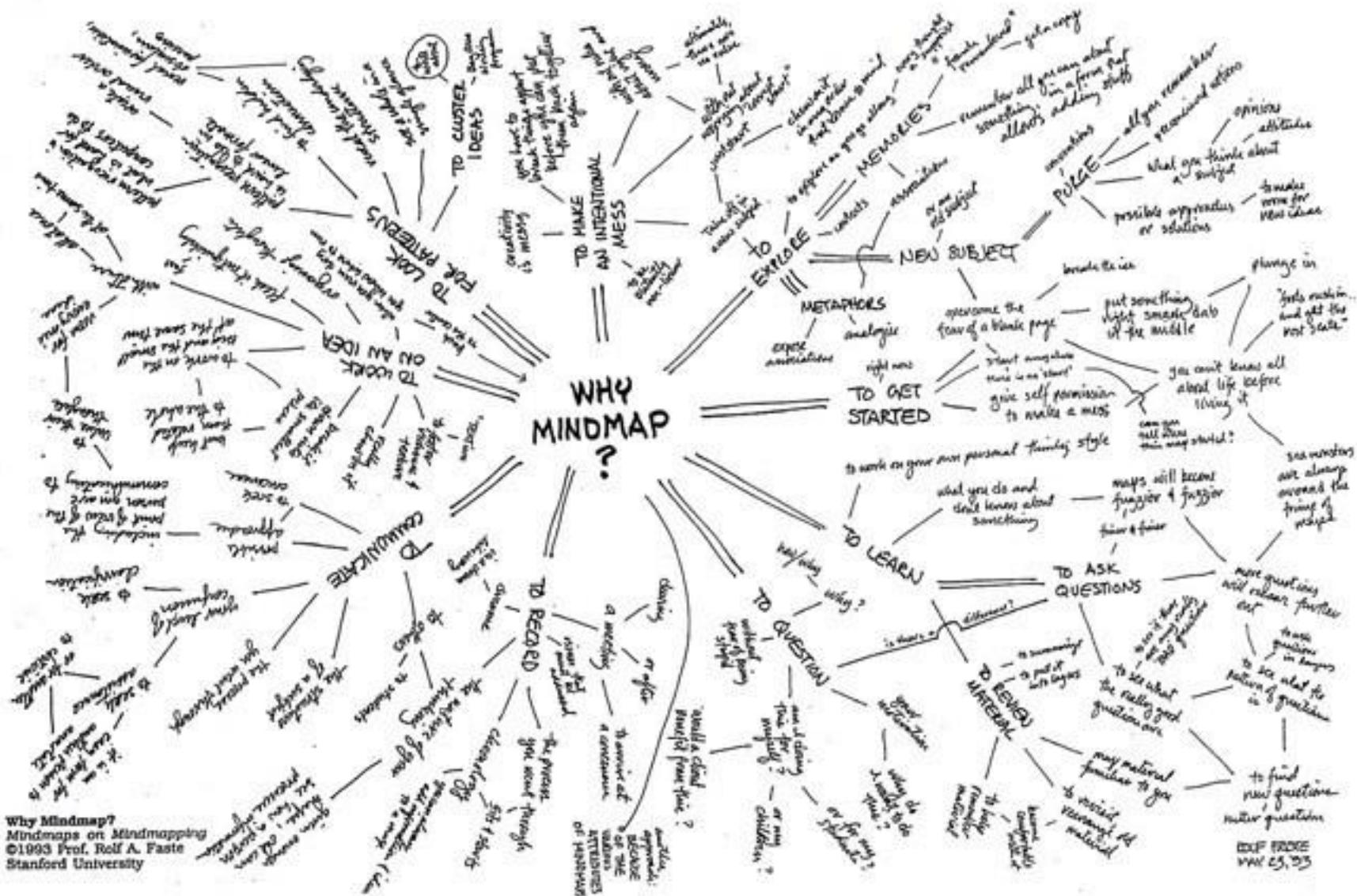
- *network*



# 1. GIỚI THIỆU

## 1.1. Một số ứng dụng của cây

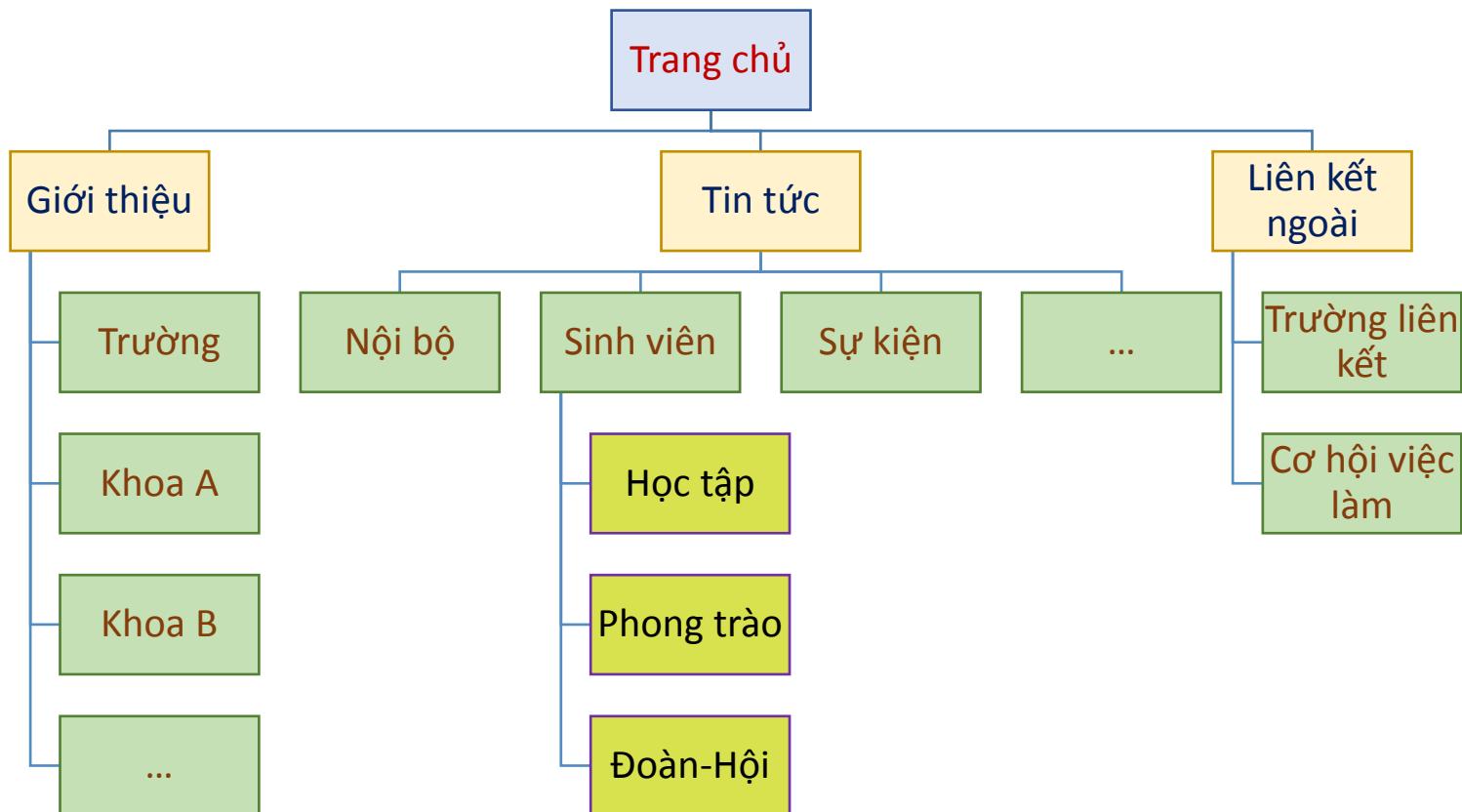
### - *MindMap*



# 1. GIỚI THIỆU

## 1.1. Một số ứng dụng của cây

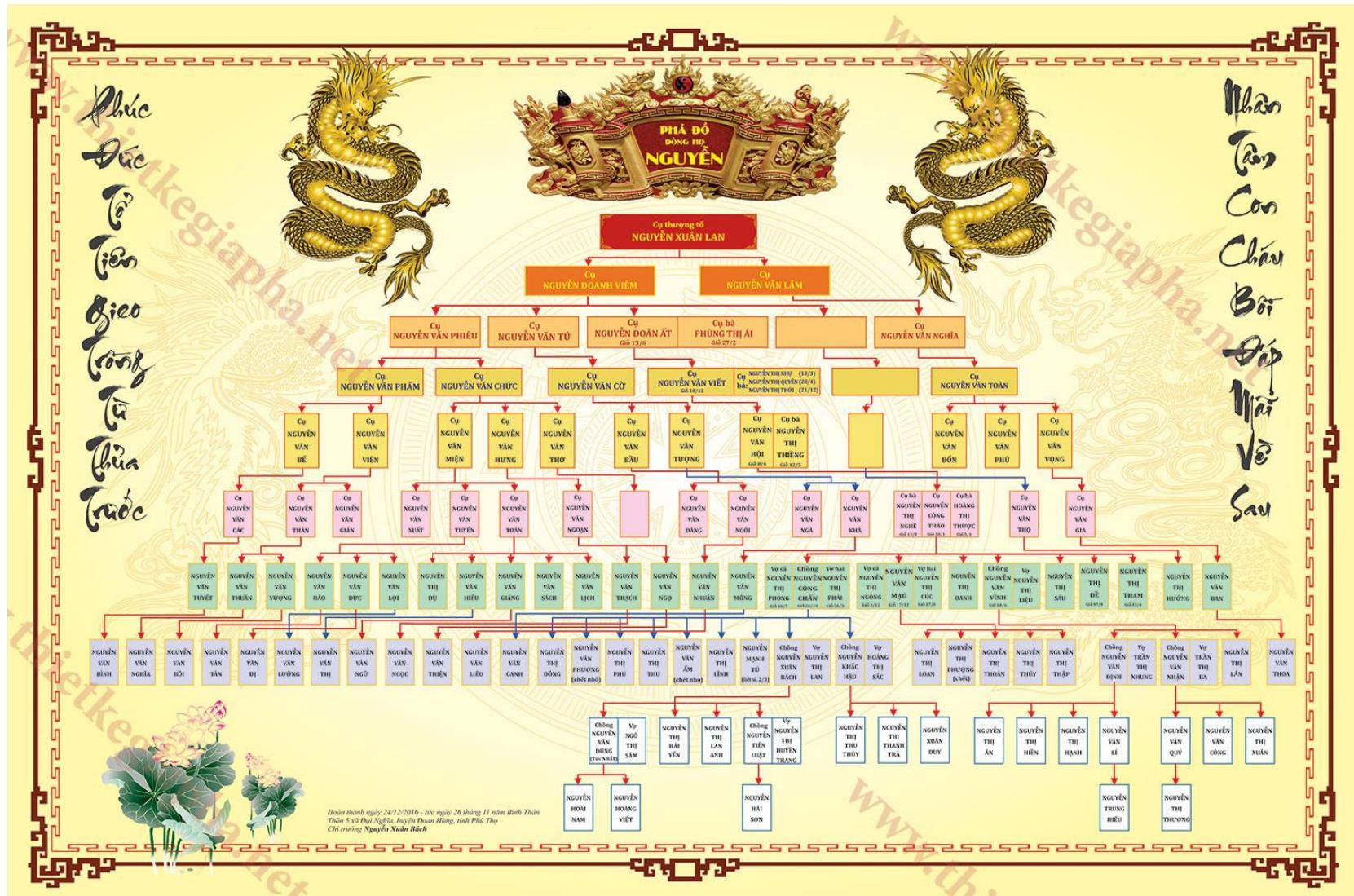
### - SiteMap



# 1. GIỚI THIỆU

## 1.1. Một số ứng dụng của cây

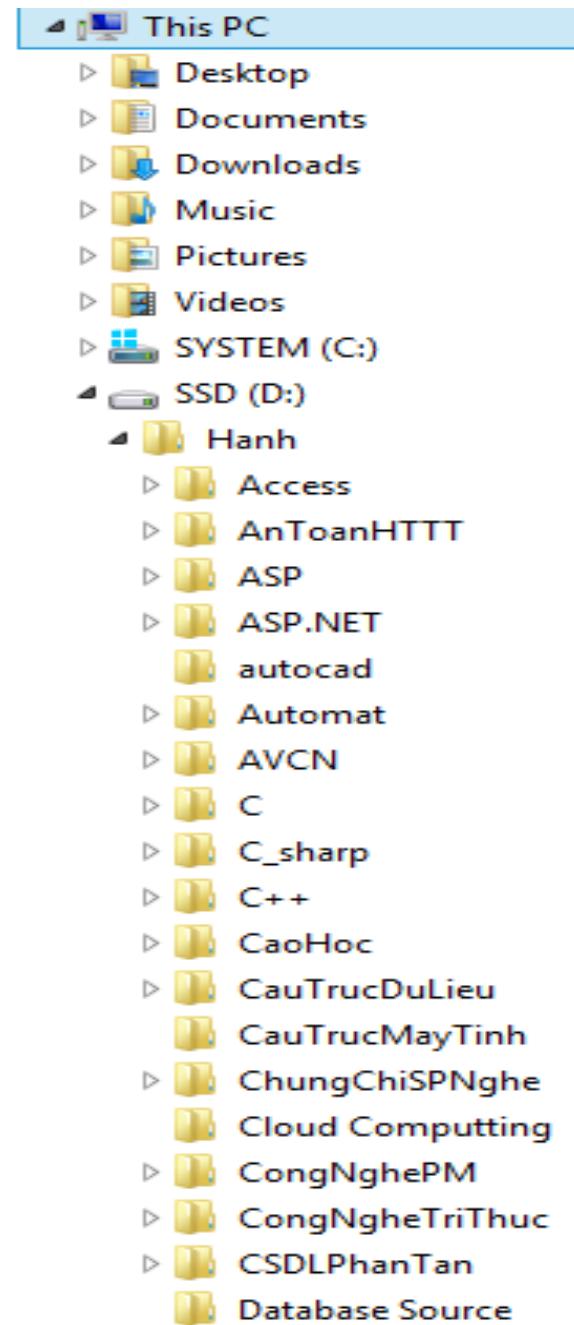
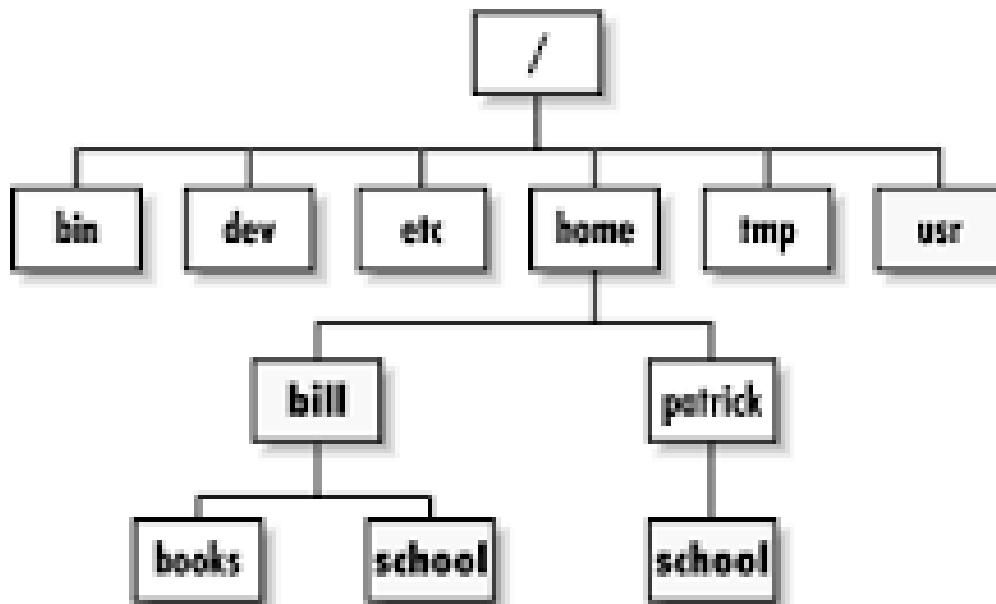
### - Cây gia phả



# 1. GIỚI THIỆU

## 1.1. Một số ứng dụng của cây

- *Directory tree*

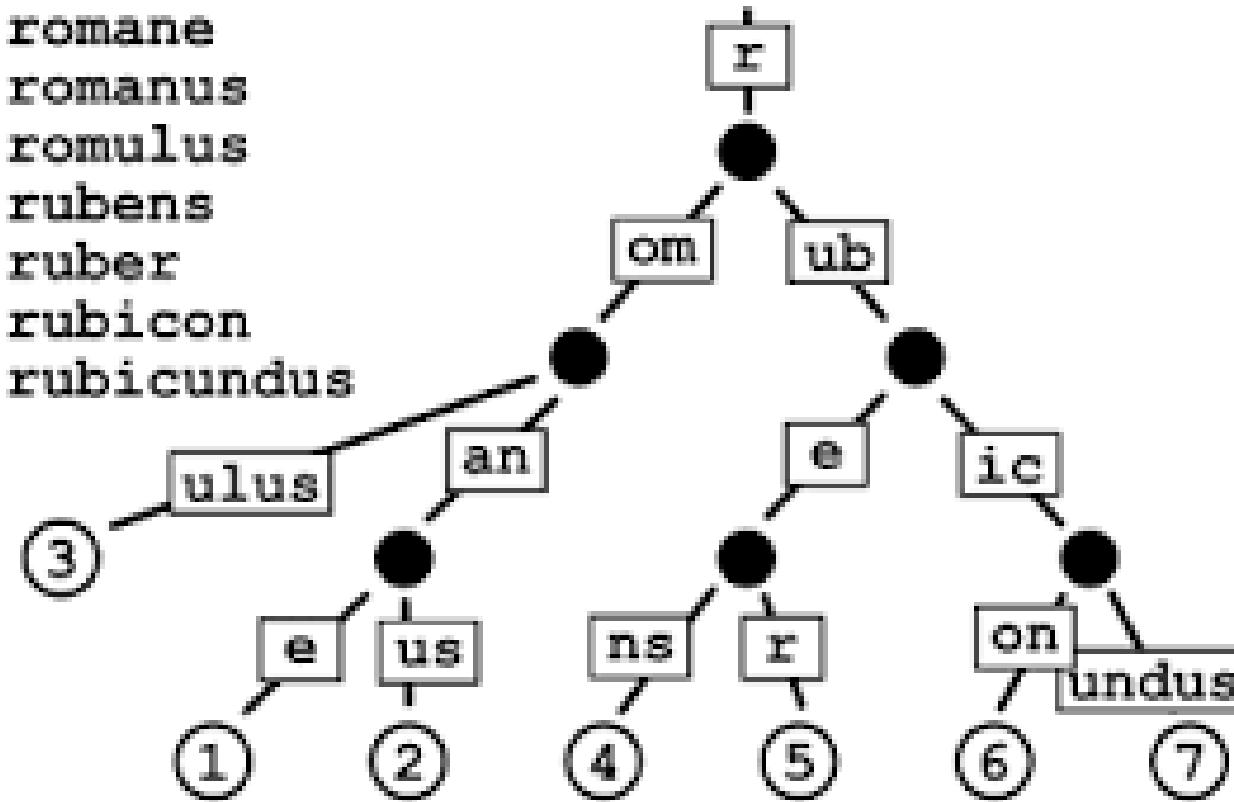


# 1. GIỚI THIỆU

## 1.1. Một số ứng dụng của cây

### - *Dictionary tree*

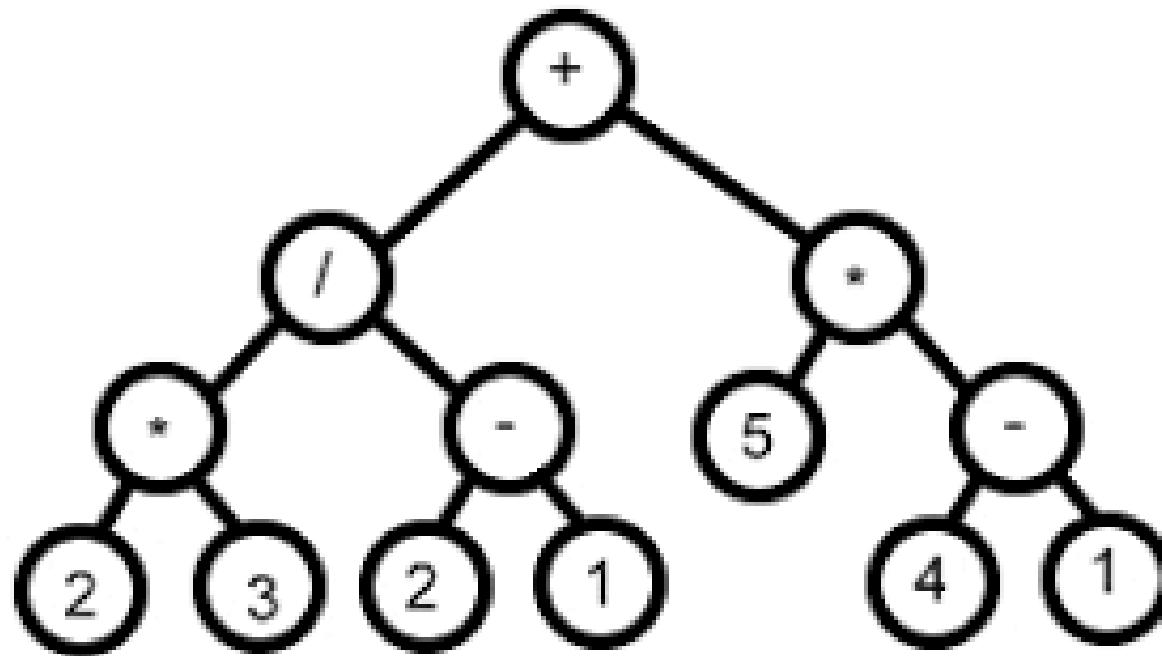
- 1 romane
- 2 romanus
- 3 romulus
- 4 rubens
- 5 ruber
- 6 rubicon
- 7 rubicundus



# 1. GIỚI THIỆU

## 1.1. Một số ứng dụng của cây

- *Expression tree*

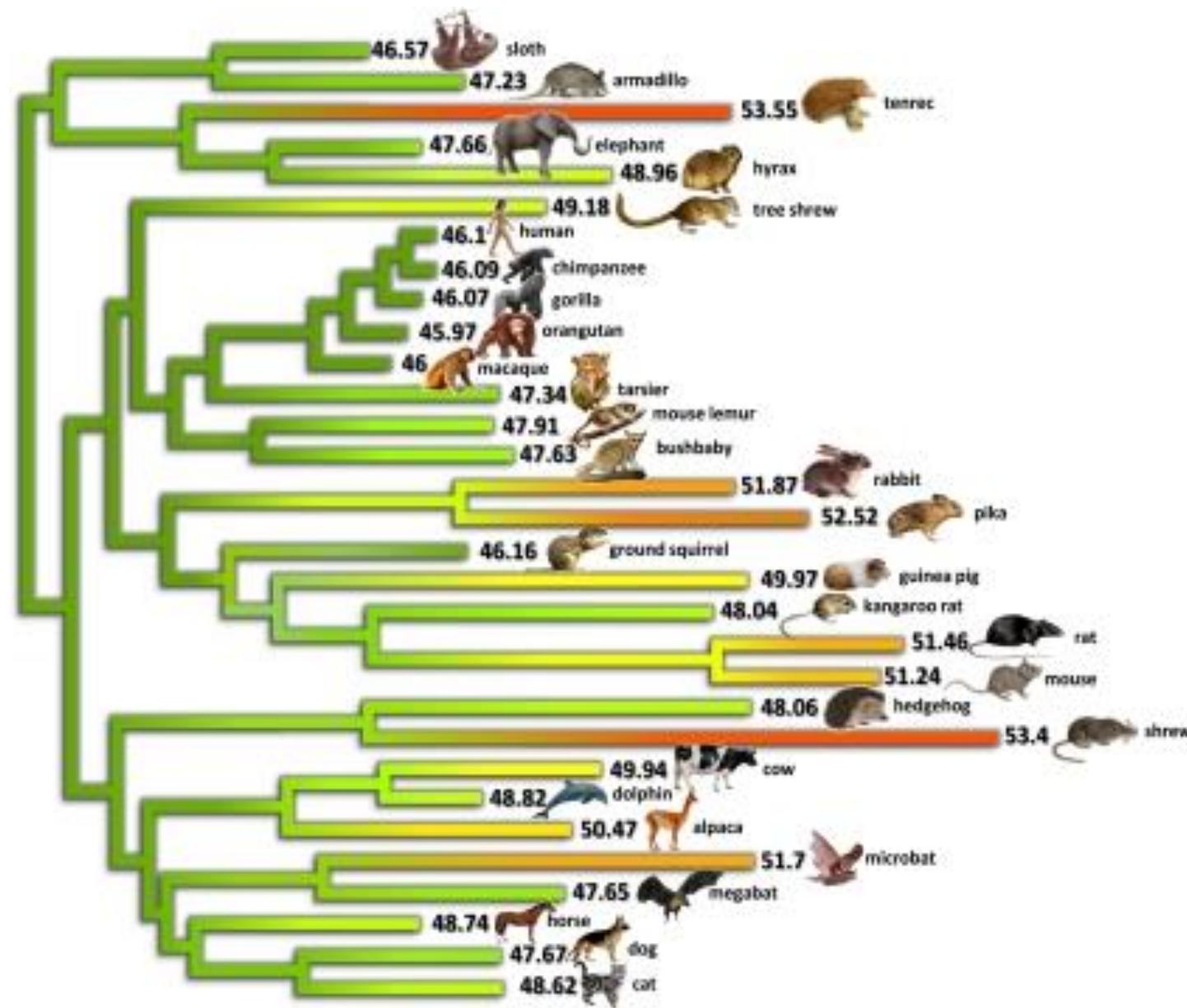


Expression tree for  $2*3/(2-1)+5*(4-1)$

# 1. GIỚI THIỆU

## 1.1. Một số ứng dụng của cây

### - Biological

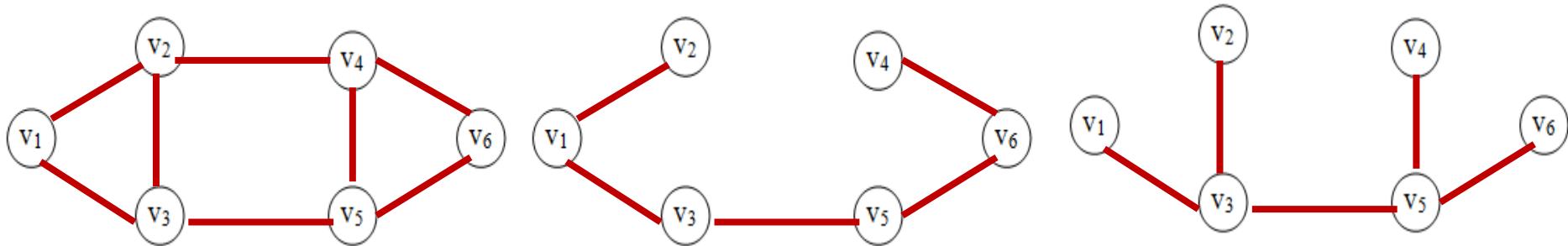


## 1.1. Một số ứng dụng của cây

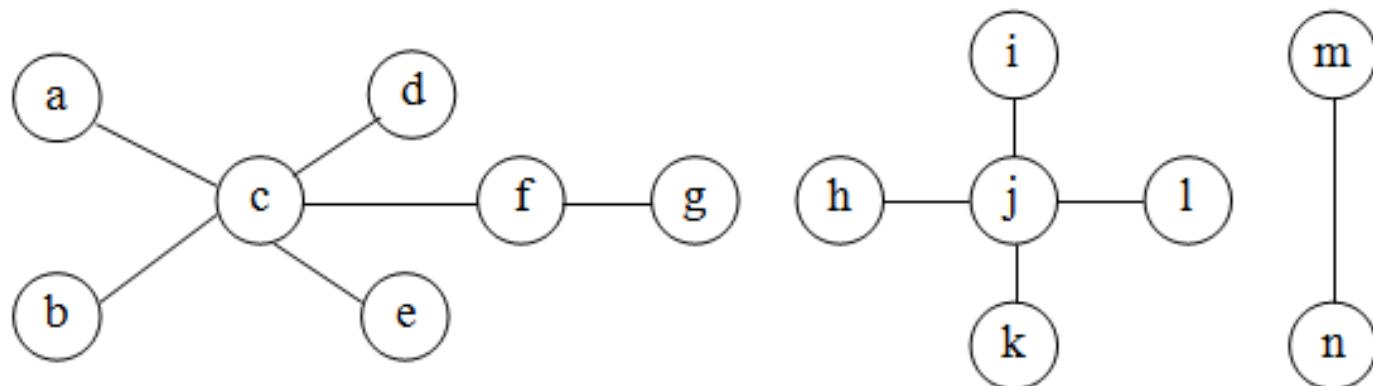
- Cây là một cấu trúc quan trọng để biểu diễn:
  - Tính kế thừa:
    - Cây gia phả (biểu diễn mối quan hệ giữa những người trong một dòng họ qua nhiều thế hệ).
    - Trong việc định nghĩa đối tượng
  - Phân cấp
    - Cây phân cấp các loài
    - Cây phân cấp quản lý mạng Internet...

## 1.2. Định nghĩa

- **Cây**: là một đồ thị vô hướng liên thông, không chứa chu trình và có ít nhất hai đỉnh.



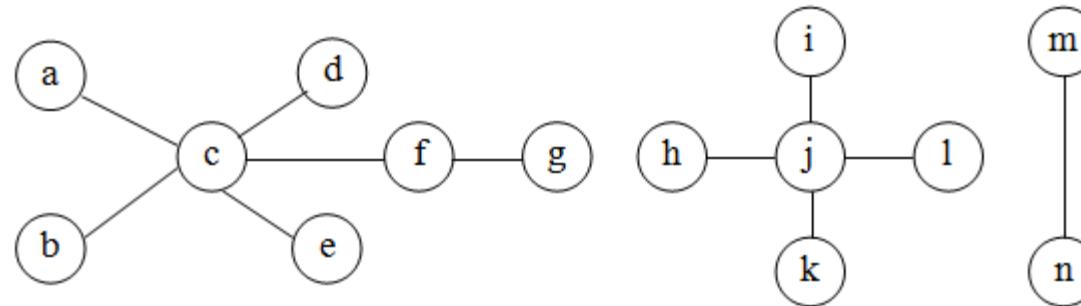
- **Rừng**: là một đồ thị vô hướng không chứa chu trình và có ít nhất hai thành phần, mỗi thành phần có ít nhất 2 đỉnh. Ví dụ: rừng sau gồm 3 cây



## 1. GIỚI THIỆU

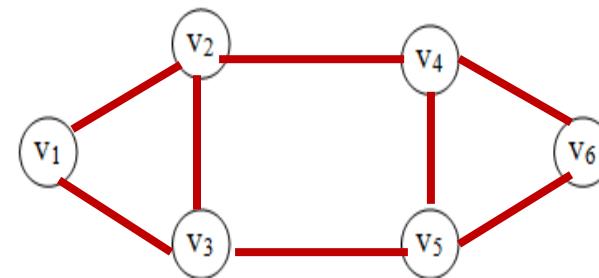
**1.3. Định lý:** Cho  $T$  là một đồ thị có  $n \geq 2$  đỉnh. Các điều sau là tương đương:

- $T$  là một cây.
- $T$  liên thông và có  $n-1$  cạnh.
- $T$  không chứa chu trình và có  $n-1$  cạnh.
- Giữa hai đỉnh phân biệt bất kỳ của  $T$  luôn có duy nhất một đường đi sơ cấp.
- $T$  không chứa chu trình nhưng khi thêm một cạnh mới thì có được một chu trình duy nhất.



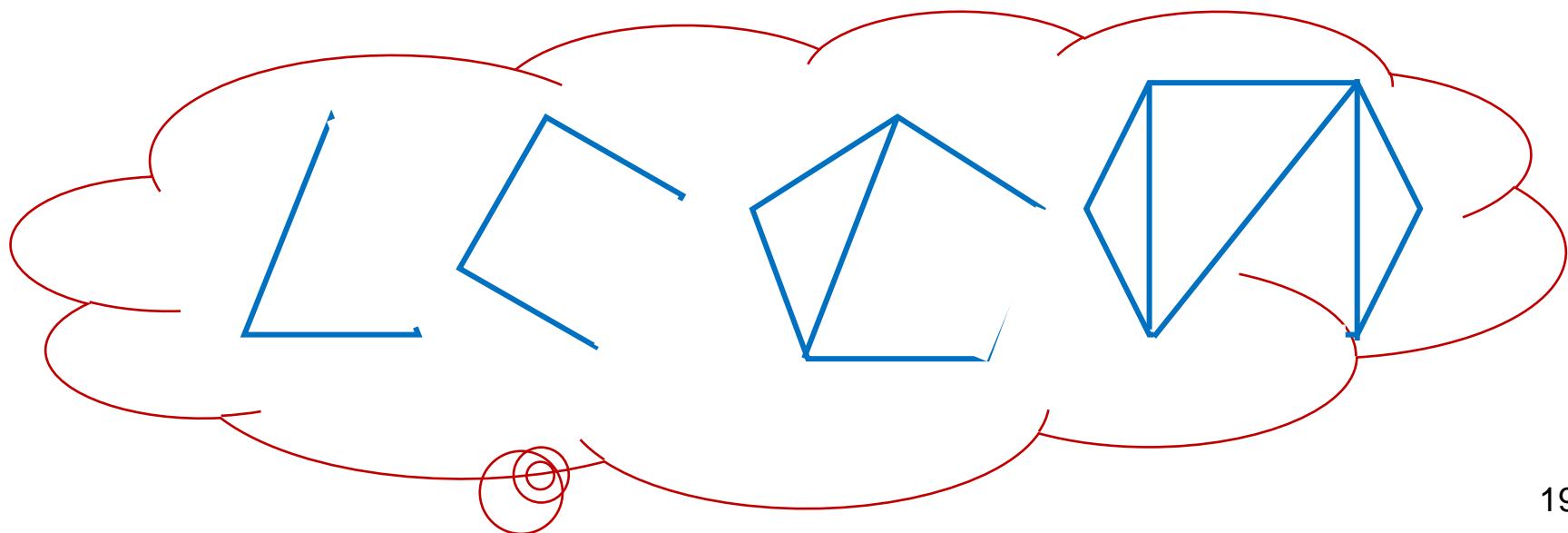
## 1.4. Cây khung

- Trong đồ thị liên thông  $G$ , nếu ta loại bỏ cạnh nằm trên chu trình nào đó thì ta sẽ được đồ thị vẫn là liên thông.
- Nếu cứ loại bỏ các cạnh ở các chu trình khác cho đến khi nào đồ thị không còn chu trình (vẫn liên thông) thì ta thu được một cây nối các đỉnh của  $G$ . Cây đó gọi là **cây khung** hay **cây bao trùm** của đồ thị  $G$ .



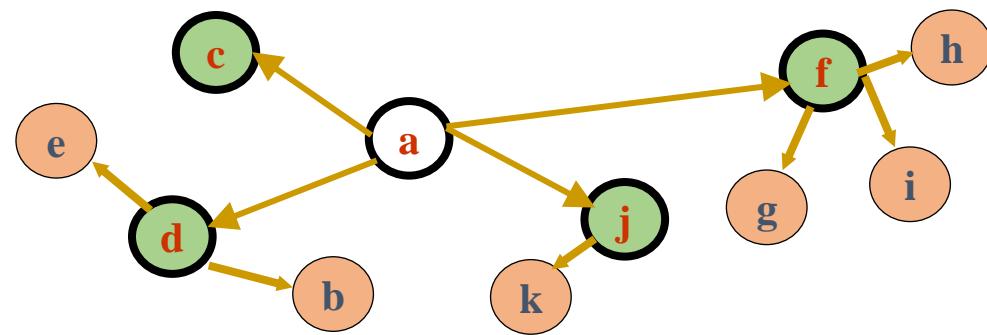
## 1.5. Rừng khung

- Nếu G là đồ thị có n đỉnh, m cạnh và k thành phần liên thông.
- G được coi là rừng khung khi các thành phần liên thông đều là cây khung.
- Để G trở thành rừng khung thì số cạnh cần bị loại bỏ là  $m-n+k$ , số này ký hiệu là  $v(G)$  và gọi là chu số của đồ thị G. Ví dụ:  $n=18, m=22, k=4 \Rightarrow v(G)=22-18+4=8$



## 1.6. CÂY CÓ GỐC

- *Cây có hướng* là đồ thị có hướng mà đồ thị vô hướng nền của nó là một cây.
- *Cây có gốc* là một cây có hướng, trong đó có một đỉnh đặc biệt, gọi là gốc, từ gốc có đường đi đến mọi đỉnh khác của cây.

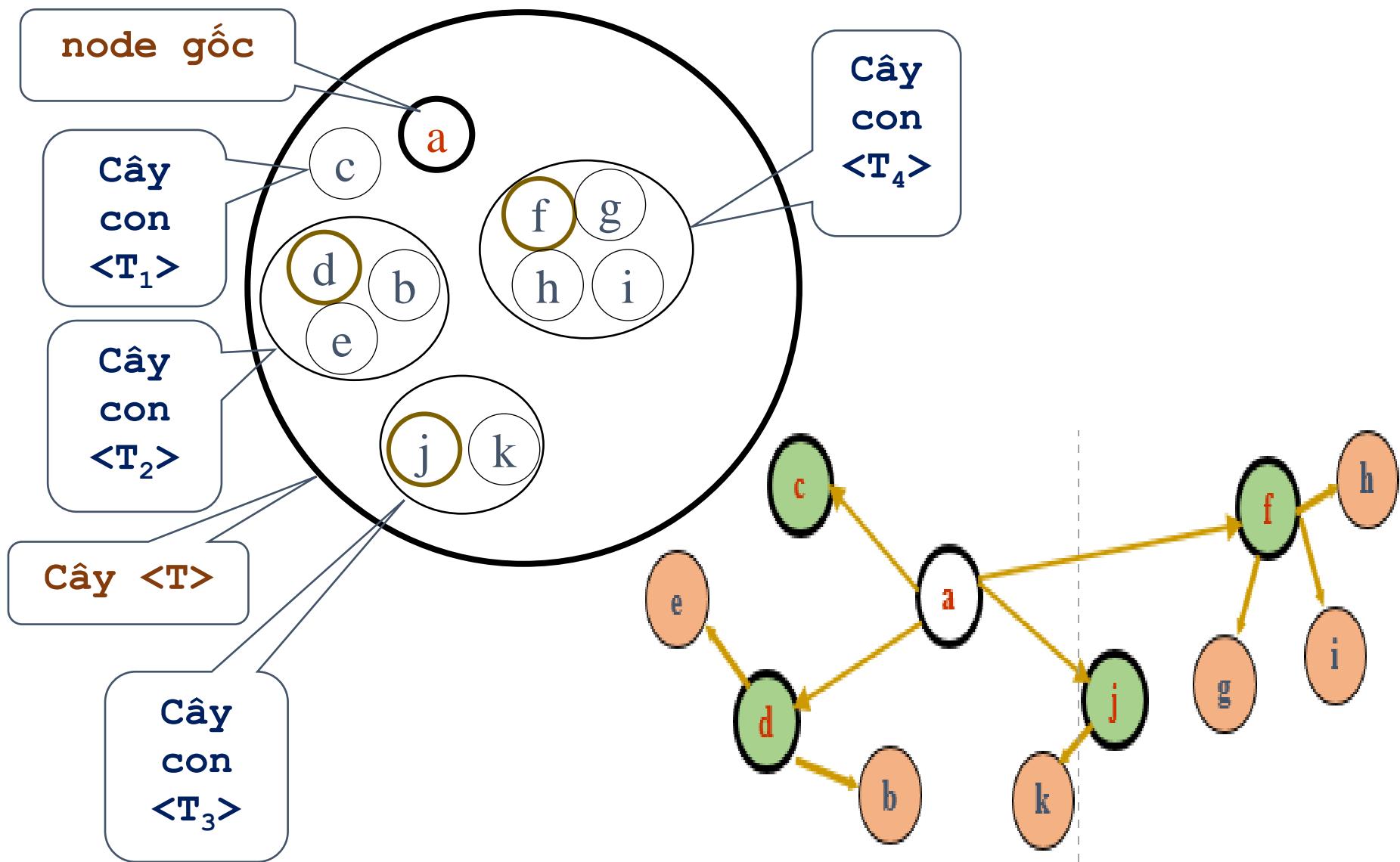


- Cấu trúc cây tổng quát:
  - Cây là một cấu trúc gồm 1 tập hữu hạn các node có cùng kiểu dữ liệu.
  - Các node được nối với nhau bởi 1 cạnh.
  - Có duy nhất 1 node gốc.
  - Có duy nhất 1 đường đi từ node gốc đến 1 node bất kỳ

## 1. GIỚI THIỆU

### 1.7. Biểu diễn cây

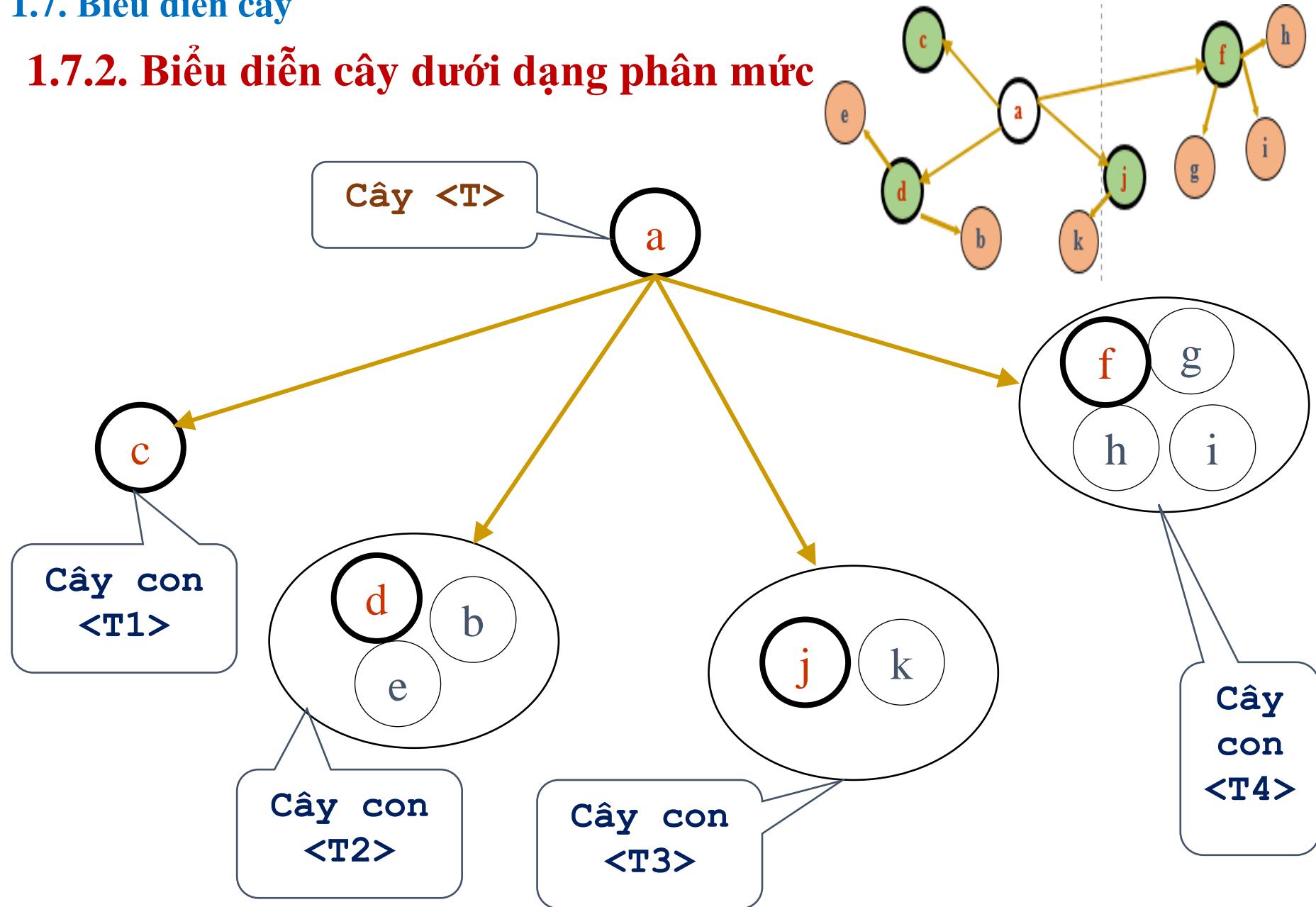
#### 1.7.1. Biểu diễn cây dưới dạng tập hợp



# 1. GIỚI THIỆU

## 1.7. Biểu diễn cây

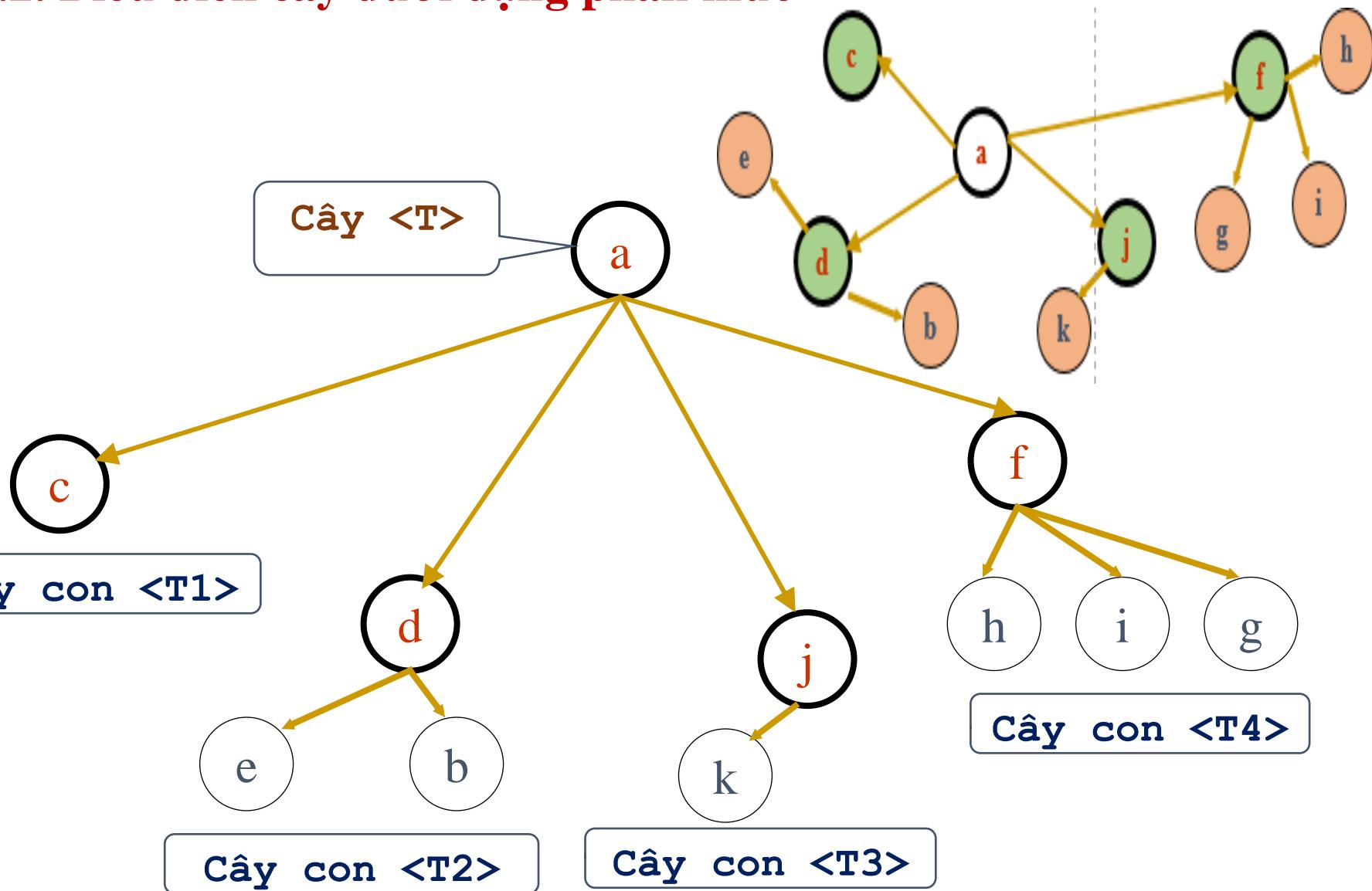
### 1.7.2. Biểu diễn cây dưới dạng phân mức



# 1. GIỚI THIỆU

## 1.7. Biểu diễn cây

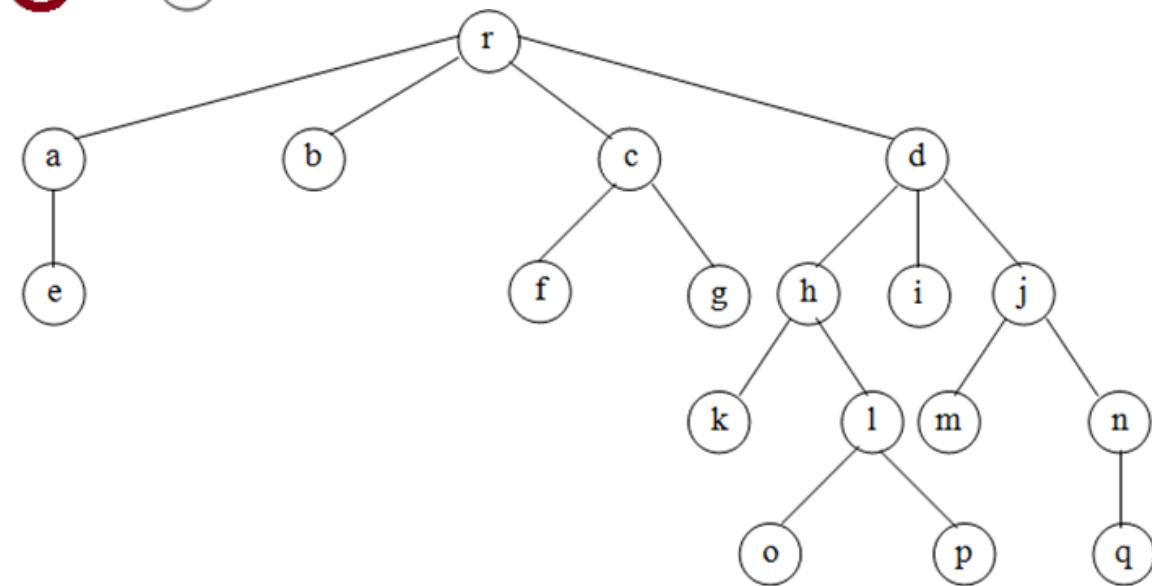
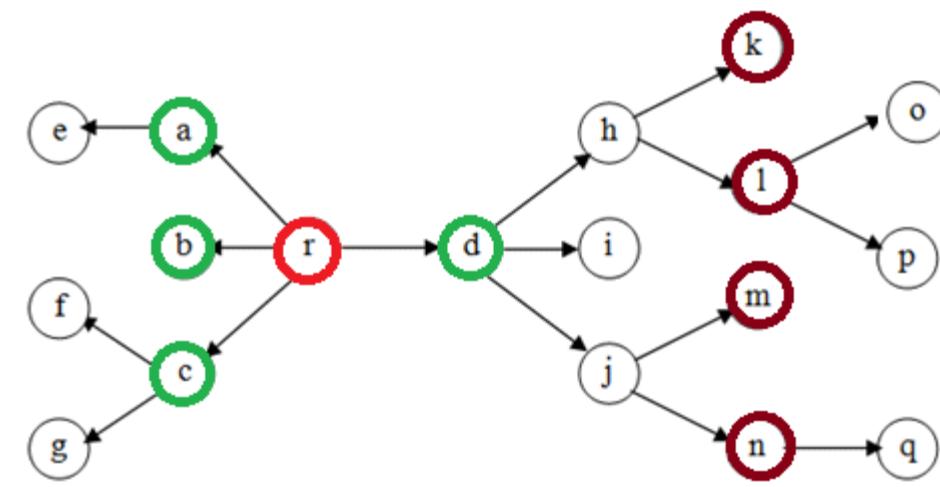
### 1.7.2. Biểu diễn cây dưới dạng phân mức



# 1. GIỚI THIỆU

## 1.7. Biểu diễn cây

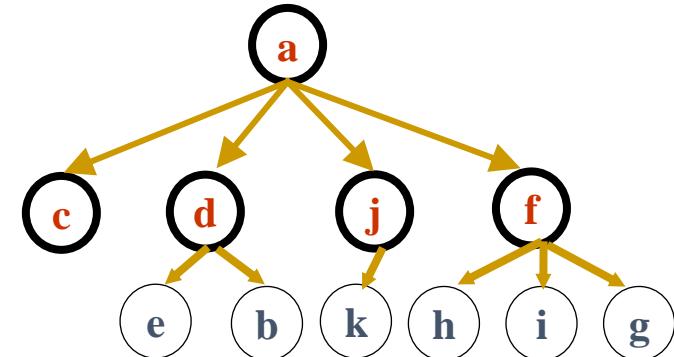
### 1.7.2. Biểu diễn cây dưới dạng phân mức



## 1.8. Cây m phân

### 1.8.1. Độ tuổi của một nút $p_i$ (*Degree of node*):

là số nút con của nút  $p_i$ .



- Độ tuổi (a) = 4
- Độ tuổi (f) = 3
- Độ tuổi (d) = 2
- Độ tuổi (j) = 1
- Độ tuổi (c) = 0 = Độ tuổi (b) = Độ tuổi (e) = Độ tuổi (g)  
= Độ tuổi (h) = Độ tuổi (i) = Độ tuổi (k)

### 1.8.2. Độ tuổi của cây (*Degree of tree*):

là độ tuổi lớn nhất của tất cả các node có trong cây

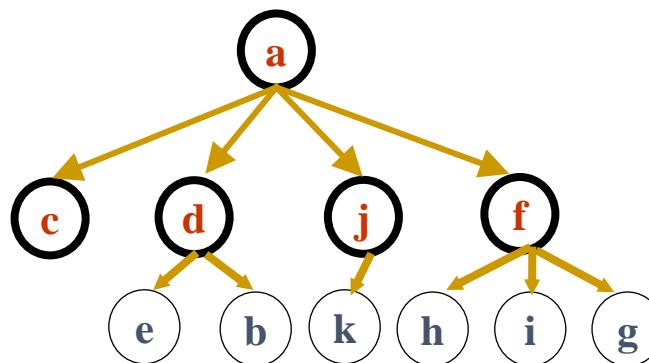
## 1.8. Cây m phân

### 1.8.3. Cây m phân

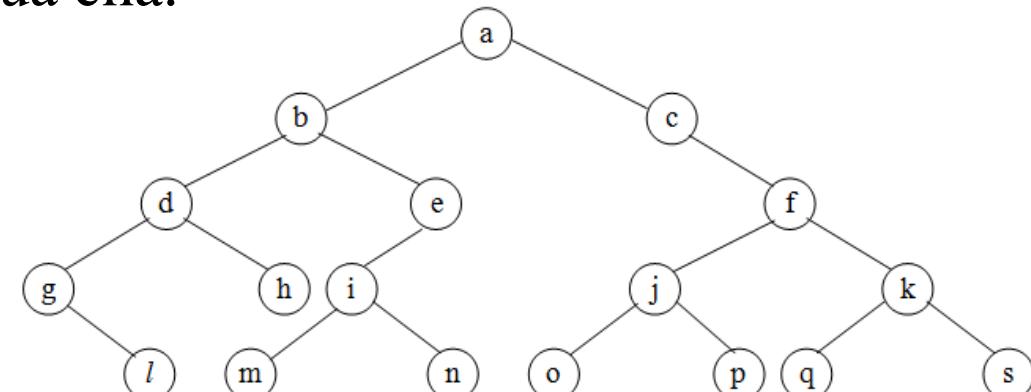
- Một cây có gốc T được gọi là cây m-phân nếu mỗi đỉnh của T có nhiều nhất là m con.
- Cây có gốc T được gọi là một cây m-phân đầy đủ nếu mỗi đỉnh trong của T đều có m con.

### 1.8.4. Cây nhị phân

- Khi cây m phân có m=2, ta có một cây nhị phân.
- Trong một cây nhị phân, mỗi con được chỉ rõ là con bên trái hay con bên phải; con bên trái (hoặc bên phải) được vẽ phía dưới về bên trái (hoặc bên phải) của cha.



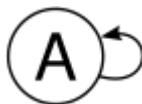
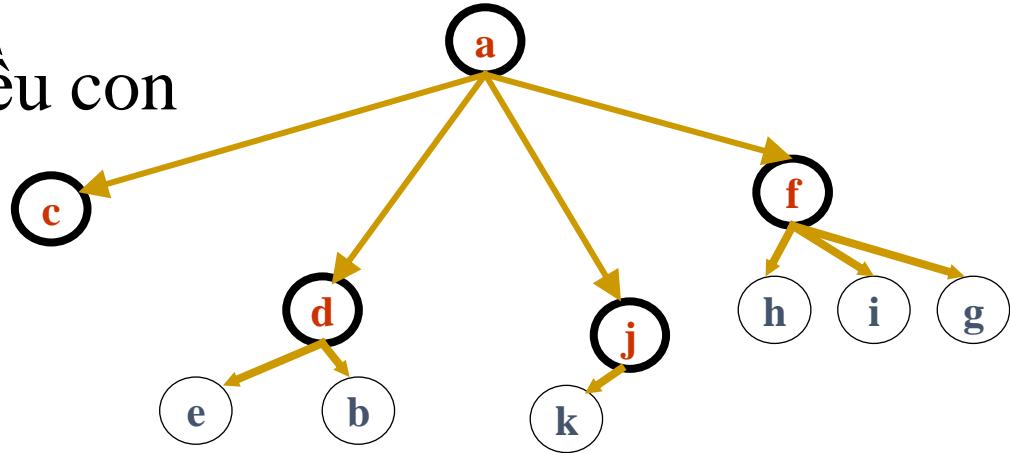
Cây tứ phân ( $m=4$ )



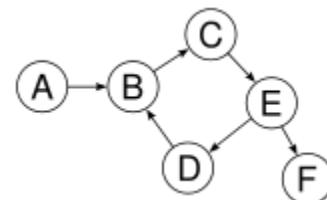
Cây nhị phân ( $m=2$ )

## 2. MỘT SỐ TÍNH CHẤT CỦA CÂY

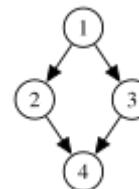
- i. Node gốc không có node cha
- ii. Mỗi node khác không phải gốc đều chỉ có duy nhất một node cha
- iii. Mỗi node có thể có nhiều con
- iv. Không có chu trình



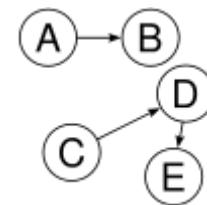
cycle  $A \rightarrow A$ . A is the root but it also has a parent.



- Cycle  $B \rightarrow C \rightarrow E \rightarrow D \rightarrow B$ .
- B has more than one parent (inbound edge).



Node 4 has more than one parent (inbound edge).

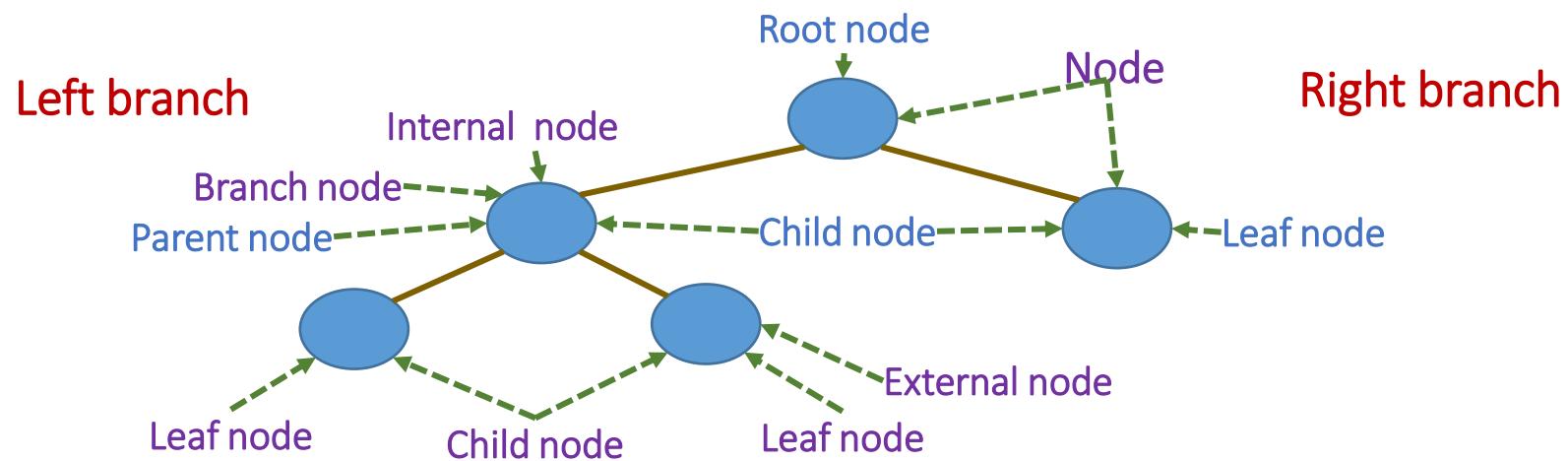


There is more than one root.

*Not a tree*

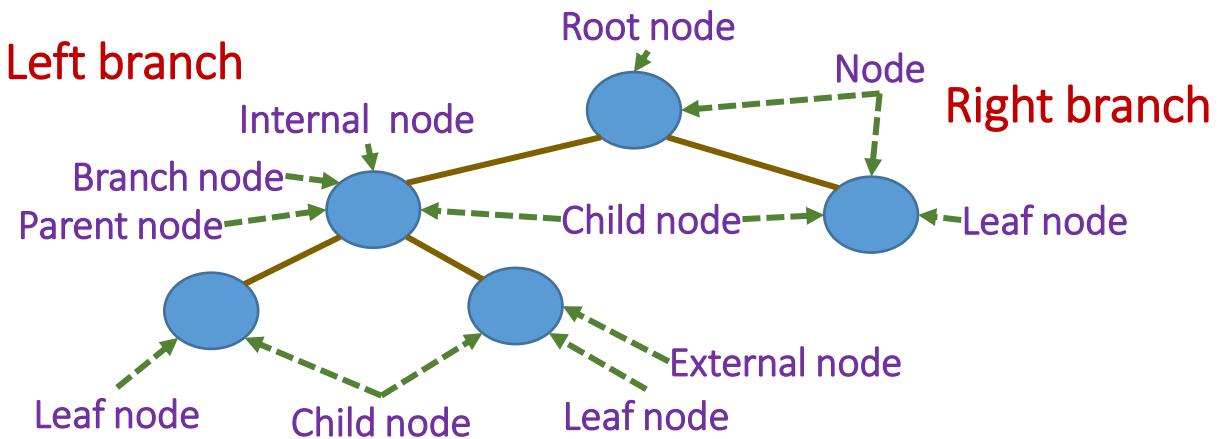
### 3. MỘT SỐ THUẬT NGỮ

- i. **Nút (Node)**: là một phần tử trong cây. Mỗi node có thể chứa một dữ liệu bất kỳ (do người dùng tự định nghĩa).
- ii. **Nút cha (Parent node)** Node có kết nối trực tiếp với một node khác ngay phía *dưới*.
- iii. **Nút con (Child node)** Node có kết nối trực tiếp với một node khác ngay phía *trên*.
- iv. **Nút gốc (Root node)**: node trên cùng trong cây, và node gốc không có node cha



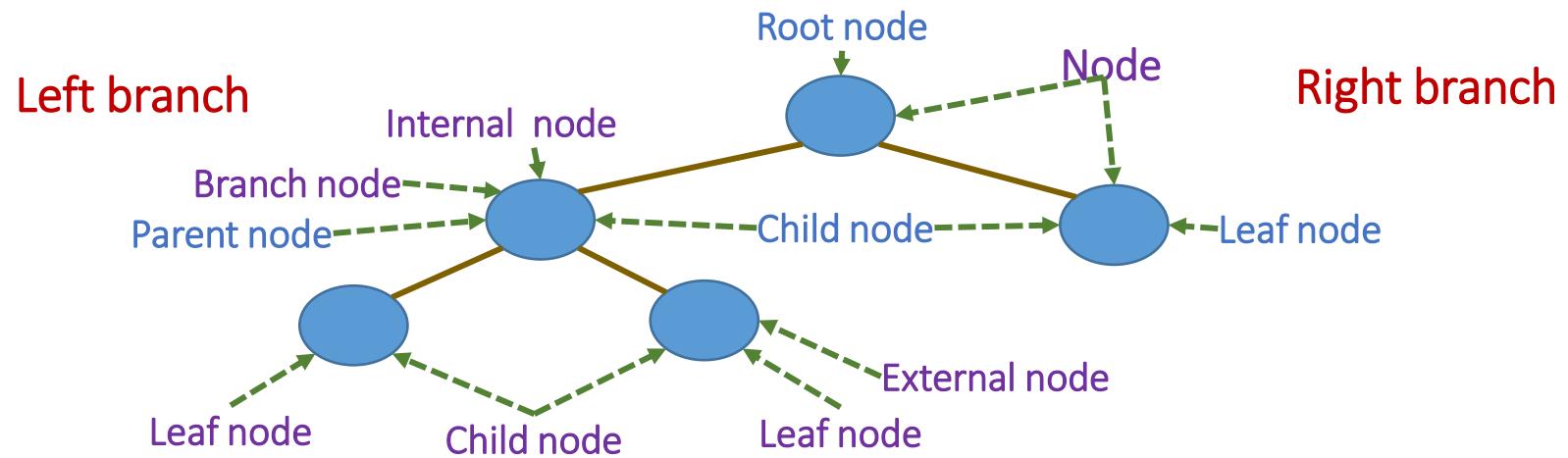
### 3. MỘT SỐ THUẬT NGỮ

- v. **Nút nội (Internal node hay node nhánh, node trung gian):** là node có *Parent node* và có *child node*
- vi. **Nút anh em (Sibling node):** là những node có cùng *Parent node*
- vii. **Nút tổ tiên (Ancestor node):** Là nút có thể truy cập bằng cách lặp đi lặp lại việc truy cập từ *child node* đến *parent node*
- viii. **Nút con cháu (Descendant node):** Là nút có thể truy cập bằng cách lặp đi lặp lại việc truy cập từ *parent node* đến *child node*.

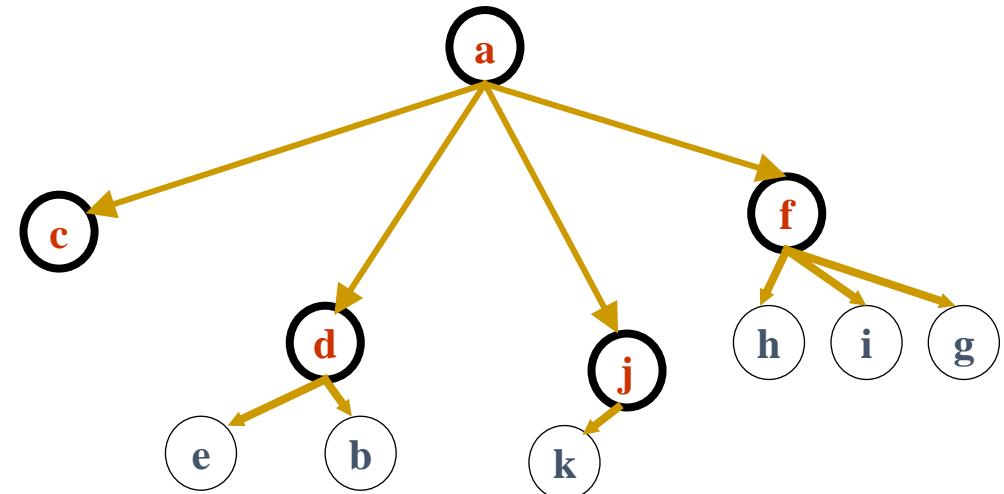


### 3. MỘT SỐ THUẬT NGỮ

- ix.** **Nút lá** (*Leaf node* hay *node ngoài – External node*): là node có bậc = 0 (nút không có node con)
- x.** **Nhánh** (*Branch - Edge*): là cạnh nối giữa Parent & Child node.



- xi.** **Cây con** (*SubTree*)

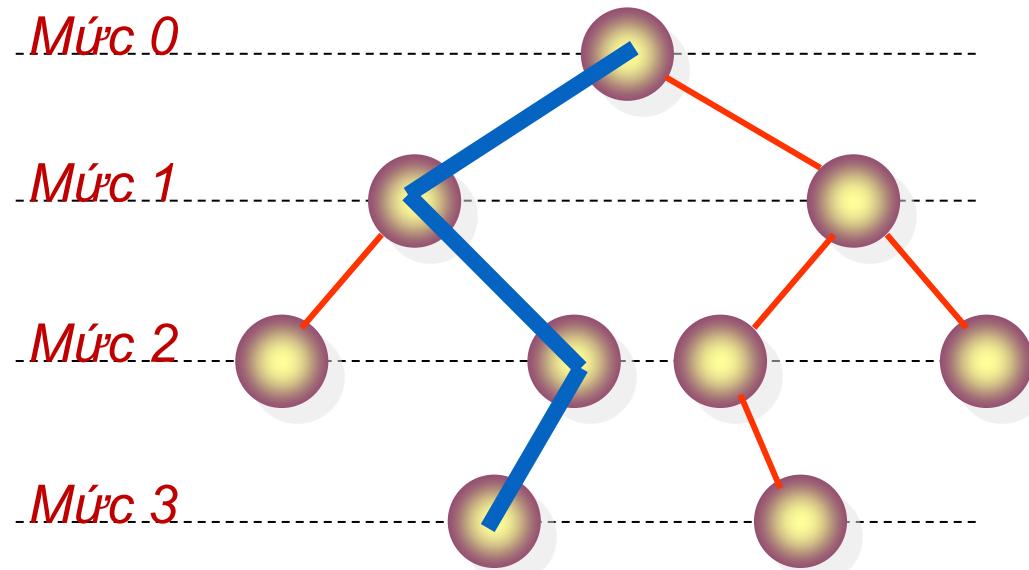


### 3. MỘT SỐ THUẬT NGỮ

**xii. Đường đi (Path):** Một chuỗi node và các cạnh kết nối một node với node hậu duệ.

**xiii. Mức (Level):** Mức của 1 node được xác định bằng số lượng kết nối giữa node đang xét và *root node*.

⇒ Mức của *root node* = 0.

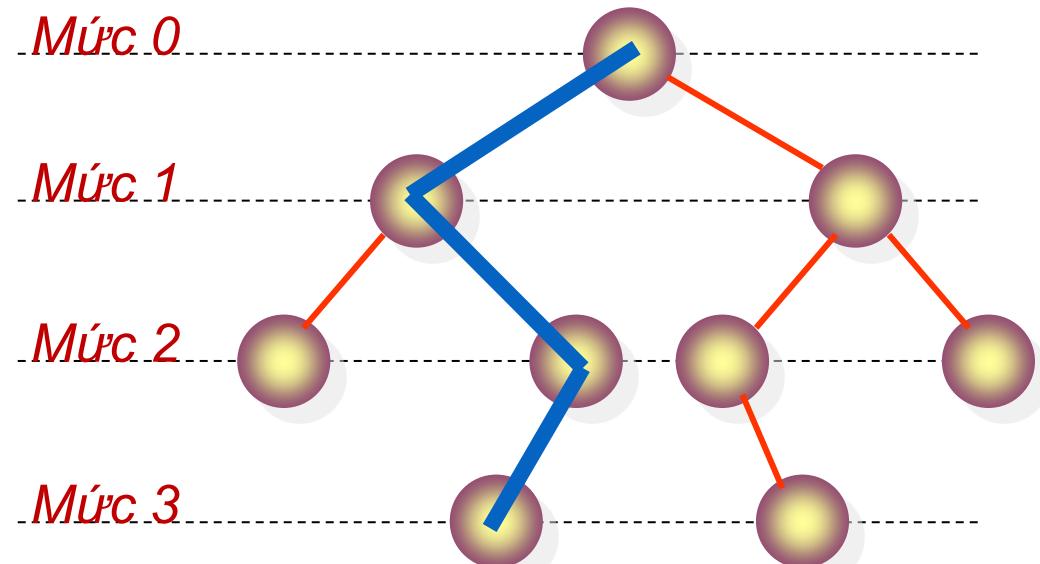


### 3. MỘT SỐ THUẬT NGỮ

**xiv. Độ sâu của node (*Depth of node*)** là số cạnh đi từ *root node* đến node đang xét.

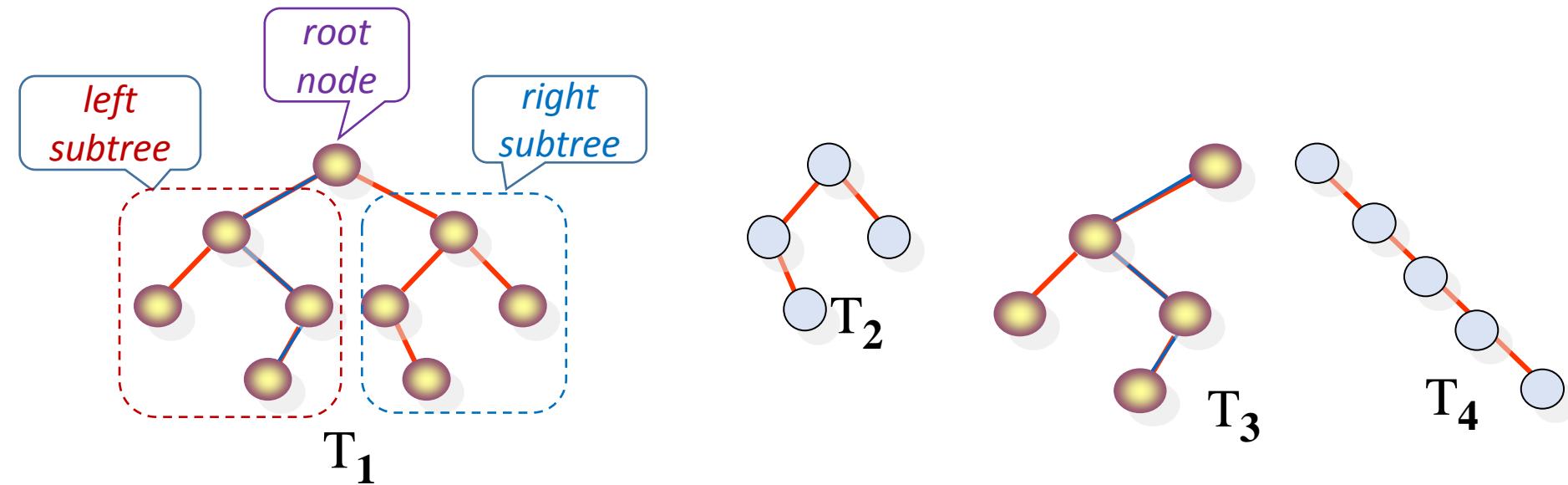
**xv. Chiều cao của node (*Heigh of node*)**: là số nhánh (cạnh) trên con đường **dài nhất** giữa nút đó và lá.

**xvi. Chiều cao của cây (*Heigh of tree*)**: chính là chiều cao của *root node*.



# 5. CÂY NHỊ PHÂN

**5.1. Khái niệm:** là cây mà các node đều có bậc tối đa là 2.



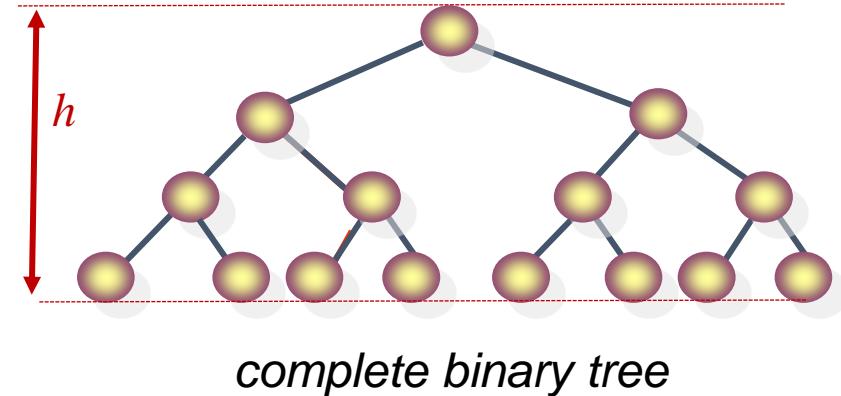
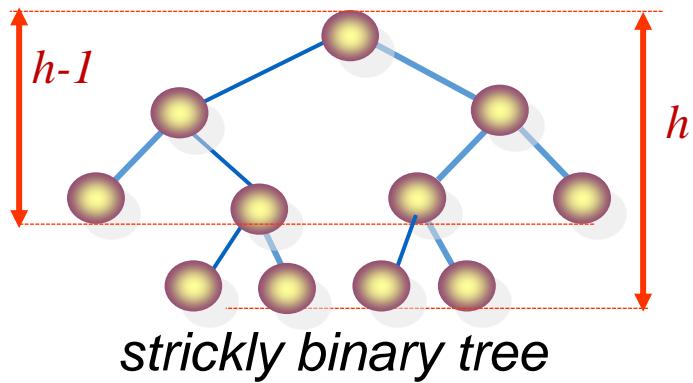
**5.2. Phân chia tập con trong cây nhị phân:** có thể phân chia 1 cây nhị phân thành 3 tập con:

- *Tập con thứ nhất*: có 1 node gọi là *root node*
  - *Tập con thứ hai*: là nhánh các cây con bên trái (*left subtree*).
  - *Tập con thứ ba*: là nhánh các cây con bên phải (*right subtree*).
- Trong đó *left subtree* và *right subtree* có thể tự thân hình thành cây nhị phân. Do đó cây (*nhánh*) này cũng có thể là rỗng.

## 5. CÂY NHỊ PHÂN

### 5.3. Các cây nhị phân đặc biệt

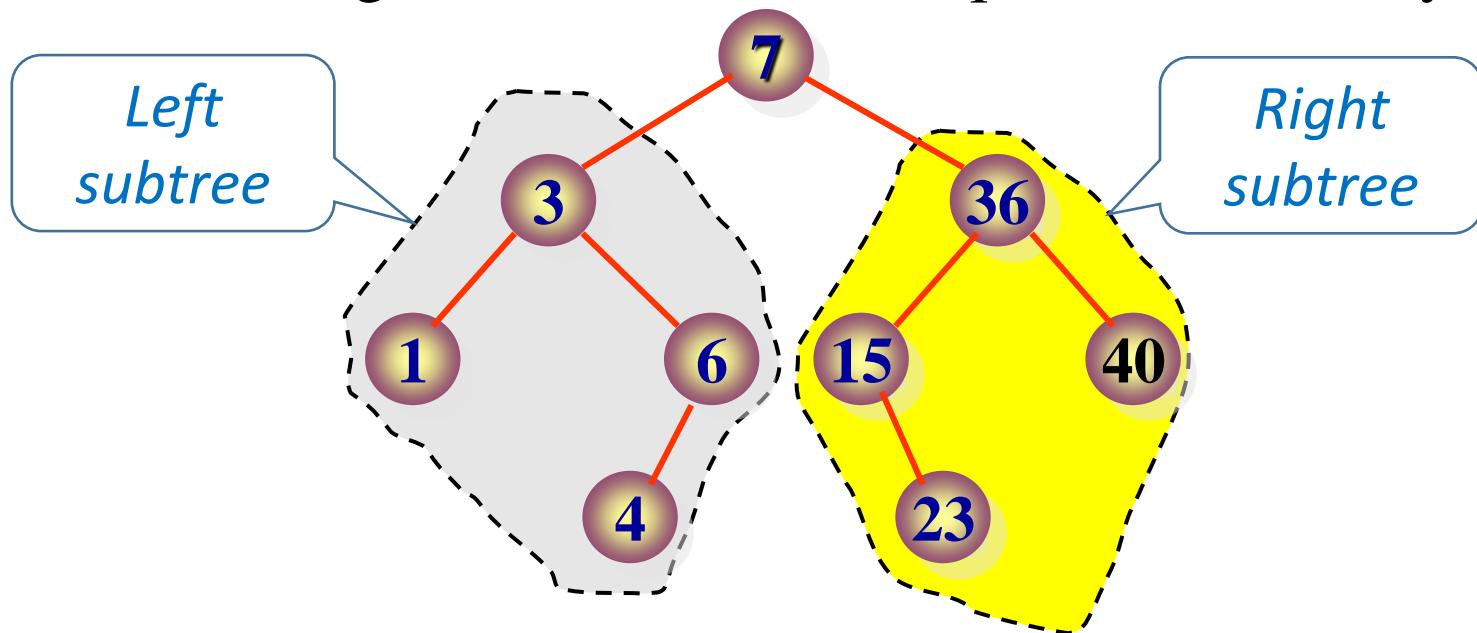
- *Cây nhị phân đúng (strictly binary tree):*
  - Là cây mà các node không phải là node lá đều có 2 node con.
  - Gọi  $n$  là số node lá của cây  $\Rightarrow$  số node trên toàn bộ cây sẽ là  $2n-1$
- *Cây nhị phân đầy đủ (complete binary tree):* là cây thỏa 2 điều kiện:
  - Là cây nhị phân đúng.
  - Tất cả các node lá đều có cùng mức  $h$ .



# 6. CÂY NHỊ PHÂN TÌM KIẾM (BST- *Binary Search Tree*)

## 6.1. Khái niệm

- Là cây nhị phân
  - Giá trị của một node bất kỳ luôn lớn hơn giá trị của tất cả các node bên trái và nhỏ hơn giá trị tất cả các node bên phải
- ⇒
- cây không chứa giá trị trùng (giống nhau)
  - node có giá trị nhỏ nhất nằm ở trái nhất của cây
  - node có giá trị lớn nhất nằm ở phải nhất của cây



### 6.1. Khái niệm

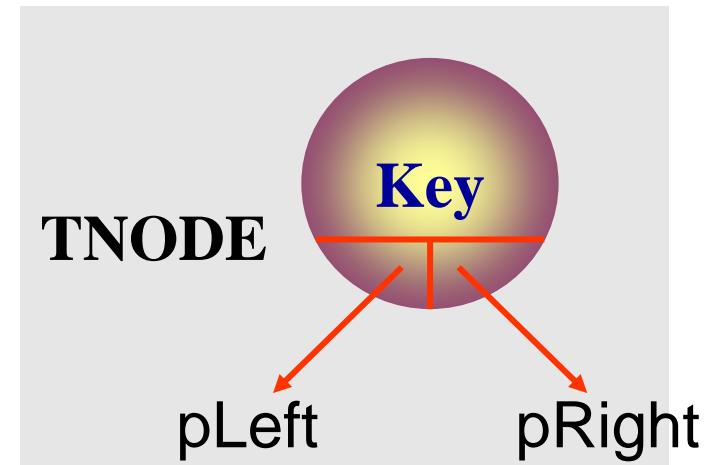
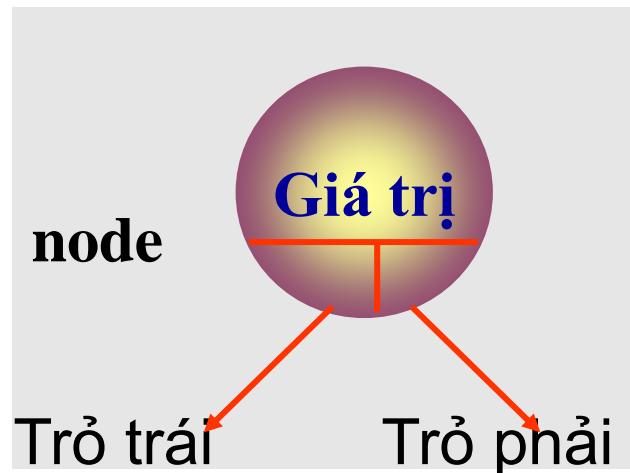
- Nhờ trật tự bố trí khóa trên cây  $\Rightarrow$  định hướng được nhanh khi tìm kiếm.
- Cây gồm N phần tử:
  - Trường hợp tốt nhất  $h = \log_2 N$
  - Trường hợp xấu nhất  $h = N$

$\Rightarrow$  Khi nào xảy ra trường hợp xấu nhất?



## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.2. Định nghĩa kiểu dữ liệu



```
struct TNODE
{
    <Data> Key;
    struct TNODE *pLeft;
    struct TNODE *pRight;
}*TREE;
```

Ví dụ khai báo cây nhị phân  
biểu diễn các node là số  
nguyên

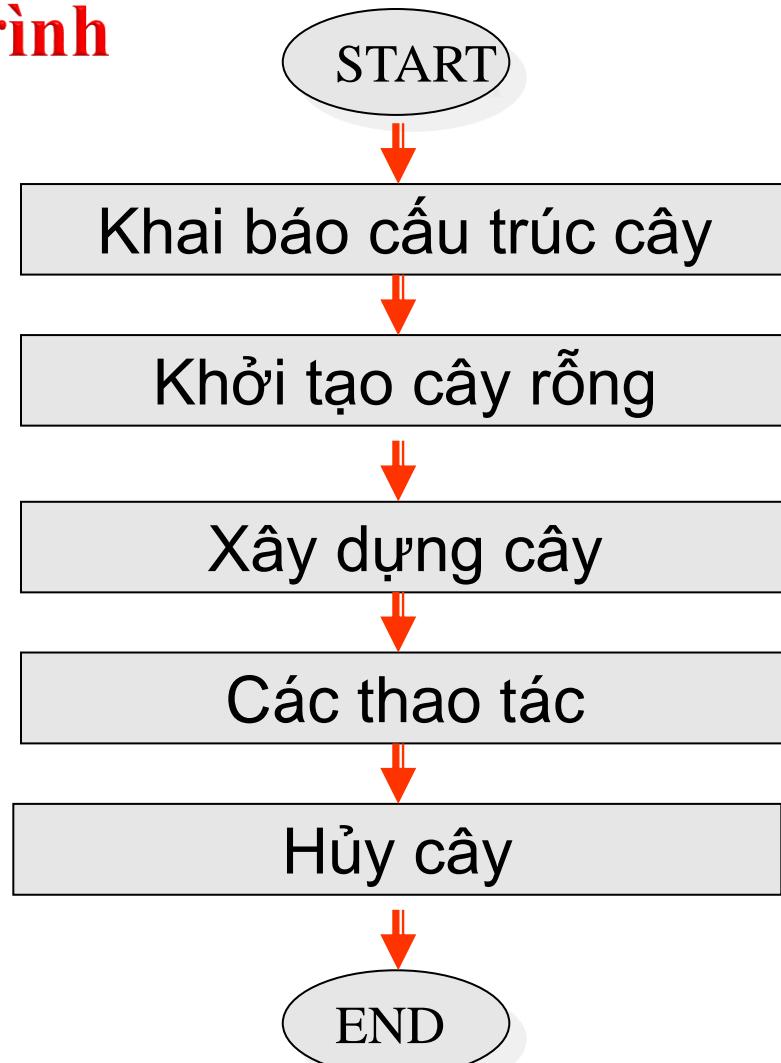
```
struct TNODE
{
    int Key;
    struct TNODE *pLeft,
    struct TNODE *pRight;
}*TREE;
```

### 6.3. Trình tự xây dựng chương trình

- **Bước 1:** Khai báo kiểu dữ liệu biểu diễn cây
- **Bước 2:** Xây dựng hàm đưa dữ liệu (nhập) vào cây
- **Bước 3:** Xây dựng các thao tác duyệt, tìm kiếm, huỷ, ...

• Lưu ý:

1. Trước khi tạo node mới phải *xin cấp phát vùng nhớ*.
2. Trước khi kết thúc chương trình phải *huỷ cây* (giải phóng vùng nhớ).



## 6.4. Các thao tác

- |   |                                   |
|---|-----------------------------------|
| 1. int <i>InitTree</i> (TREE &t)            | : khởi tạo cây                    |
| 2. int <i>IsEmptyTree</i> (TREE t)          | : kiểm tra cây rỗng               |
| 3. NODE <i>CreateNode</i> (int x)           | : cấp phát 1 node có dữ liệu là x |
| 4. int <i>AddNode</i> (TREE &t, int x)      | : Đưa node vào cây                |
| 5. void <i>TreeTraversal</i> (TREE t)       | : Duyệt cây                       |
| 6. void <i>Info</i> (TREE t)                | : Cho biết các thông tin của cây  |
| 7. int <i>Search</i> (TREE t)               | : Tìm kiếm                        |
| 8. int <i>DeleteNode</i> (TREE &t, NODE p): | Xoá node trên cây                 |
| 9. void <i>ClearTree</i> ((TREE &t)         | : xóa toàn bộ cây                 |

## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.1. Khởi tạo cây

```
int InitTree (TREE &t)
{
    t = NULL;
}
```

#### 6.4.2. Kiểm tra cây rỗng

```
int IsEmptyTree (TREE t)
{
    return (t == NULL)
}
```

## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.3. Tạo 1 node mới

```
Node* CreateNode(int x)
```

```
{
```

```
Node *p;
```

```
p = new Node; //Cấp phát vùng nhớ cho phần tử
```

```
if ( p==NULL)
```

```
    exit(1);
```

```
p->Info = x; //gán dữ liệu cho node
```

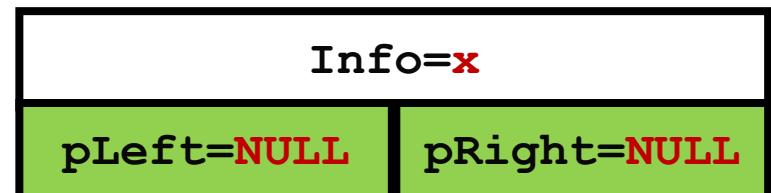
```
p->pLeft = NULL;
```

```
p->pRight = NULL;
```

```
return p;
```

```
}
```

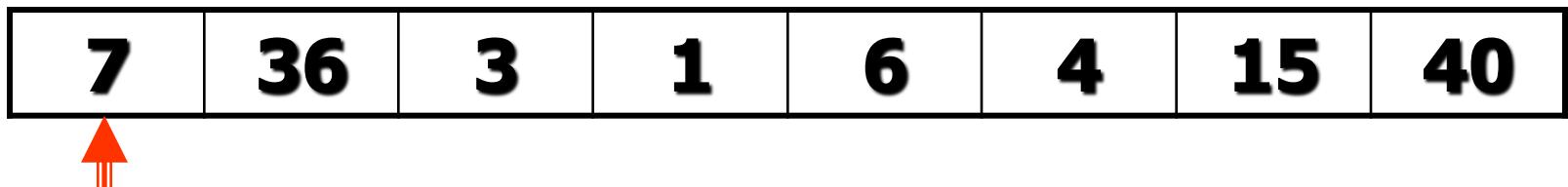
*p*



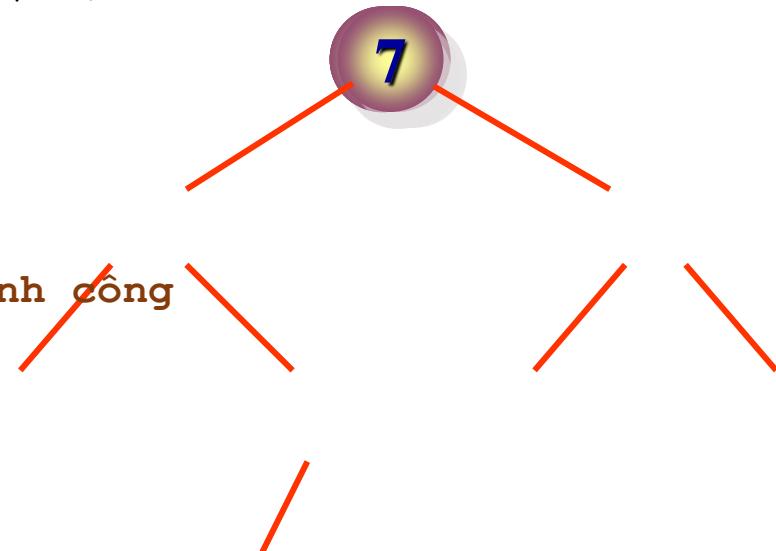
## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.4. Đưa node vào cây



```
int AddNode (TREE &t, int x)
{  if(t!=NULL)
   {   if(x==t->Key)    //giá trị x đã có trên cây
       return 0;
   else
       if(x<t->Key) AddNode(t->pLeft, x);
       else            AddNode(t->pRight, x);
   }
   else
   {   NODE* p;
       p= CreateNode(x);
       if(t==NULL)
           return -1;//thêm KHÔNG thành công
       else
           t=p;
       return 1;
   }
}
```



## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.4. Đưa node vào cây

1. Vẽ hình ảnh của cây khi biết thứ tự các khoá nhập vào như sau:

- a. 8, 3, 5, 2, 20, 11, 30, 9, 18, 4.
- b. 27, 19, 10, 21, 3, 15, 41, 50, 30, 7
- c. H, B, C, A, E, D, T, M, X, O

2. Từ kết quả của bài 1, lần lượt thực hiện:

- a. Tính chiều cao của node có giá trị là 9 (1.a), 30(1.b), O(1.c)
- b. Tính chiều cao của cây trong 3 cây có ở bài tập 1.
- c. Đếm số lượng node lá của cây trong 3 cây có ở bài tập 1?
- d. Tính tổng các node không phải node lá của cây trong 2 cây có ở bài tập 1.a và 1.b

## 6. CÂY NHỊ PHÂN TÌM KIẾM

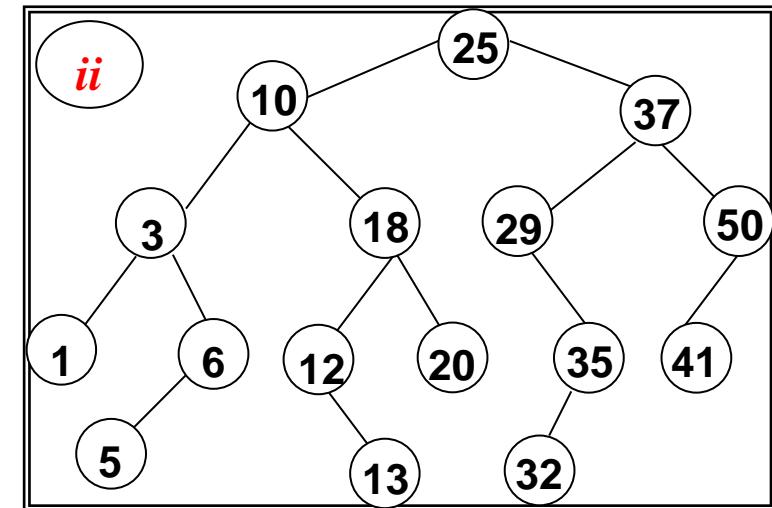
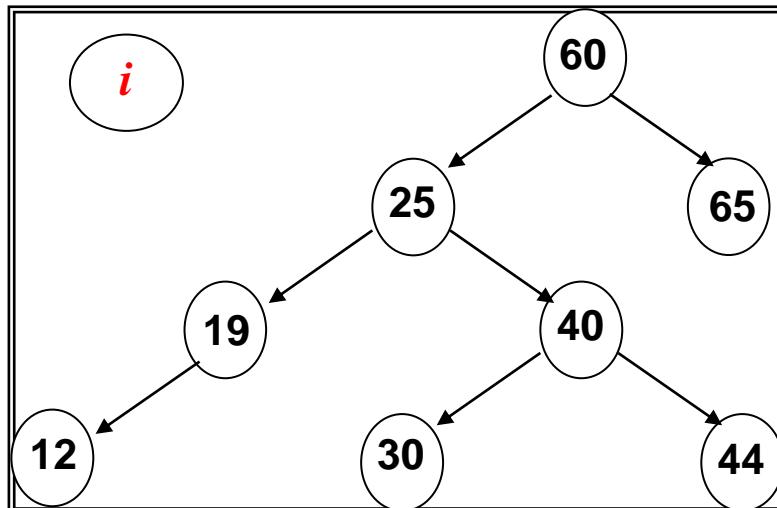
### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.4. Đưa node vào cây

3. Cho BST có 10 node chứa các số từ 0 đến 9. Vẽ cây cho các trường hợp sau:

<i>Trường hợp</i>	<i>Tổng các nút lá</i>
1	12
2	18
3	19

4. Cho biết thứ tự của các phần tử khi thêm vào cây ra sao để có được cấu trúc này? (Giả sử lúc đầu cây rỗng)



## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.4. Đưa node vào cây

5. Cho một cây nhị phân gồm 9 node với key có giá trị từ 1 đến 9. Lần lượt vẽ cây cho mỗi trường hợp sau:
- i. Biết tổng các node lá bằng 25.
  - ii. Vẽ cây khi cây có 2 node lá, tổng các node lá là 12.
  - iii. Vẽ cây khi biết tổng các node không lá là 34
  - iv. Vẽ cây có 4 node lá, sao cho tổng các node lá là nhỏ nhất.
  - v. Vẽ cây sao cho các lá đều là số lẻ.
  - vi. Xác định số node lá tối thiểu, số node lá tối đa của cây. Vẽ hình minh họa cho 2 trường hợp trên. Từ đó, đưa ra công thức tính số node lá tối đa khi biết số node của cây.

## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.4. Đưa node vào cây

6. Vẽ một cây nhị phân tìm kiếm gồm 9 node với key có giá trị từ 1 đến 9 và chiều cao của cây =3 trong các trường hợp sau:
  - i. Biết tổng các node lá bằng 20.
  - ii. Biết tổng các node lá bằng 25.
7. Vẽ một cây nhị phân đúng (*strickly binary tree*) gồm 9 node với key có giá trị từ 1 đến 9 trong các trường hợp sau:
  - i. Biết tổng các node lá bằng 20.
  - ii. Biết tổng các node lá bằng 24.
  - iii. Biết tổng các node lá bằng 25.
8. Vẽ một cây nhị phân đầy đủ (*complete binary tree*) gồm 15 node với key có giá trị từ 0 đến 14 và tổng nút lá là 56.

## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.5. Duyệt cây (*Tree traversal*)

Có 3 thành phần tham gia vào quá trình duyệt cây là *Node, Left, Right*

⇒ Có 6 cách duyệt

i. ~~Node - Right - Left~~

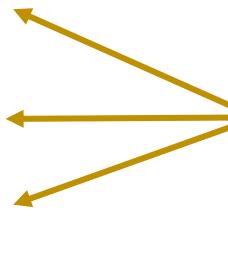
ii. *Node* - Left -Right

iii. *Left-Node*-Right

iv. *Left-Right-Node*

v. ~~Right-Node-Left~~

vi. ~~Right - Left -Node~~



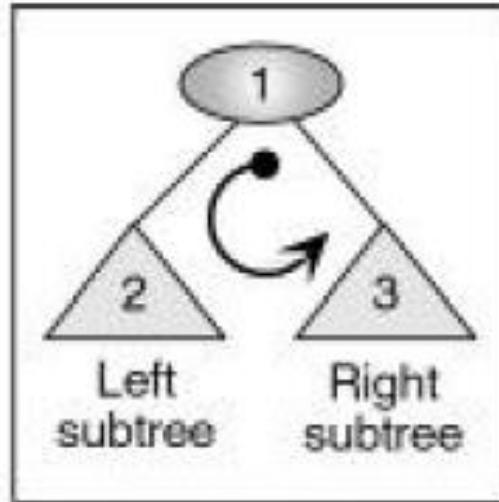
Trong thực tế,  
người ta chỉ quan  
tâm đến 3 cách dựa  
trên vị trí của root  
node (trong đó *Left*  
luôn được duyệt  
trước *Right*)

## 6. CÂY NHỊ PHÂN TÌM KIẾM

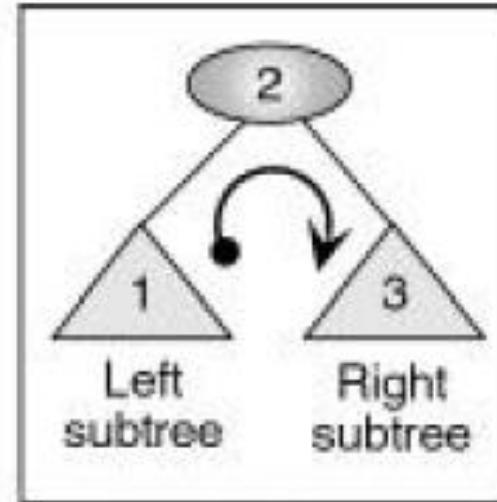
### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.5. Duyệt cây (*Tree traversal*)

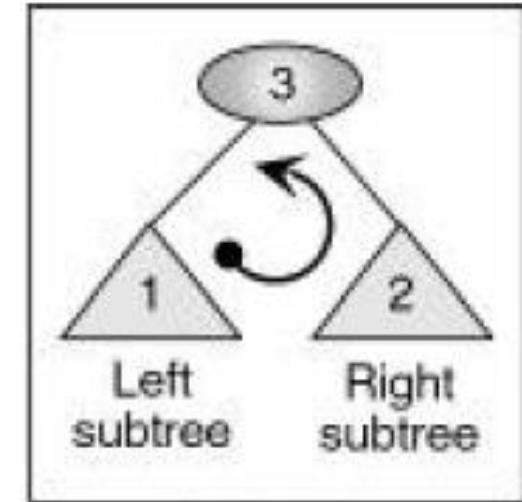
- i. Duyệt gốc trước (*pre-order traversal* hay *Node-Left-Right*)
- ii. Duyệt gốc giữa (*in-order traversal* hay *Left-Node-Right*)
- iii. Duyệt gốc sau (*post-order traversal* hay *Left-Right-Node*)



*pre-order traversal*



*in-order traversal*



*post-order traversal*

## 6. CÂY NHỊ PHÂN TÌM KIẾM

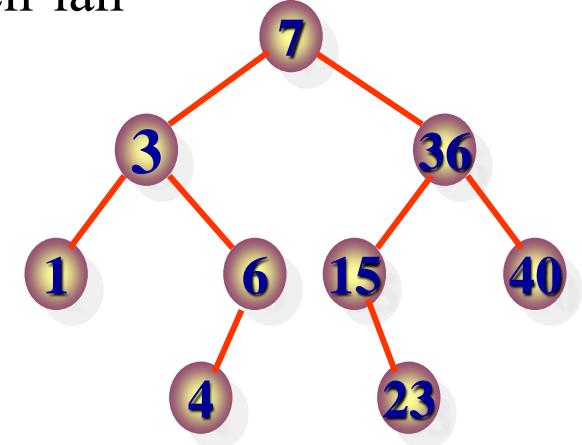
### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.5. Duyệt cây (*Tree traversal*)

##### 6.4.5.1. NLR

Tại node  $t$  đang xét, nếu khác rỗng thì thực hiện lần lượt

- i. In giá trị của  $t$
- ii. Duyệt cây con bên **trái** của  $t$  theo thứ tự NLR
- iii. Duyệt cây con bên **phải** của  $t$  theo thứ tự NLR



## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.5. Duyệt cây (*Tree traversal*)

##### 6.4.5.1. NLR

Tại node t đang xét, nếu khác rỗng thì thực hiện lần lượt

- i. In giá trị của t
- ii. Duyệt cây con bên trái của t theo thứ tự NLR
- iii. Duyệt cây con bên phải của t theo thứ tự NLR

```
void NLR (TREE t)
{ if(t!=NULL)
    { printf("%5d",t->Key;
        NLR(t->pLeft);
        NLR(t->pRight);
    }
}
```

## 6. CÂY NHỊ PHÂN TÌM KIẾM

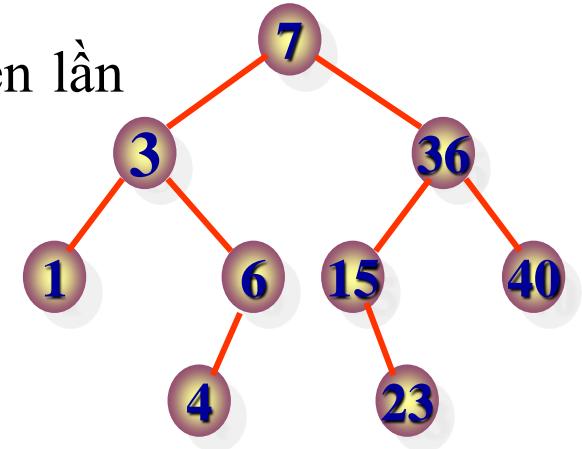
### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.5. Duyệt cây (*Tree traversal*)

##### 6.4.5.2. LNR

Tại node  $t$  đang xét, nếu khác rỗng thì thực hiện lần lượt

- i. Duyệt cây con bên **trái** của  $t$  theo thứ tự LNR
- ii. In giá trị của  $t$
- iii. Duyệt cây con bên **phải** của  $t$  theo thứ tự LNR



⇒ Viết lại các giá trị trên cây theo thứ tự tăng dần

⇒ Nếu yêu cầu in giá trị trên cây theo thứ tự **giảm dần** thì duyệt bằng cách nào?

## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.5. Duyệt cây (*Tree traversal*)

##### 6.4.5.2. LNR

Tại node t đang xét, nếu khác rỗng thì:

- i. Duyệt cây con bên trái của t theo thứ tự **LNR**
- ii. In giá trị của t
- iii. Duyệt cây con bên phải của t theo thứ tự **LNR**

```
void LNR (TREE t)
{
    if(t!=NULL)
    {
        LNR(t->pLeft) ;
        printf ("%5D", t->Key) ;
        LNR(t->pRight) ;
    }
}
```

## 6. CÂY NHỊ PHÂN TÌM KIẾM

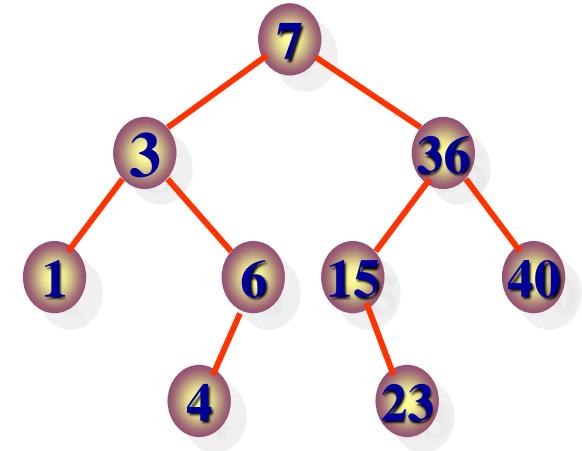
### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.5. Duyệt cây (*Tree traversal*)

##### 6.4.5.3. LRN

Tại node  $t$  đang xét, nếu khác rỗng thì thực hiện lần lượt

- i. Duyệt cây con bên **trái** của  $t$  theo thứ tự **LRN**
- ii. Duyệt cây con bên **phải** của  $t$  theo thứ tự **LRN**
- iii. In giá trị của  $t$



---

## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.5. Duyệt cây (*Tree traversal*)

##### 6.4.5.3. LRN

Tại node t đang xét, nếu khác rỗng thì thực hiện lần lượt

- i. Duyệt cây con bên trái của t theo thứ tự **LRN**
- ii. Duyệt cây con bên phải của t theo thứ tự **LRN**
- iii. In giá trị của t

```
void LRN (TREE t)
{
    if(t!=NULL)
    {
        LRN (t->pLeft) ;
        LRN (t->pRight) ;
        printf ("%5d", t->Key) ;
    }
}
```

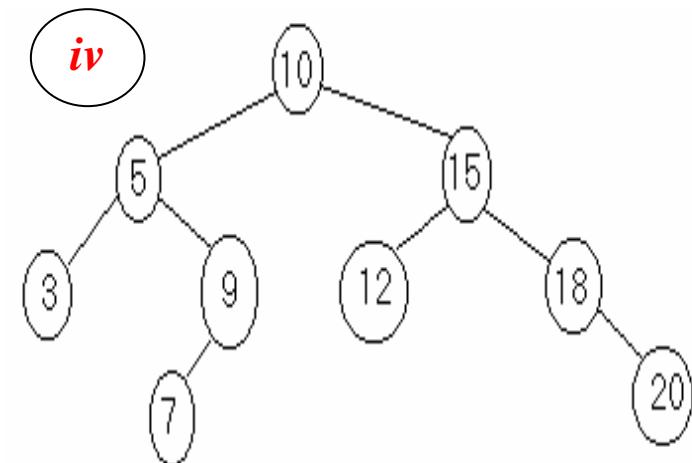
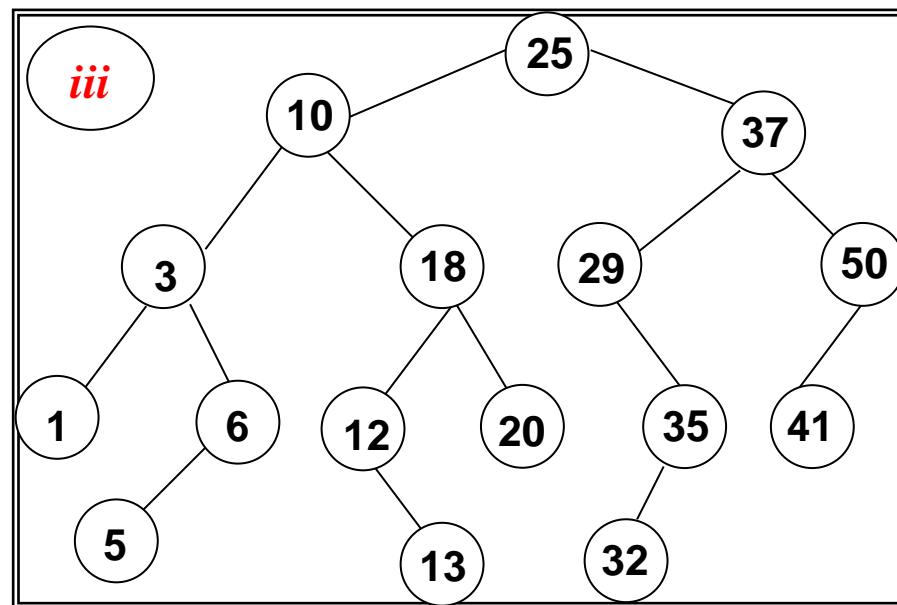
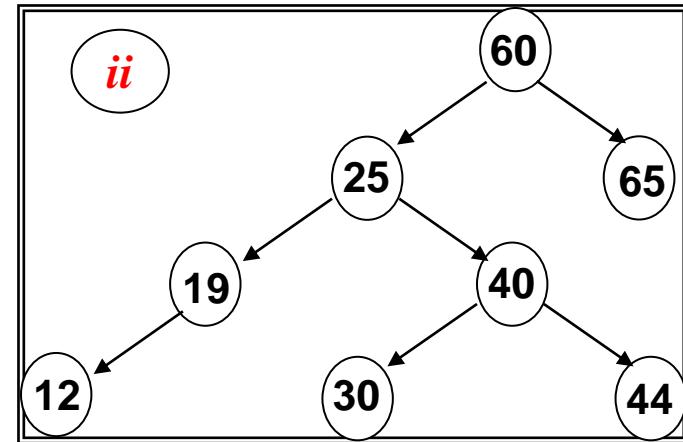
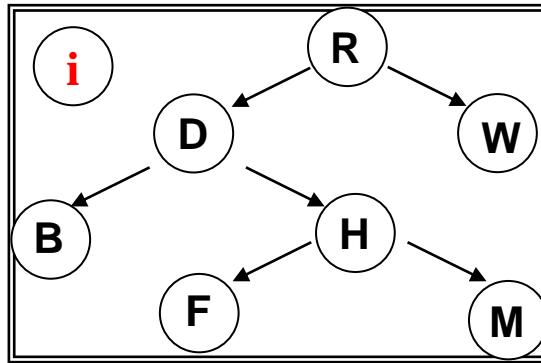
## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.5. Duyệt cây (*Tree traversal*)

## BÀI TẬP

1. Duyệt các cây trong hình sau theo 3 cách **NLR**, **LNR** và **LRN**



## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.5. Duyệt cây (*Tree traversal*)

##### Bài tập

2. Vẽ BST cho mỗi trường hợp, biết mỗi phần tử của cây là một số nguyên và nếu áp dụng phương pháp duyệt ta có kết quả:

Trường hợp	Cách duyệt	Kết quả
1	NLR	5 3 2 0 4 6 7 9 8
2	NLR	5 2 1 0 4 3 7 6 8 9
3	LRN	0 1 4 3 2 6 5 8 9 7
4	LRN	0 1 3 4 5 6 7 8 9 2
5	NLR	9, 4, 1, 3, 8, 6, 5, 7, 10, 14, 12, 13, 16, 19
6	LRN	1, 4, 7, 5, 3, 16, 18, 15, 29, 25, 30, 20, 8
7		
8		

## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.6. Cho biết các thông tin của cây

1. Số node lá (node bậc 0)
2. Số node có 1 cây con (node bậc 1)
3. Số node chỉ có 1 cây con phải
4. Số node có 1 cây con trái
5. Số node 2 cây con (node bậc 2)
6. Độ cao của cây
7. Số node của cây
8. Các node trên cùng mức của cây
9. Độ dài đường đi từ gốc đến node x
10. Kiểm tra xem cây có phải là cây nhị phân đúng (*strictly binary tree*) hay không?
11. Kiểm tra xem cây có phải là cây nhị phân đầy đủ (*complete binary tree*) hay không?

## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.6. Cho biết các thông tin của cây

##### 6.4.3.1. Số node lá

Nếu node t khác rỗng thì

- . Nếu node t có bậc 0 thì

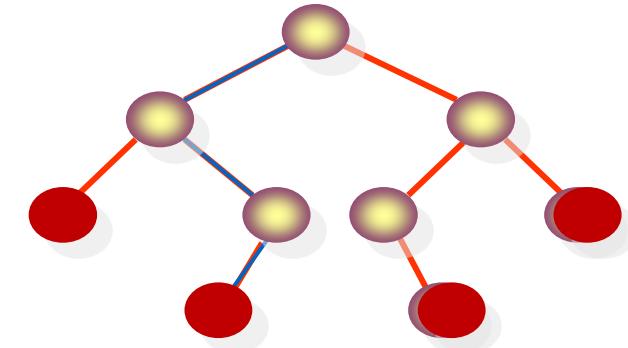
Trả về 1

- . Ngược lại

Trả về Số node lá cây trái t + Số node lá cây phải t

Nếu node t rỗng thì trả về 0

```
int DemNutLa (TREE t)
{
    if(t)
    {
        if(t->pLeft==NULL && t->pRight==NULL)
            return 1;
        else
            return DemNutLa (t->pLeft)+DemNutLa (t->pRight);
    }
    else
        return 0;
}
```



## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.6. Cho biết các thông tin của cây

##### 6.4.6.2. Số node có 1 cây con

Nếu node t khác rỗng thì

$d = \text{Số node bậc 1 của cây trái } t$

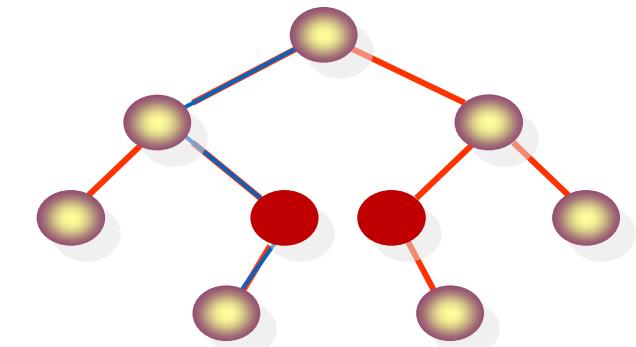
+ Số node bậc 1 của cây phải  $t$

Nếu node  $t$  có bậc 1 thì trả về  $d+1$

Ngược lại trả về  $d$

Nếu node  $t$  rỗng thì trả về 0

```
int DemNode1Con (TREE t)
{
    if(t)
    {
        int d=DemNut1Con (t->pLeft) + DemNut1Con (t->pRight);
        if((t->pLeft!=NULL && t->pRight==NULL)
            || (t->pLeft==NULL && t->pRight!=NULL))
            return d+1;
        else
            return d;
    }
    else
        return 0;
}
```



## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.6. Cho biết các thông tin của cây

### 6.4.6.3. Số node chỉ có 1 cây con phải

Nếu node t khác rỗng thì

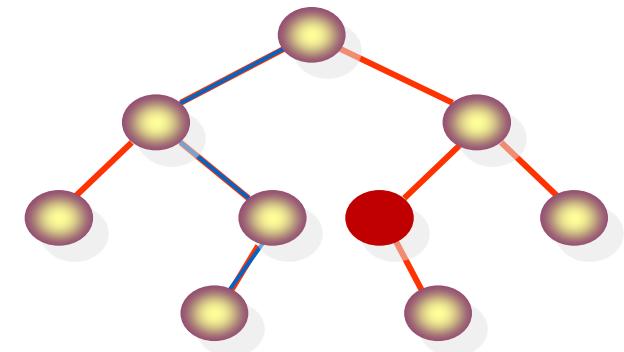
$$d = \text{Số node chỉ có 1 cây con phải của cây con trái } t$$
$$+ \text{Số node chỉ có 1 cây con phải của cây con phải } t$$

Nếu node t chỉ có 1 cây con phải thì trả về  $d+1$

Ngược lại trả về  $d$

Nếu node t rỗng thì trả về 0

```
int DemNut1ConPhai (TREE t)
{
    if(t)
    {
        int d = DemNut1ConPhai (t->pLeft)
                +DemNut1ConPhai (t->pRight) ;
        if(t->pLeft==NULL && t->pRight!=NULL)
            return d+1;
        else
            return d;
    }
    else
        return 0;
}
```



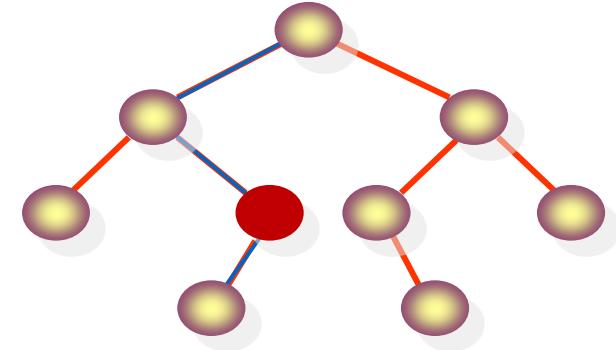
## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.6. Cho biết các thông tin của cây

##### 6.4.6.4. Số node chỉ có 1 cây con trái

Sinh viên tự cài đặt



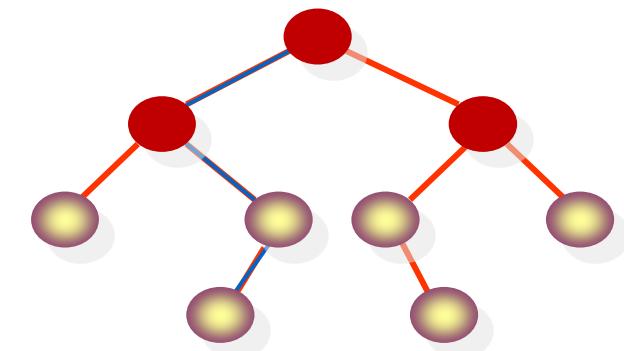
## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.6. Cho biết các thông tin của cây

##### 6.4.6.5. Số node có 2 cây con

Sinh viên tự cài đặt



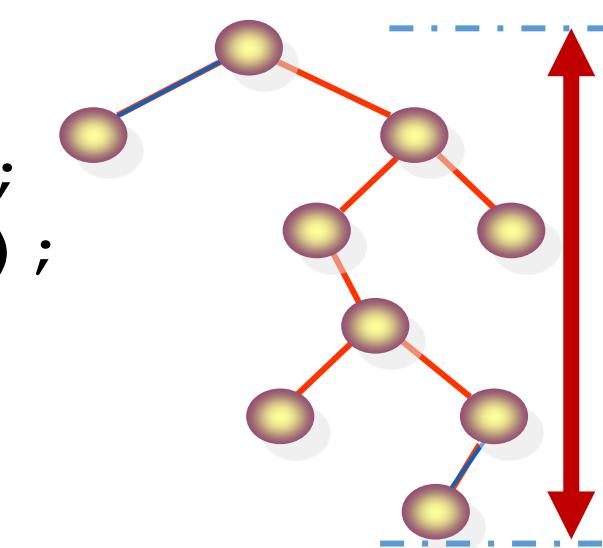
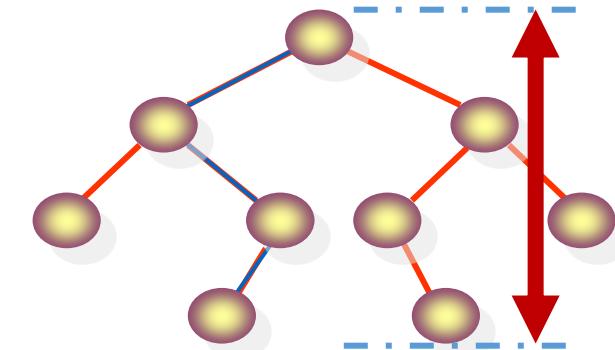
## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.6. Cho biết các thông tin của cây

##### 6.4.6.6. Độ cao của cây

```
int DoCaoCay (TREE t)
{
    if (t)
    {
        int t1=DoCaoCay (t->pLeft) ;
        int t2=DoCaoCay (t->pRight) ;
        return (t1>t2?t1:t2) + 1;
    }
    else
        return 0;
}
```



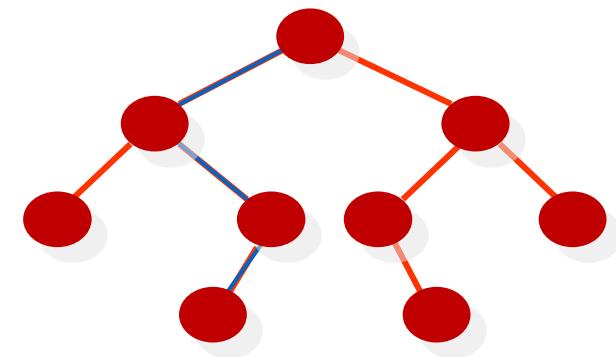
## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.6. Cho biết các thông tin của cây

##### 6.4.6.7. Số node của cây

Sinh viên tự cài đặt



## 6. CÂY NHỊ PHÂN TÌM KIẾM

### 6.4. CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

#### 6.4.6. Cho biết các thông tin của cây

##### 6.4.6.8. Các node trên từng mức

**Yêu cầu** in ra màn hình các node trên từng mức.

VD: với cây hiện có như hình bên, chương trình sẽ in ra kết quả như sau:

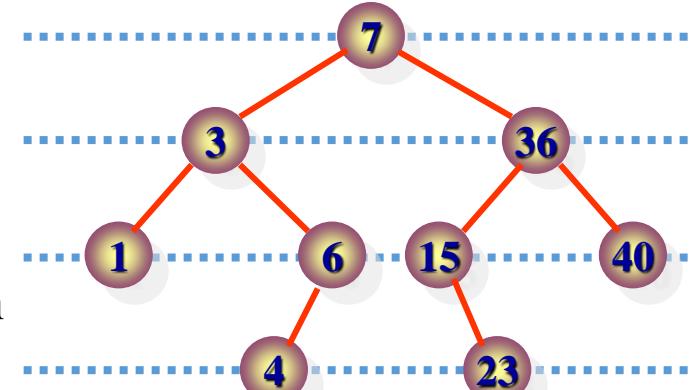
Mức 0: 7

Mức 1: 3 36

Mức 2: 1 6 15 40

Mức 3: 4 23

```
void InMucK(TREE t, int start, int level)
{
    if(t)
        { if(start == level) {
            cout<<t->Key;
            return;
        }
        else { start++;
            InMucK(t->pLeft, start, level);
            InMucK(t->pRight, start, level);
        }
    }
}
```



**Trong hàm main() :**

```
int ChieuCaoCay = DoCaoCay(T);
for (int i = 0; i < ChieuCaoCay; i++)
{
    cout << "\nMuc "<<i<<": ";
    InMucK(T, 0, i);
}
```

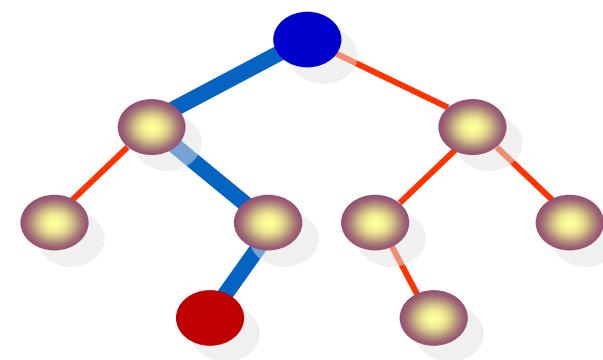
## 6. Cây nhị phân tìm kiếm

### 6.4. Các thao tác trên cây nhị phân tìm kiếm

#### 6.4.6. Cho biết các thông tin của cây

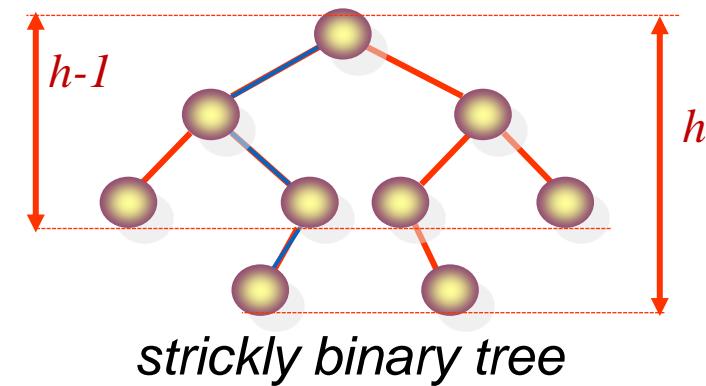
#### 6.4.6.9. Độ dài đường đi từ gốc đến node x

Sinh viên tự cài đặt



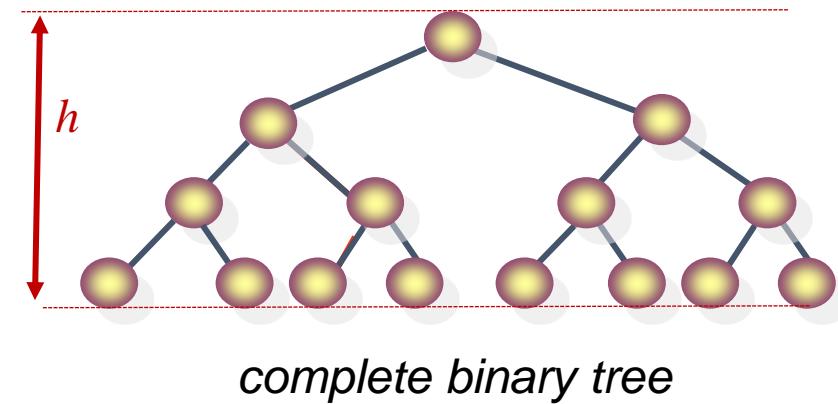
## 6.4.6.10. Kiểm tra xem cây có phải là cây nhị phân đúng (*strictly binary tree*)

Sinh viên tự cài đặt



## 6.4.6.11. Kiểm tra xem cây có phải là cây nhị phân đầy đủ (*complete binary tree*)

Sinh viên tự cài đặt



## 6. Cây nhị phân tìm kiếm

### 6.4. Các thao tác trên cây nhị phân tìm kiếm

#### 6.4.7. TÌM KIẾM

1. Tìm node chứa giá trị x
2. Tìm min
3. Tìm min của cây con bên phải
4. Tìm max
5. Tìm max của cây con bên trái
6. Tìm parent node của node có khóa là x.

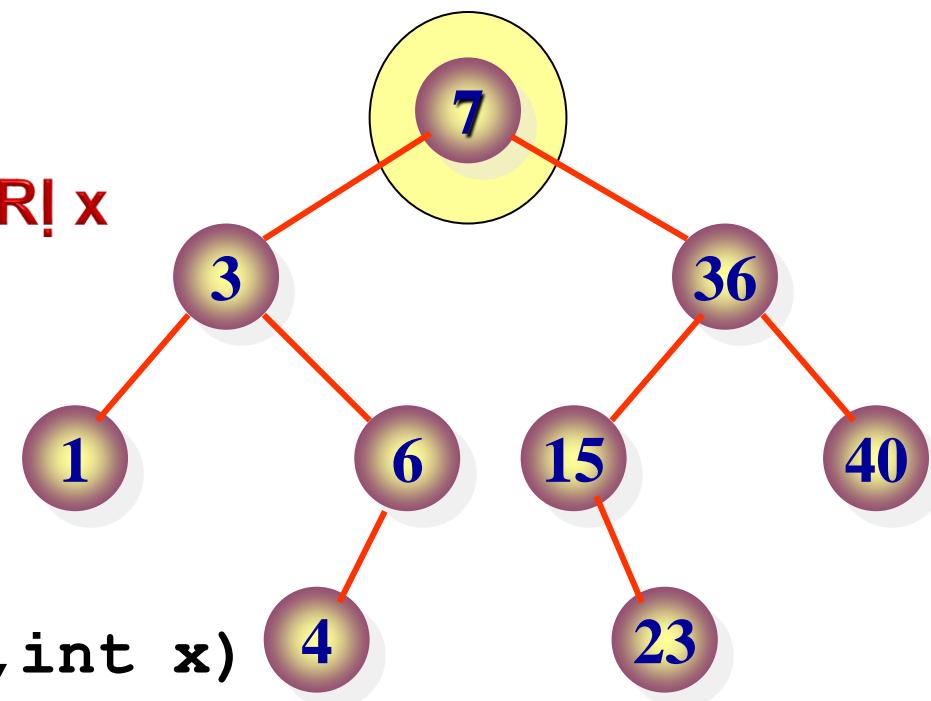
## 6. Cây nhị phân tìm kiếm

### 6.4. Các thao tác trên cây nhị phân tìm kiếm

#### 6.4.7. Tìm kiếm

##### 6.4.7.1. TÌM NODE CHỨA GIÁ TRỊ x

Ví dụ tìm  $x = 23$



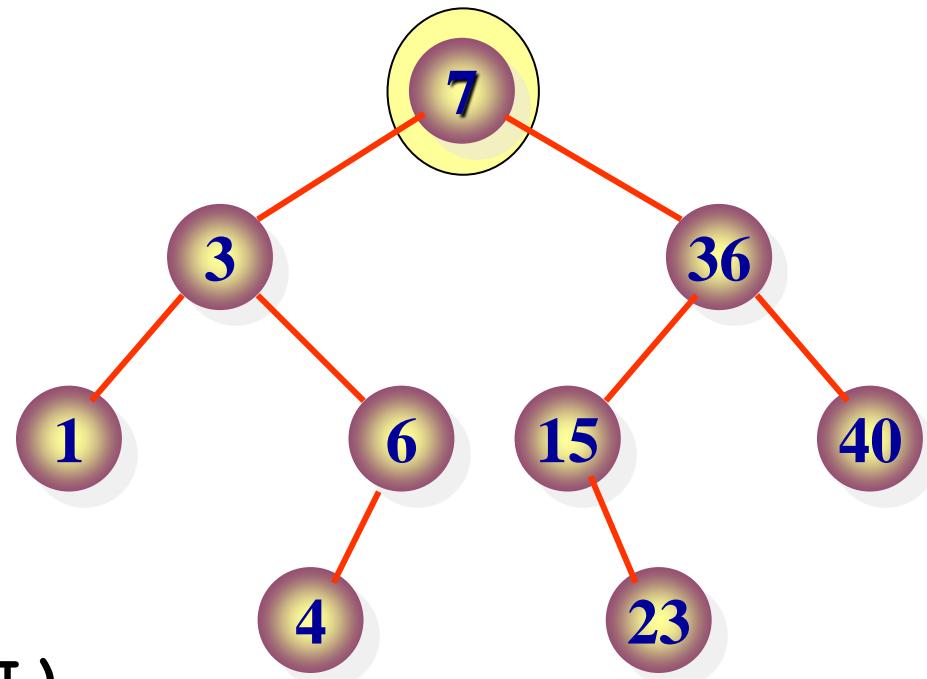
```
TNODE * TimKiem(TREE t, int x)
{
    if(t!=NULL)
    {
        if(t->Key==x)    return t;
        if(x<t->Key)
            return TimKiem(t->pLeft, x);
        else
            return TimKiem (t->pRight, x);
    }
    return t;
}
```

## 6. Cây nhị phân tìm kiếm

### 6.4. Các thao tác trên cây nhị phân tìm kiếm

#### 6.4.7. Tìm kiếm

##### 6.4.7.2. TÌM GIÁ TRỊ MIN CỦA CÂY



```
TNODE* Min(TREE t)
{   while(t->pLeft!=NULL)
    t=t->pLeft;
    return t;
}
```

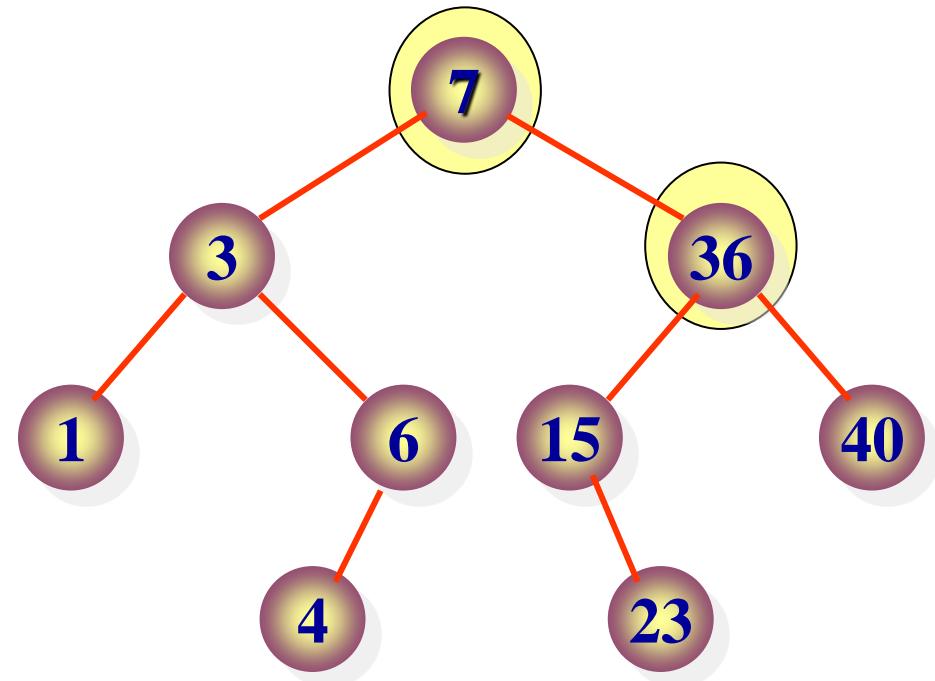
## 6. Cây nhị phân tìm kiếm

### 6.4. Các thao tác trên cây nhị phân tìm kiếm

#### 6.4.7. Tìm kiếm

#### 6.4.7.3. TÌM MIN CỦA CÂY CON BÊN PHẢI

Sinh viên tự cài đặt

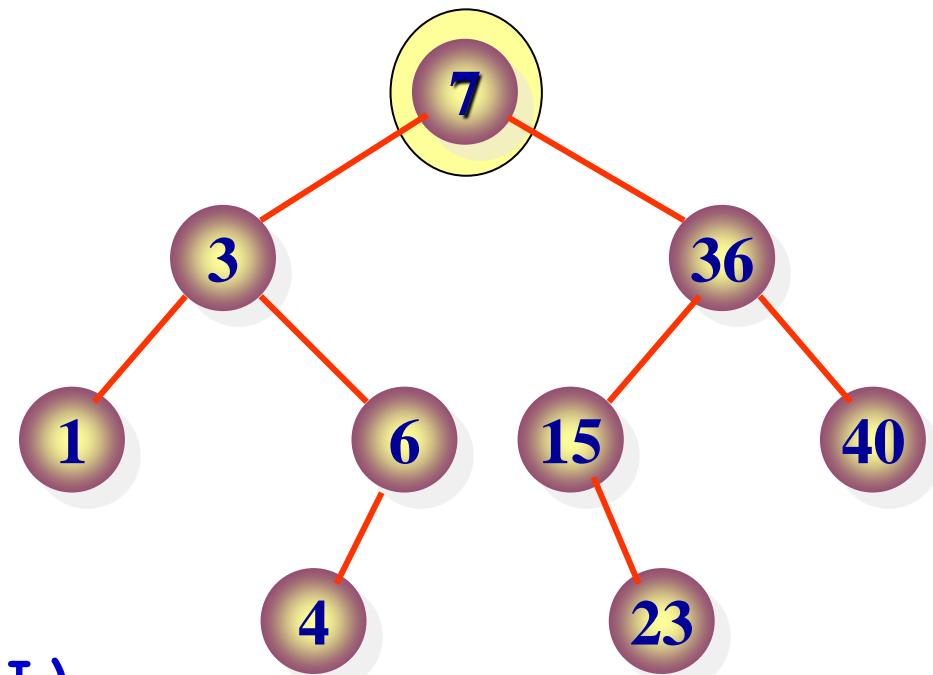


## 6. Cây nhị phân tìm kiếm

### 6.4. Các thao tác trên cây nhị phân tìm kiếm

#### 6.4.7. Tìm kiếm

#### 6.4.7.4. TÌM GIÁ TRỊ MAX CỦA CÂY



```
TNODE* Max (TREE t)
{   while(t->pRight!=NULL)
    t=t->pRight;
    return t;
}
```

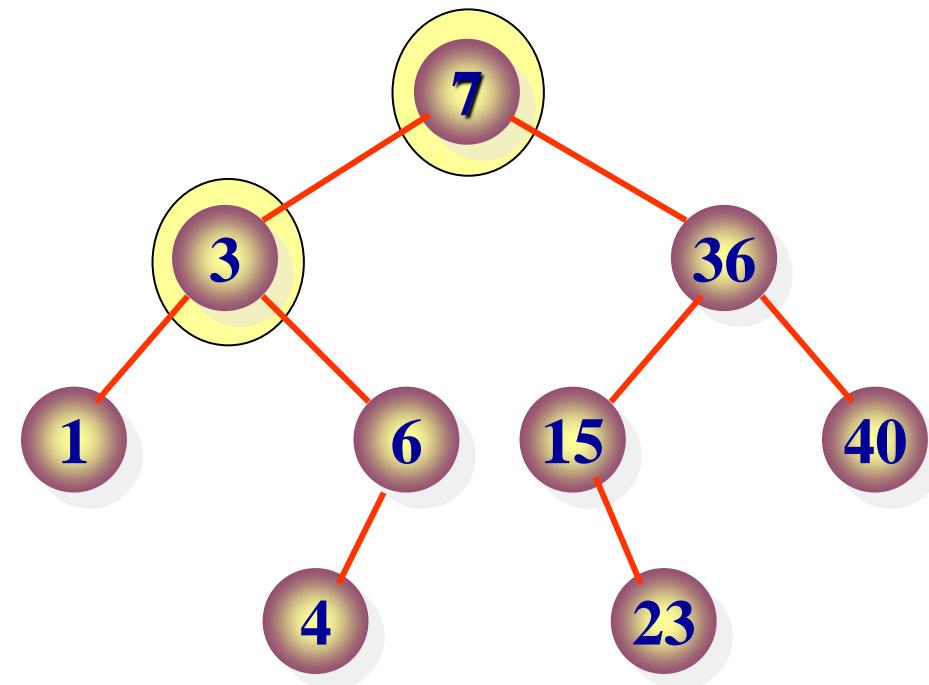
## 6. Cây nhị phân tìm kiếm

### 6.4. Các thao tác trên cây nhị phân tìm kiếm

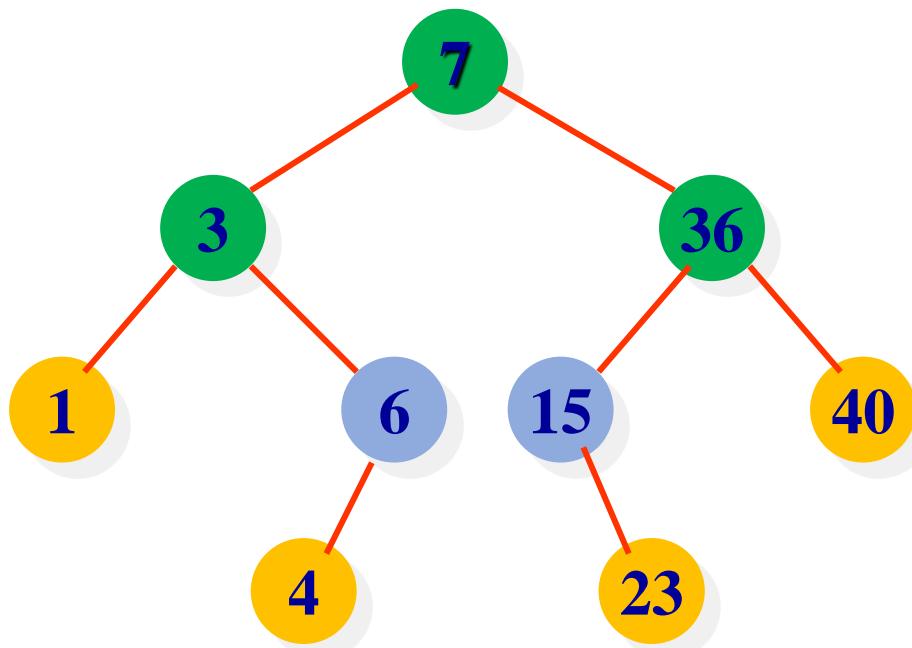
#### 6.4.7. Tìm kiếm

#### 6.4.7.5. TÌM MAX CỦA CÂY CON BÊN TRÁI

Sinh viên tự cài đặt



## 6.4.8. XÓA NODE TRÊN CÂY



1. Node lá
2. Node có 1 cây con
3. Node có 2 cây con

## 6. Cây nhị phân tìm kiếm

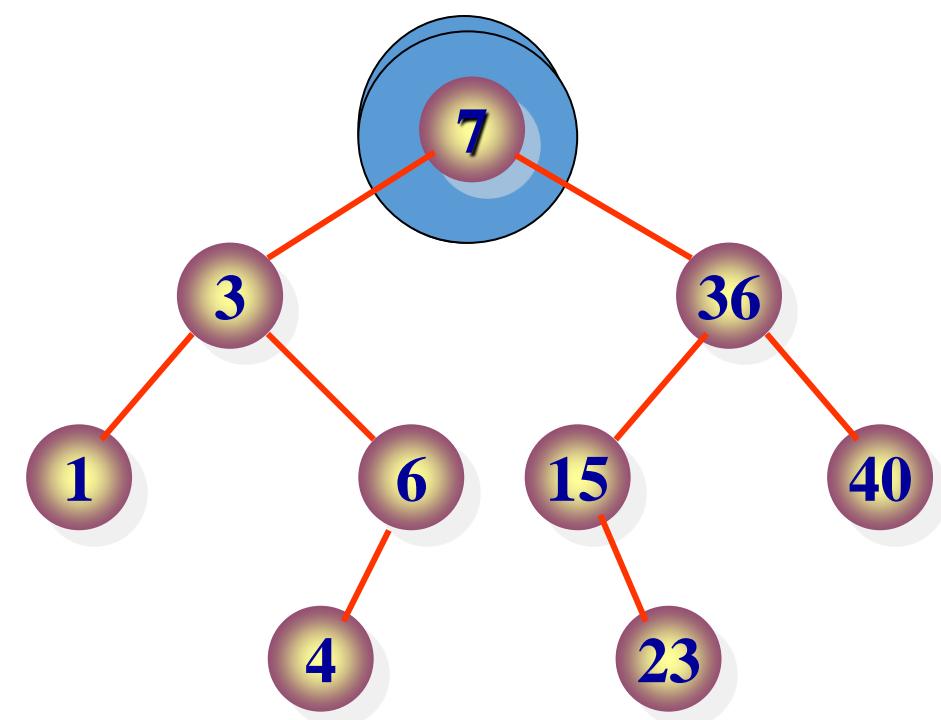
### 6.4. Các thao tác trên cây nhị phân tìm kiếm

#### 6.4.8. Xóa node trên cây

##### 6.4.8.1. Xóa node lá

Xóa 1

Xóa 23



Sơ bộ về mã lệnh sẽ thực hiện

```
void DelNode (TREE & t, int x)
{ if(t!=NULL)
  { if(x<t->Key)
      DelNode(t->pLeft,x) ;
    else
      if(x>t->Key)
        DelNode(t->pRight,x) ;
    delete pHuy;
  }
}
```

## 6. Cây nhị phân tìm kiếm

### 6.4. Các thao tác trên cây nhị phân tìm kiếm

#### 6.4.8. Xóa node trên cây

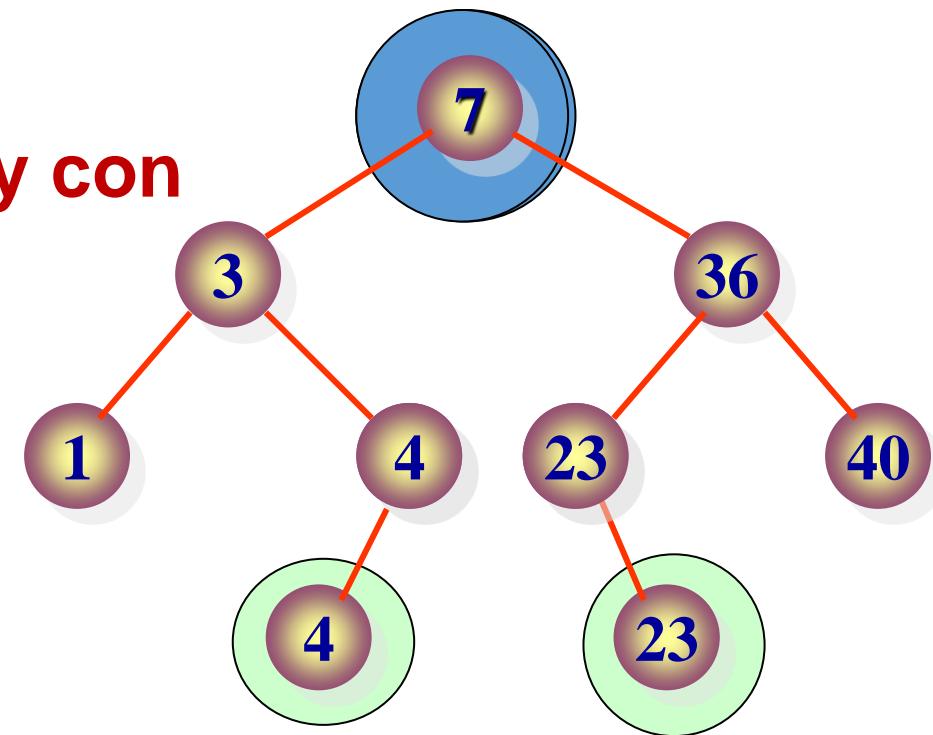
##### 6.4.8.2. Xóa node có 1 cây con

Xóa 6

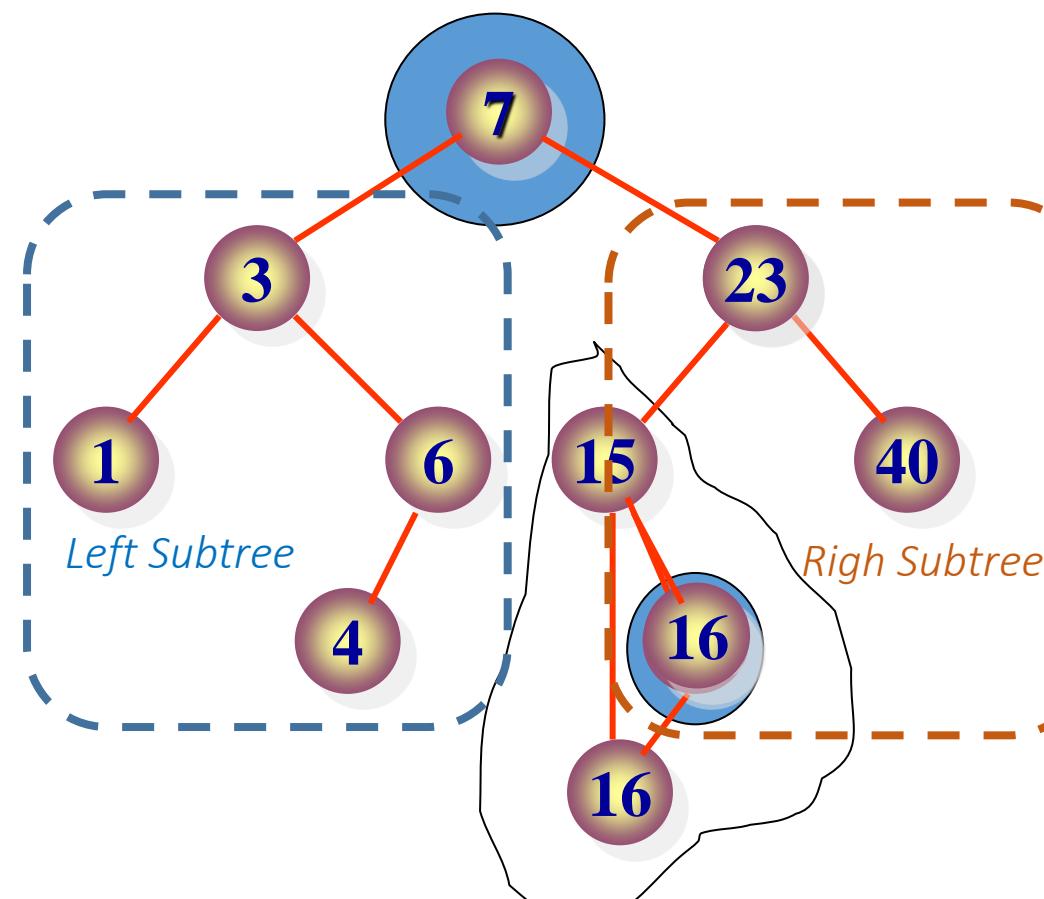
Xóa 15

Sơ bộ về mã lệnh sẽ thực hiện

```
void DelNode (TREE & t, int x)
{  if(t!=NULL)
   {  if(x<t->Key)
       DelNode (t->pLeft,x) ;
      else
       {  if(x>t->Key)     DelNode (t->pRight,x) ;
          else
           {  NODE * pHuy=t;
              if(t->pLeft==NULL)           t=t->pRight;
              else
               if(t->pRight==NULL)         t=t->pLeft;
               delete pHuy;
           }
       }
   }
```



### 6.4.8.3. Xóa node có 2 cây con



Tìm node thế mạng

Cách 1: Tìm node trái  
nhất của cây con phải

Cách 2: Tìm node phải  
nhất của cây con trái

Xóa 36 (Cách 2)

## 6. Cây nhị phân tìm kiếm

### 6.4. Các thao tác trên cây nhị phân tìm kiếm

#### 6.4.8. Xóa node trên cây

##### 6.4.8.3. Xóa node có 2 cây con

## Tìm node thế mạng

```
void TimTheMang (TREE &pHuy, TREE & q)
{   if (q->pLeft)
        TimTheMang (pHuy, q->pLeft) ;
    else
    {   pHuy->Key=q->Key ;
        pHuy=q;
        q=q->pRight;
    }
}
```

## 6. Cây nhị phân tìm kiếm

### 6.4. Các thao tác trên cây nhị phân tìm kiếm

#### 6.4.8. Xóa node trên cây

##### 6.4.8.3. Xóa node có 2 cây con

## Xóa một node có giá trị x

```
void DelNode (TREE & t, int x)
{ if(t!=NULL)
{ if(x<t->Key)
    DelNode (t->pLeft,x) ;
  else
{ if(x>t->Key)      DelNode (t->pRight,x) ;
  else
{ TNODE * pHuy=t;
  if(t->pLeft==NULL)          t=t->pRight;
  else
    if(t->pRight==NULL)        t=t->pLeft;
    else      TimTheMang (pHuy,t->pRight) ;
  delete pHuy;
}
}
}
}
```

## 6.4.9. Hủy toàn bộ cây

*Giải thuật*

Nếu node khác rỗng

- Hủy cây bên trái t
- Hủy cây bên phải t
- Hủy node t

## 7.BÀI TẬP

### 7.1. Tổng hợp về BST

1. Cho dãy số theo thứ tự nhập từ trái sang phải:

**20, 15, 35, 30, 11, 13, 17, 36, 47, 16, 38, 28, 14**

- i. Vẽ cây nhị phân tìm kiếm cho dãy số trên
- ii. Lần lượt cho biết kết quả duyệt cây trên theo thứ tự **NLR** và **LRN**
- iii. Cho biết độ cao của cây, số lượng node lá, số lượng các node có bậc 2.
- iv. Vẽ lại cây sau khi thêm node: **25** và **91**
- v. Trình bày từng bước và vẽ lại cây sau khi lần lượt xoá các node: **11** và **35**. Nếu cần tìm phần tử thế mạng cho node cần xóa, hãy chọn node trái nhất của cây con phải.

## 7.1. Tổng hợp về BST

2. Biết thứ tự các khoá nhập vào khi tạo cây là: **17, 4, 9, 5, 6, 22, 91, 18, 32, 45, 87, 35, 6, 4**. Lần lượt vẽ hình ảnh của cây sau mỗi thao tác sau:

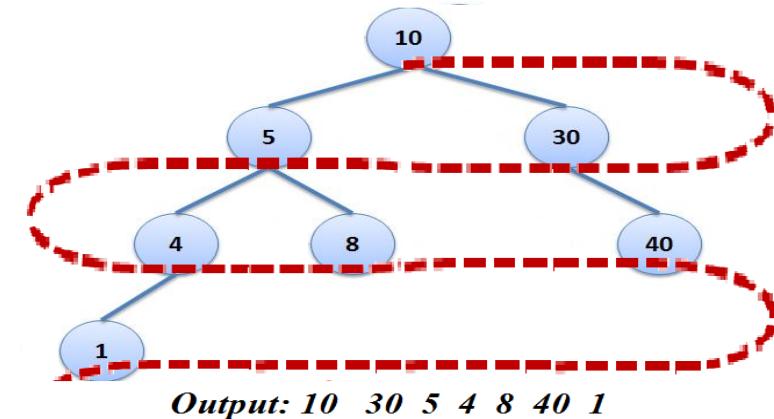
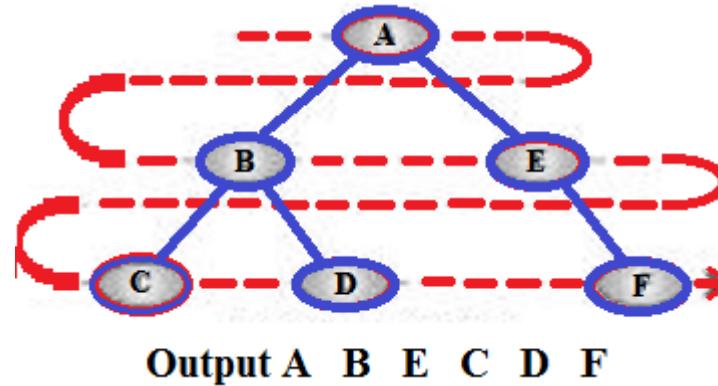
TT	Thao tác	Giá trị
1	Thêm	42, 50, 27
2	Xoá	29, 32. <i>Yêu cầu:</i> Chọn node thế mạng (nếu có) là phải nhất của cây con trái.
3	Thêm	67, 55, 81
4	Xoá	17, 4, 22. <i>Yêu cầu:</i> Chọn node thế mạng (nếu có) là trái nhất của cây con phải.
5	Thêm	18, 25, 49, 33
6	Xoá	Cần xoá những giá trị nào để tất cả các node không lá đều có bậc là 2
7	Thêm	Từ kết quả của thao tác 5, cần thêm những giá trị nào để tất cả các node không lá đều có bậc là 2
8	Thêm	Từ kết quả của thao tác 7, cần thêm những giá trị nào để chiều cao của cây đạt là 12. <i>Yêu cầu</i> số lượng node được thêm là ít nhất.
9	Thêm	Từ kết quả của thao tác 7, cần thêm những giá trị nào (số node cần thêm ít nhất là 1) để bậc của tất cả các node không lá đều là 2 và tổng các node lá >600.
10	Thêm	Từ kết quả của thao tác 7, có thể thêm những node nào để cây trở thành cây cân bằng hay không? <ul style="list-style-type: none"> <li>▫ Nếu có, vẽ hình minh họa.</li> <li>▫ Ngược lại, thực hiện cân bằng cây đang có. Vẽ hình minh họa.</li> </ul>

## 7.4. Lập trình

1. Viết chương trình thực hiện các yêu cầu sau:

- i. Hàm khởi tạo cây
- ii. Hàm thêm node vào cây
- iii. Hàm duyệt cây theo các cách:

- post-order
- in-order
- pre-order
- Duyệt cây theo mức (level) theo 2 dạng sau:



## 7.BÀI TẬP

### 7.4. Lập trình

2. Viết chương trình thực hiện các yêu cầu sau:

i. Cho biết các thông tin của cây

- Đếm số node trên cây
- Đếm số node trên từng mức của cây
- Tính tổng tất cả các khóa trên cây
- Đếm số node lá trên cây
- Đếm số node có duy nhất 1 cây con (không phân biệt trái hay phải)
- Số node chỉ có 1 cây con phải
- Số node có 1 cây con trái
- Đếm số node có 2 cây con
- Độ dài đường đi từ gốc đến node chứa giá trị X

ii. Tìm kiếm

- Cho biết khoá lớn nhất trong cây là bao nhiêu
- Liệt kê các số nguyên tố có trong cây

iii. Xoá node trên cây

- Xóa 1 node với giá trị do người dùng cung cấp
- Xóa toàn bộ cây.

