



TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



Cấu trúc dữ liệu và giải thuật

Nguyễn Khánh Phương

**Computer Science department
School of Information and Communication technology
E-mail: phuongnk@soict.hust.edu.vn**

Course outline

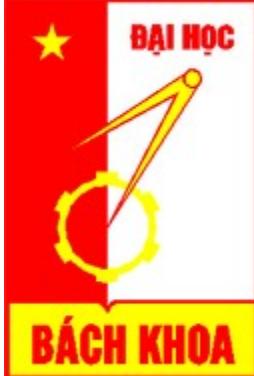
Chương 1. Các khái niệm cơ bản

Chương 2. Các cấu trúc dữ liệu cơ bản

Chương 3. Cây

Chương 4. Sắp xếp

Chương 5. Tìm kiếm



TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG



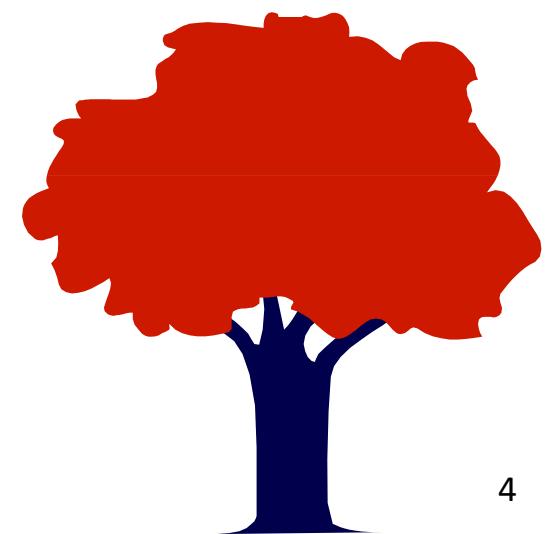
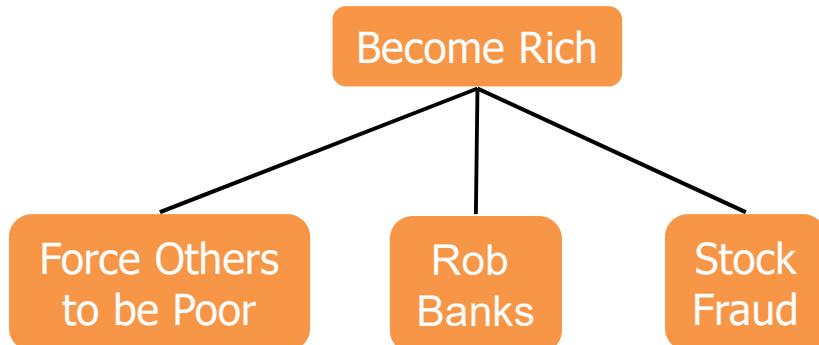
Chương 3. Cây

Nguyễn Khánh Phương

**Computer Science department
School of Information and Communication technology
E-mail: phuongnk@soict.hust.edu.vn**

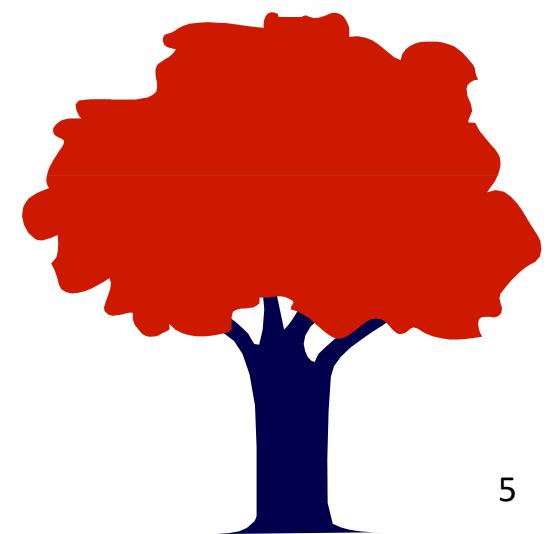
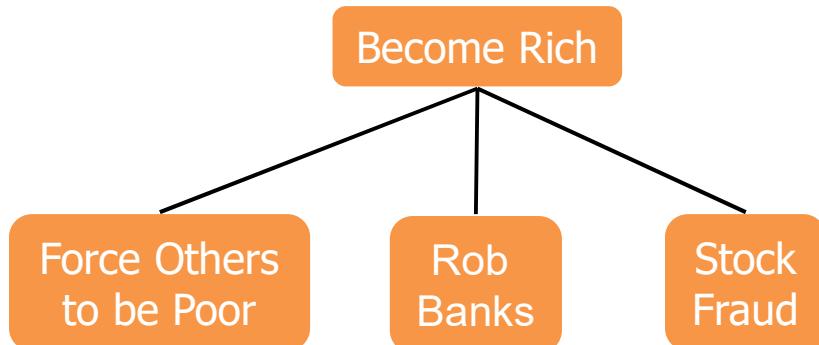
Nội dung

1. Định nghĩa và các khái niệm
2. Biểu diễn cây
3. Duyệt cây
4. Cây nhị phân



Nội dung

1. Định nghĩa và các khái niệm
2. Biểu diễn cây
3. Duyệt cây
4. Cây nhị phân



1. Định nghĩa và các khái niệm

- 1.1. Định nghĩa
- 1.2. Các thuật ngữ
- 1.3. Cây có thứ tự (Ordered tree)



1. Định nghĩa và các khái niệm

1.1. Định nghĩa

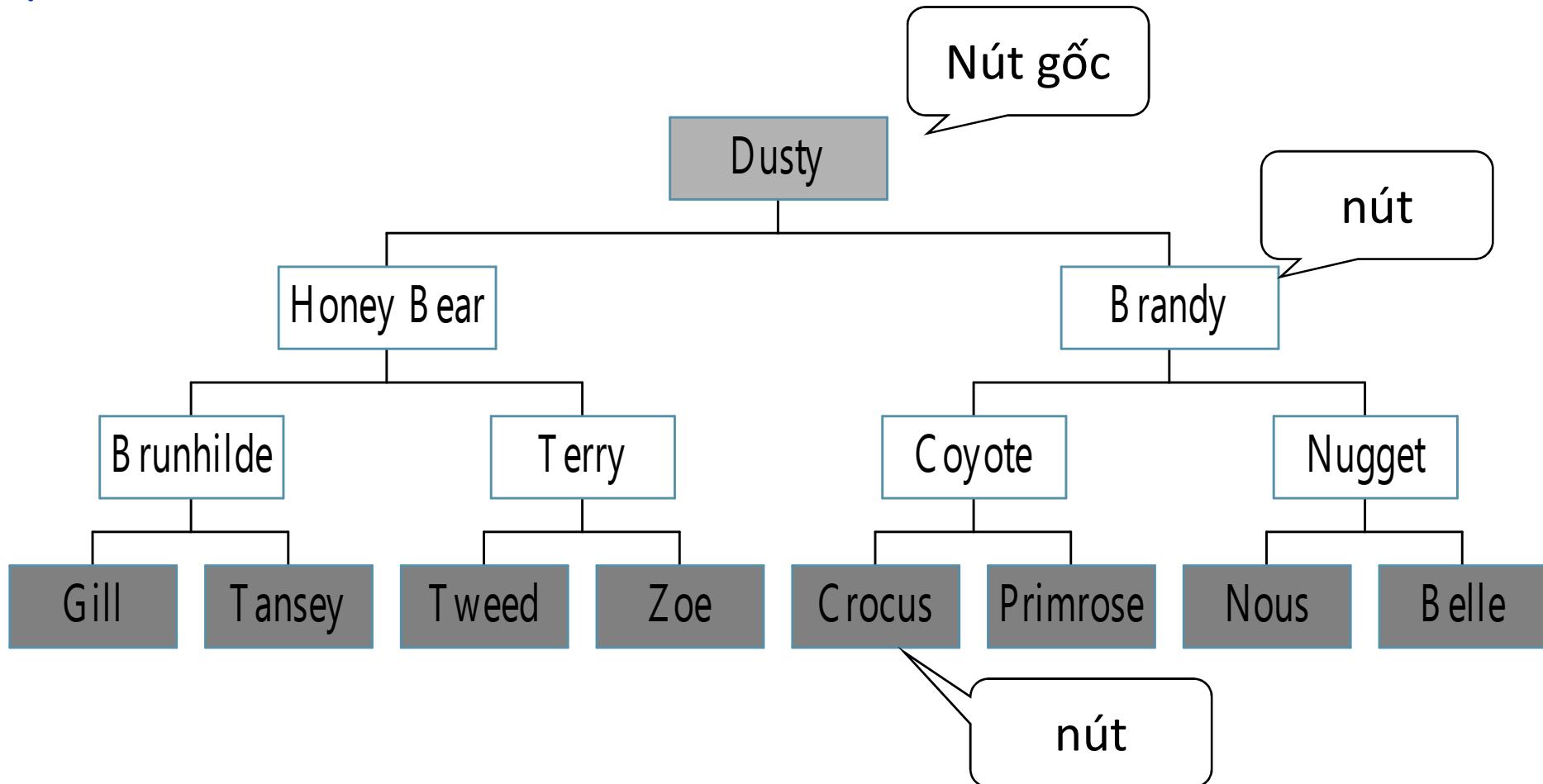
1.2. Các thuật ngữ

1.3. Cây có thứ tự (Ordered tree)



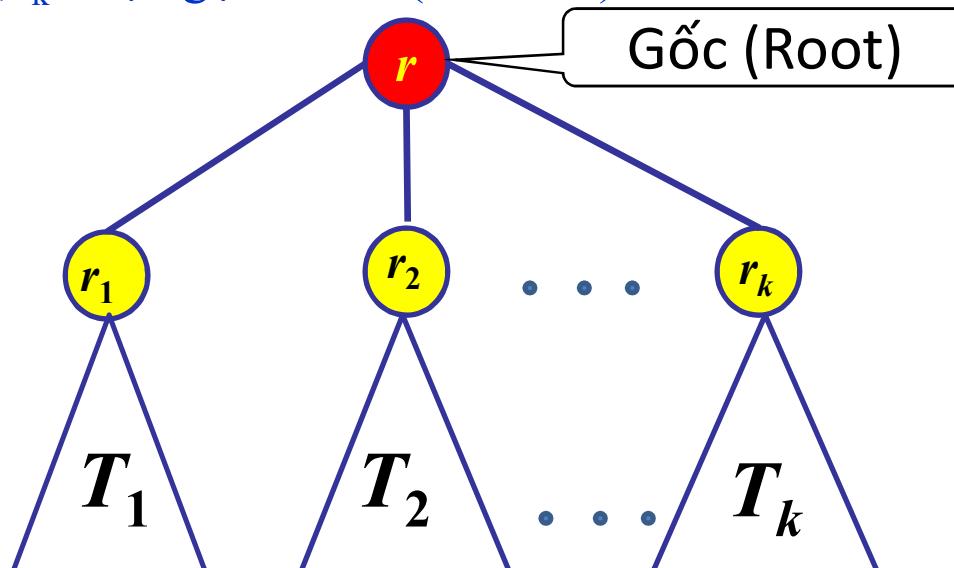
1.1. Định nghĩa

Cây gồm một tập các nút, có 1 nút đặc biệt gọi là gốc (**root**) và các cạnh nối các nút.



Cây: Định nghĩa đệ quy

- Bước cơ sở:
 - Cây rỗng: cây không có nút nào
 - Cây có 1 nút r : nút r được gọi là gốc của cây
- Bước đệ quy:
 - Giả sử T_1, T_2, \dots, T_k là các cây với gốc tương ứng là r_1, r_2, \dots, r_k
 - Ta có thể xây dựng cây mới bằng cách đặt nút r làm cha (parent) của các nút r_1, r_2, \dots, r_k :
 - Trong cây mới này: r là gốc; T_1, T_2, \dots, T_k là các cây con của gốc r ; Các nút r_1, r_2, \dots, r_k được gọi là con (children) của nút r .

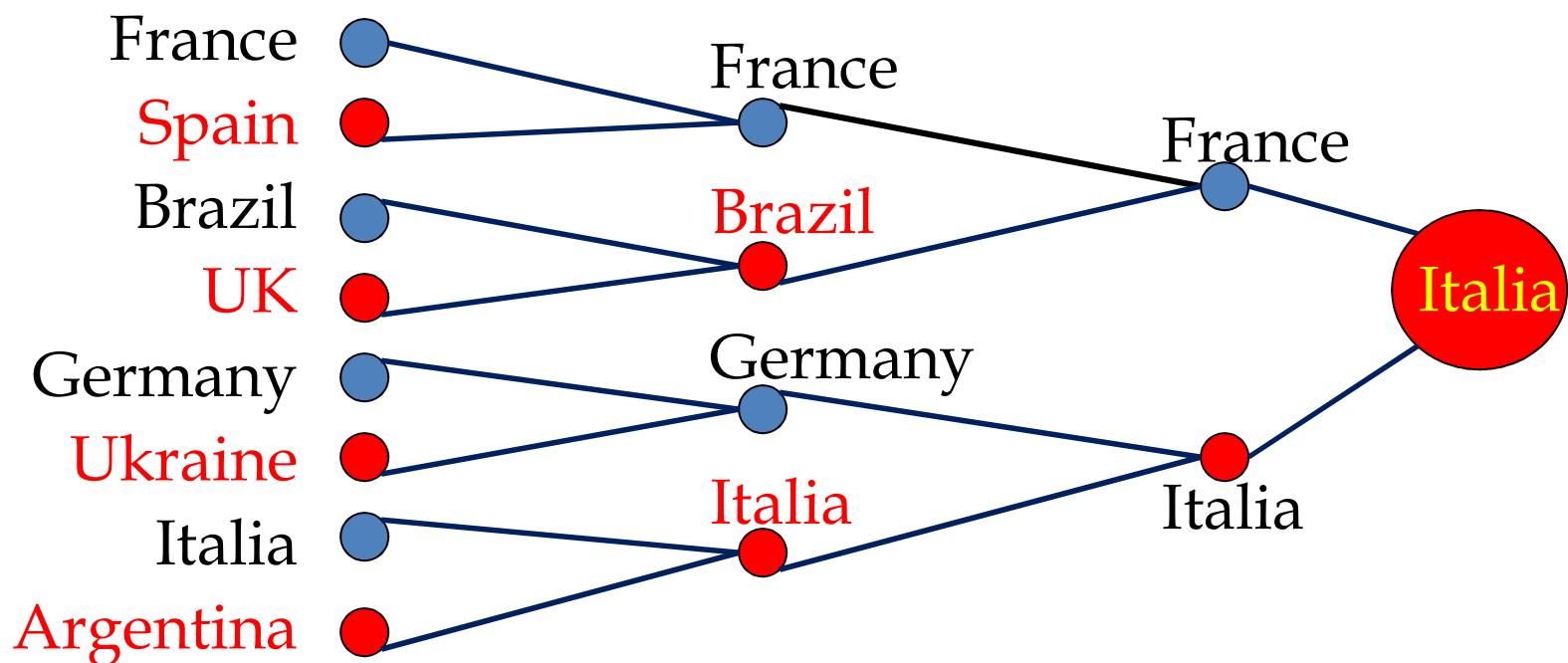


Các ứng dụng của cây

- Lịch thi đấu
- Cây gia phả
- Cây thư mục
- Cấu trúc của 1 quyển sách
- Cây biểu thức
- Sơ đồ phân cấp của 1 cơ quan
-

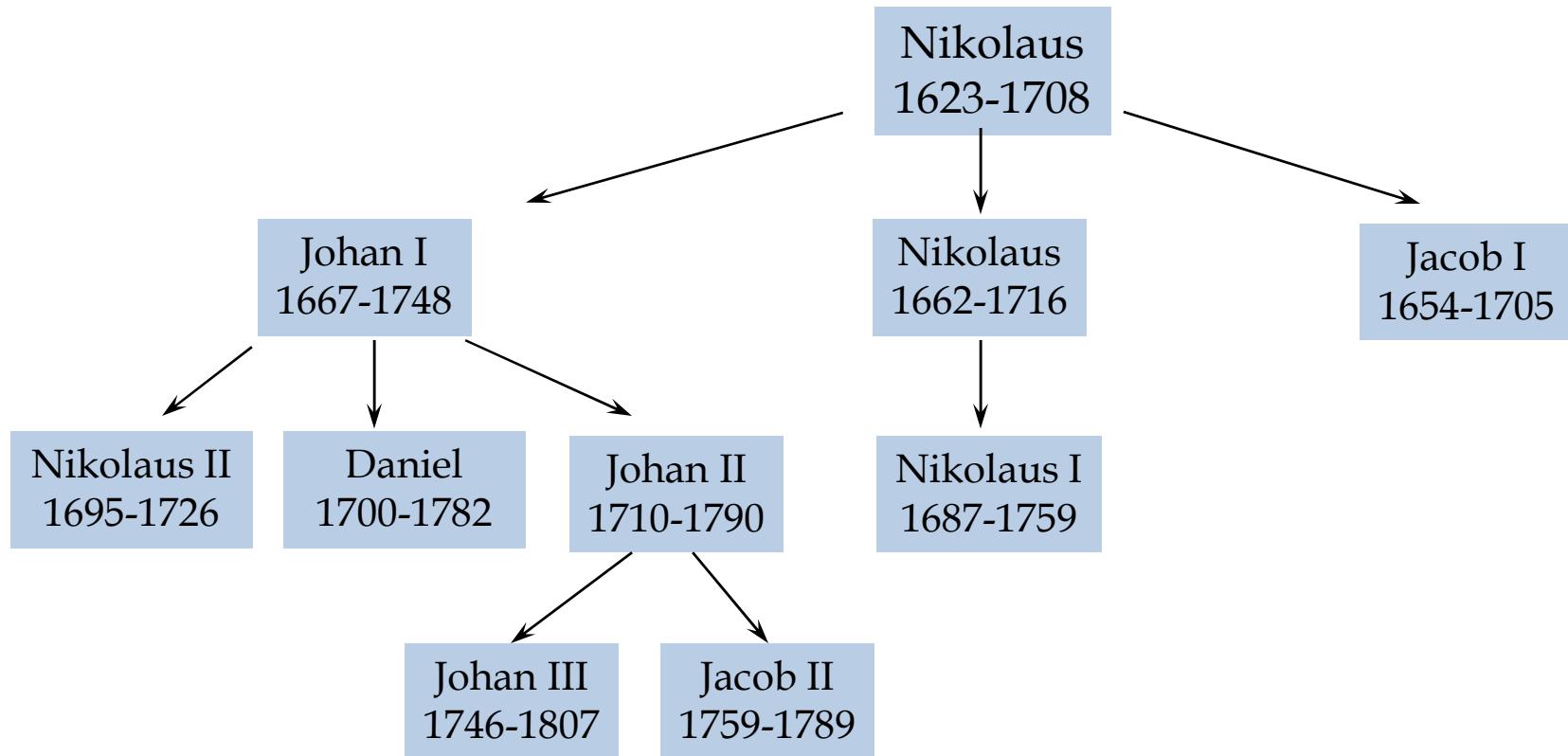
Các ứng dụng của cây: Biểu đồ lịch thi đấu

- Trong đời thường, cây thường được sử dụng để mô tả lịch thi đấu của các giải thể thao theo thể thức đấu loại trực tiếp



Các ứng dụng của cây: Cây gia phả

- Cây gia phả của các nhà toán học dòng họ Bernoulli



Các ứng dụng của cây: Cây thư mục

- Cây thư mục



Các ứng dụng của cây: Mục lục 1 quyển sách

Book

Chapter 1

Section 1.1

Section 1.2

Subsection 1.2.1

Subsection 1.2.2

Subsection 1.2.3

Section 1.3

Chapter 2

Section 2.1

Section 2.2

Section 2.3

Chapter 3

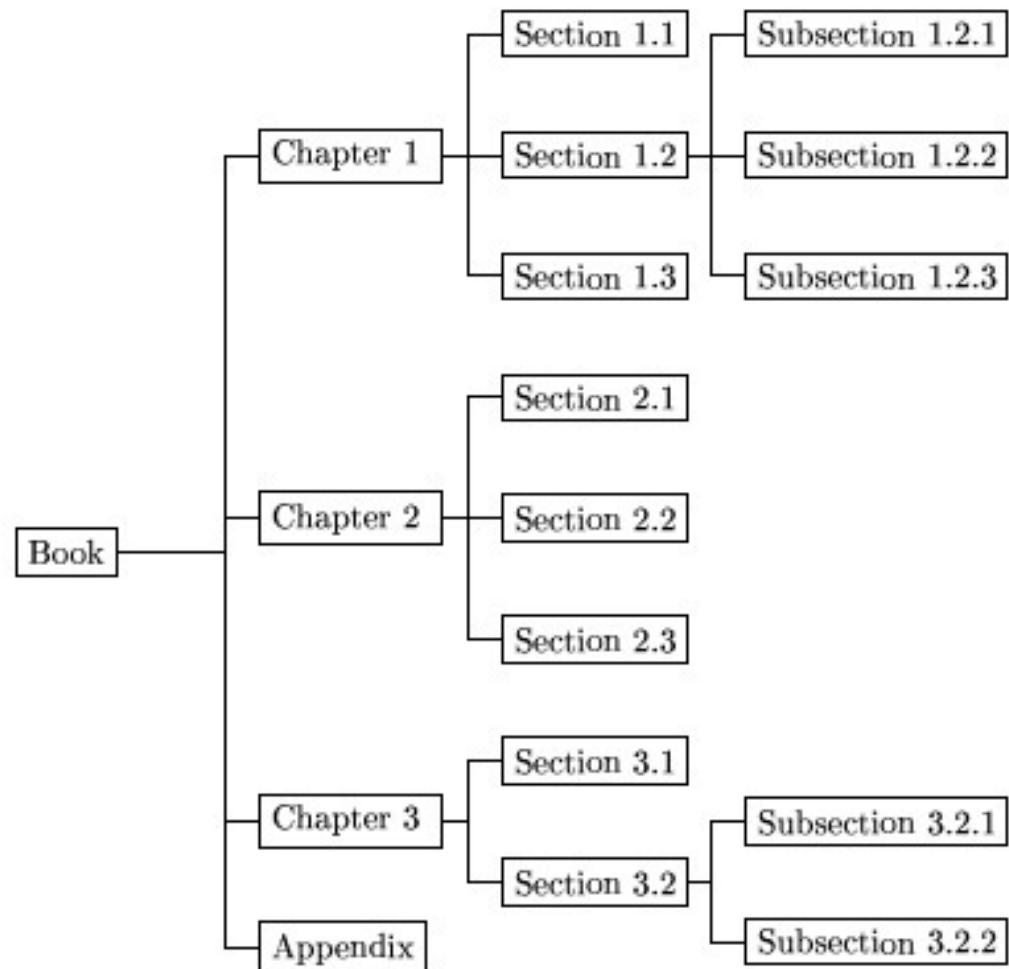
Section 3.1

Section 3.2

Subsection 3.2.1

Subsection 3.2.2

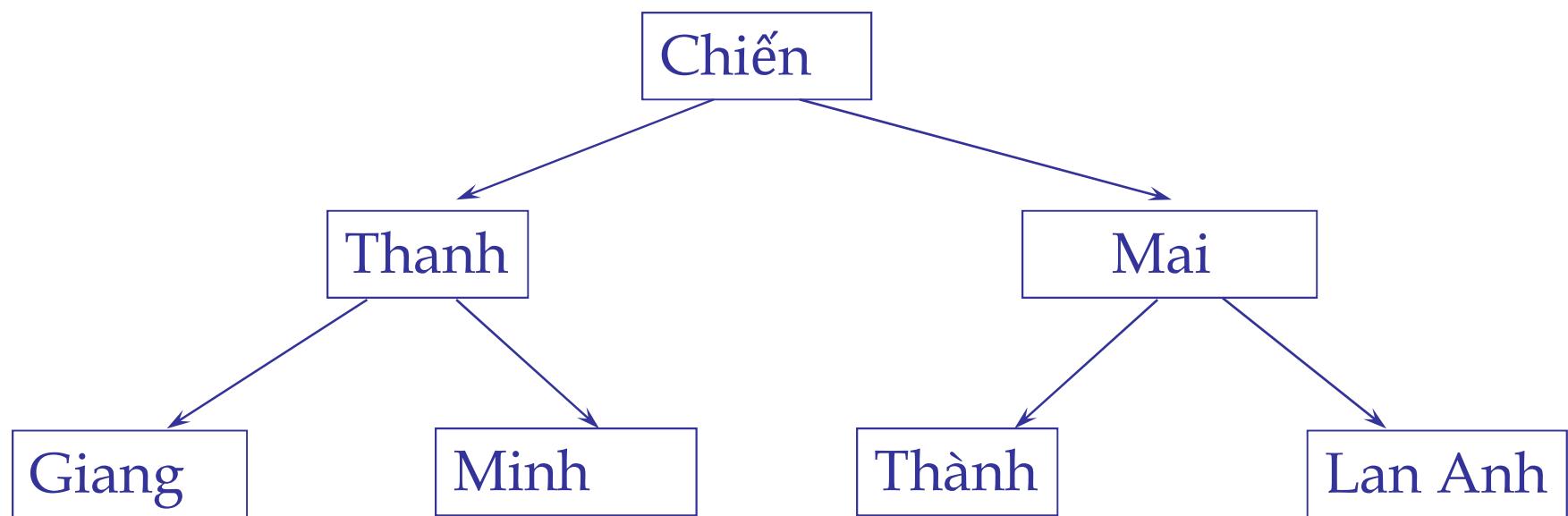
Appendix



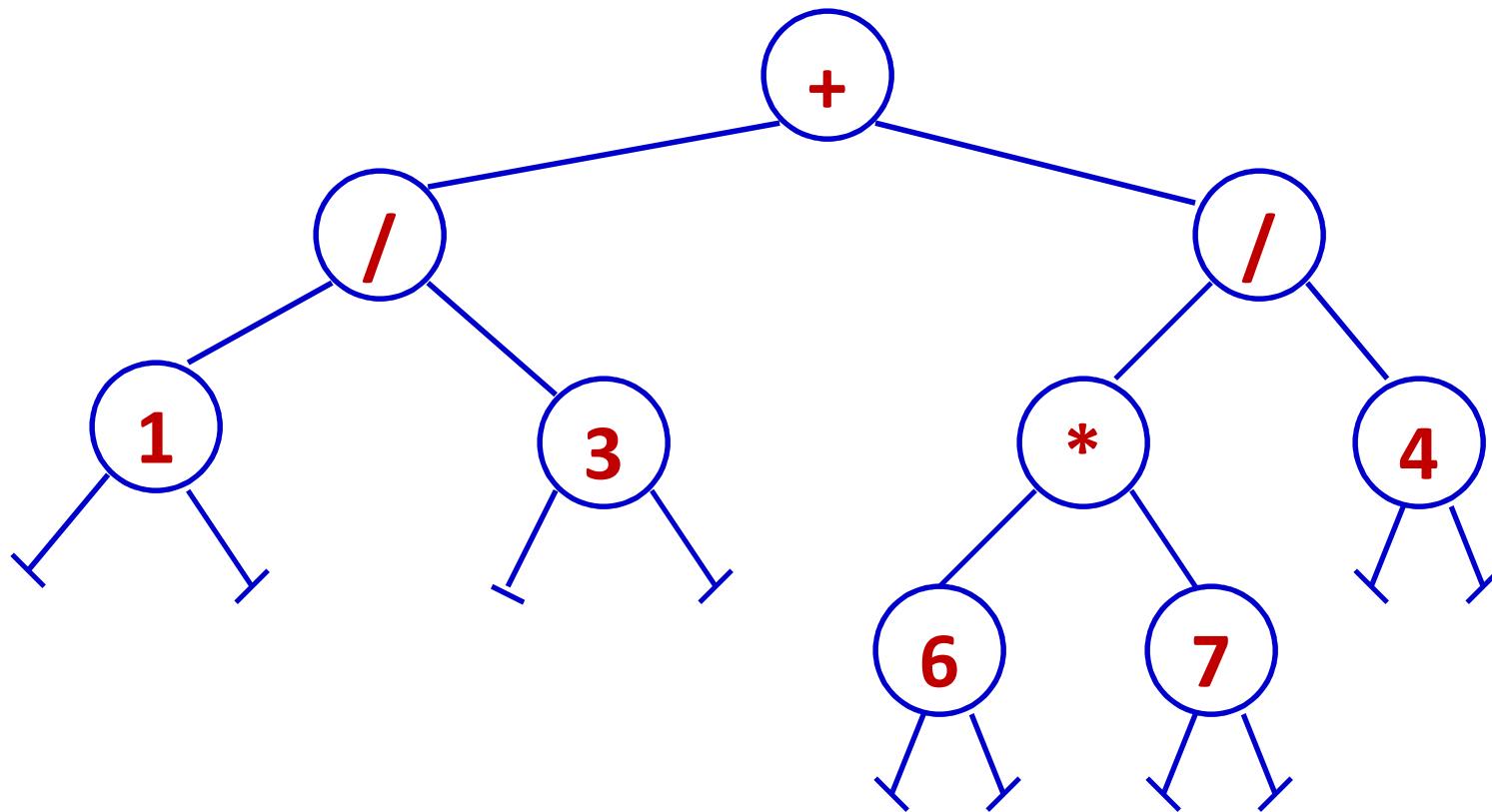
Các ứng dụng của cây: Cây gia phả ngược (Ancestor Tree)

- Mỗi người đều có bố mẹ:

Ví dụ: Chiến có cha mẹ là: Thanh và Mai

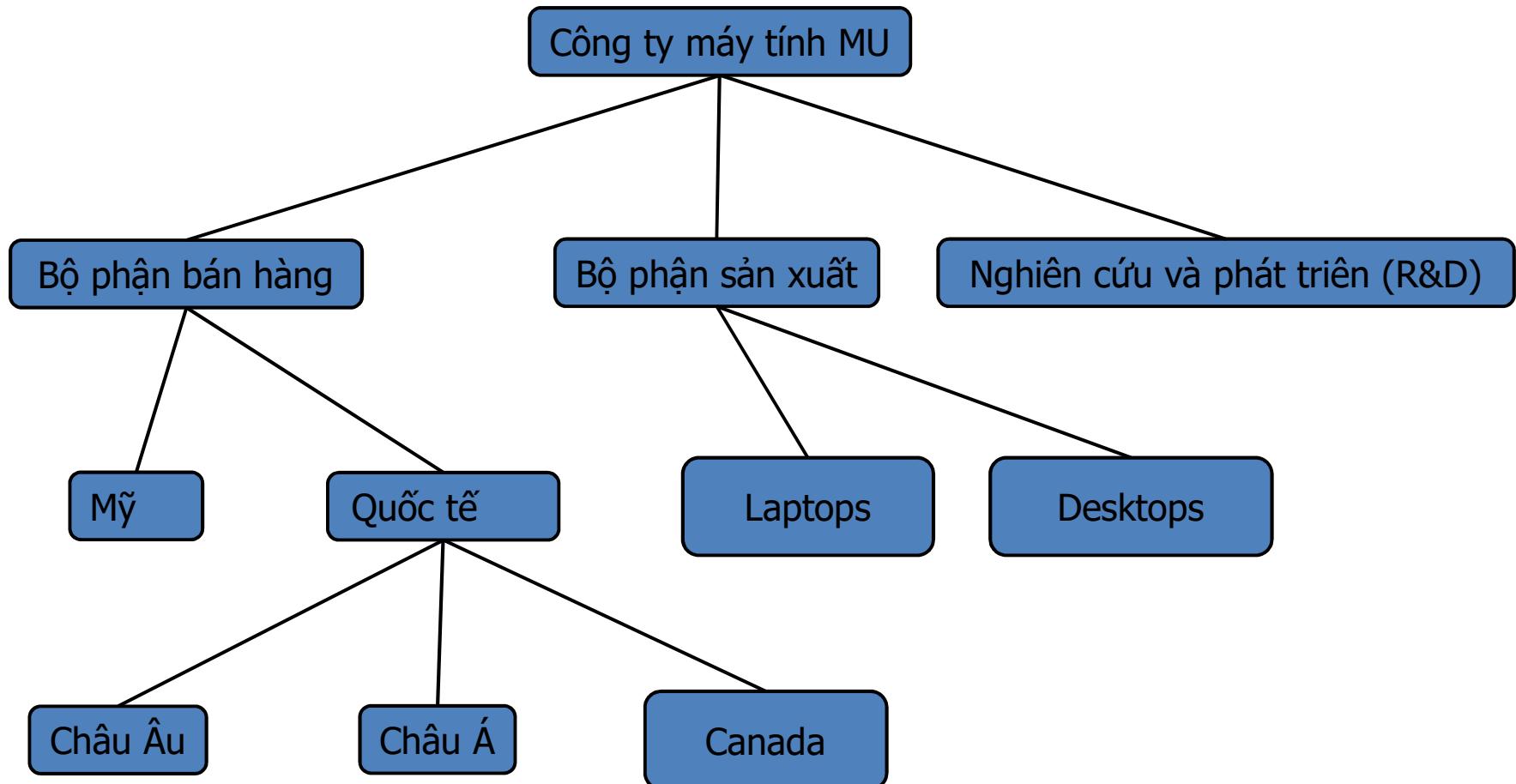


Các ứng dụng của cây: Cây biểu thức



$$1/3 + 6*7 / 4$$

Các ứng dụng của cây: Sơ đồ tổ chức của cơ quan



1. Định nghĩa và các khái niệm

1.1. Định nghĩa

1.2. Các thuật ngữ

1.3. Cây có thứ tự (Ordered tree)



1.2. Các thuật ngữ

- Nút (node)
- Gốc (Root)
- Lá (leaf)
- Con (child)
- Cha (Parent)
- Tổ tiên (Ancestors)
- Hậu duệ (Descendants)
- Anh em (Sibling)
- Nút trong (Internal node)
- Chiều cao (Height)
- Chiều sâu (Depth)

1.2. Các thuật ngữ

- **Gốc:** nút không có cha (ví dụ: nút A)
- **Anh em:** nút có cùng cha (ví dụ: B, C, D là anh em; I, J, K là anh em; E và F là anh em; G và H là anh em)
- **Nút trong:** nút có ít nhất 1 con (ví dụ: các nút màu xanh dương: A, B, C, F)
- **Nút ngoài (lá):** nút không có con (ví dụ: các nút xanh lá: E, I, J, K, G, H, D)
- **Tổ tiên** của 1 nút: là các nút cha / ông bà / cụ.. của nút đó.
- **Hậu duệ** của 1 nút: là các nút con/cháu/chắt... của nút đó
- **Cây con** của 1 nút: là cây có gốc là con của nút đó

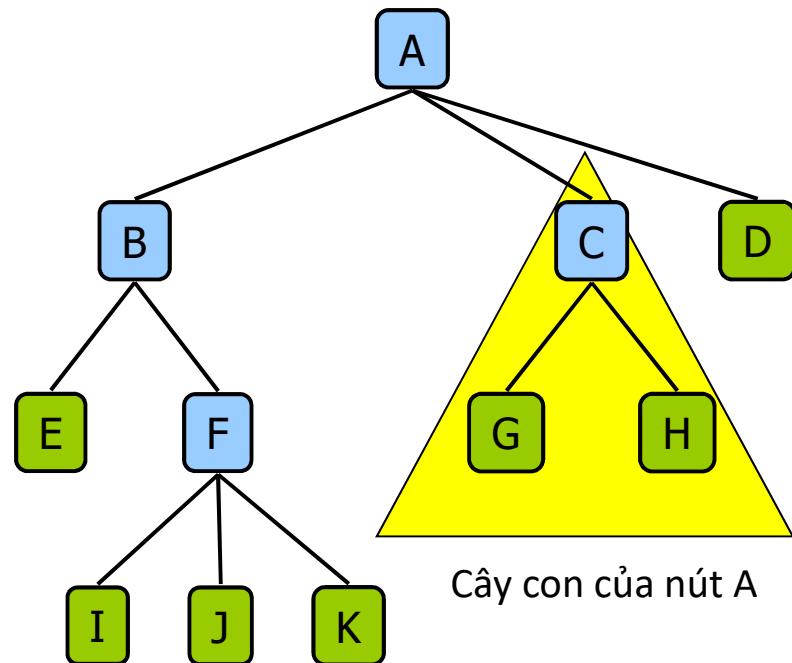
Ví dụ:

Con của B: E, F

Cha của E: B

Tổ tiên của F: B, A

Hậu duệ của B: E, F, I, J, K

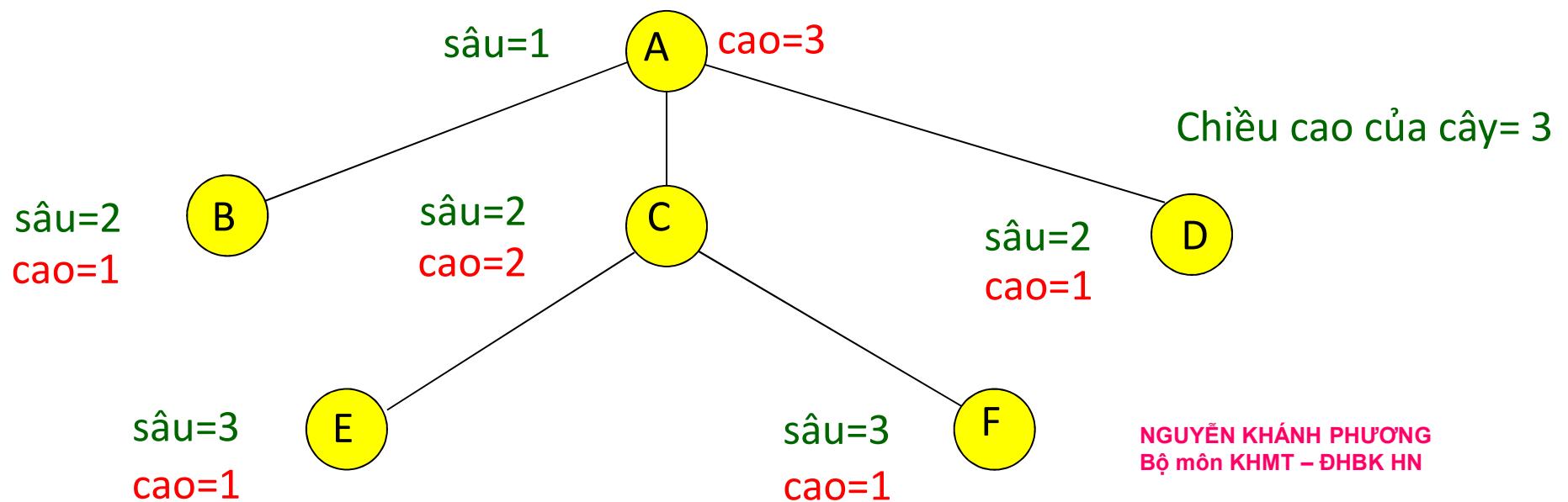


1.2. Các thuật ngữ

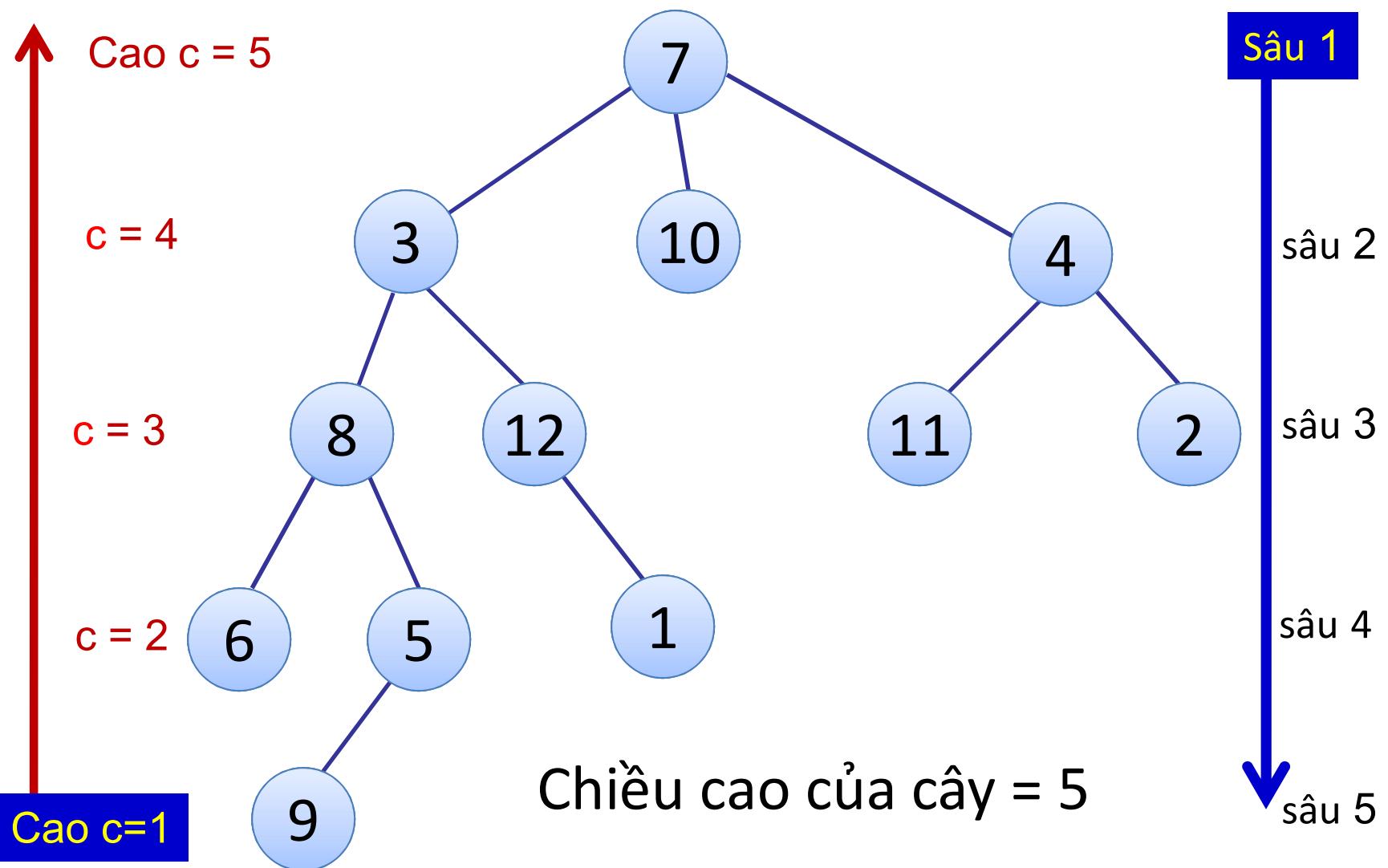
- Đường đi (Path): là 1 dãy các nút và các cạnh nối 1 nút đến 1 hậu duệ của nó
- Độ dài (Length) của đường đi = số lượng cạnh trên đường đi = số lượng nút trên đường đi - 1
(Ví dụ: Độ dài đường đi ($A \rightarrow C \rightarrow E$) = 2; độ dài đường đi ($C \rightarrow F$) = 1)

Như vậy, đường đi độ dài 0 là đường đi từ 1 nút đến chính nó

- Độ sâu/mức (Depth/level) của 1 nút N = 1 + độ dài đường đi từ gốc đến nút N
- Chiều cao (Height) của nút N = 1 + độ dài đường đi dài nhất từ N đến lá
- Chiều cao (sâu) của cây = chiều cao của gốc (= chiều cao lớn nhất của tất cả các nút trên cây)

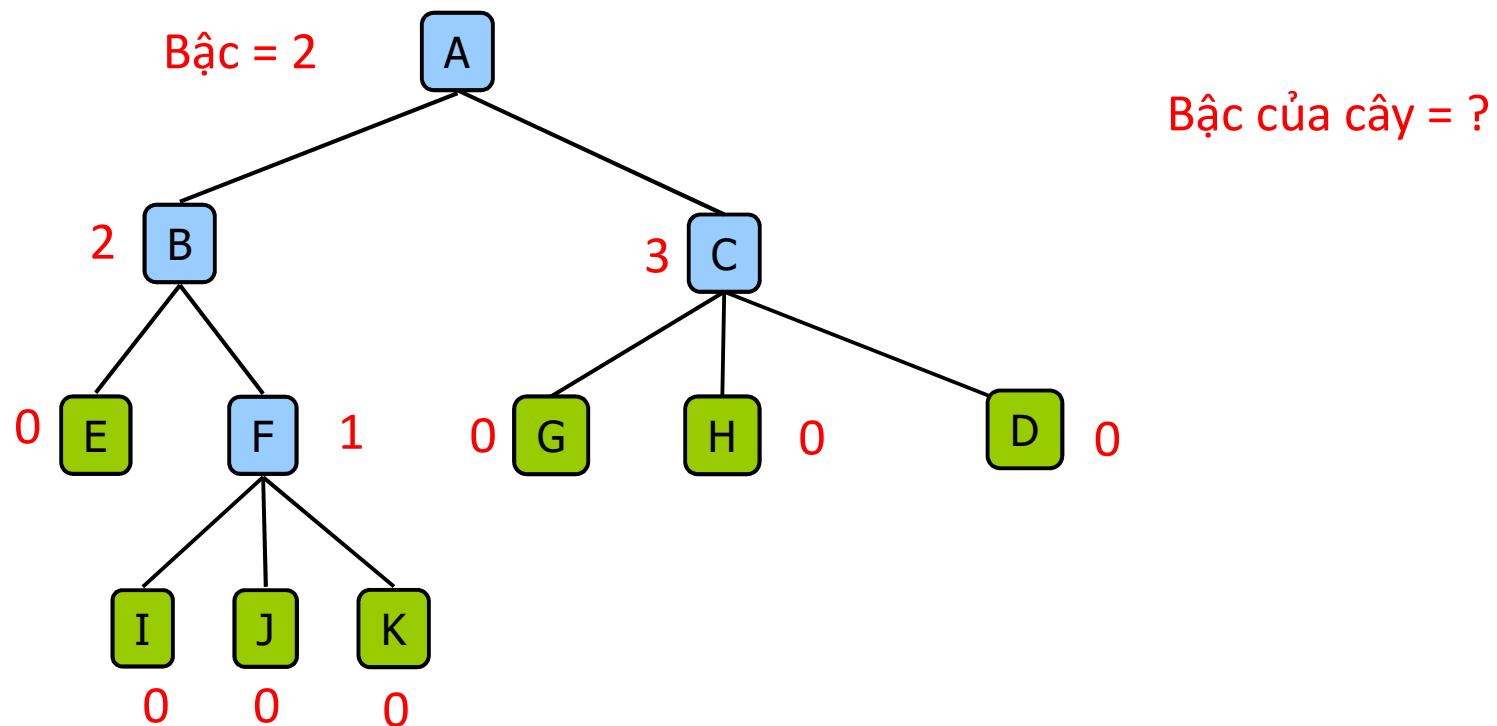


Chiều cao và độ sâu/mức

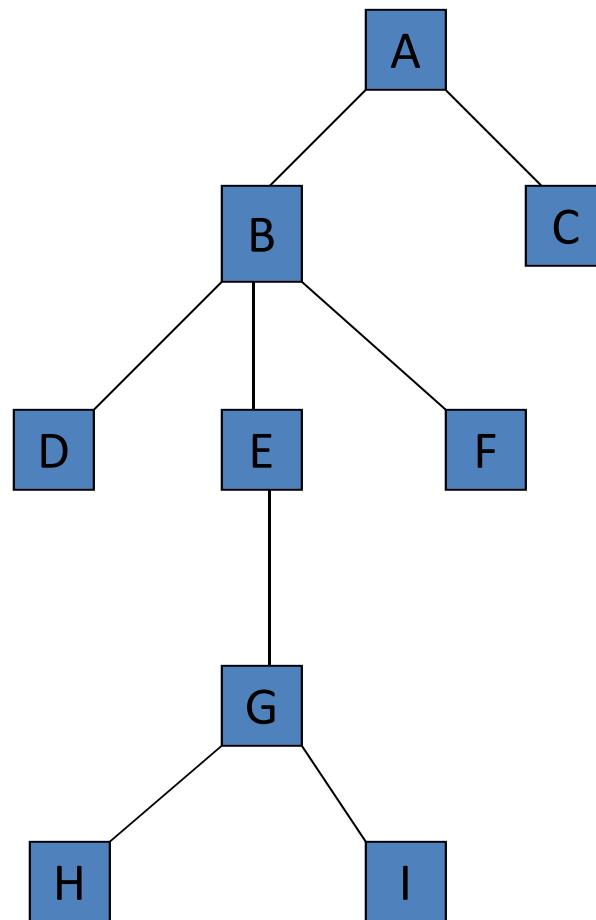


1.2. Các thuật ngữ

- **Bậc (Degree)** của 1 nút: số lượng nút con của nút đó (= số lượng cây con của nút đó)
- **Bậc (Degree)** của 1 cây: bậc lớn nhất của tất cả các nút trên cây

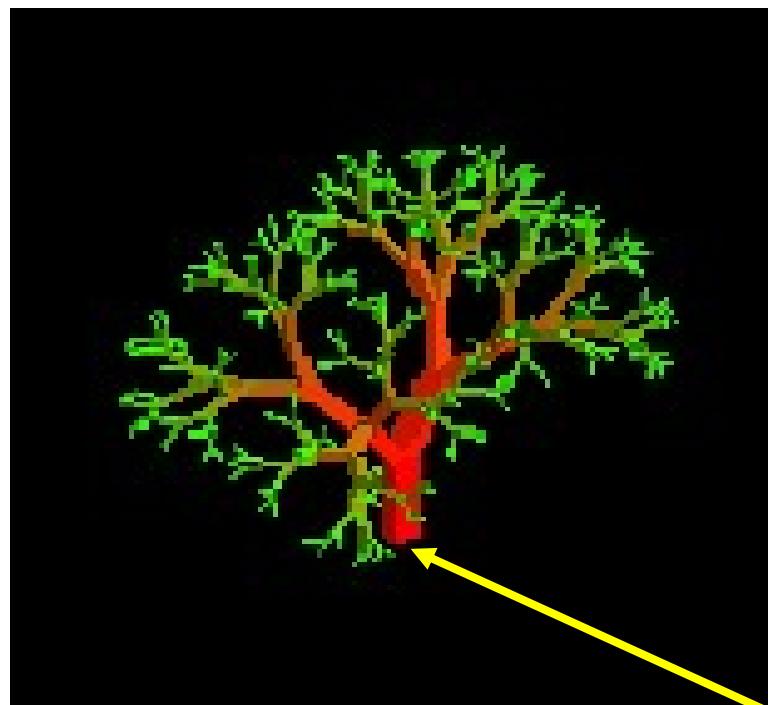


Bài tập: Các thuộc tính của cây



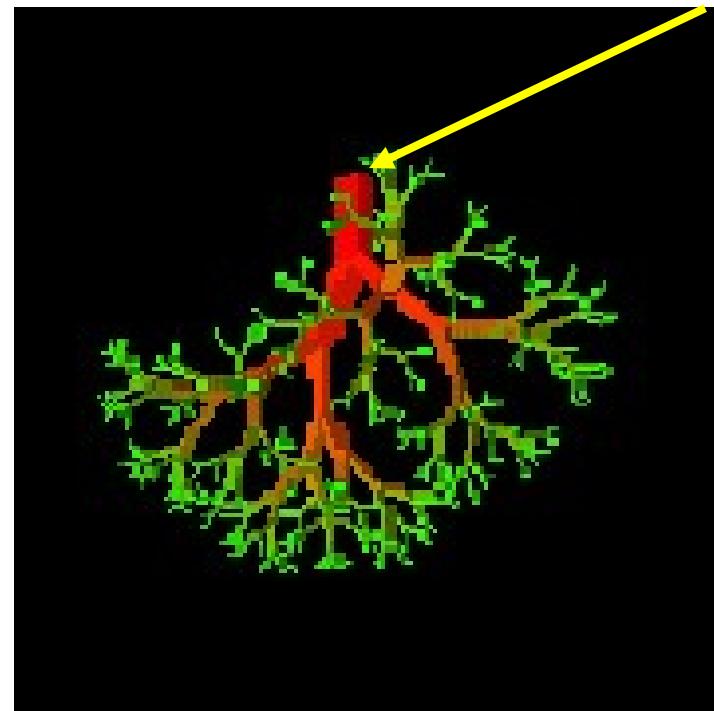
Thuộc tính	Giá trị
1) Số nút	
2) Nút gốc	A
3) Các nút lá	D, C, F, H, I
4) Các nút trong	B, E, G
5) Tổ tiên của H	A, B, E
6) Con cháu của B	D, E, F
7) Anh em của E	F
8) Cây con phải của A	C
9) Cây con trái của A	B
10) Chiều cao của cây	3
11) Độ sâu của cây	3

How we view a tree



root

Nature Lovers View



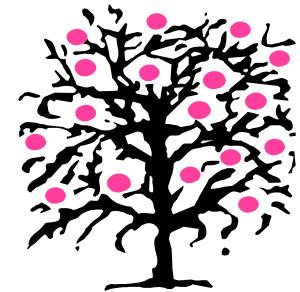
root

Computer Scientists View

1. Định nghĩa và các khái niệm

- 1.1. Định nghĩa
- 1.2. Các thuật ngữ

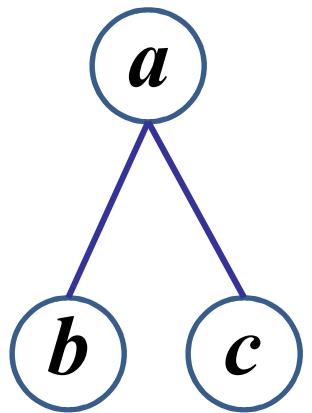
1.3. Cây có thứ tự (Ordered tree)



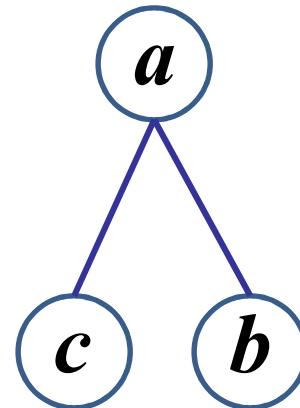
1.3. Cây có thứ tự (Ordered tree)

Cây có thứ tự là cây mà các nút được sắp xếp theo thứ tự

Ví dụ: Nếu T_1 và T_2 là cây có thứ tự thì $T_1 \neq T_2$; nếu không $T_1 = T_2$



T_1



T_2

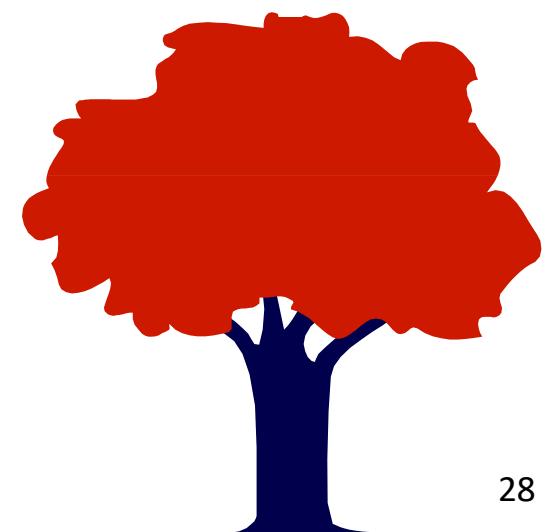
Nội dung

1. Định nghĩa và các khái niệm

2. Biểu diễn cây

3. Duyệt cây

4. Cây nhị phân

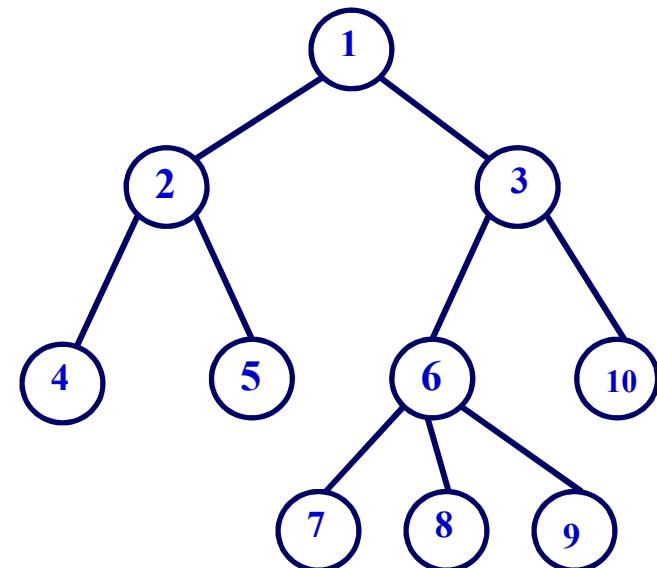


2. Biểu diễn cây

- Một số cách cài đặt cây trên máy tính
 - Mảng
 - Danh sách các con (Lists of Children)
 - Con trái nhất, anh em bên phải (The Leftmost-Child, Right-Sibling)

2. Biểu diễn cây: Mảng

- Giả sử cây T có n nút, được đánh số $1, 2, \dots, n$.
- Ta sử dụng 1 mảng A để biểu diễn cây T như sau:
 - $A[i] = j$ nếu nút j là cha của nút i
 - Gốc của cây T không có cha, do đó $A[1] = 0$

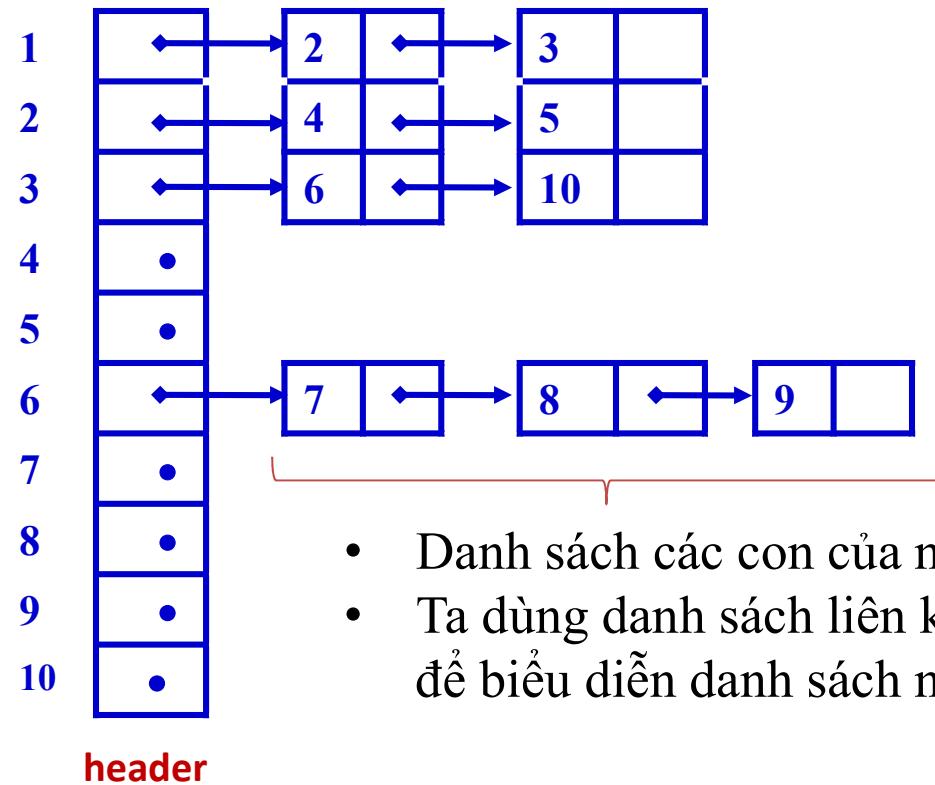
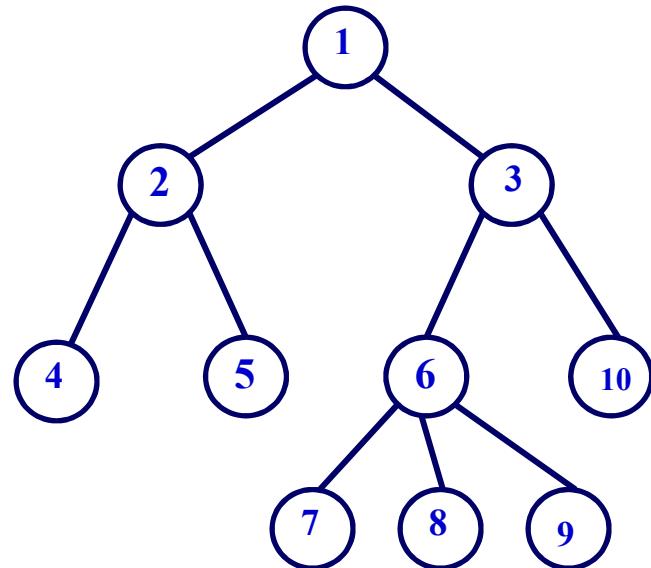


A	0	1	1	2	2	3	6	6	3
	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

2. Biểu diễn cây

- Một số cách cài đặt cây trên máy tính
 - Mảng
 - **Danh sách các con (Lists of Children)**
 - Con trái nhất, anh em bên phải (The Leftmost-Child, Right-Sibling)

2. Biểu diễn cây: Danh sách các con



Header: mảng các con trỏ

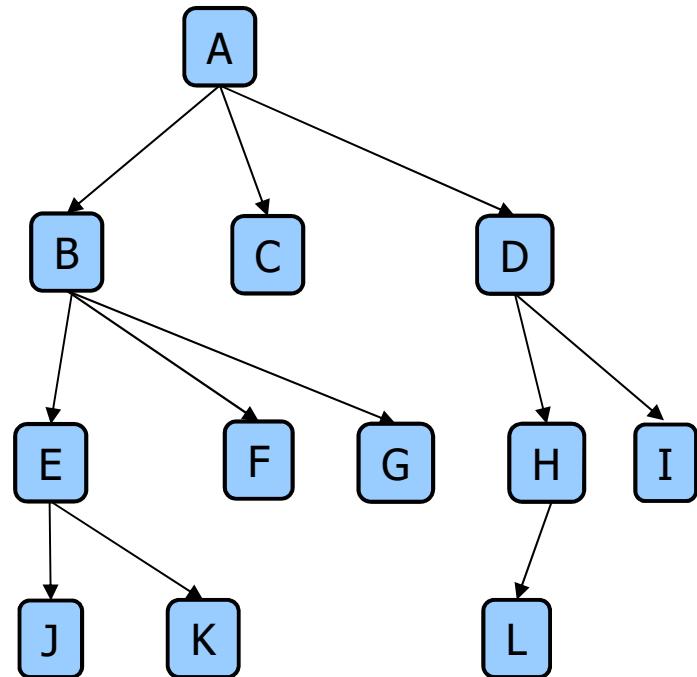
- Mỗi con trỏ **header** [*i*] trỏ đến đầu danh sách chứa các con của nút *i* trên cây
- Danh sách các con của nút *i*: được biểu diễn bởi 1 danh sách liên kết

2. Biểu diễn cây

- Một số cách cài đặt cây trên máy tính
 - Mảng
 - Danh sách các con (Lists of Children)
 - **Con trái nhất, em kê cận bên phải (The Leftmost-Child, Right-Sibling)**

2. Biểu diễn cây : Con trái nhất, em kế cận bên phải

- Mỗi nút trên cây, có thể:
 - Không có con nào, hoặc có duy nhất 1 con trái nhất
 - Không có anh em nào, hoặc có duy nhất 1 anh em bên phải



Ví dụ 1: nút B

- Con trái nhất của B: E
- Anh em bên phải của B: C

Ví dụ 2: nút C

- Không có con
- Con trái nhất của C:
- Anh em bên phải của C: D

Ví dụ 3: nút I

- Không có con
- Không có anh em bên phải

2. Biểu diễn cây : Con trái nhất, em kế cận bên phải

- Mỗi nút trên cây, có thể:
 - Không có con nào, hoặc có duy nhất 1 con trái nhất
 - Không có anh em nào, hoặc có duy nhất 1 anh em bên phải

Do đó, để biểu diễn cây, ta sẽ lưu trữ các thông tin về con trái nhất và anh em bên phải của mỗi nút:

```
typedef struct
{
    int data; // dữ liệu có trên mỗi nút
    struct treeNode * con_trai_nhat;
    struct treeNode * em_ke_can_phai;
}treeNode;
treeNode * Root;
```

Data	
Con trái nhất	Em kế cận phải

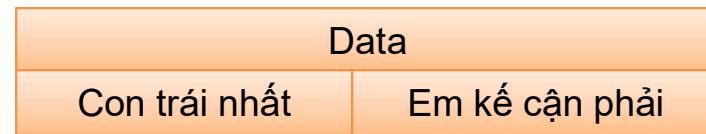
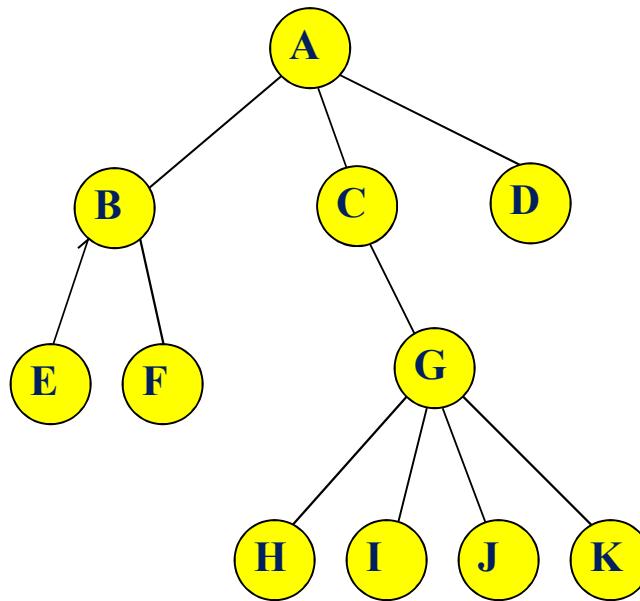
2. Biểu diễn cây : Con trái nhất, em kế cận bên phải

- Mỗi nút trên cây, có thể:
 - Không có con nào, hoặc có duy nhất 1 con trái nhất
 - Không có anh em nào, hoặc có duy nhất 1 anh em bên phải

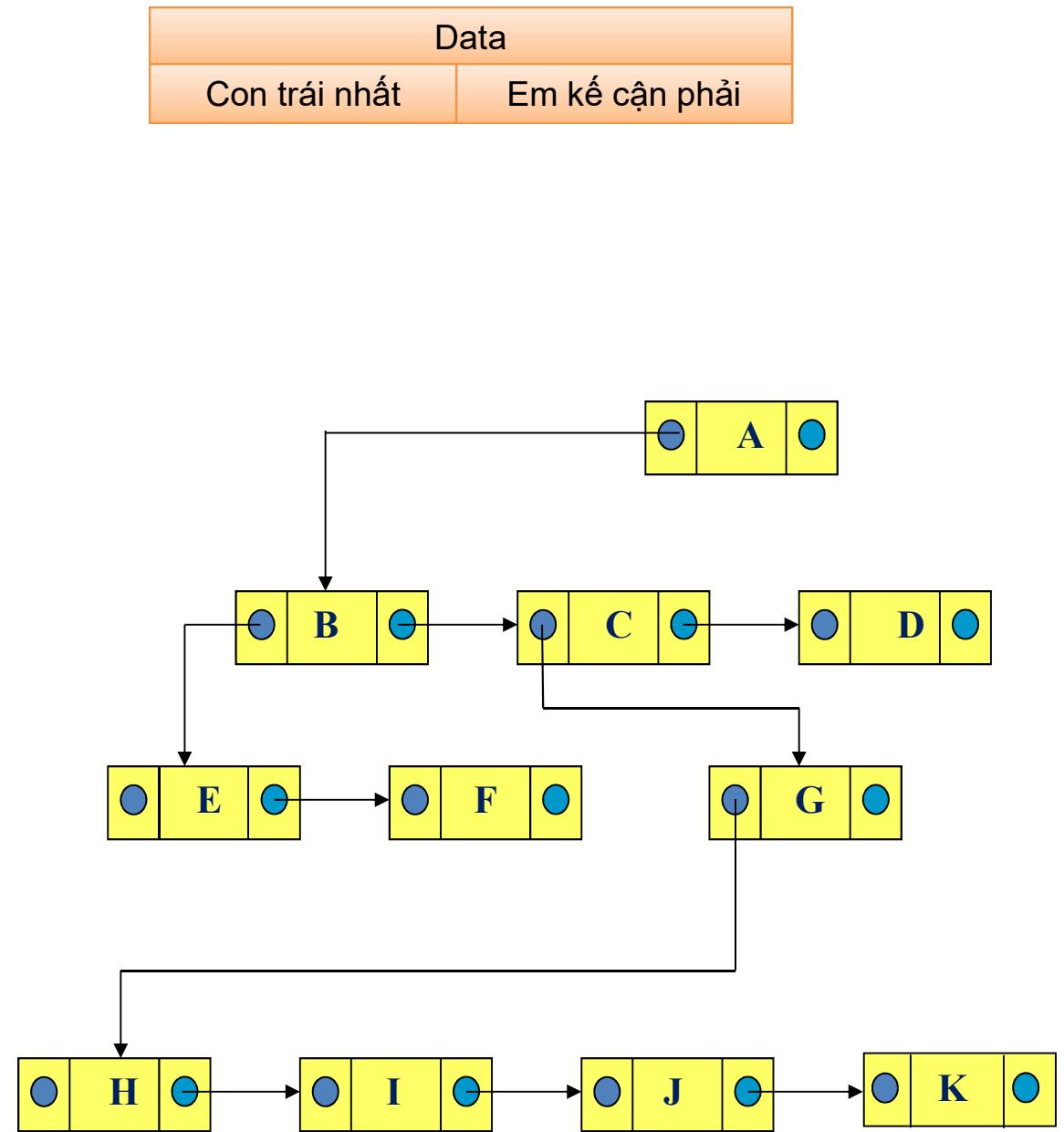
Do đó, để biểu diễn cây, ta sẽ lưu trữ các thông tin về con trái nhất và anh em bên phải của mỗi nút:

```
typedef struct
{
    int data; // dữ liệu có trên mỗi nút
    struct treeNode * con_trai_nhat;
    struct treeNode * em_ke_can_phai;
}treeNode;
treeNode * Root;
```

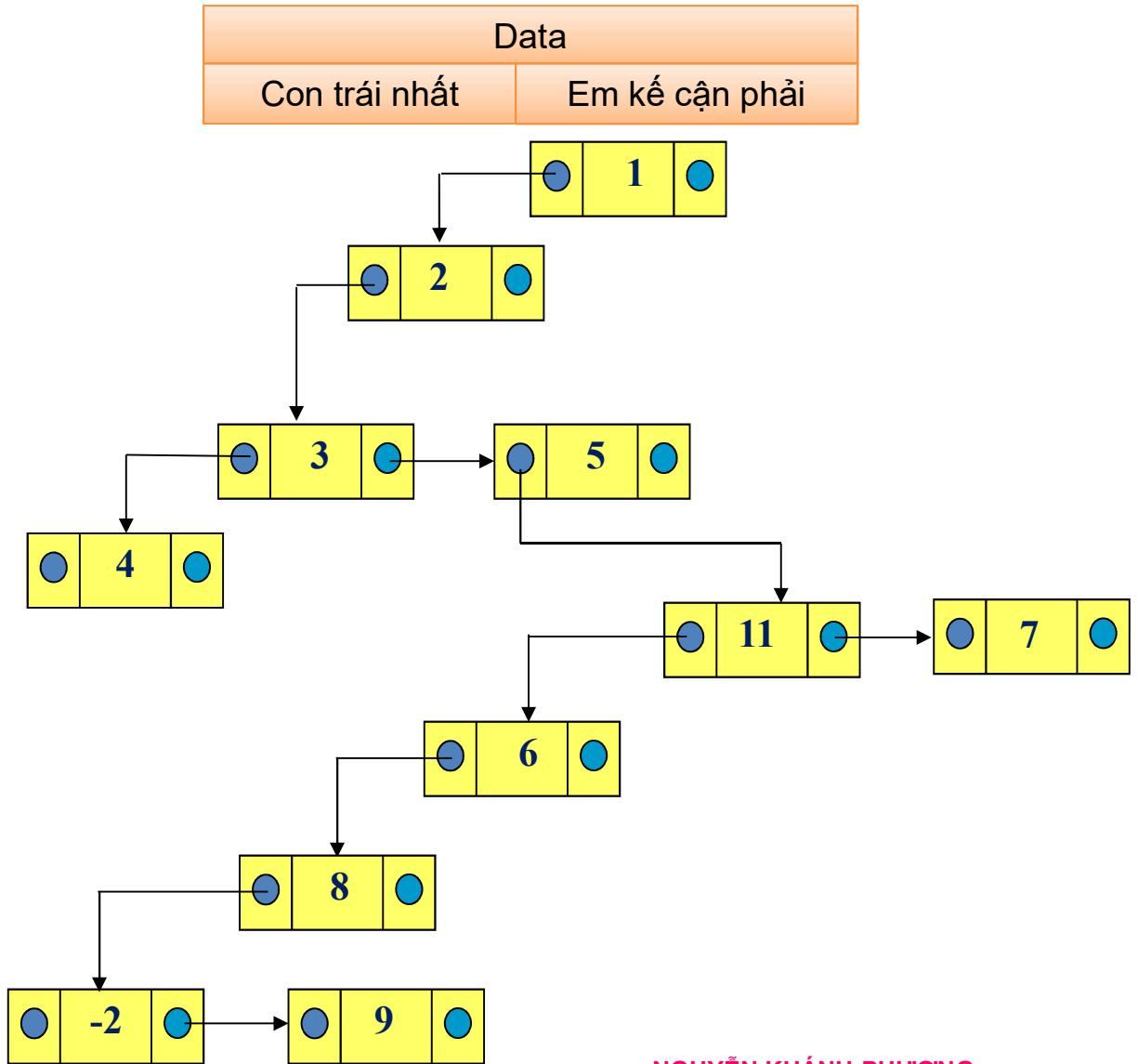
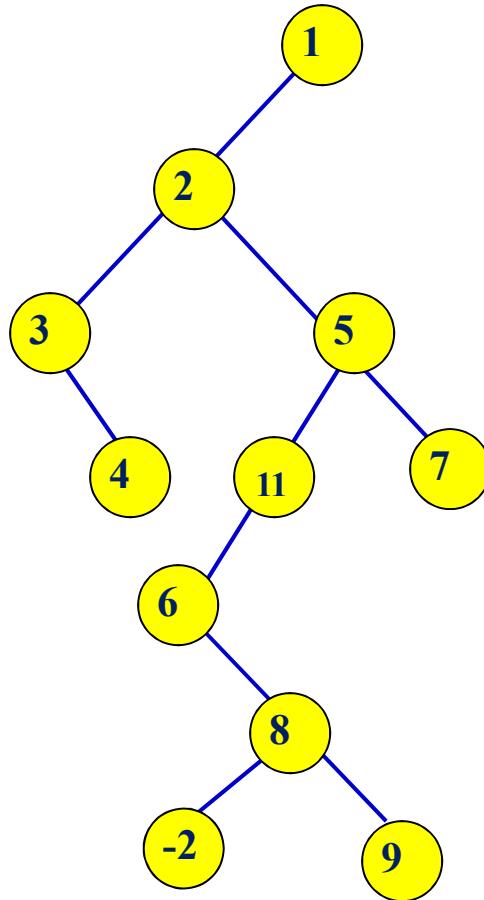
2. Biểu diễn cây : Con trái nhất, em kế cận bên phải



```
typedef struct
{
    int data; // dữ liệu có trên mỗi nút
    struct treeNode * con_trái_nhất;
    struct treeNode * em_kế_cận_phải;
}treeNode;
treeNode * Root;
```



2. Biểu diễn cây : Con trái nhất, em kế cận phải

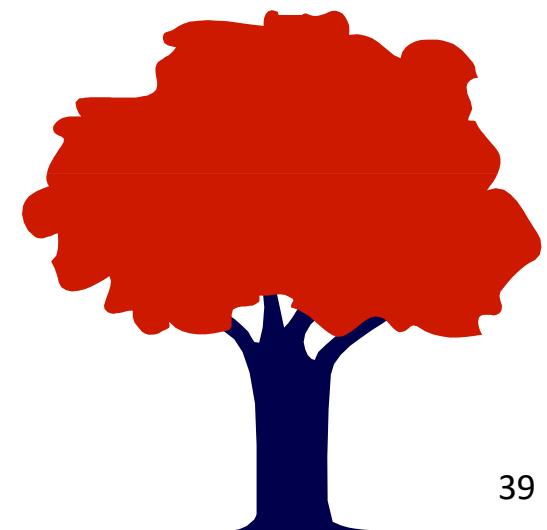


```

typedef struct
{
    int data; // dữ liệu có trên mỗi nút
    struct treeNode * con_trái_nhất;
    struct treeNode * em_kế_cận_phải;
}treeNode;
treeNode * Root;
  
```

Nội dung

1. Định nghĩa và các khái niệm
2. Biểu diễn cây
- 3. Duyệt cây**
4. Cây nhị phân



3. Duyệt cây (Tree Traversal)

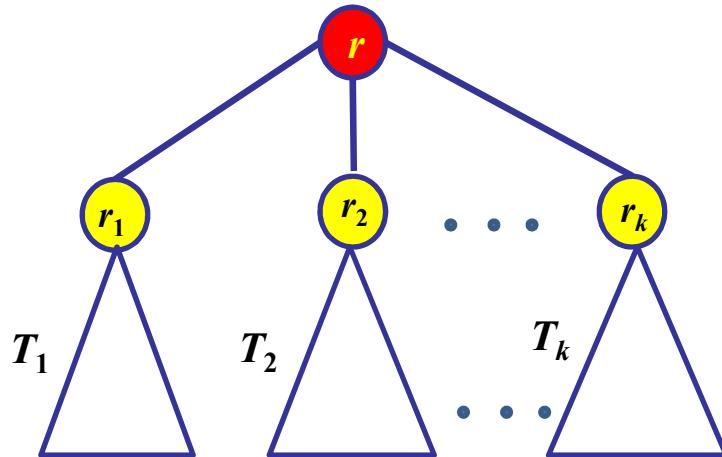
Có 3 phương pháp chính:

- Thú tự trước (Preorder):
 - Thăm nút gốc
 - Duyệt theo thứ tự trước các nút con (cây con)
- Thú tự sau (Postorder):
 - Duyệt các con (cây con) theo thứ tự sau
 - Thăm nút gốc
- Thú tự sau (Inorder):
 - Duyệt con trái nhất (cây con trái nhất) theo thứ tự sau
 - Thăm nút gốc
 - Duyệt các con còn lại (không phải con trái nhất) theo thứ tự sau

Duyệt thứ tự trước (Preorder Traversal)

Khi duyệt cây theo thứ tự trước, mỗi nút đều được thăm trước con cháu của nó.

Ví dụ: Duyệt cây T theo thứ tự trước:



```
procedure preorder( $T$ : ordered rooted tree)
     $r :=$  root of  $T$ 
    visit  $r$  Step 1
    for each child  $c$  of  $r$  from left to right
        begin
             $T(c) :=$  subtree with  $c$  as its root
            preorder( $T(c)$ ) Step 2, 3...
        end
```

- Bước 1: thăm nút gốc r ,
- Bước 2: thăm cây con T_1 theo thứ tự trước,
- Bước 3: thăm cây con T_2 theo thứ tự trước
-
- Bước $k+1$: thăm cây con T_k theo thứ tự trước

Duyệt thứ tự trước (Preorder Traversal)

- Ứng dụng: in ra cấu trúc của một quyển sách

```
procedure preorder( $T$ : ordered rooted tree)
```

```
     $r :=$  root of  $T$ 
```

```
    visit  $r$ 
```

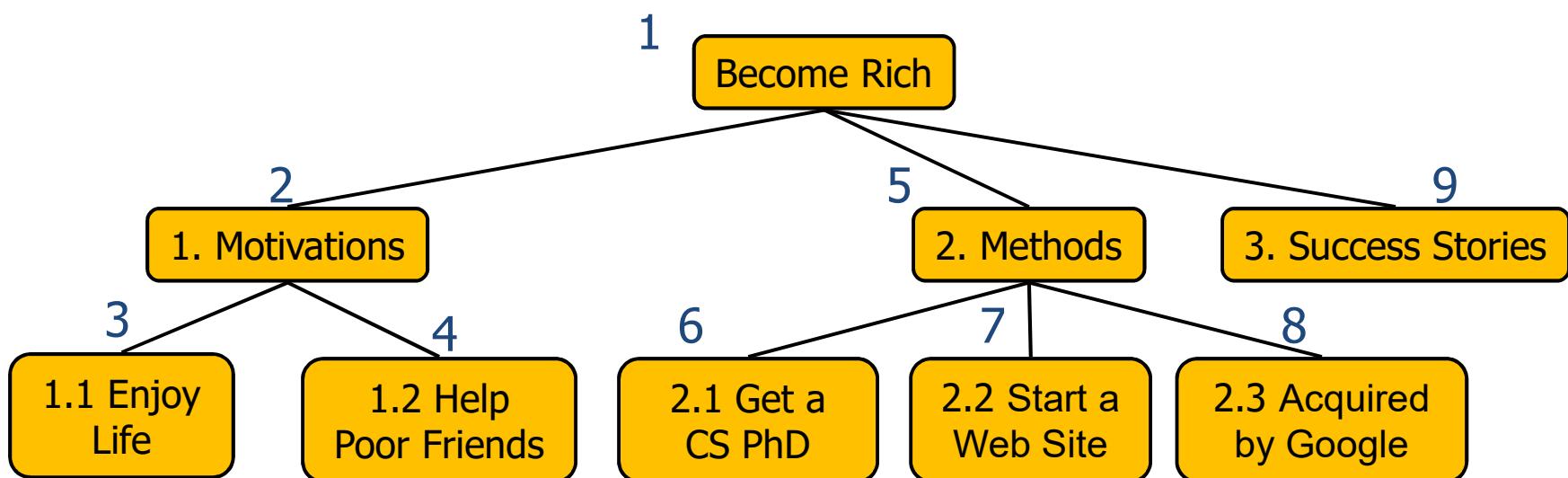
```
    for each child  $c$  of  $r$  from left to right
```

```
        begin
```

```
             $T(c) :=$  subtree with  $c$  as its root
```

```
            preorder( $T(c)$ )
```

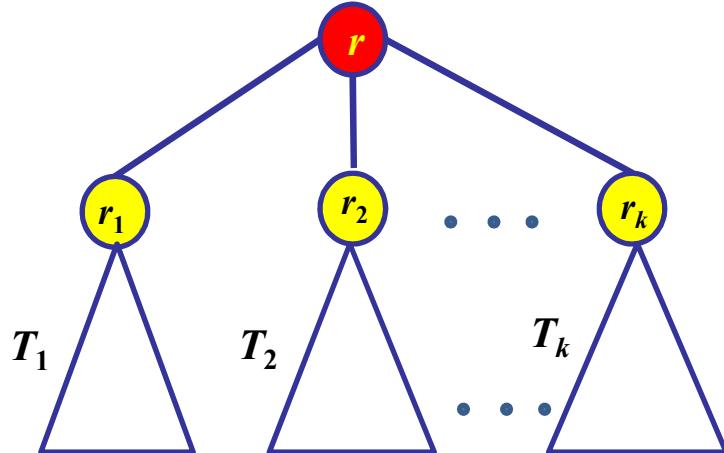
```
        end
```



Duyệt thứ tự sau (Postorder Traversal)

Khi duyệt cây theo thứ tự sau, mỗi nút đều được thăm trước con cháu của nó.

Ví dụ: Duyệt cây T theo thứ tự sau:



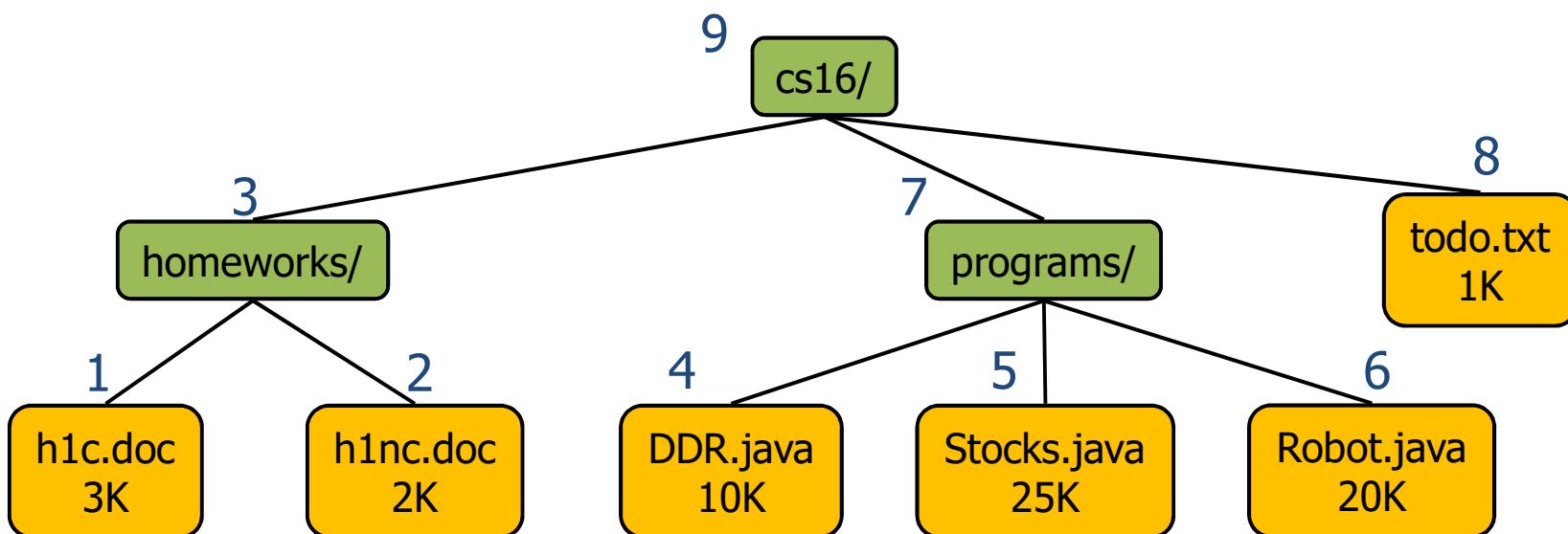
```
procedure postorder( $T$ : ordered rooted tree)
   $r :=$  root of  $T$ 
  for each child  $c$  of  $r$  from left to right
    begin
       $T(c) :=$  subtree with  $c$  as its root
      postorder( $T(c)$ )
    end
    visit  $r$ 
```

- Bước 1: thăm cây con T_1 theo thứ tự sau,
- Bước 2: thăm cây con T_2 theo thứ tự sau,
-
- Bước k : thăm cây con T_k theo thứ tự sau,
- Bước $k+1$: thăm nút gốc r

Duyệt thứ tự sau (Postorder Traversal)

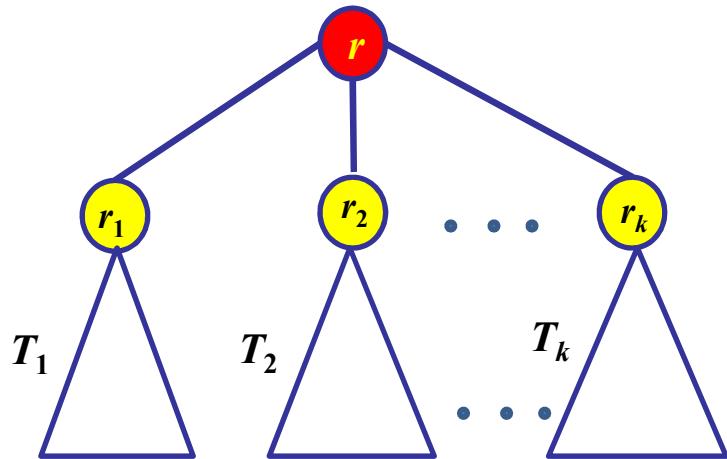
- Ứng dụng: tính dung lượng bộ nhớ bị chiếm giữ bởi các file trong một thư mục và các thư mục con của nó

```
procedure postorder( $T$ : ordered rooted tree)
   $r :=$  root of  $T$ 
  for each child  $c$  of  $r$  from left to right
    begin
       $T(c) :=$  subtree with  $c$  as its root
      postorder( $T(c)$ )
    end
    visit  $r$ 
```



Duyệt thứ tự giữa (Inorder Traversal)

- Duyệt cây T theo thứ tự giữa:

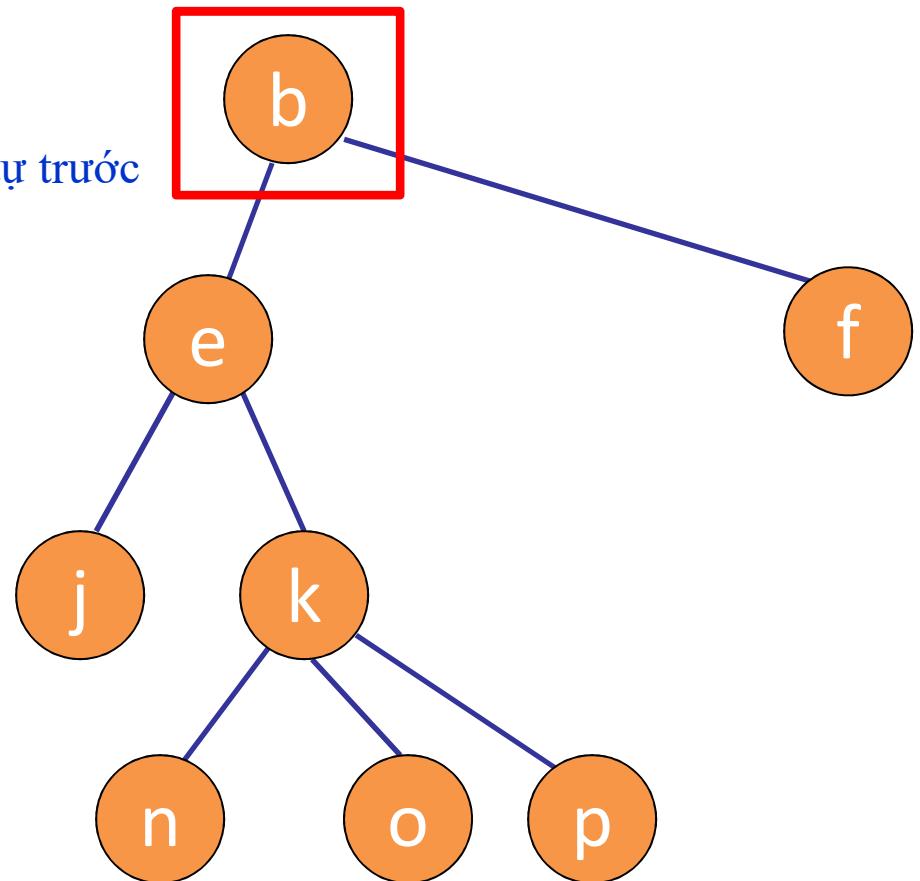


- Bước 1: thăm T_1 theo thứ tự giữa,
- Bước 2: thăm nút gốc r ,
- Bước 3: thăm T_2 theo thứ tự giữa,
-
- Bước $k+1$: thăm T_k theo thứ tự giữa

```
procedure inorder( $T$ : ordered rooted tree)
   $r :=$  root of  $T$ 
  if  $r$  is a leaf then visit  $r$ 
  else
    begin
       $l :=$  first child of  $r$  from left to right
       $T(l) :=$  subtree with  $l$  as its root
      inorder( $T(l)$ ) Step 1
      visit  $r$  Step 2
      for each child  $c$  of  $r$  except for  $l$  left to right
         $T(c) :=$  subtree with  $c$  as its root
        inorder( $T(c)$ ) Step 3, 4...
    end
```

Thứ tự trước (PreOrder)

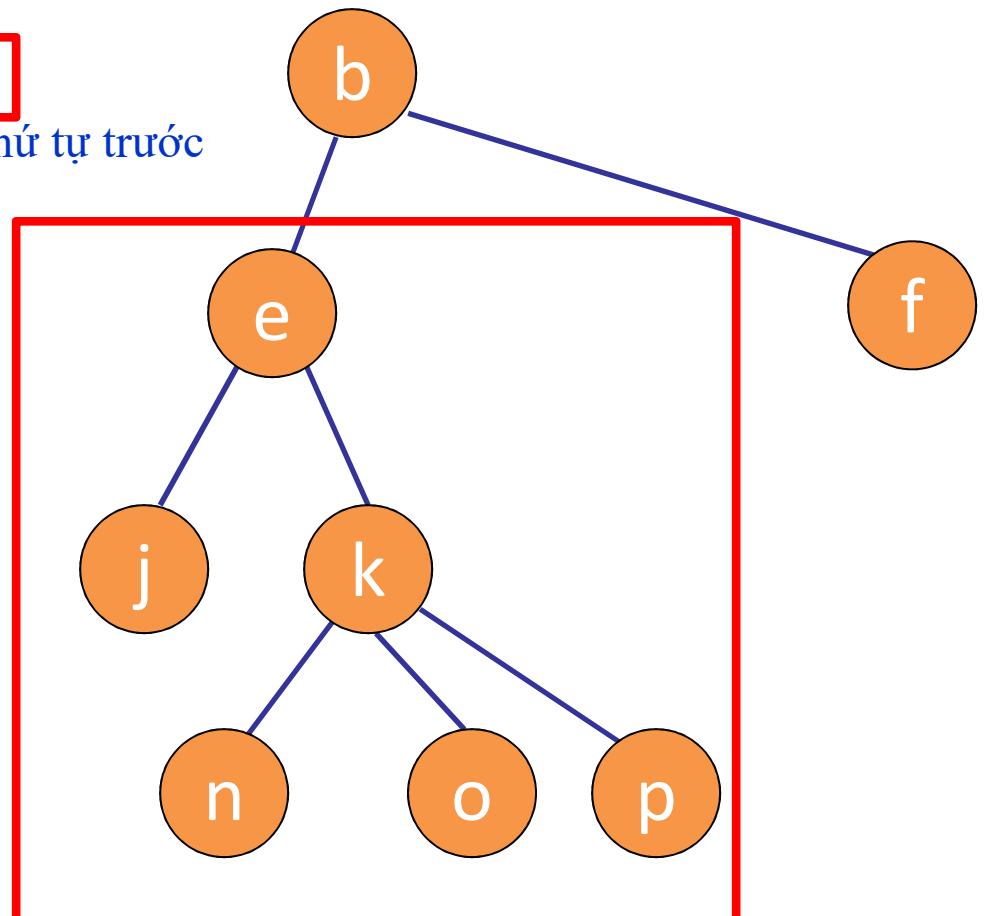
1. Thăm gốc
2. Thăm cây con trái nhất theo thứ tự trước
3. Lần lượt thăm các cây con còn lại theo thứ tự trước



b

PreOrder

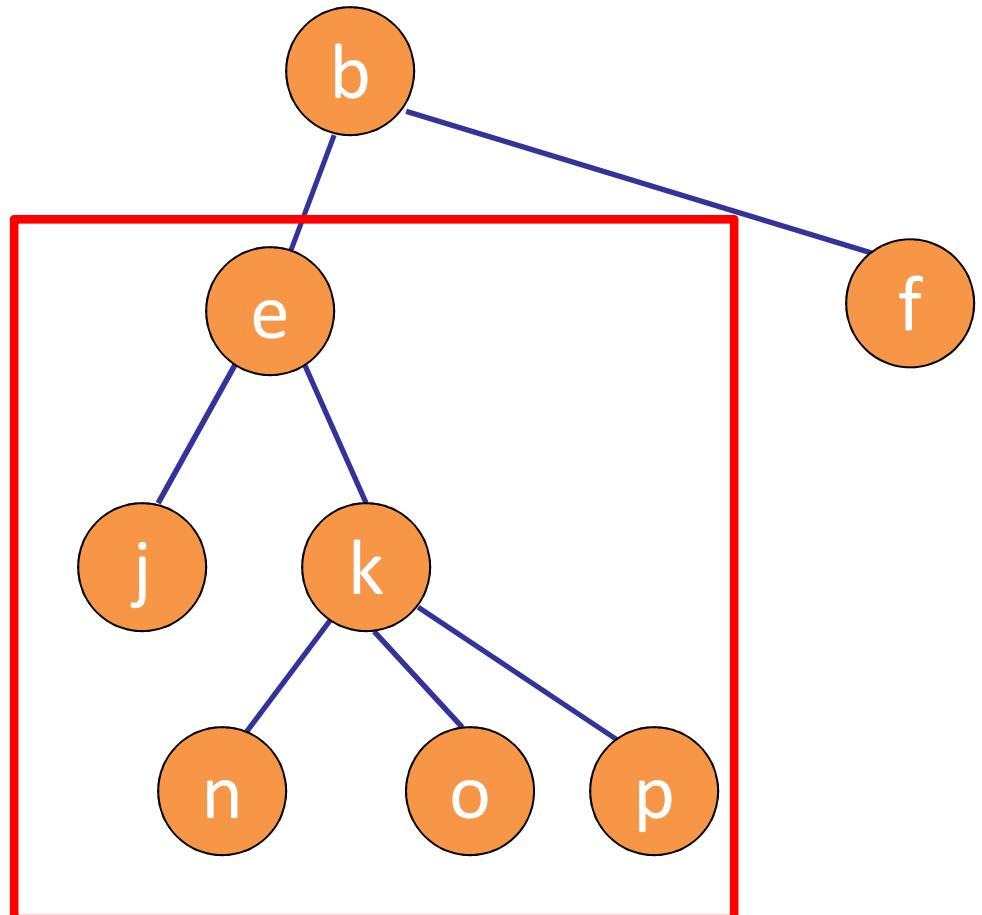
1. Thăm gốc
2. Thăm cây con trái nhất theo thứ tự trước
3. Lần lượt thăm các cây con còn lại theo thứ tự trước



b

PreOrder

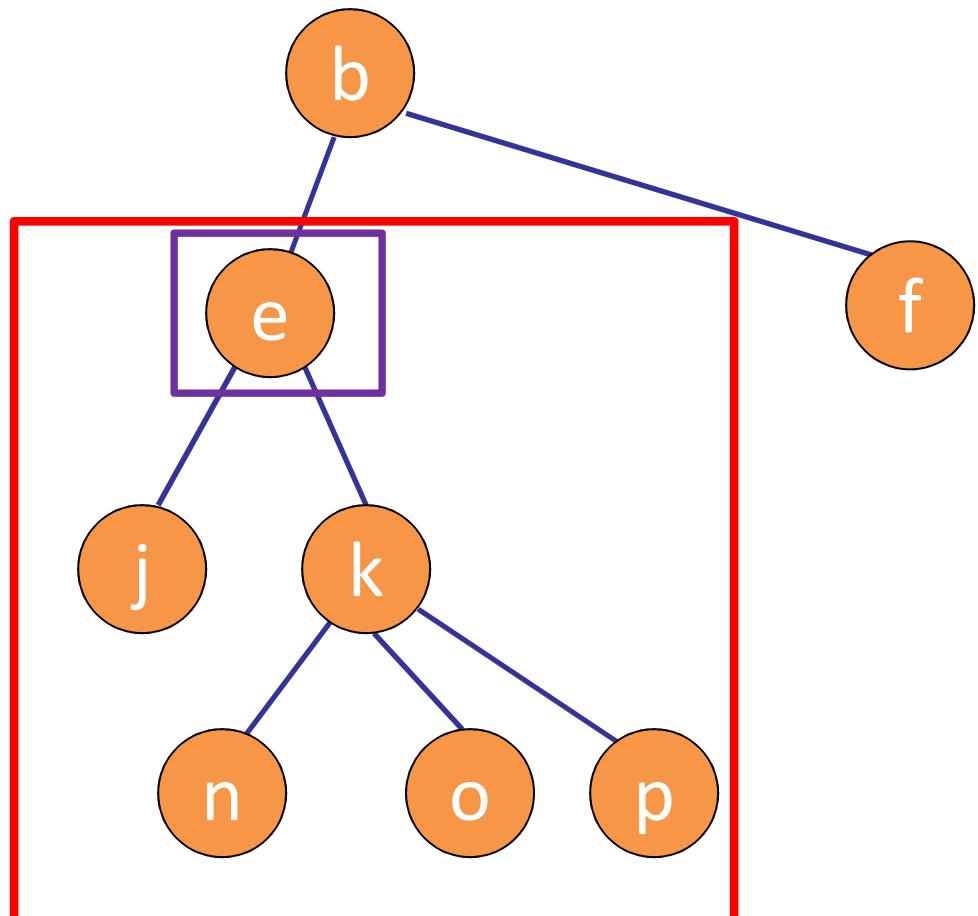
1. Thăm gốc
2. Thăm cây con trái nhất theo thứ tự trước
 - 2.1. Thăm gốc
 - 2.2. Thăm cây con trái nhất theo preorder
 - 2.3. Lần lượt thăm các cây con còn lại theo preorder
3. Lần lượt thăm các cây con còn lại theo preorder



b

PreOrder

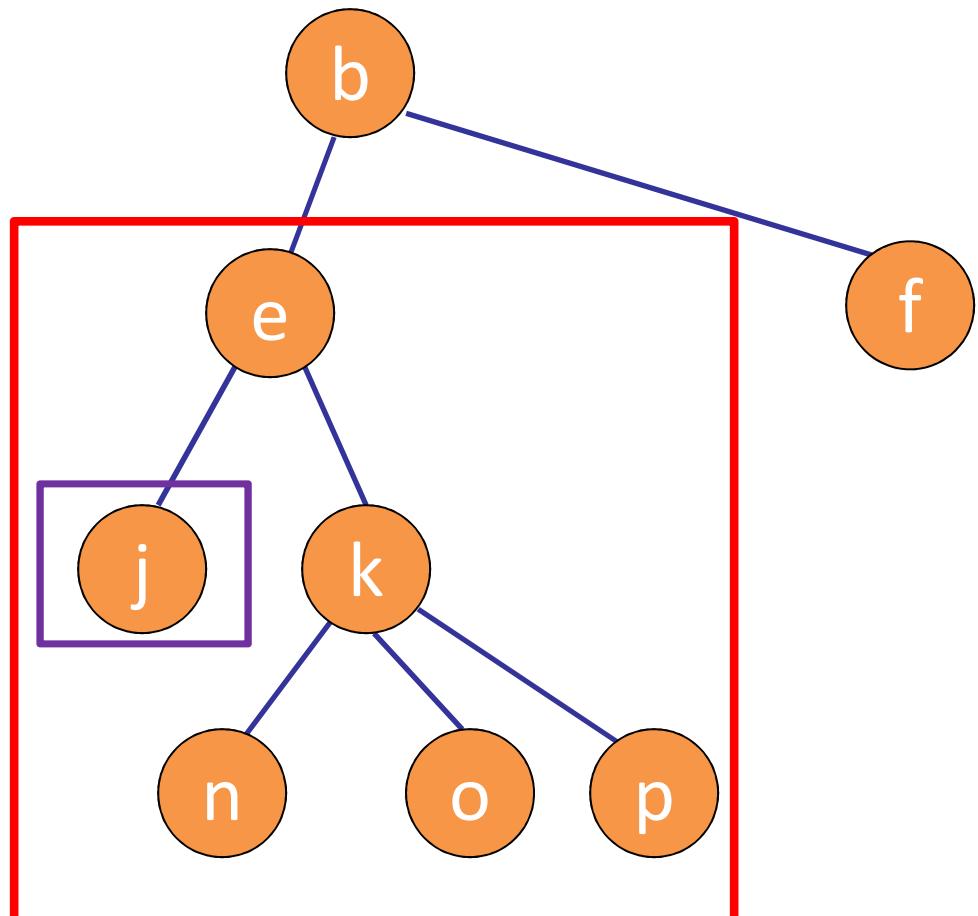
1. Thăm gốc
2. Thăm cây con trái nhất theo preorder
 - 2.1. Thăm gốc
 - 2.2. Thăm cây con trái nhất theo preorder
 - 2.3. Lần lượt thăm các cây con còn lại theo preorder
3. Lần lượt thăm các cây con còn lại theo preorder



b e

PreOrder

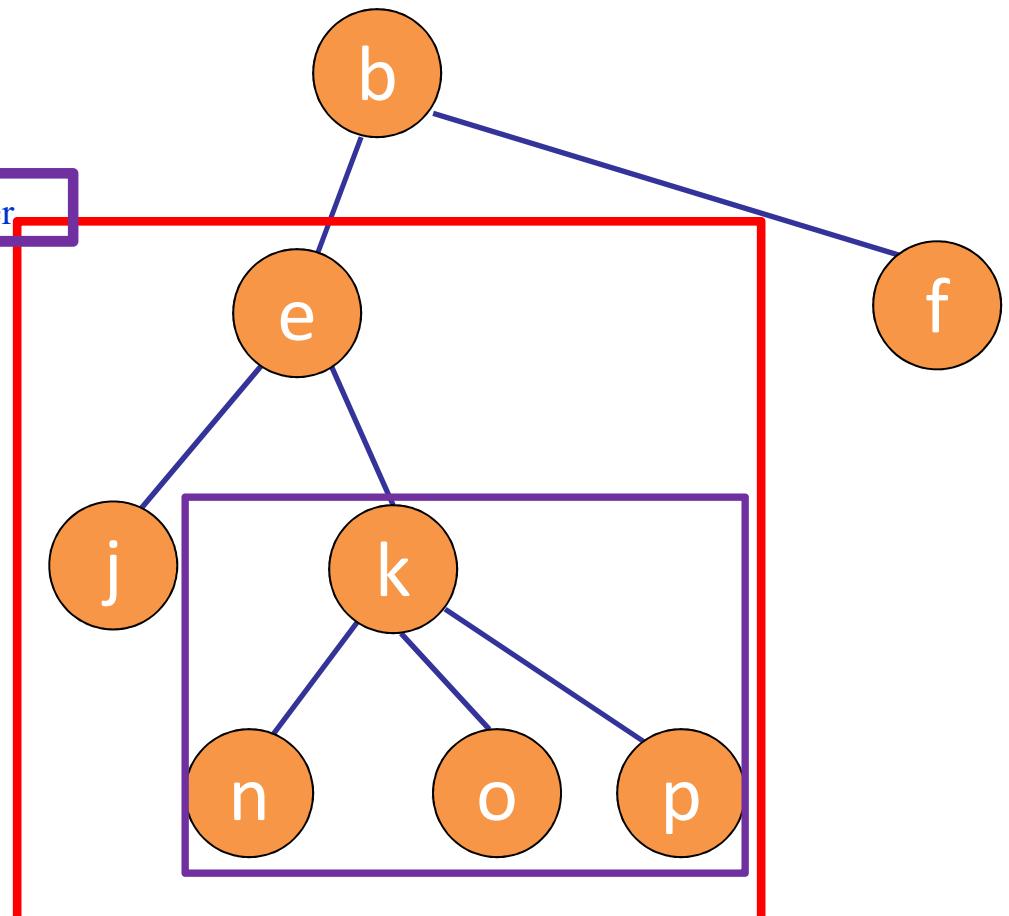
1. Thăm gốc
2. Thăm cây con trái nhất theo preorder
 - 2.1 Thăm gốc
 - 2.2. Thăm cây con trái nhất theo preorder
 - 2.3. Lần lượt thăm các cây con còn lại theo preorder
3. Lần lượt thăm các cây con còn lại theo preorder



b e j

PreOrder

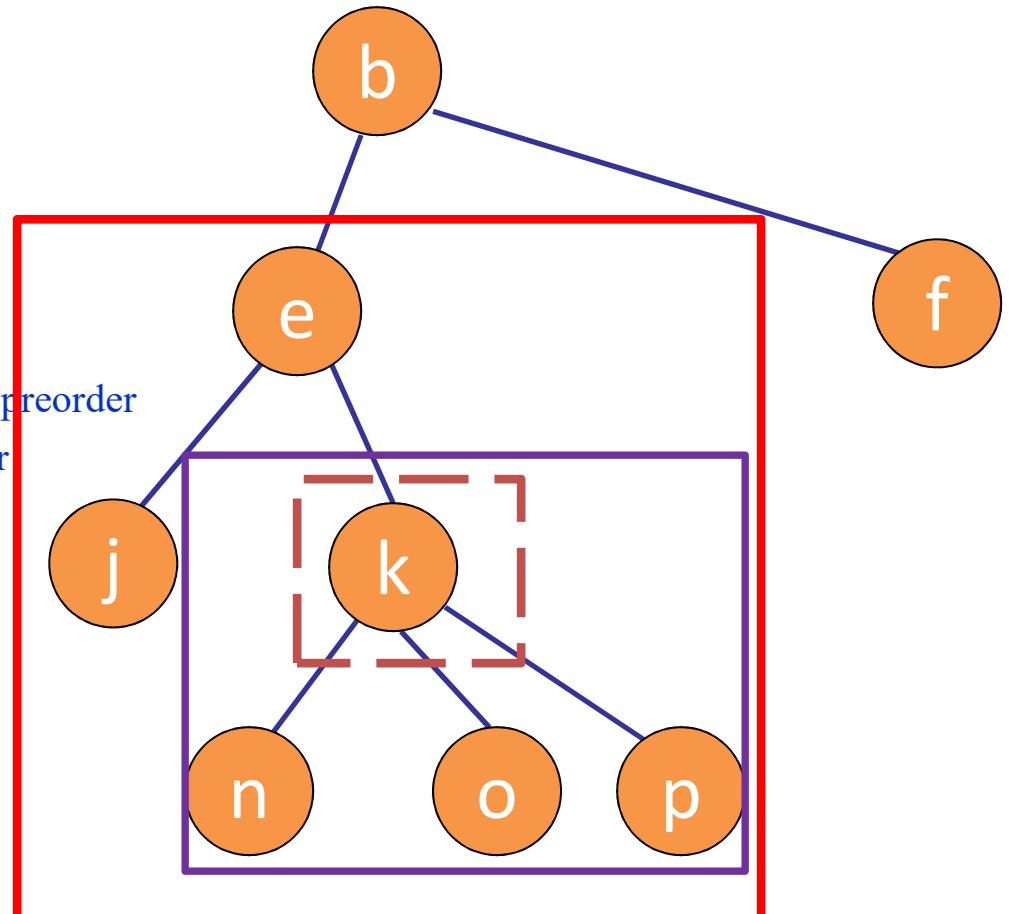
1. Thăm gốc
2. Thăm cây con trái nhất theo preorder
 - 2.1. Thăm gốc
 - 2.2. Thăm cây con trái nhất theo preorder
 - 2.3. Lần lượt thăm các cây con còn lại theo preorder
3. Lần lượt thăm các cây con còn lại theo preorder



b e j

PreOrder

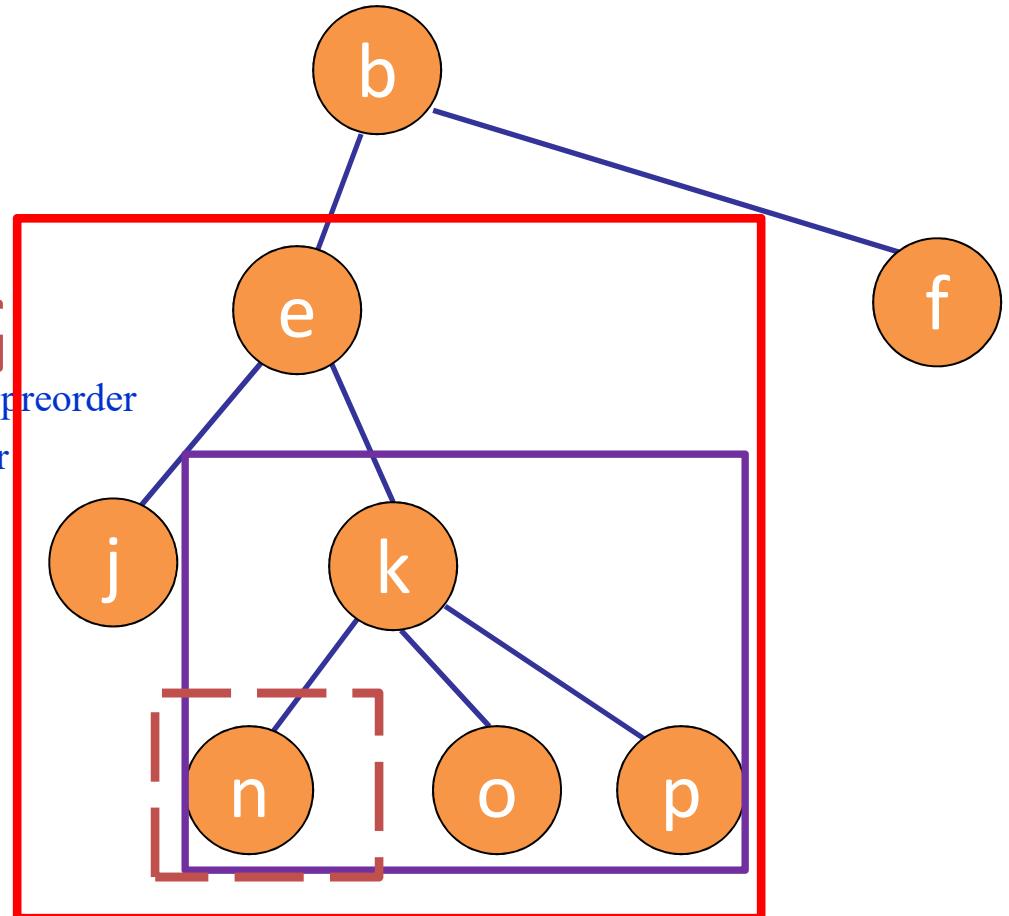
1. Thăm gốc
2. Thăm cây con trái nhất theo preorder
 - 2.1. Thăm gốc
 - 2.2. Thăm cây con trái nhất theo preorder
 - 2.3. Lần lượt thăm các cây con theo preorder
 - 2.3.1. Thăm gốc
 - 2.3.2. Thăm cây con trái nhất theo preorder
 - 2.3.3. Lần lượt thăm các cây con còn lại theo preorder
3. Lần lượt thăm các cây con còn lại theo preorder



b e j k

PreOrder

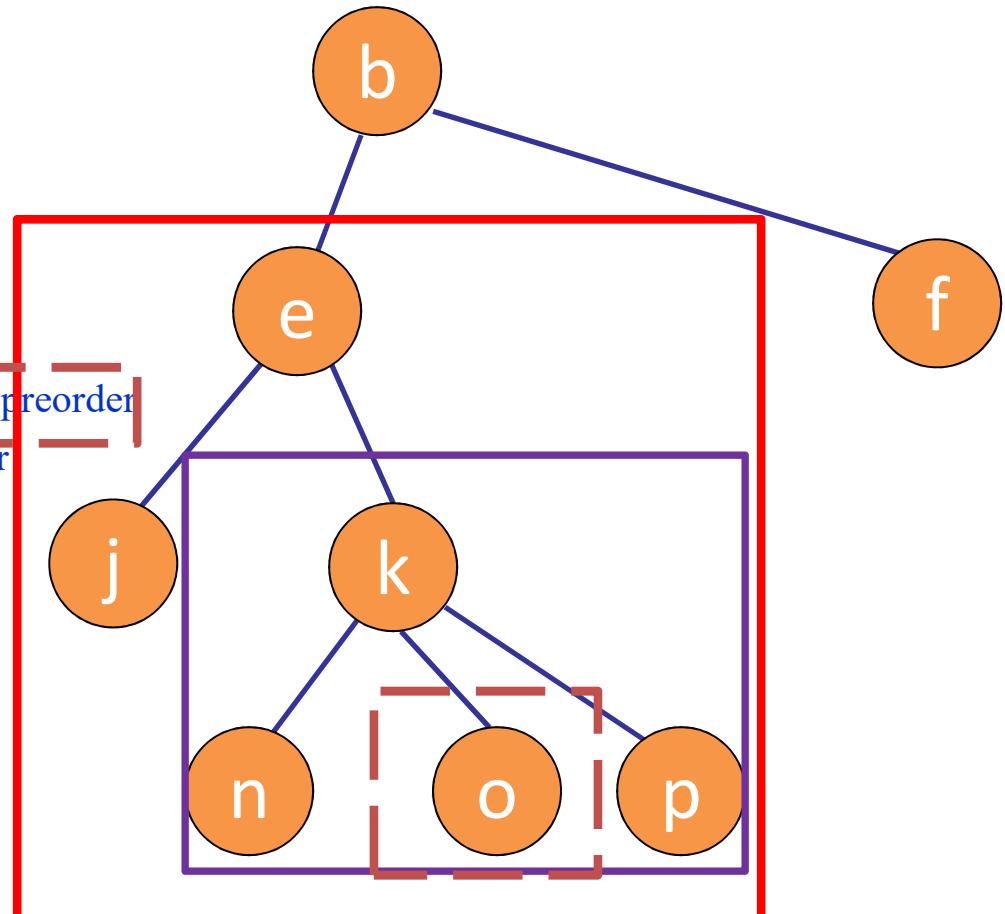
1. Thăm gốc
2. Thăm cây con trái nhất theo preorder
 - 2.1. Thăm gốc
 - 2.2. Thăm cây con trái nhất theo preorder
 - 2.3. Lần lượt thăm các cây con theo preorder
 - 2.3.1. Thăm gốc
 - 2.3.2. Thăm cây con trái nhất theo preorder
 - 2.3.3. Lần lượt thăm các cây con còn lại theo preorder
3. Lần lượt thăm các cây con còn lại theo preorder



b e j k n

PreOrder

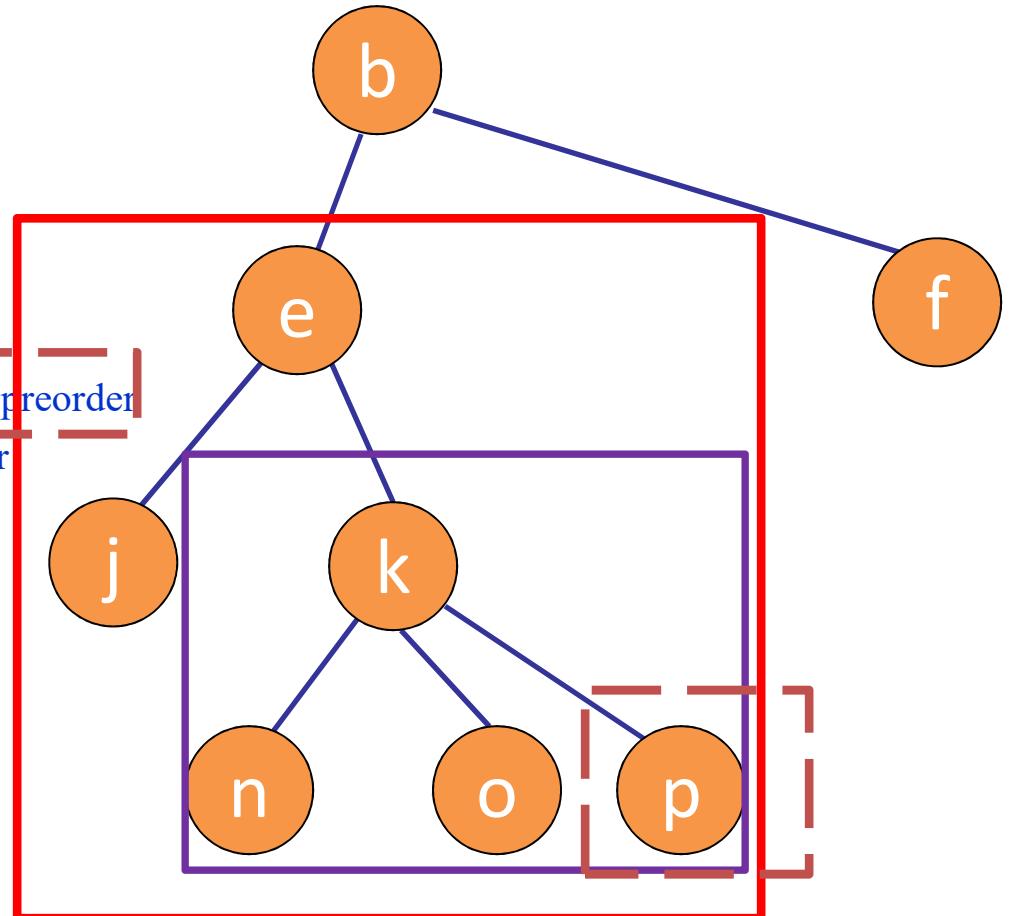
1. Thăm gốc
2. Thăm cây con trái nhất theo preorder
 - 2.1. Thăm gốc
 - 2.2. Thăm cây con trái nhất theo preorder
 - 2.3. Lần lượt thăm các cây con theo preorder
 - 2.3.1. Thăm gốc
 - 2.3.2. Thăm cây con trái nhất theo preorder
 - 2.3.3. Lần lượt thăm các cây con còn lại theo preorder
3. Lần lượt thăm các cây con còn lại theo preorder



b e j k n o

PreOrder

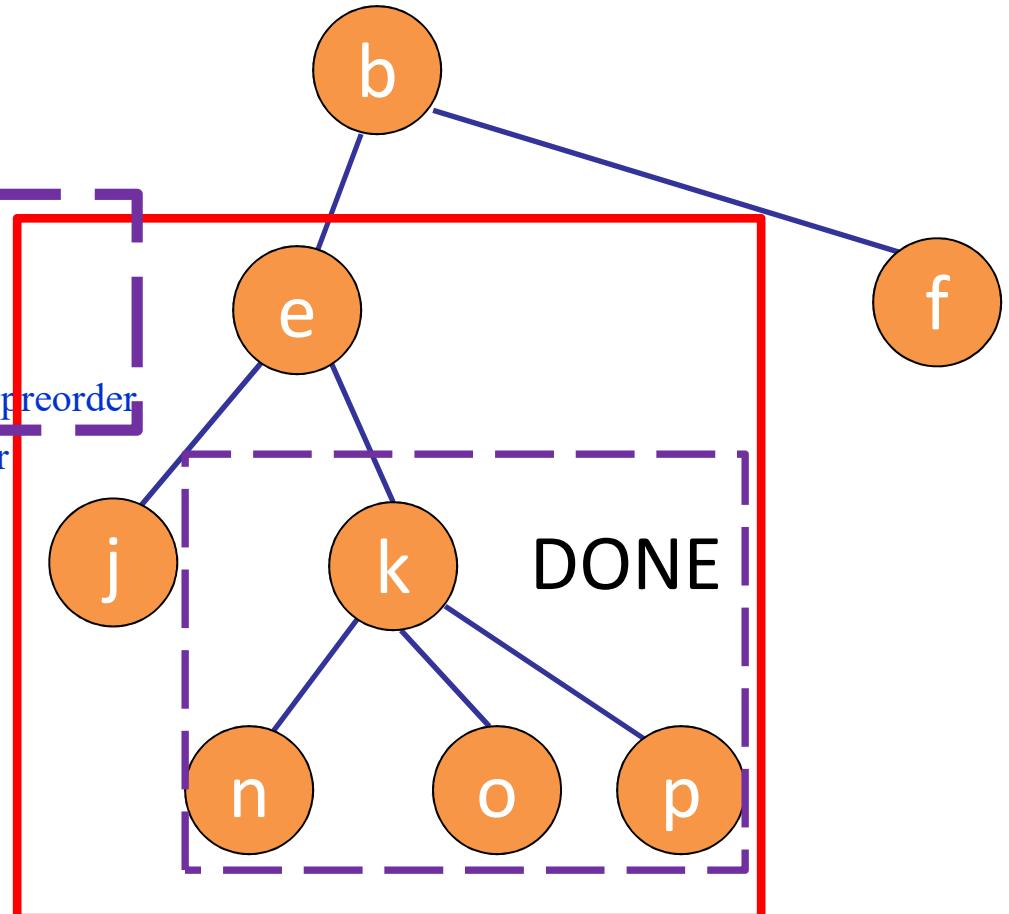
1. Thăm gốc
2. Thăm cây con trái nhất theo preorder
 - 2.1. Thăm gốc
 - 2.2. Thăm cây con trái nhất theo preorder
 - 2.3. Lần lượt thăm các cây con theo preorder
 - 2.3.1. Thăm gốc
 - 2.3.2. Thăm cây con trái nhất theo preorder
 - 2.3.3. Lần lượt thăm các cây con còn lại theo preorder
3. Lần lượt thăm các cây con còn lại theo preorder



b e j k n o p

PreOrder

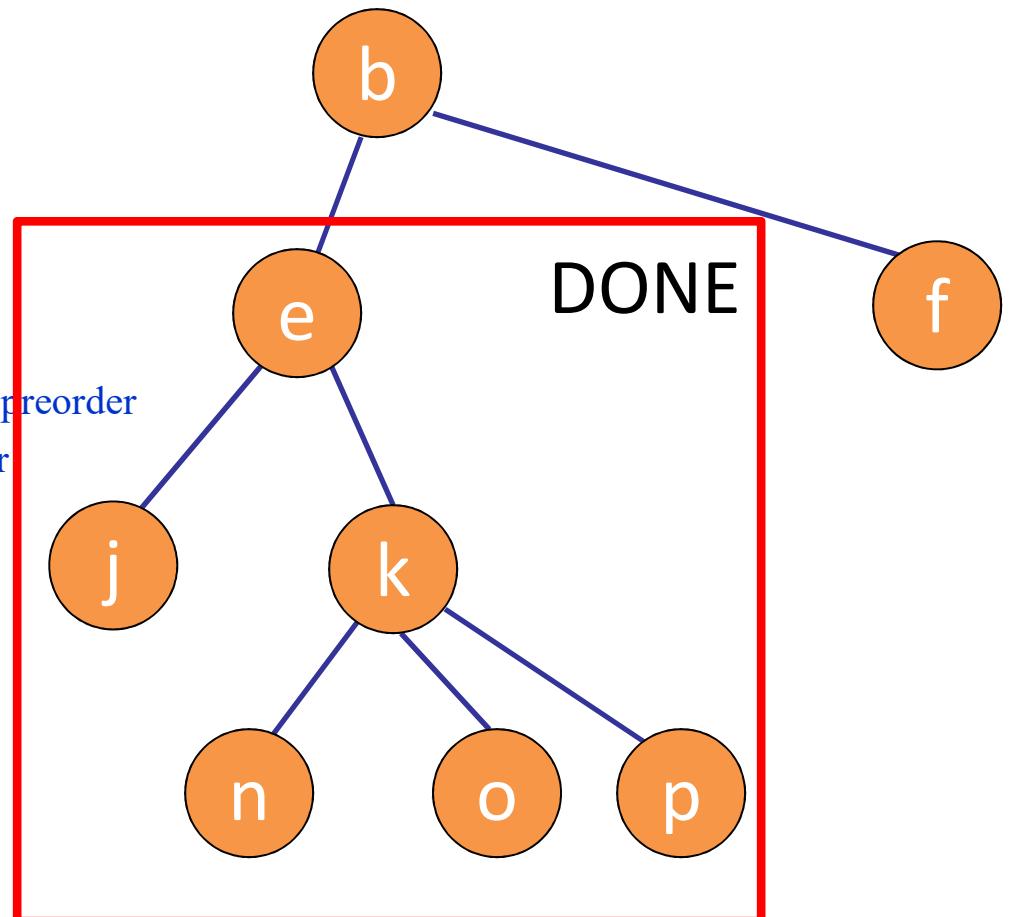
1. Thăm gốc
2. Thăm cây con trái nhất theo preorder
 - 2.1. Thăm gốc
 - 2.2. Thăm cây con trái nhất theo preorder
 - 2.3. Lần lượt thăm các cây con theo preorder
 - 2.3.1. Thăm gốc
 - 2.3.2. Thăm cây con trái nhất theo preorder
 - 2.3.3. Lần lượt thăm các cây con còn lại theo preorder
3. Lần lượt thăm các cây con còn lại theo preorder



b e j k n o p

PreOrder

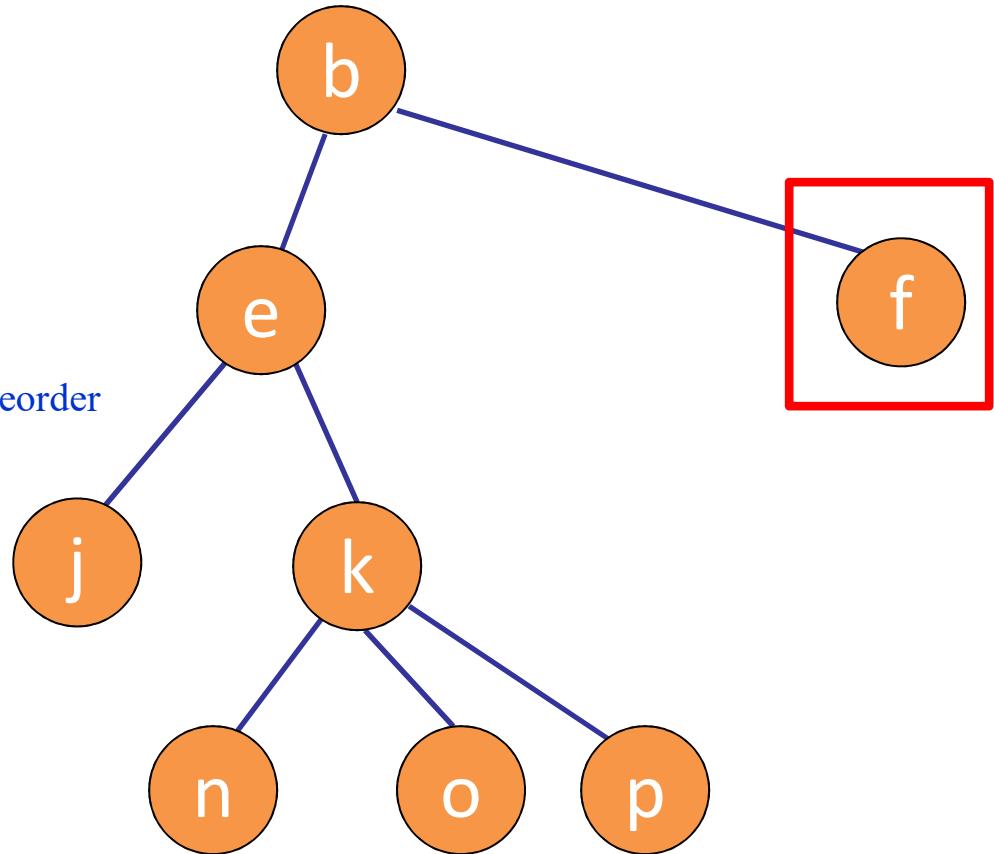
1. Thăm gốc
2. Thăm cây con trái nhất theo preorder
 - 2.1. Thăm gốc
 - 2.2. Thăm cây con trái nhất theo preorder
 - 2.3. Lần lượt thăm các cây con theo preorder
 - 2.3.1. Thăm gốc
 - 2.3.2. Thăm cây con trái nhất theo preorder
 - 2.3.3. Lần lượt thăm các cây con còn lại theo preorder
3. Lần lượt thăm các cây con còn lại theo preorder



b e j k n o p

PreOrder

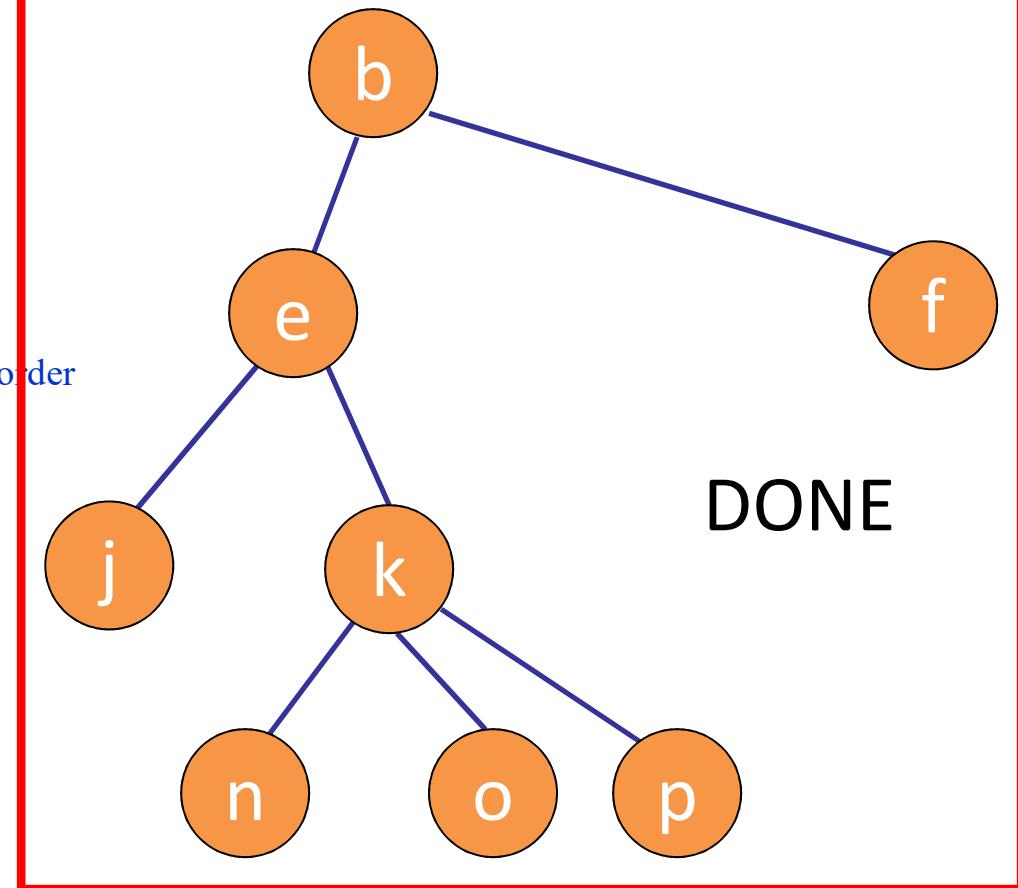
1. Thăm gốc
2. Thăm cây con trái nhất theo preorder
 - 2.1. Thăm gốc
 - 2.2. Thăm cây con trái nhất theo preorder
 - 2.3. Lần lượt thăm các cây con theo preorder
 - 2.3.1. Thăm gốc
 - 2.3.2. Thăm cây con trái nhất theo preorder
 - 2.3.3. Lần lượt thăm các cây con còn lại theo preorder
3. Lần lượt thăm các cây con còn lại theo preorder



b e j k n o p f

PreOrder

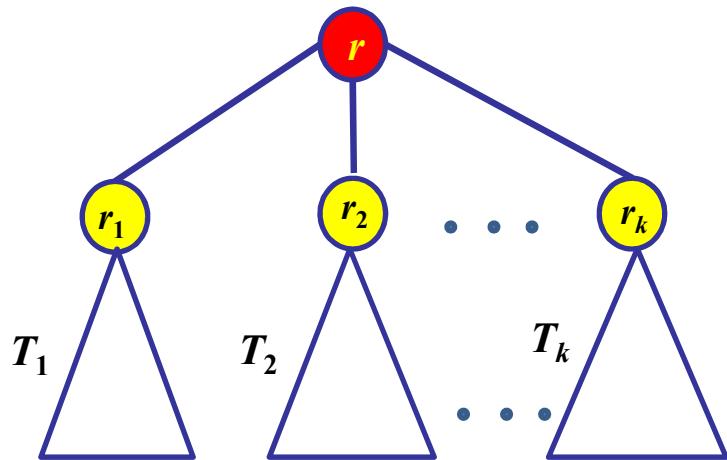
1. Thăm gốc
2. Thăm cây con trái nhất theo preorder
 - 2.1. Thăm gốc
 - 2.2. Thăm cây con trái nhất theo preorder
 - 2.3. Lần lượt thăm các cây con theo preorder
 - 2.3.1. Thăm gốc
 - 2.3.2. Thăm cây con trái nhất theo preorder
 - 2.3.3. Lần lượt thăm các cây con còn lại theo preorder
3. Lần lượt thăm các cây con còn lại theo preorder



b e j k n o p f

Duyệt thứ tự giữa (Inorder Traversal)

- Duyệt cây T theo thứ tự giữa:

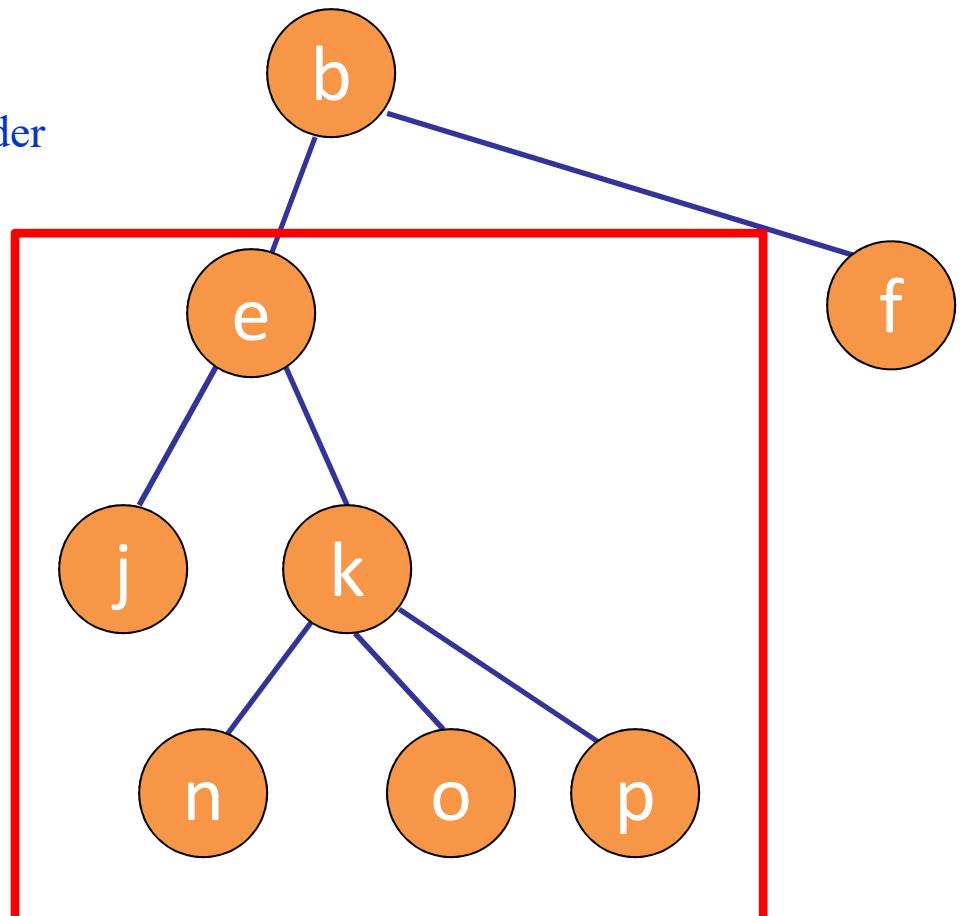


- Bước 1: thăm T_1 theo thứ tự giữa,
- Bước 2: thăm nút gốc r ,
- Bước 3: thăm T_2 theo thứ tự giữa,
-
- Bước $k+1$: thăm T_k theo thứ tự giữa

```
procedure inorder( $T$ : ordered rooted tree)
   $r :=$  root of  $T$ 
  if  $r$  is a leaf then visit  $r$ 
  else
    begin
       $l :=$  first child of  $r$  from left to right
       $T(l) :=$  subtree with  $l$  as its root
      inorder( $T(l)$ ) Step 1
      visit  $r$  Step 2
      for each child  $c$  of  $r$  except for  $l$  left to right
         $T(c) :=$  subtree with  $c$  as its root
        inorder( $T(c)$ ) Step 3, 4...
    end
```

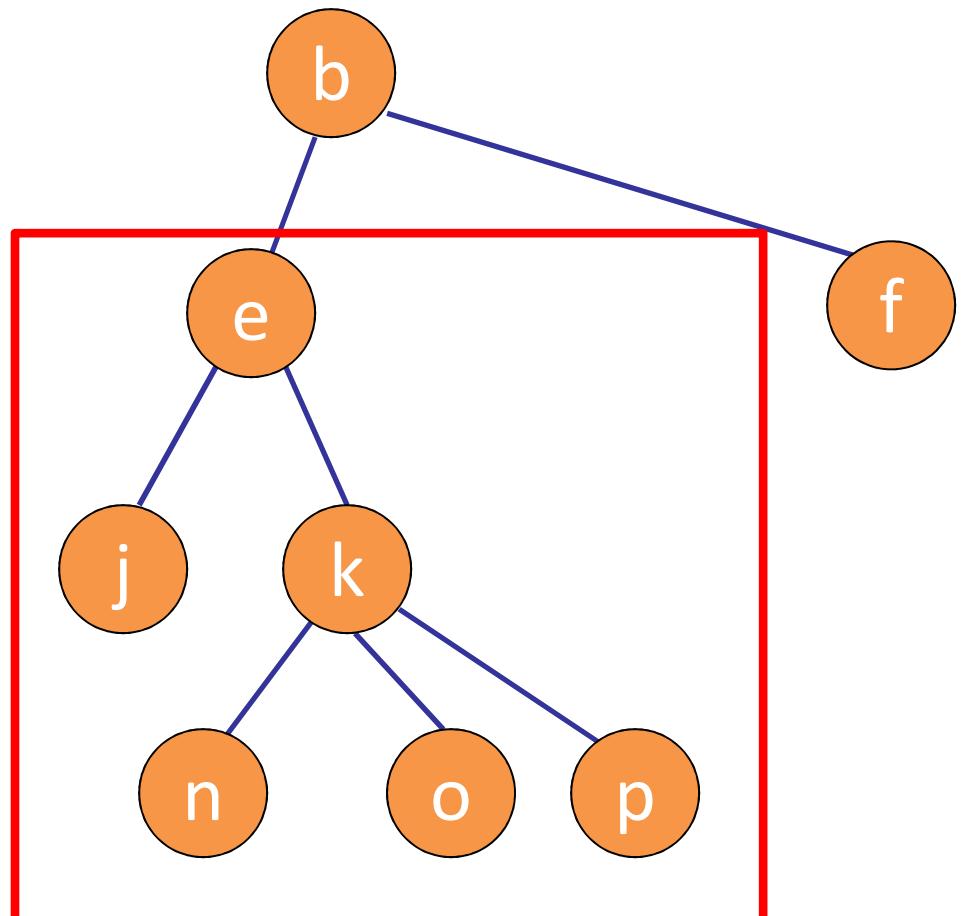
InOrder

1. Thăm cây con trái nhất theo InOrder
2. Thăm gốc
3. Lần lượt thăm các cây con còn lại theo Inorder



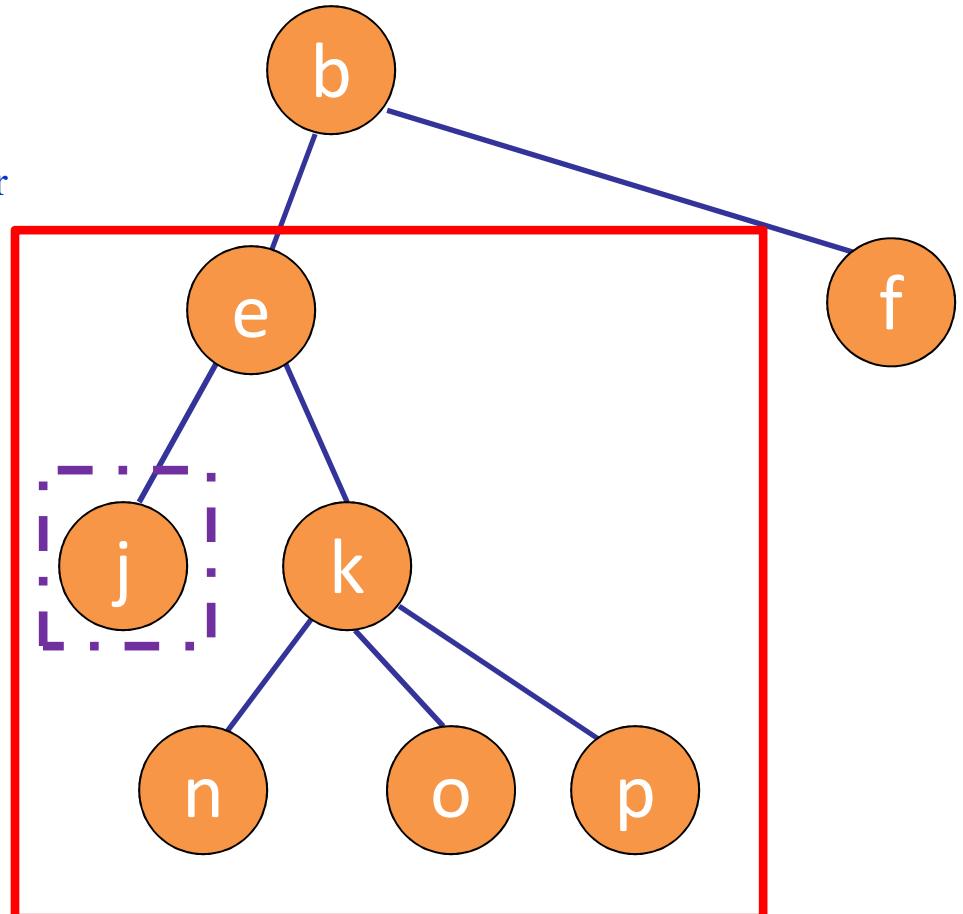
InOrder

1. Thăm cây con trái nhất theo Inorder
 - 1.1. Thăm cây con trái nhất theo Inorder
 - 1.2. Thăm gốc
 - 1.3. Thăm các cây con còn lại theo Inorder
2. Thăm gốc
3. Thăm các cây con còn lại theo Inorder



InOrder

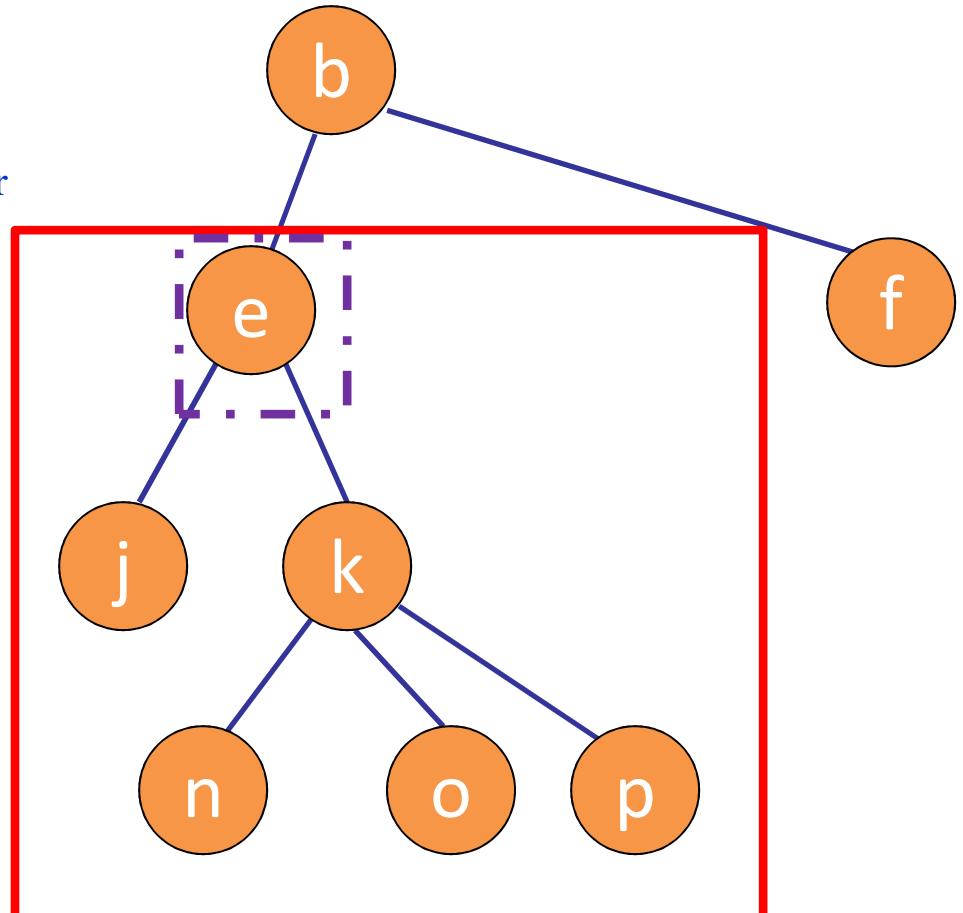
- 1. Thăm cây con trái nhất theo Inorder
 - 1.1. Thăm cây con trái nhất theo Inorder
 - 1.2. Thăm gốc
 - 1.3. Thăm lần lượt các cây con còn lại theo Inorder
- 2. Thăm gốc
- 3. Thăm lần lượt các cây con còn lại theo Inorder



j

InOrder

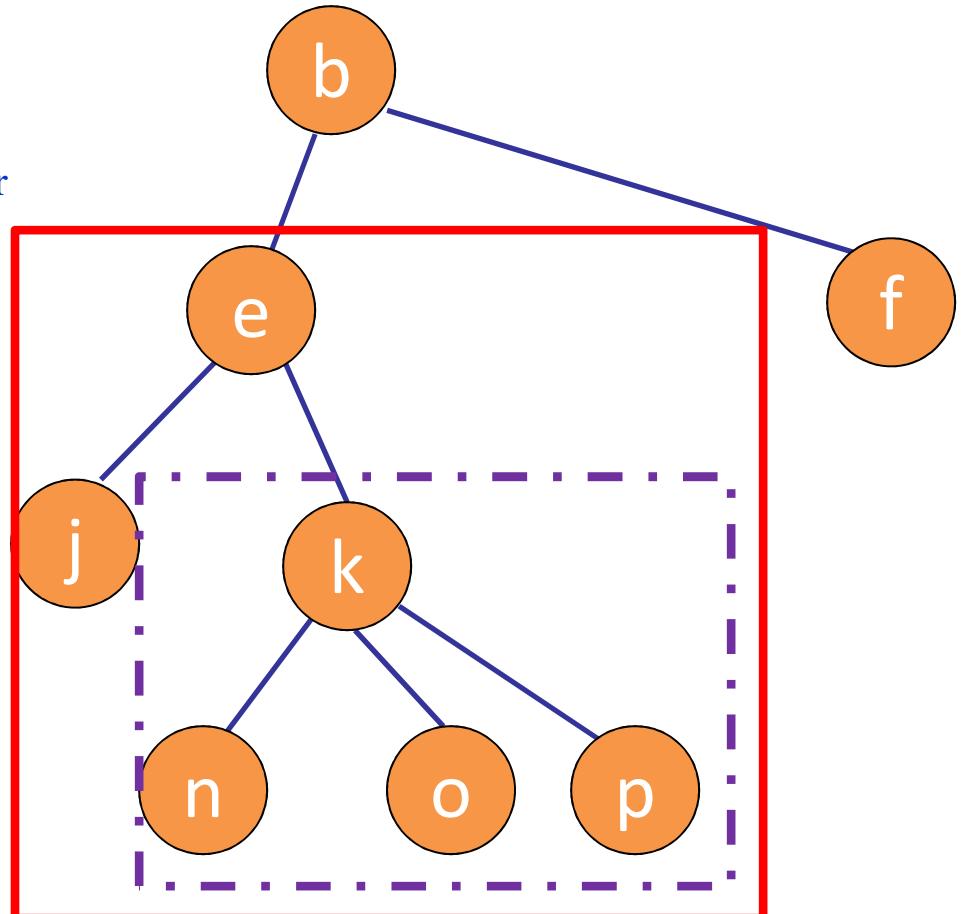
1. Thăm cây con trái nhất theo Inorder
 - 1.1. Thăm cây con trái nhất theo Inorder
 - 1.2. Thăm gốc
 - 1.3. Thăm lần lượt các cây con còn lại theo Inorder
2. Thăm gốc
3. Thăm lần lượt các cây con còn lại theo Inorder



j e

InOrder

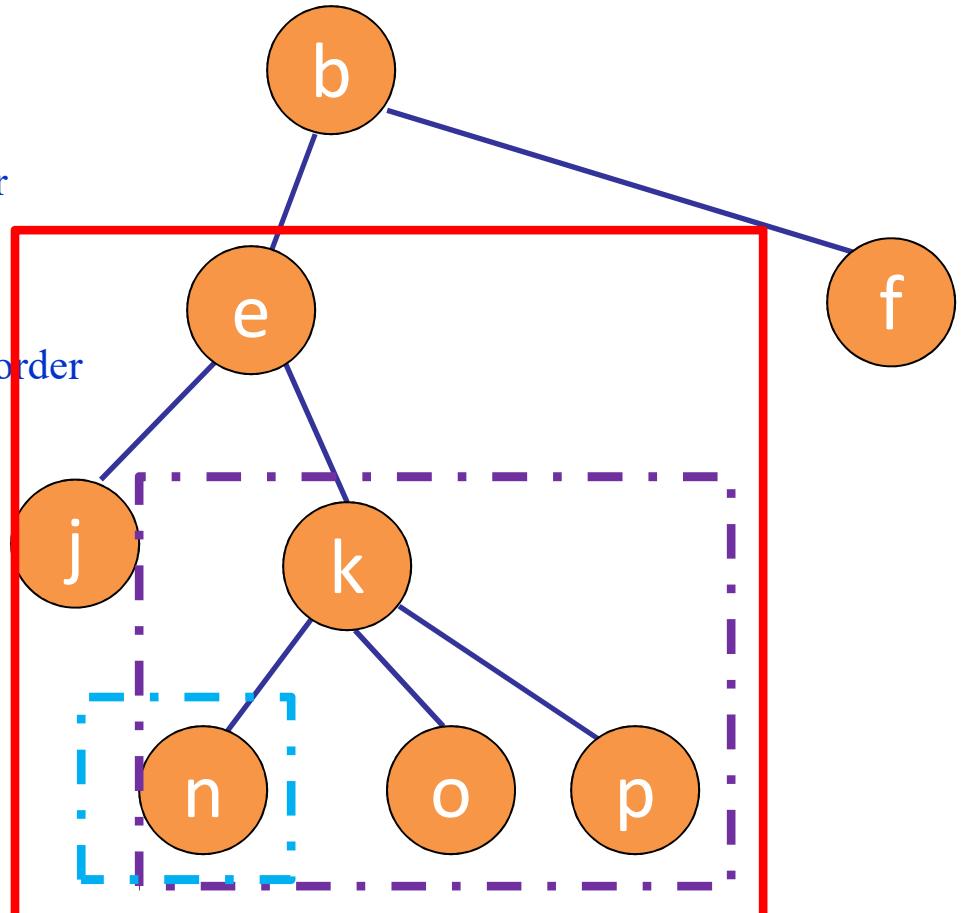
1. Thăm cây con trái nhất theo Inorder
 - 1.1. Thăm cây con trái nhất theo Inorder
 - 1.2. Thăm gốc
 - 1.3. Thăm lần lượt các cây con còn lại theo Inorder
2. Thăm gốc
3. Thăm lần lượt các cây con còn lại theo Inorder



j e

InOrder

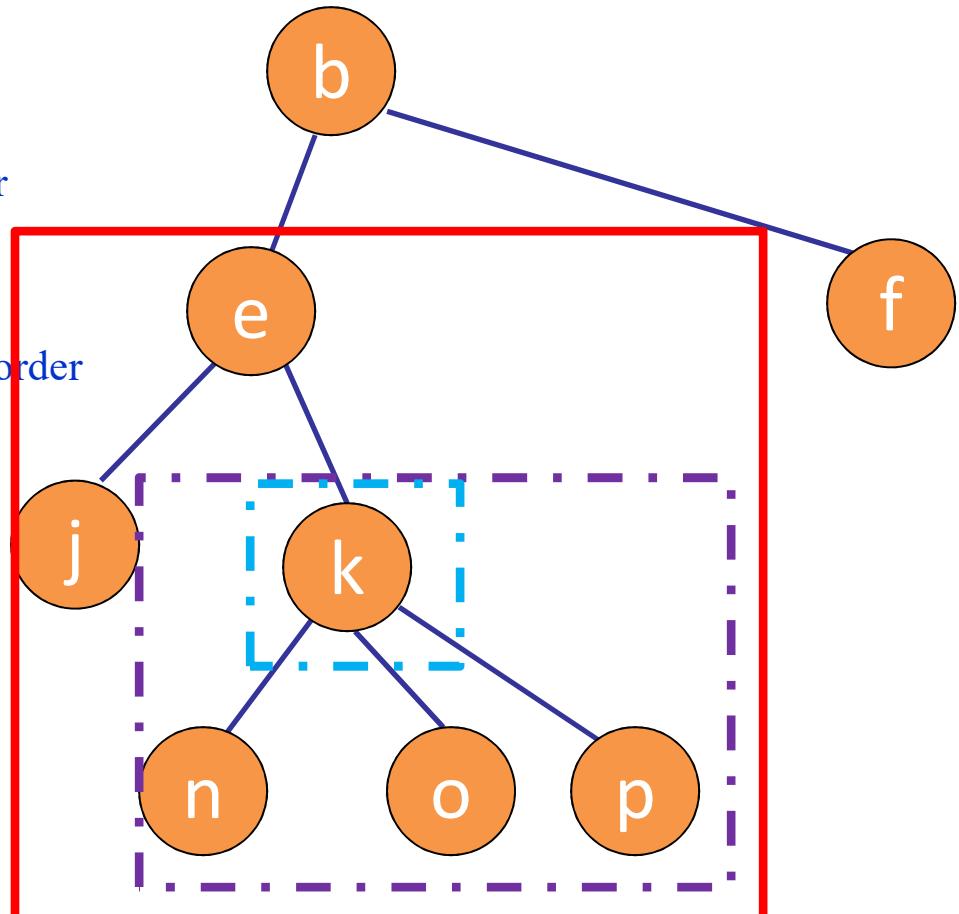
1. Thăm cây con trái nhất theo Inorder
 - 1.1. Thăm cây con trái nhất theo Inorder
 - 1.2. Thăm gốc
 - 1.3. Thăm lần lượt các cây con còn lại theo Inorder
 - 1.3.1. Thăm cây con trái nhất theo Inorder
 - 1.3.2. Thăm gốc
 - 1.3.3. Thăm lần lượt các cây con còn lại theo Inorder
2. Thăm gốc
3. Thăm lần lượt các cây con còn lại theo Inorder



j e n

InOrder

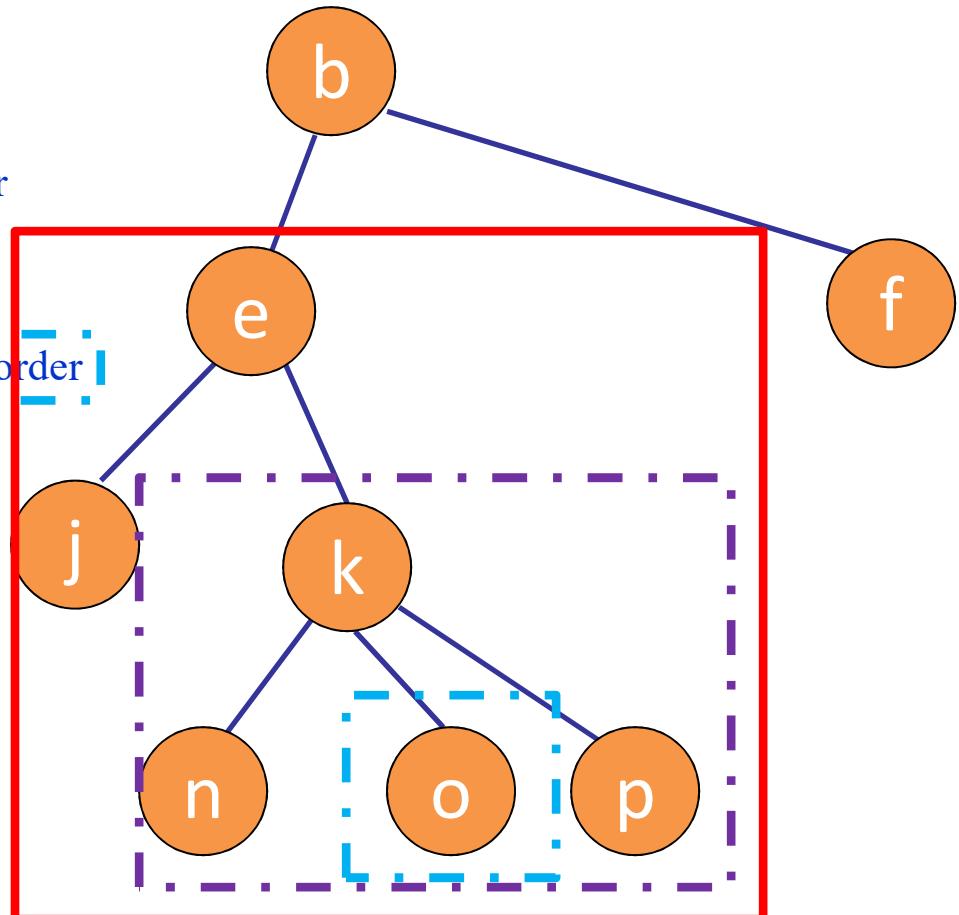
1. Thăm cây con trái nhất theo Inorder
 - 1.1. Thăm cây con trái nhất theo Inorder
 - 1.2. Thăm gốc
 - 1.3. Thăm lần lượt các cây con còn lại theo Inorder
 - 1.3.1. Thăm cây con trái nhất theo Inorder
 - 1.3.2. Thăm gốc
 - 1.3.3. Thăm lần lượt các cây con còn lại theo Inorder
2. Thăm gốc
3. Thăm lần lượt các cây con còn lại theo Inorder



j e n k

InOrder

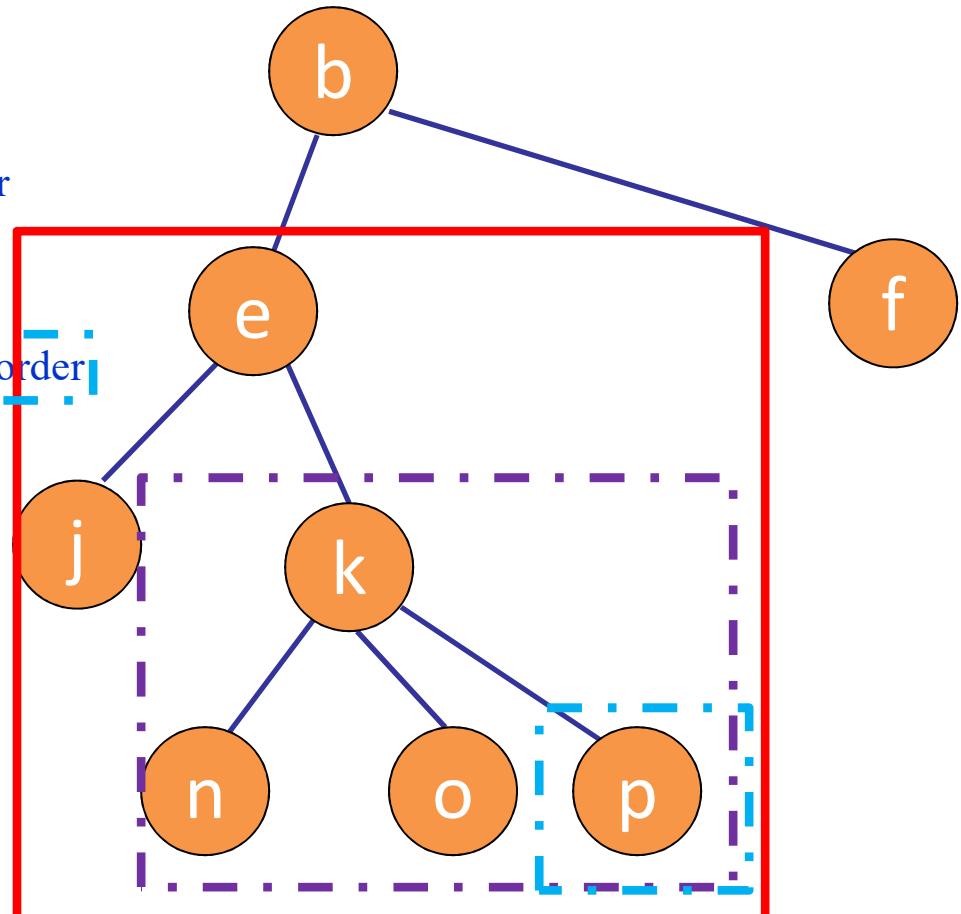
1. Thăm cây con trái nhất theo Inorder
 - 1.1. Thăm cây con trái nhất theo Inorder
 - 1.2. Thăm gốc
 - 1.3. Thăm lần lượt các cây con còn lại theo Inorder
 - 1.3.1. Thăm cây con trái nhất theo Inorder
 - 1.3.2. Thăm gốc
 - 1.3.3. Thăm lần lượt các cây con còn lại theo Inorder
2. Thăm gốc
3. Thăm lần lượt các cây con còn lại theo Inorder



j e n k o

InOrder

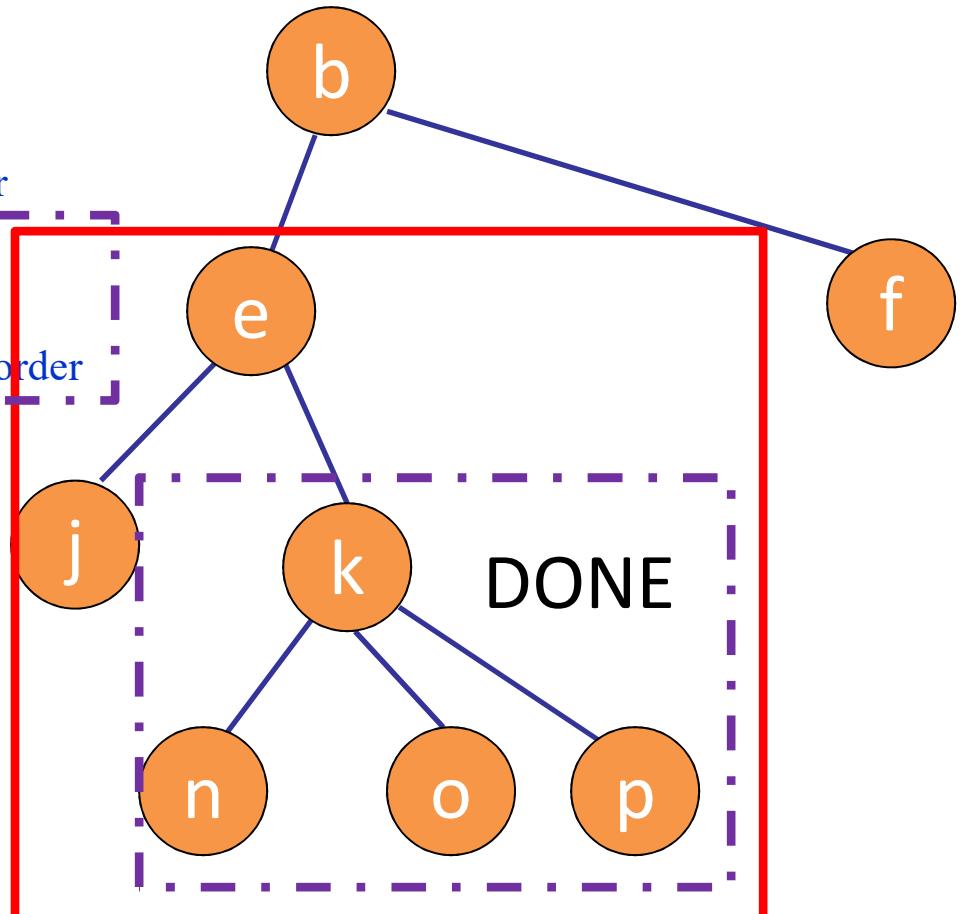
1. Thăm cây con trái nhất theo Inorder
 - 1.1. Thăm cây con trái nhất theo Inorder
 - 1.2. Thăm gốc
 - 1.3. Thăm lần lượt các cây con còn lại theo Inorder
 - 1.3.1. Thăm cây con trái nhất theo Inorder
 - 1.3.2. Thăm gốc
 - 1.3.3. Thăm lần lượt các cây con còn lại theo Inorder
2. Thăm gốc
3. Thăm lần lượt các cây con còn lại theo Inorder



j e n k o p

InOrder

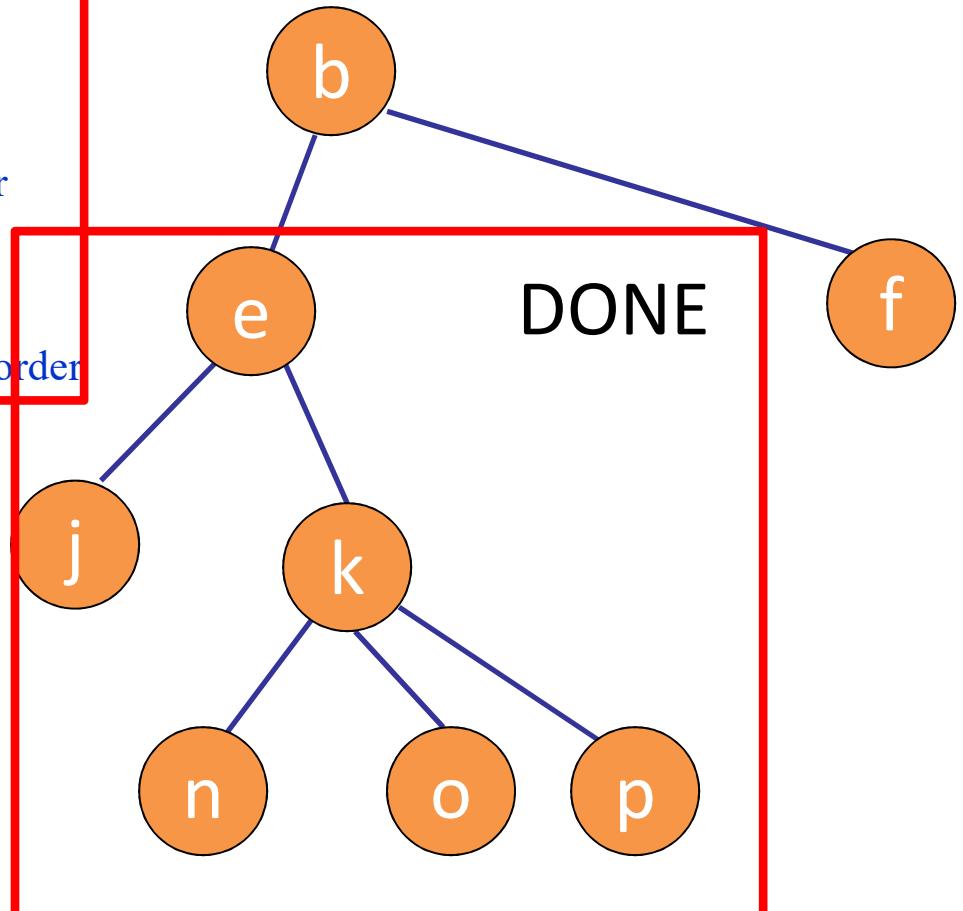
1. Thăm cây con trái nhất theo Inorder
 - 1.1. Thăm cây con trái nhất theo Inorder
 - 1.2. Thăm gốc
 - 1.3. Thăm lần lượt các cây con còn lại theo Inorder
 - 1.3.1. Thăm cây con trái nhất theo Inorder
 - 1.3.2. Thăm gốc
 - 1.3.3. Thăm lần lượt các cây con còn lại theo Inorder
2. Thăm gốc
3. Thăm lần lượt các cây con còn lại theo Inorder



j e n k o p

InOrder

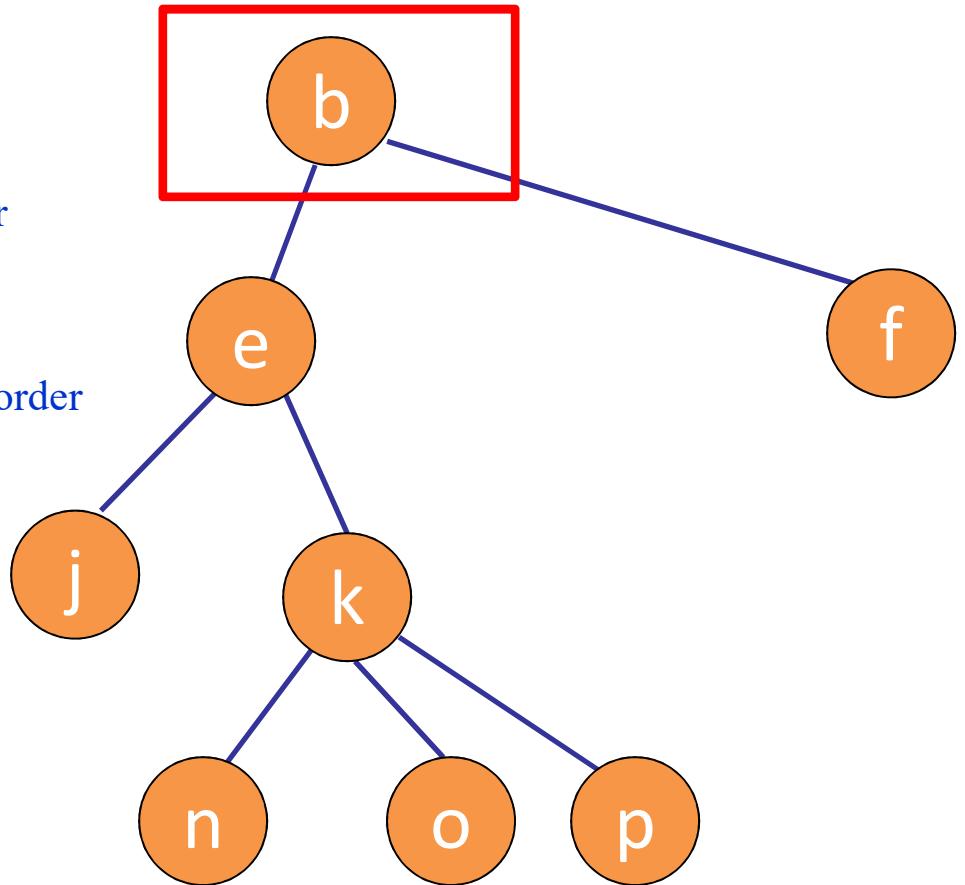
1. Thăm cây con trái nhất theo Inorder
 - 1.1. Thăm cây con trái nhất theo Inorder
 - 1.2. Thăm gốc
 - 1.3. Thăm lần lượt các cây con còn lại theo Inorder
 - 1.3.1. Thăm cây con trái nhất theo Inorder
 - 1.3.2. Thăm gốc
 - 1.3.3. Thăm lần lượt các cây con còn lại theo Inorder
2. Thăm gốc
3. Thăm lần lượt các cây con còn lại theo Inorder



j e n k o p

InOrder

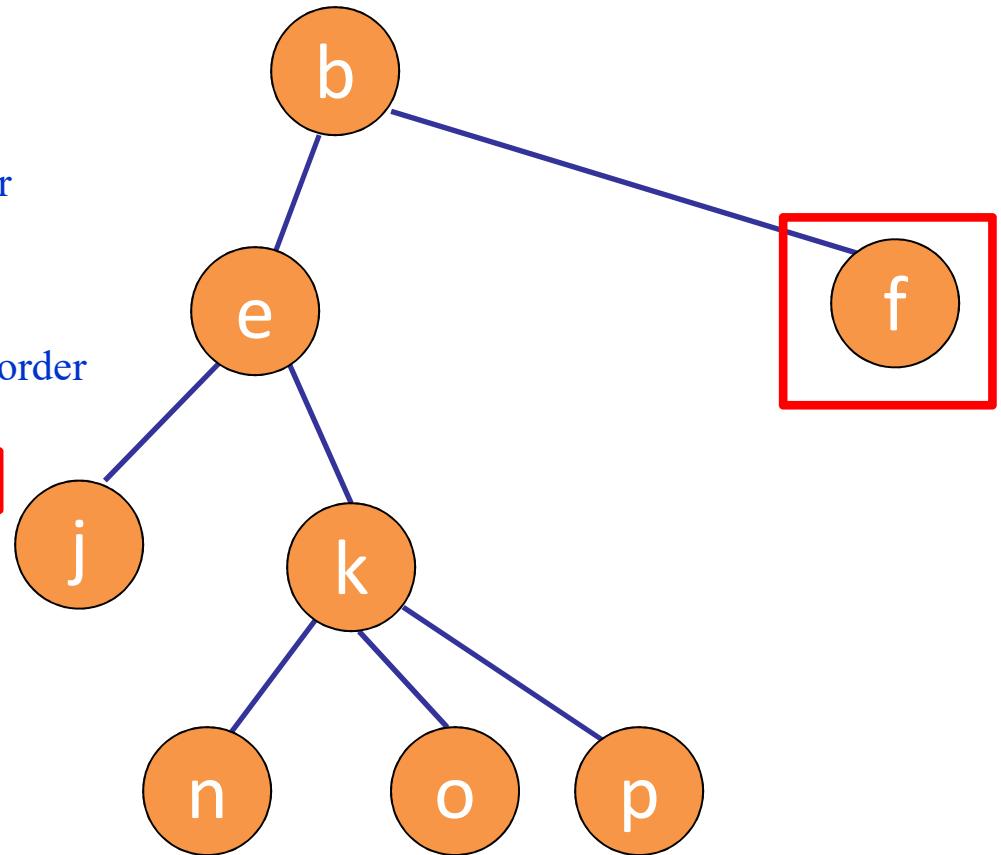
1. Thăm cây con trái nhất theo Inorder
 - 1.1. Thăm cây con trái nhất theo Inorder
 - 1.2. Thăm gốc
 - 1.3. Thăm lần lượt các cây con còn lại theo Inorder
 - 1.3.1. Thăm cây con trái nhất theo Inorder
 - 1.3.2. Thăm gốc
 - 1.3.3. Thăm lần lượt các cây con còn lại theo Inorder
2. Thăm gốc
3. Thăm lần lượt các cây con còn lại theo Inorder



j e n k o p b

InOrder

1. Thăm cây con trái nhất theo Inorder
 - 1.1. Thăm cây con trái nhất theo Inorder
 - 1.2. Thăm gốc
 - 1.3. Thăm lần lượt các cây con còn lại theo Inorder
 - 1.3.1. Thăm cây con trái nhất theo Inorder
 - 1.3.2. Thăm gốc
 - 1.3.3. Thăm lần lượt các cây con còn lại theo Inorder
2. Thăm gốc
3. Thăm lần lượt các cây con còn lại theo Inorder

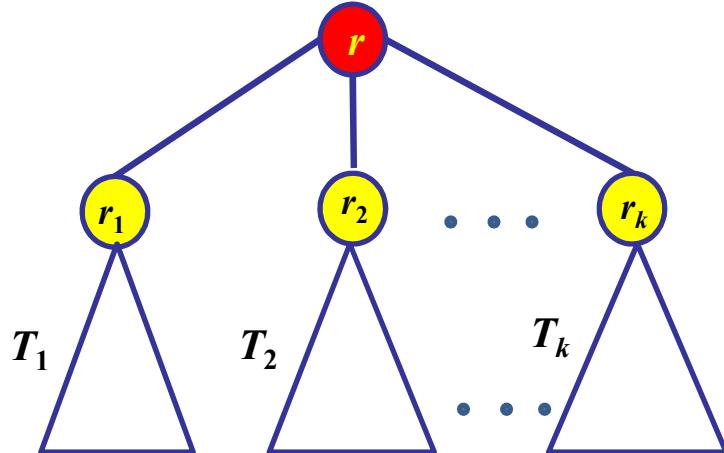


j e n k o p b f

Duyệt thứ tự sau (Postorder Traversal)

Khi duyệt cây theo thứ tự sau, mỗi nút đều được thăm trước con cháu của nó.

Ví dụ: Duyệt cây T theo thứ tự sau:

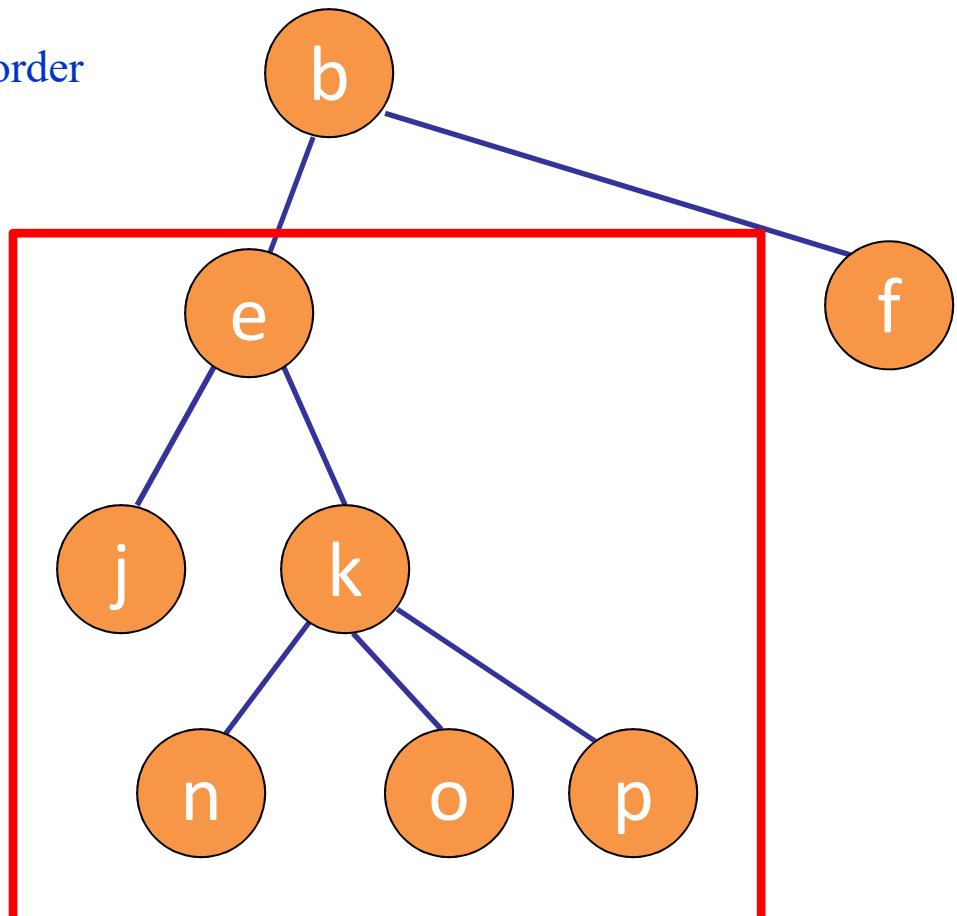


```
procedure postorder( $T$ : ordered rooted tree)
   $r :=$  root of  $T$ 
  for each child  $c$  of  $r$  from left to right
    begin
       $T(c) :=$  subtree with  $c$  as its root
      postorder( $T(c)$ )
    end
    visit  $r$ 
```

- Bước 1: thăm cây con T_1 theo thứ tự sau,
- Bước 2: thăm cây con T_2 theo thứ tự sau,
-
- Bước k : thăm cây con T_k theo thứ tự sau,
- Bước $k+1$: thăm nút gốc r

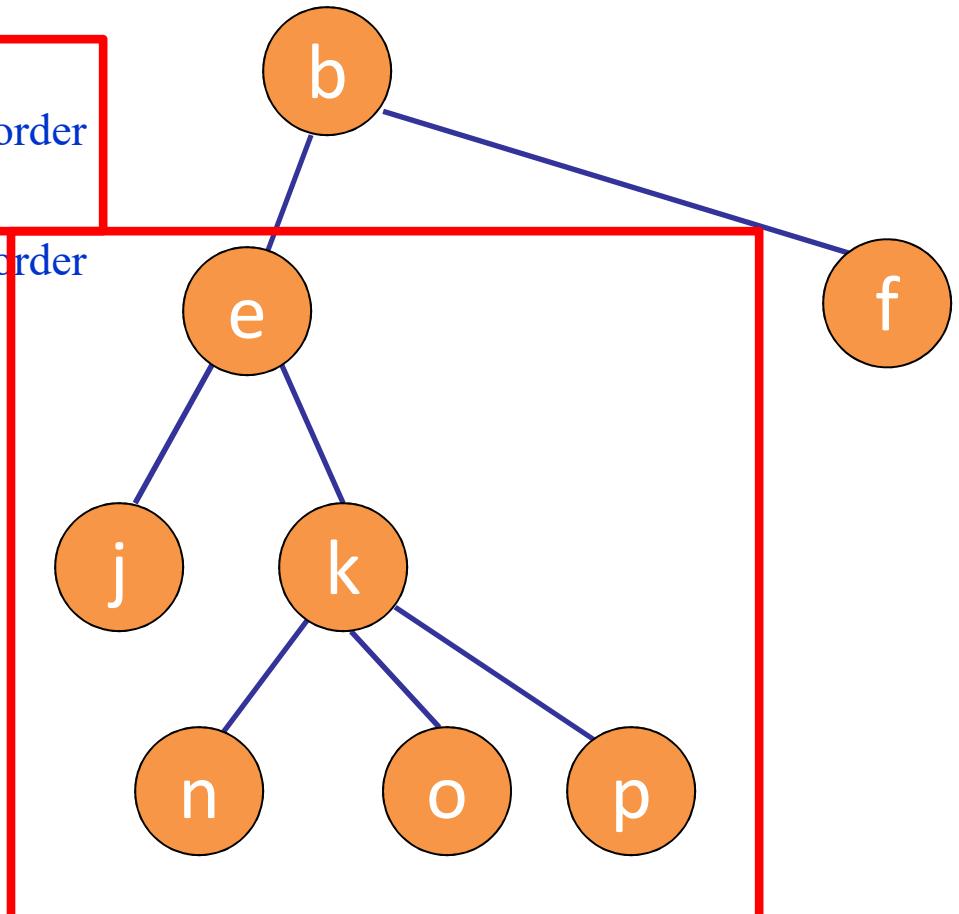
PostOrder

1. Thăm cây con trái nhất theo Postorder
2. Thăm lân lượt các cây con còn lại theo Postorder
3. Thăm gốc



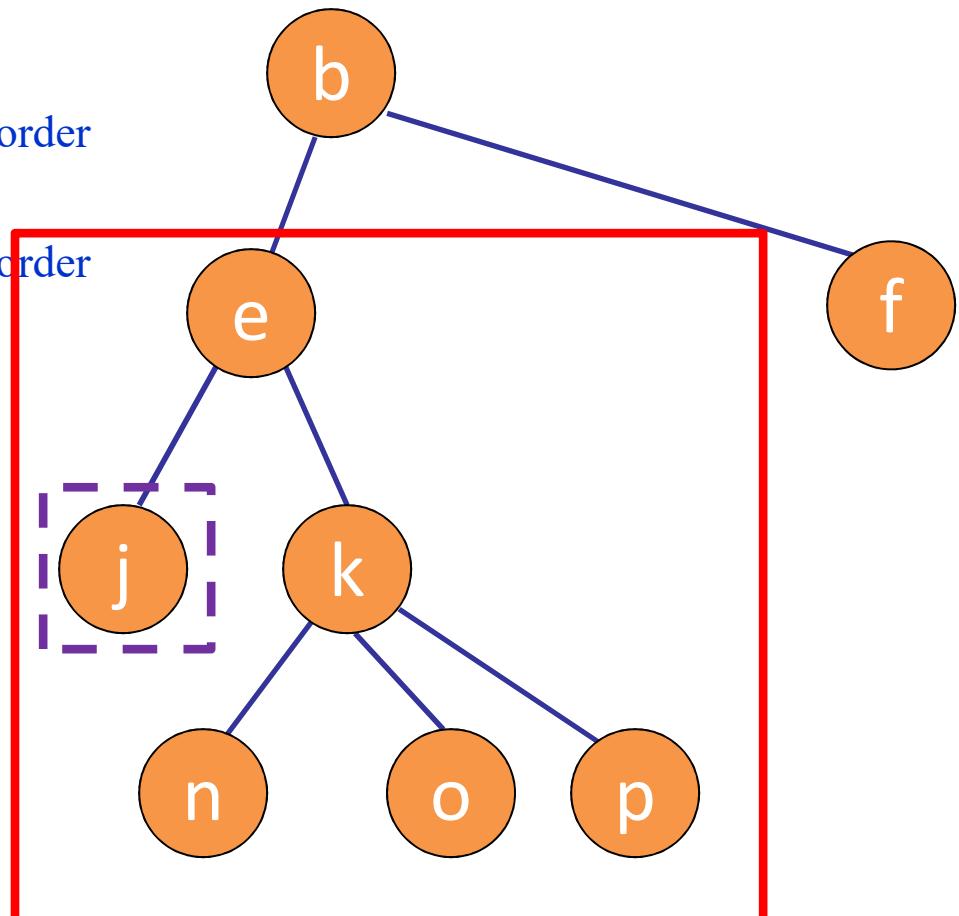
PostOrder

1. Thăm cây con trái nhất theo Postorder
 - 1.1. Thăm cây con trái nhất theo Postorder
 - 1.2. Thăm lần lượt các cây con còn lại theo Postorder
 - 1.3. Thăm gốc
2. Thăm lần lượt các cây con còn lại theo Postorder
3. Thăm gốc



PostOrder

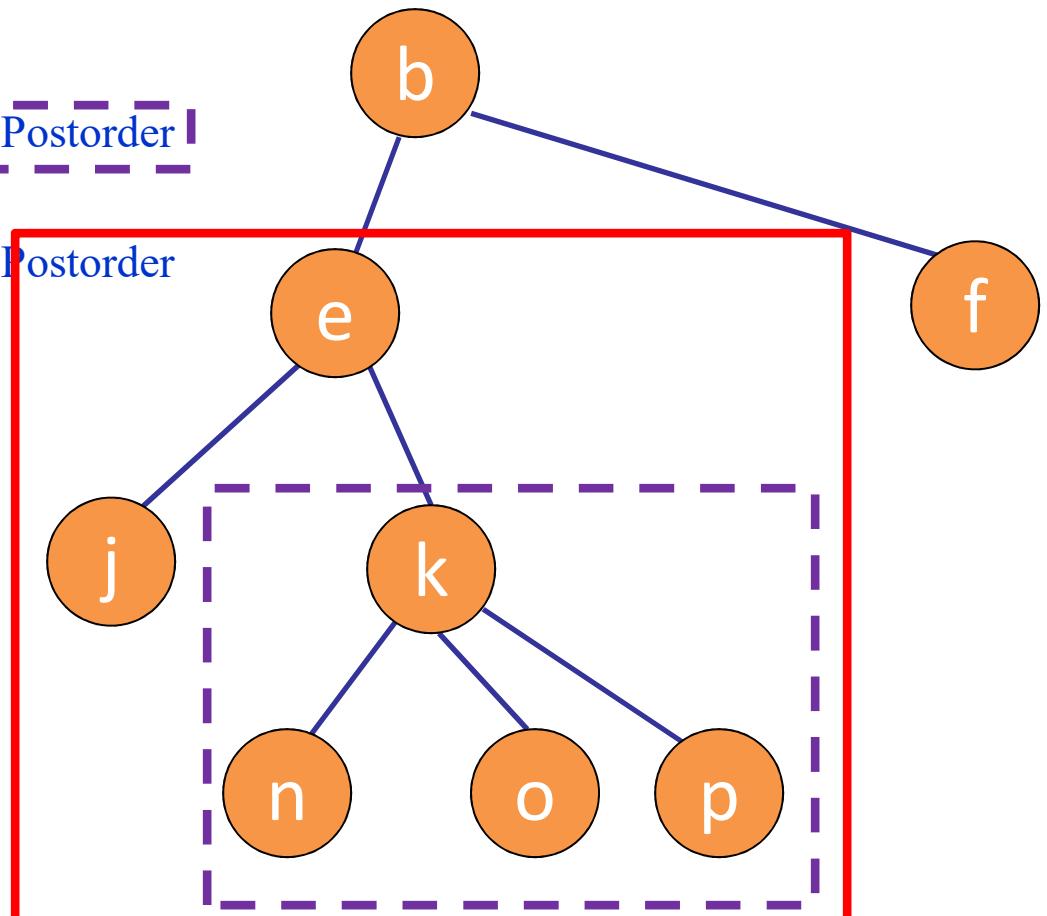
1. Thăm cây con trái nhất theo Postorder
 - 1.1. Thăm cây con trái nhất theo Postorder
 - 1.2. Thăm lần lượt các cây con còn lại theo Postorder
- 1.3. Thăm gốc
2. Thăm lần lượt các cây con còn lại theo Postorder
3. Thăm gốc



j

PostOrder

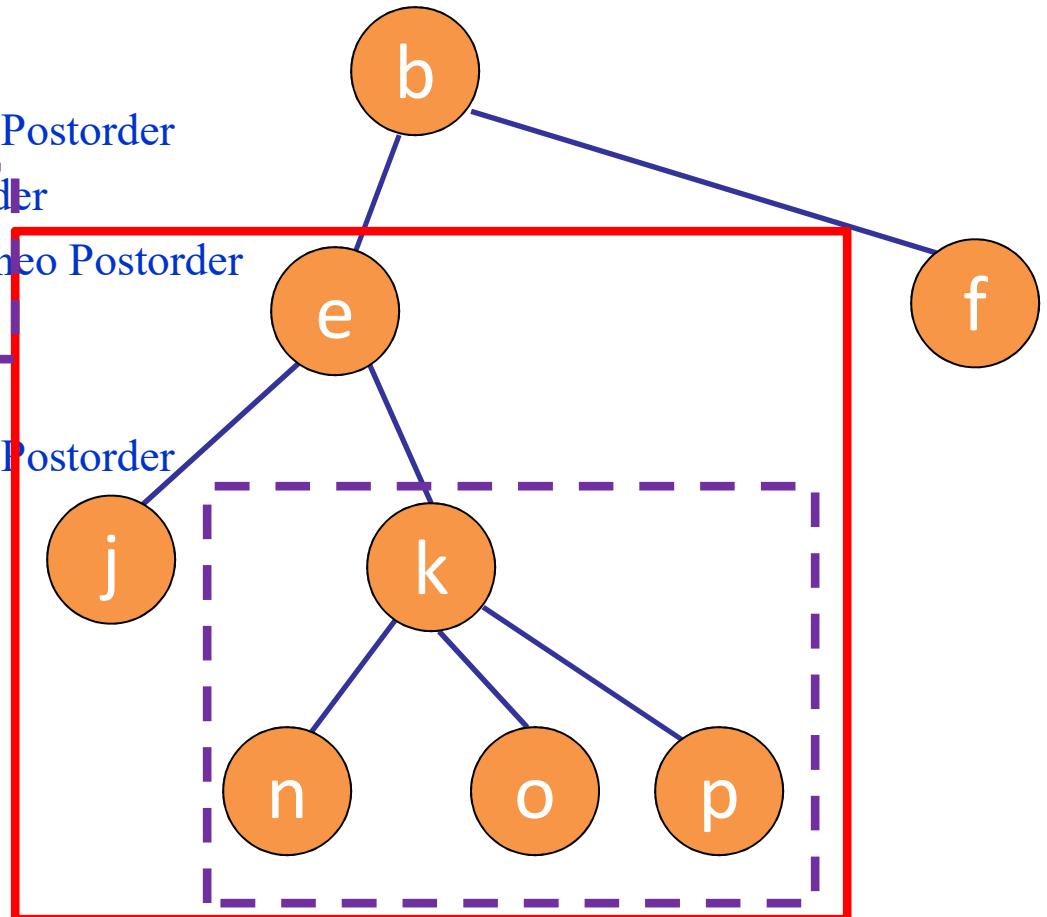
1. Thăm cây con trái nhất theo Postorder
 - 1.1. Thăm cây con trái nhất theo Postorder
 - 1.2. Thăm lần lượt các cây con còn lại theo Postorder
 - 1.3. Thăm gốc
2. Thăm lần lượt các cây con còn lại theo Postorder
3. Thăm gốc



j

PostOrder

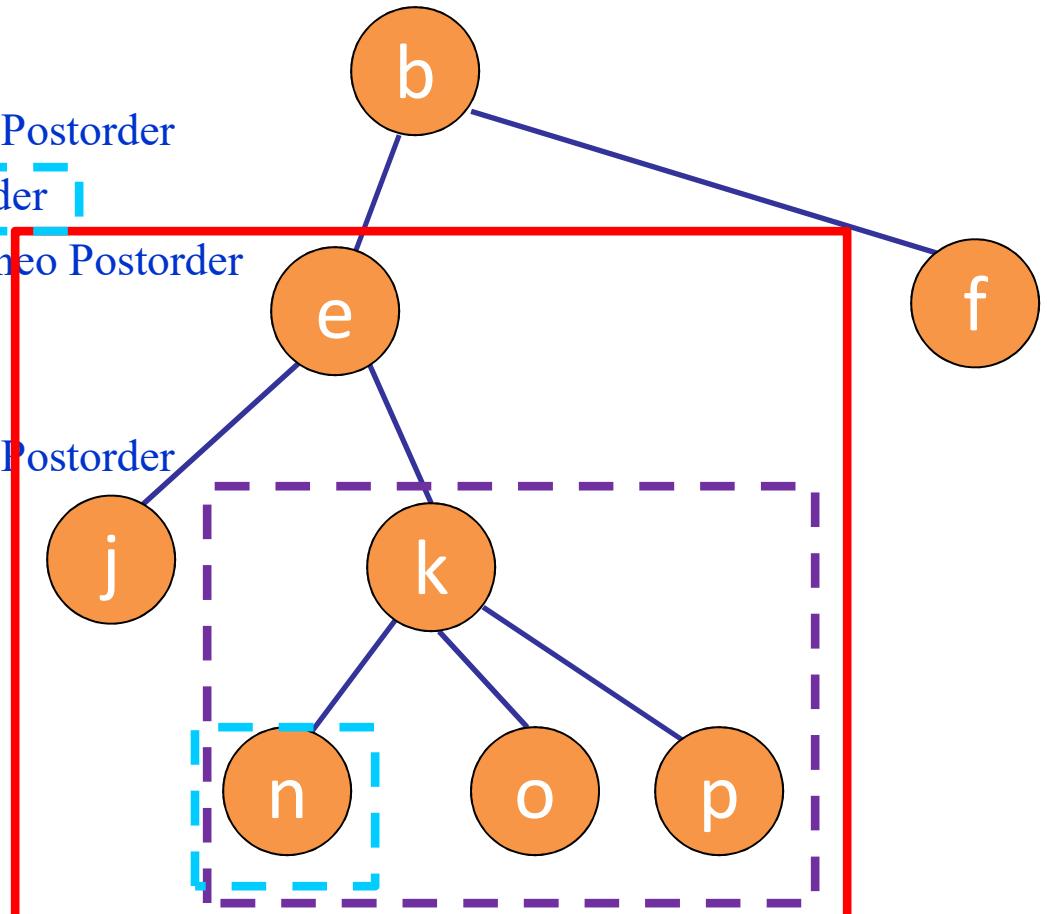
1. Thăm cây con trái nhất theo Postorder
 - 1.1. Thăm cây con trái nhất theo Postorder
 - 1.2. Thăm lần lượt các cây con còn lại theo Postorder
 - 1.2.1. Thăm cây con trái nhất theo Postorder
 - 1.2.2. Thăm lần lượt các cây con còn lại theo Postorder
 - 1.2.3. Thăm gốc
 - 1.3. Thăm gốc
2. Thăm lần lượt các cây con còn lại theo Postorder
3. Thăm gốc



j

PostOrder

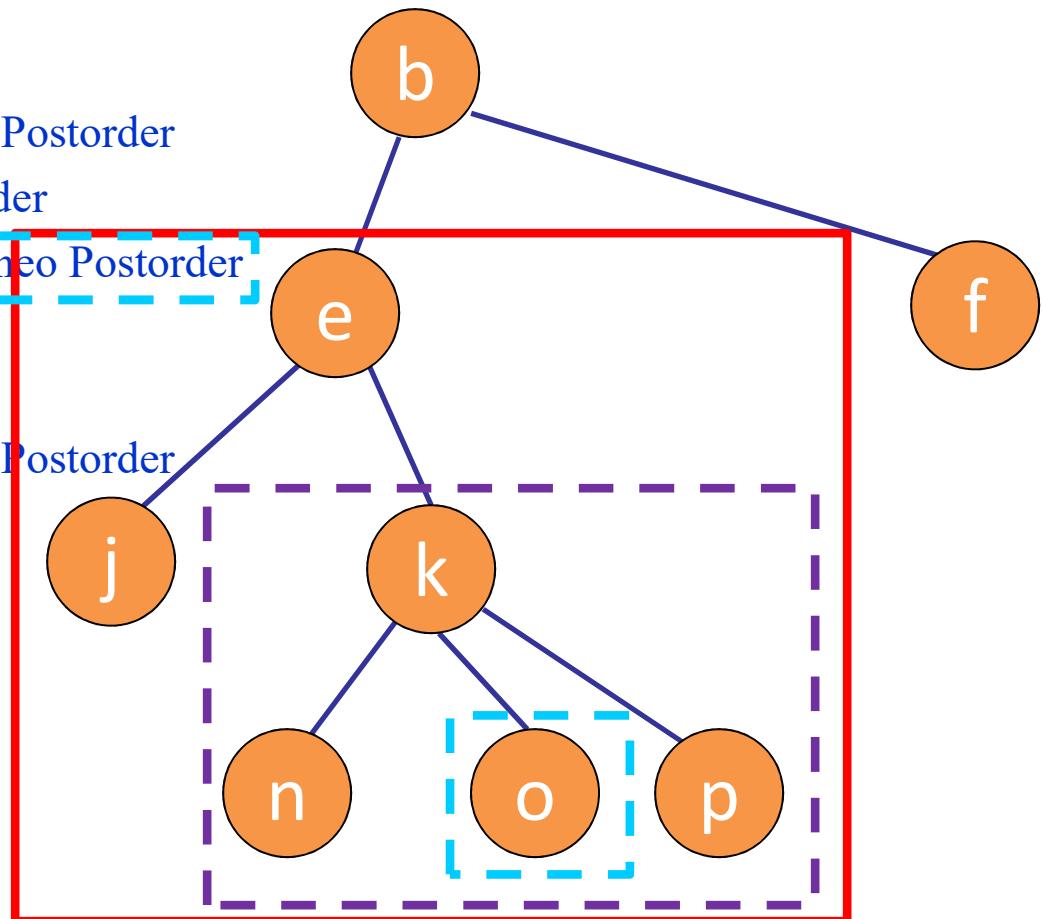
1. Thăm cây con trái nhất theo Postorder
 - 1.1. Thăm cây con trái nhất theo Postorder
 - 1.2. Thăm lần lượt các cây con còn lại theo Postorder
- 1.2.1. Thăm cây con trái nhất theo Postorder
- 1.2.2. Thăm lần lượt các cây con còn lại theo Postorder
- 1.2.3. Thăm gốc
- 1.3. Thăm gốc
2. Thăm lần lượt các cây con còn lại theo Postorder
3. Thăm gốc



j n

PostOrder

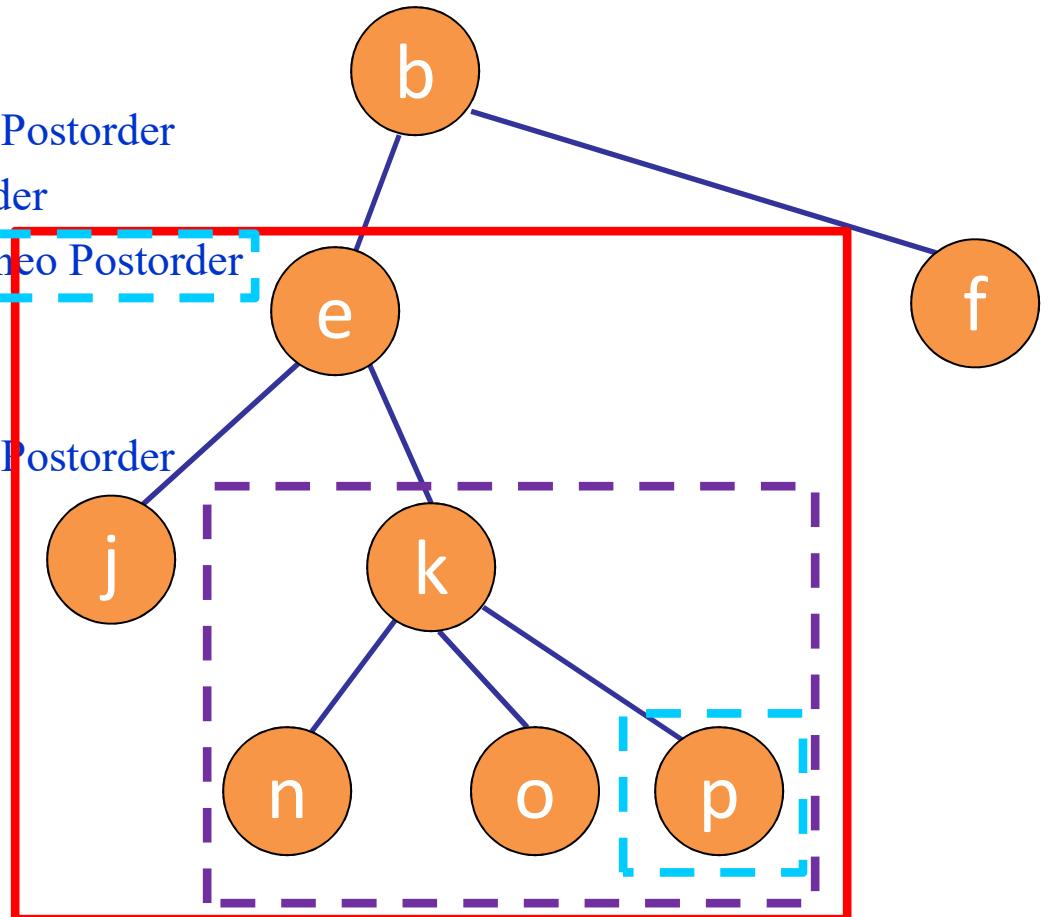
1. Thăm cây con trái nhất theo Postorder
 - 1.1. Thăm cây con trái nhất theo Postorder
 - 1.2. Thăm lần lượt các cây con còn lại theo Postorder
 - 1.2.1. Thăm cây con trái nhất theo Postorder
 - 1.2.2. Thăm lần lượt các cây con còn lại theo Postorder
 - 1.2.3. Thăm gốc
 - 1.3. Thăm gốc
2. Thăm lần lượt các cây con còn lại theo Postorder
3. Thăm gốc



j n o

PostOrder

1. Thăm cây con trái nhất theo Postorder
 - 1.1. Thăm cây con trái nhất theo Postorder
 - 1.2. Thăm lần lượt các cây con còn lại theo Postorder
 - 1.2.1. Thăm cây con trái nhất theo Postorder
 - 1.2.2. Thăm lần lượt các cây con còn lại theo Postorder
 - 1.2.3. Thăm gốc
 - 1.3. Thăm gốc
2. Thăm lần lượt các cây con còn lại theo Postorder
3. Thăm gốc



j n o p

PostOrder

1. Thăm cây con trái nhất theo Postorder

1.1. Thăm cây con trái nhất theo Postorder

1.2. Thăm lần lượt các cây con còn lại theo Postorder

1.2.1. Thăm cây con trái nhất theo Postorder

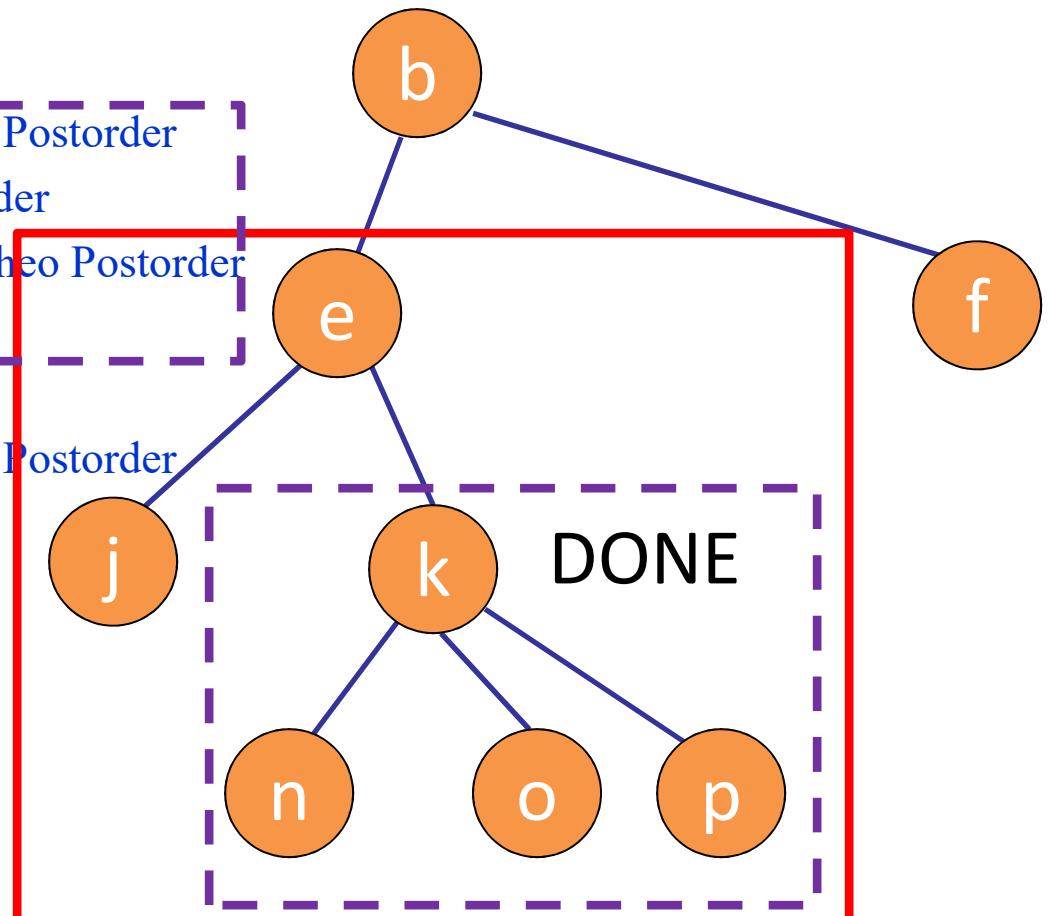
1.2.2. Thăm lần lượt các cây con còn lại theo Postorder

1.2.3. Thăm gốc

1.3. Thăm gốc

2. Thăm lần lượt các cây con còn lại theo Postorder

3. Thăm gốc

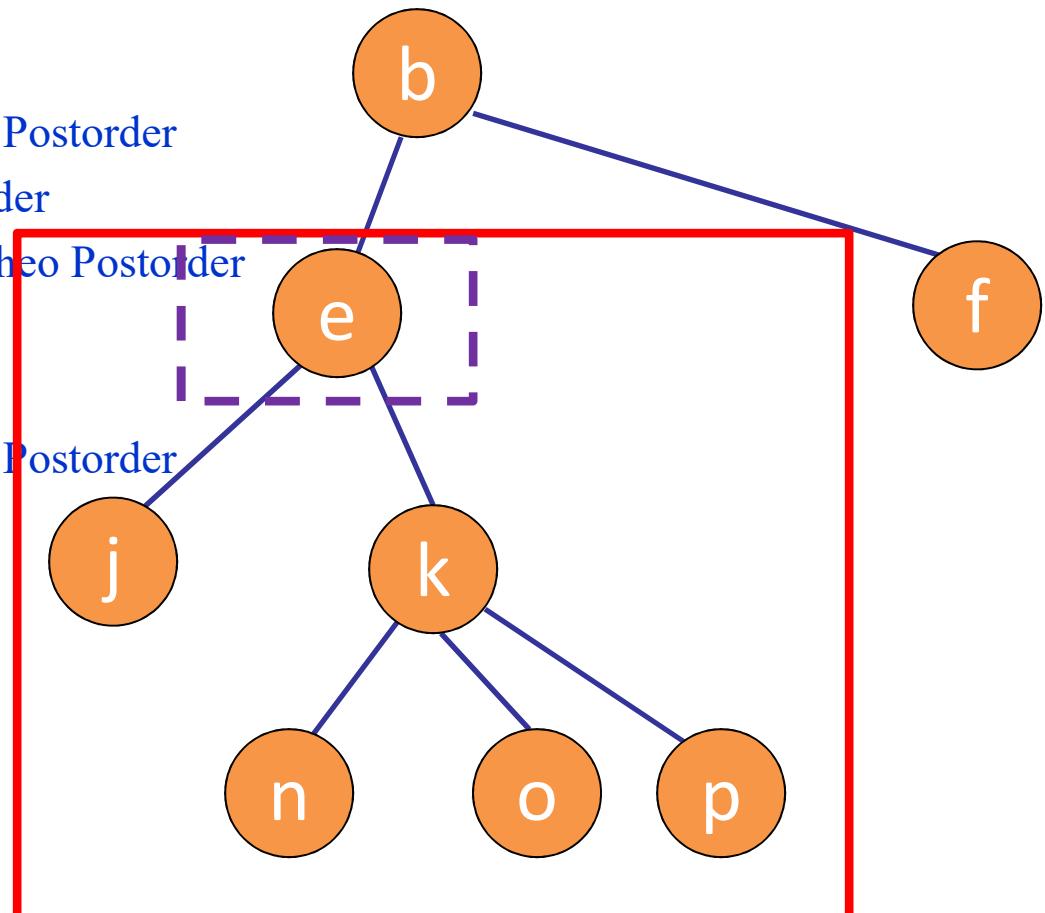


j n o p k

NGUYỄN KHÁNH PHƯƠNG
Bộ môn KHMT – ĐHBK HN

PostOrder

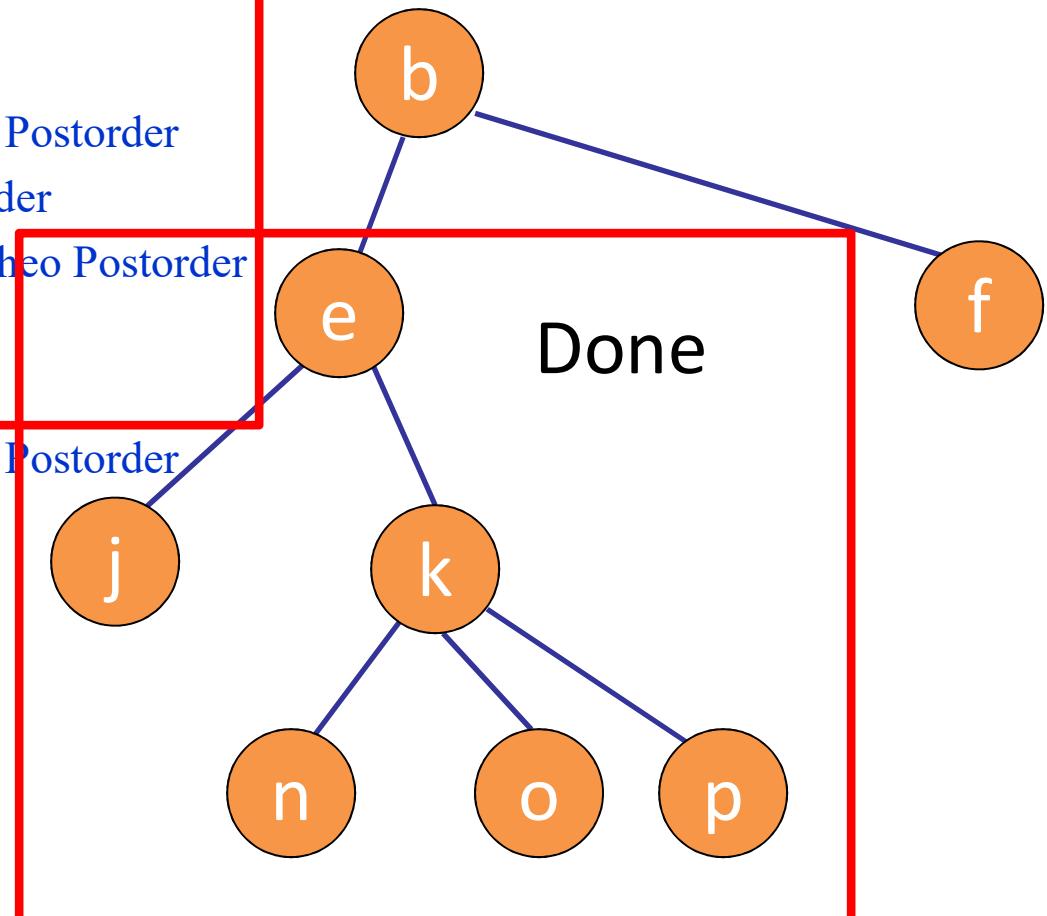
1. Thăm cây con trái nhất theo Postorder
 - 1.1. Thăm cây con trái nhất theo Postorder
 - 1.2. Thăm lần lượt các cây con còn lại theo Postorder
 - 1.2.1. Thăm cây con trái nhất theo Postorder
 - 1.2.2. Thăm lần lượt các cây con còn lại theo Postorder
 - 1.2.3. Thăm gốc
- 1.3. Thăm gốc
2. Thăm lần lượt các cây con còn lại theo Postorder
3. Thăm gốc



j n o p k e

PostOrder

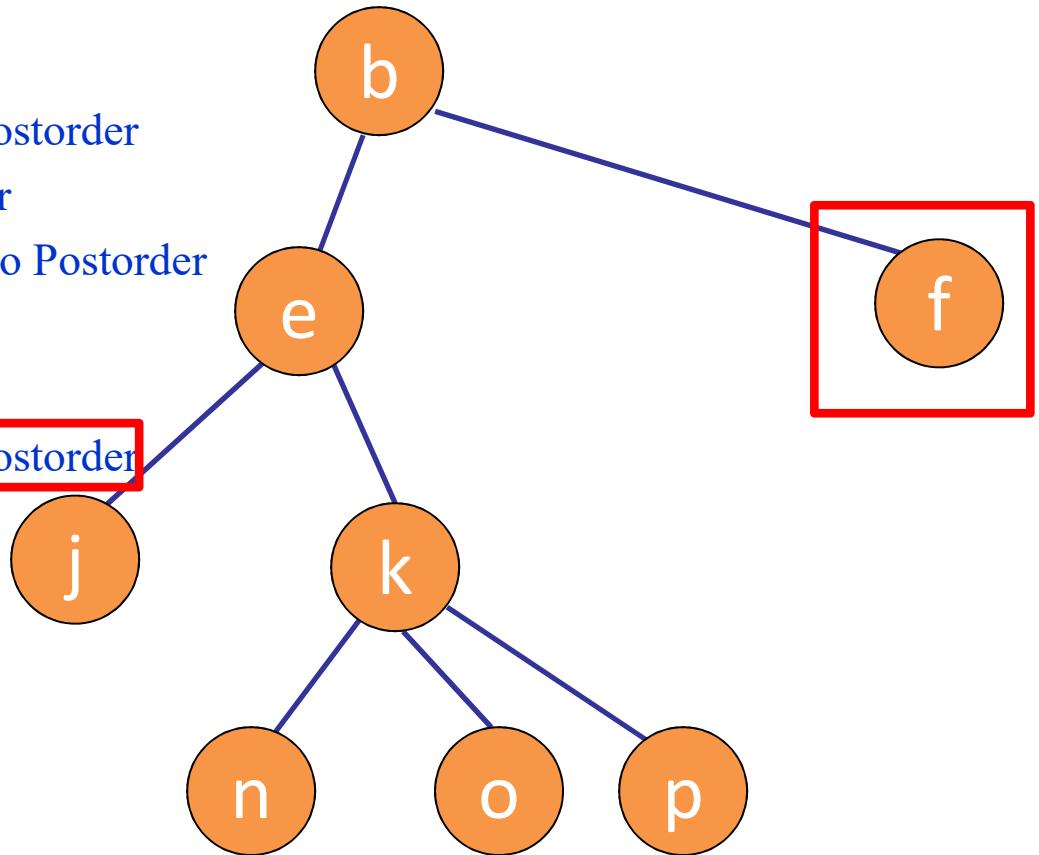
1. Thăm cây con trái nhất theo Postorder
 - 1.1. Thăm cây con trái nhất theo Postorder
 - 1.2. Thăm lần lượt các cây con còn lại theo Postorder
 - 1.2.1. Thăm cây con trái nhất theo Postorder
 - 1.2.2. Thăm lần lượt các cây con còn lại theo Postorder
 - 1.2.3. Thăm gốc
 - 1.3. Thăm gốc
2. Thăm lần lượt các cây con còn lại theo Postorder
3. Thăm gốc



j n o p k e

PostOrder

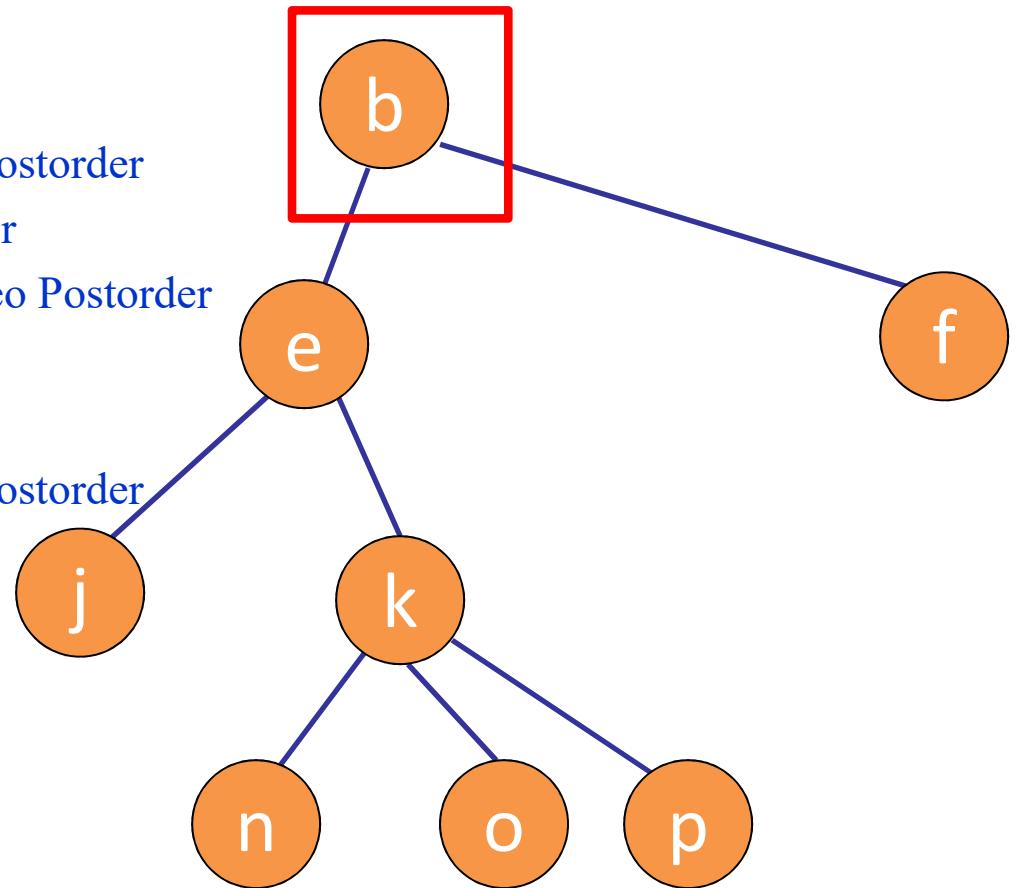
1. Thăm cây con trái nhất theo Postorder
 - 1.1. Thăm cây con trái nhất theo Postorder
 - 1.2. Thăm lần lượt các cây con còn lại theo Postorder
 - 1.2.1. Thăm cây con trái nhất theo Postorder
 - 1.2.2. Thăm lần lượt các cây con còn lại theo Postorder
 - 1.2.3. Thăm gốc
 - 1.3. Thăm gốc
2. Thăm lần lượt các cây con còn lại theo Postorder
3. Thăm gốc



j n o p k e f

PostOrder

1. Thăm cây con trái nhất theo Postorder
 - 1.1. Thăm cây con trái nhất theo Postorder
 - 1.2. Thăm lần lượt các cây con còn lại theo Postorder
 - 1.2.1. Thăm cây con trái nhất theo Postorder
 - 1.2.2. Thăm lần lượt các cây con còn lại theo Postorder
 - 1.2.3. Thăm gốc
 - 1.3. Thăm gốc
2. Thăm lần lượt các cây con còn lại theo Postorder
3. Thăm gốc

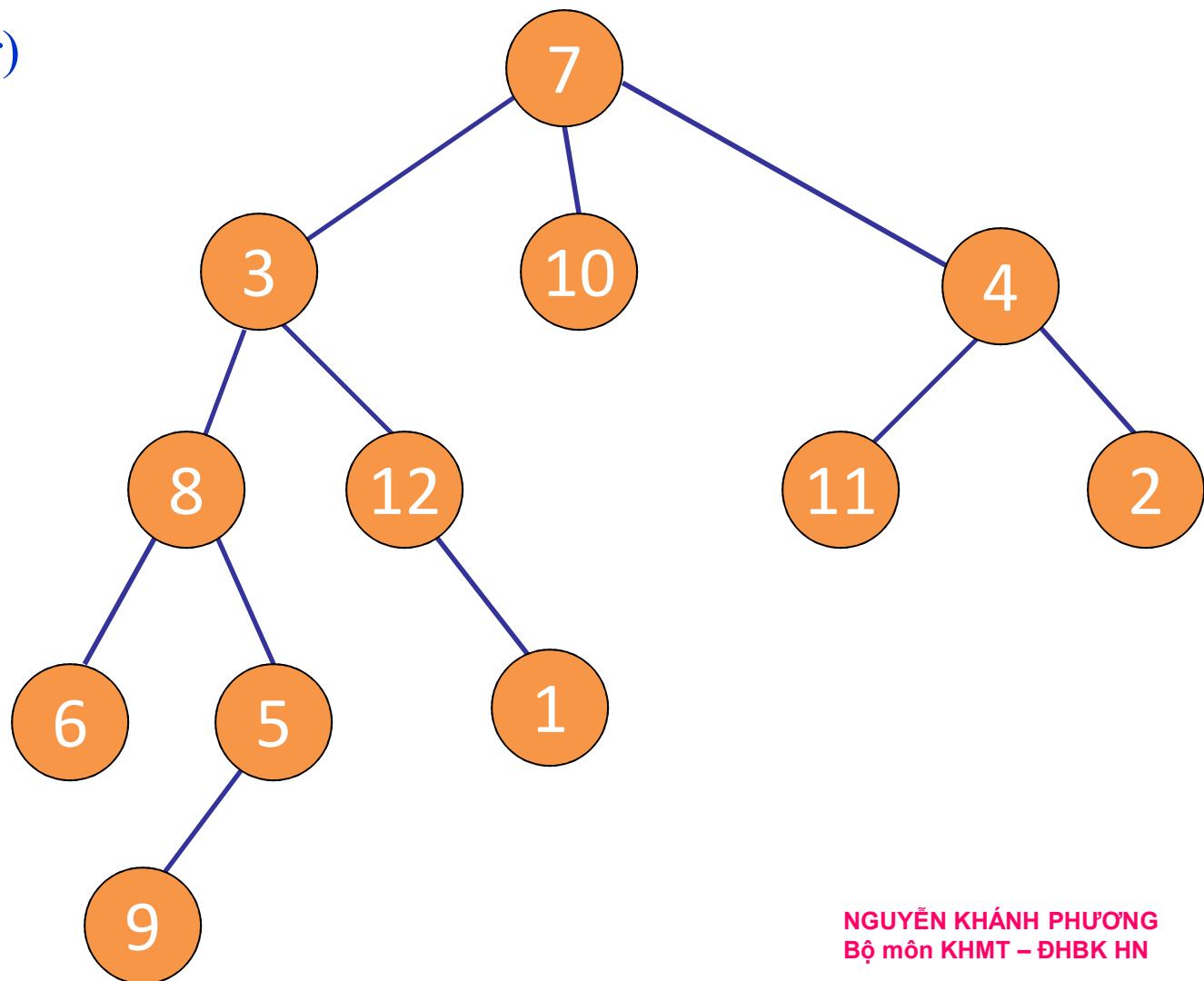


j n o p k e f b

Ví dụ 1: Duyệt cây

Cho cây có thứ tự T có nút gốc 7. Hãy đưa ra thứ tự các nút được thăm nếu ta duyệt cây T theo thứ tự:

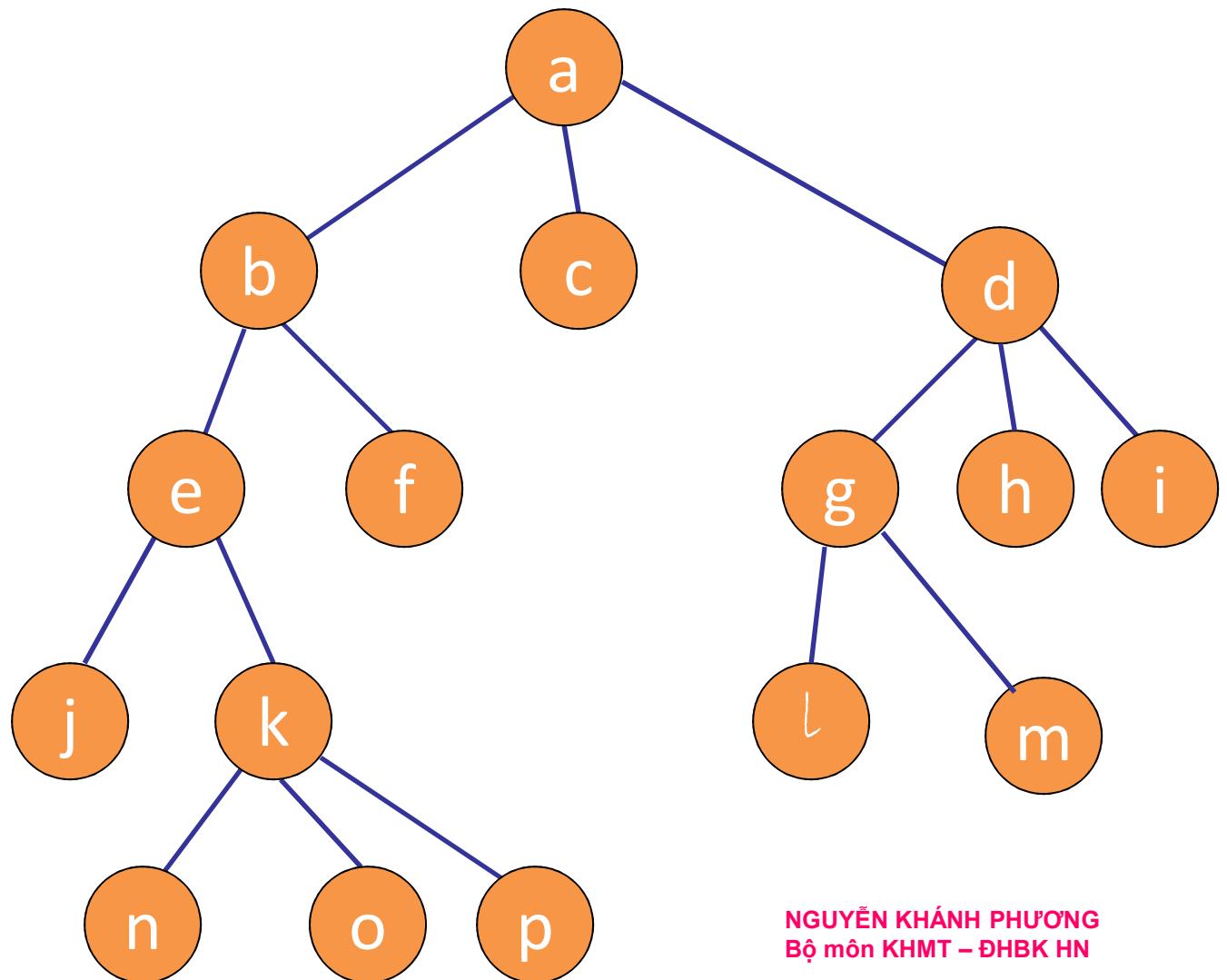
- Trước (Preorder)
- Giữa (Inorder)
- Sau (Postorder)



Ví dụ 2: Duyệt cây

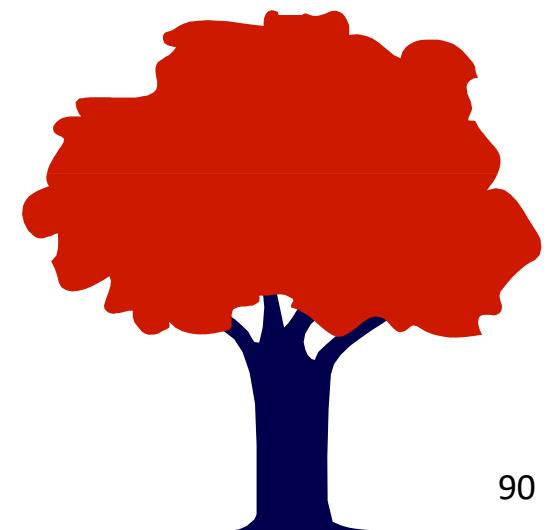
Cho cây có thứ tự T có nút gốc a. Hãy đưa ra thứ tự các nút được thăm nếu ta duyệt cây T theo thứ tự:

- Trước (Preorder)
- Giữa (Inorder)
- Sau (Postorder)



Nội dung

1. Định nghĩa và các khái niệm
2. Biểu diễn cây
3. Duyệt cây
- 4. Cây nhị phân (Binary tree)**



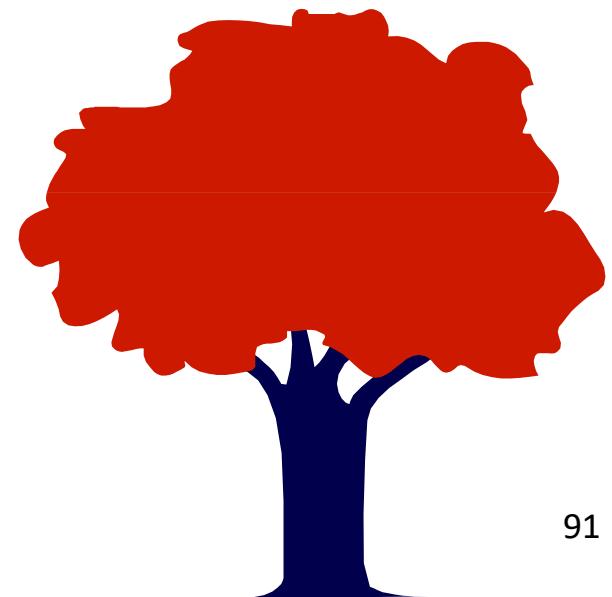
4. Cây nhị phân (Binary tree)

4.1. Định nghĩa và tính chất

4.2. Biểu diễn cây nhị phân

4.3. Duyệt cây nhị phân

4.4. Một số ứng dụng



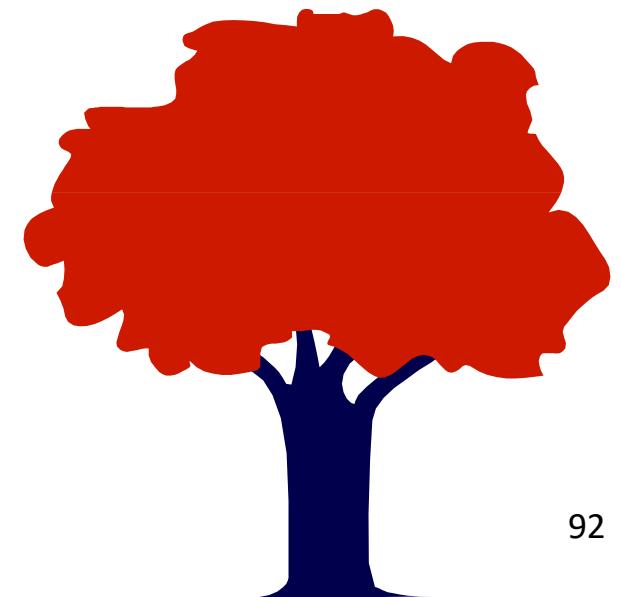
4. Cây nhị phân (Binary tree)

4.1. Định nghĩa và tính chất

4.2. Biểu diễn cây nhị phân

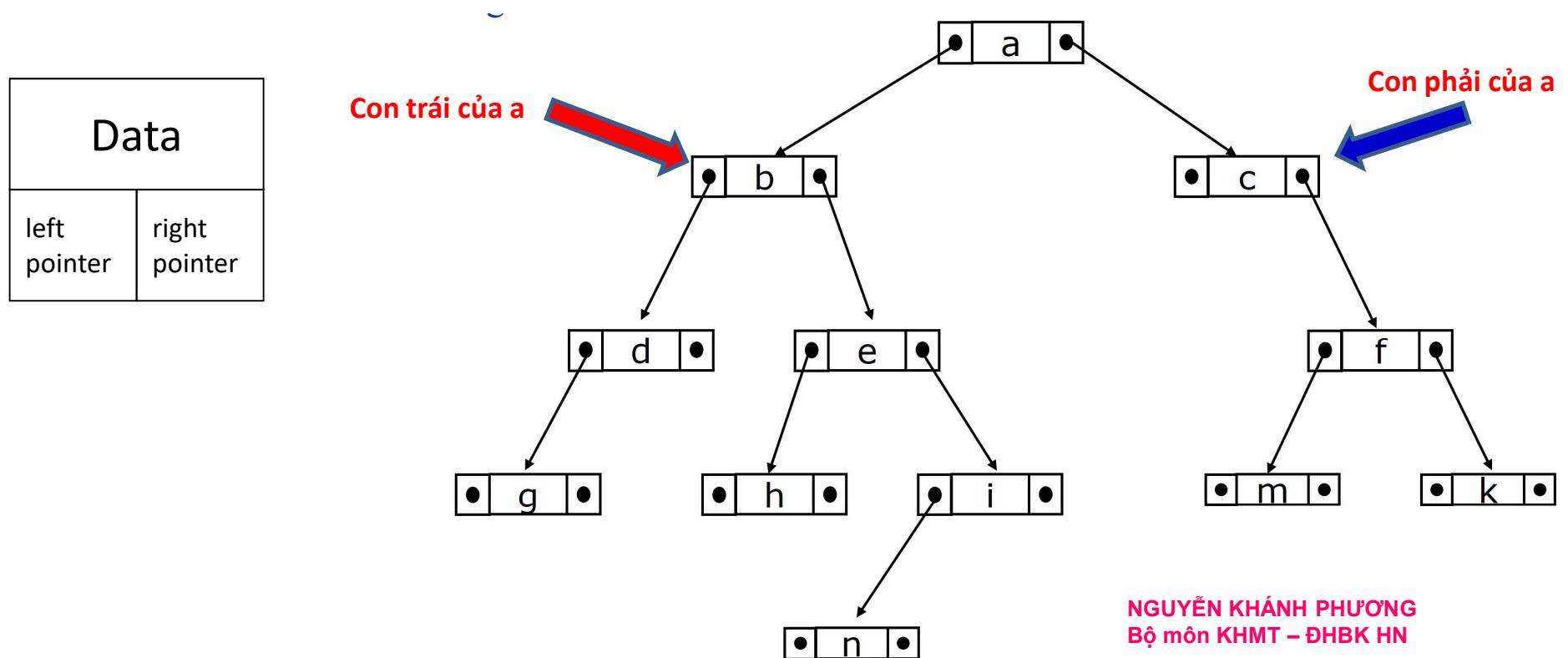
4.3. Duyệt cây nhị phân

4.4. Một số ứng dụng

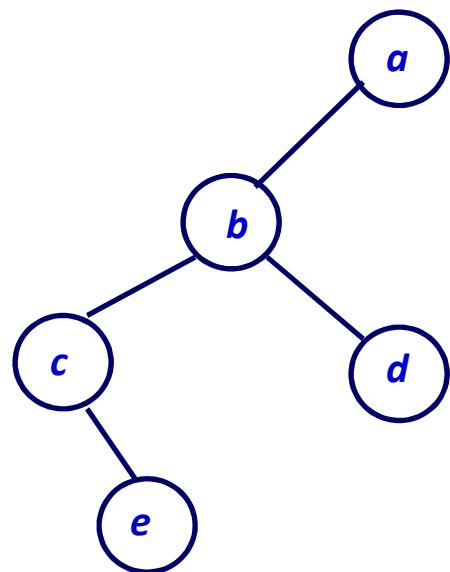


4.1. Định nghĩa: Binary tree

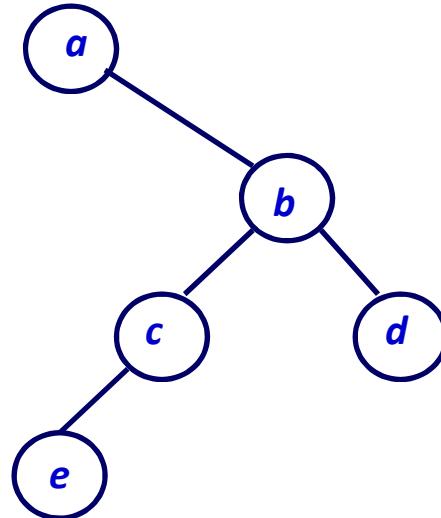
- Cây nhị phân là cây mà mỗi nút có nhiều nhất là hai con.
- ➔ Mỗi nút trên cây hoặc là: (1) không có con nào, (2) hoặc chỉ có con trái, (3) hoặc chỉ có con phải, (4) hoặc có cả con trái và con phải
- Mỗi nút đều có dữ liệu (data), một con trỏ trỏ đến con trái và một con trỏ trỏ đến con phải.



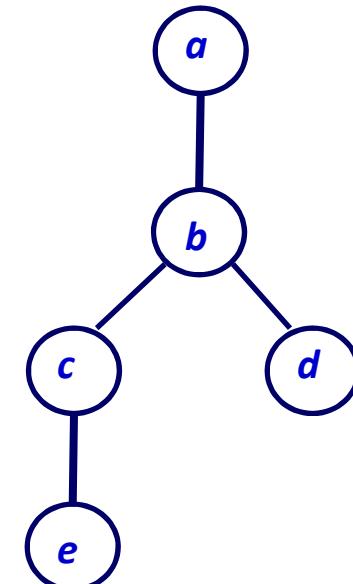
Chú ý



Cây nhị phân T_1



Cây nhị phân T_2



Cây tổng quát

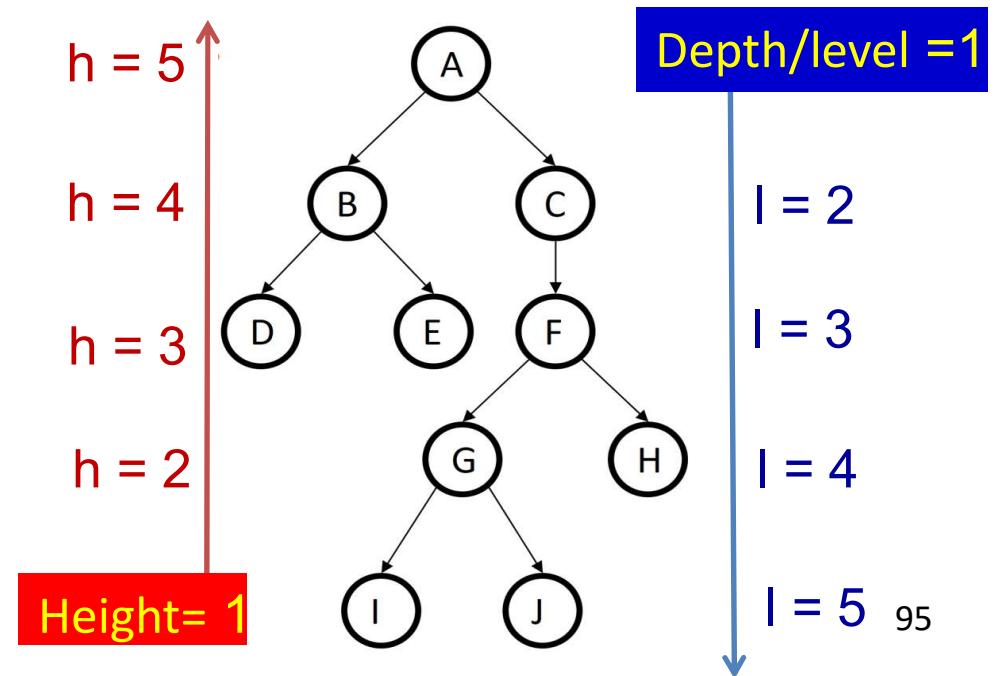
$$T_1 \neq T_2$$

Tính chất của cây nhị phân

$$\begin{aligned} \text{height}(cây) &= \text{depth}(cây) \\ &= \text{MAX } \{\text{depth(lá)}\} = \text{height(gốc)} \end{aligned}$$

Bố đề 1:

- 1) max # số nút trên mức $i = 2^{i-1}$ ($i \geq 1$)
 - 1) max # số lá = $2^{\text{depth(tree)}-1}$
- 2) max # số nút = $2^{\text{depth(tree)}} - 1$
- 3) Cây nhị phân có n nút: $\text{depth(cây)} \geq \lceil \log_2(n) \rceil$



Tính chất của cây nhị phân

Bố đề 1:

- 1) $\max \#$ số nút trên mức $i = 2^{i-1}$ ($i \geq 1$) : Mức i của cây nhị phân có số nút nhiều nhất là 2^{i-1}

Chứng minh: Bằng qui nạp theo i .

- **Cơ sở:** Gốc là nút duy nhất trên mức $i=1$. Như vậy số nút lớn nhất trên mức $i=1$ là $2^0 = 2^{i-1}$.
- **Chuyển qui nạp:** Giả sử với mọi j , $1 \leq j < i$: số nút lớn nhất trên mức j là 2^{j-1} .
 - Do số nút trên mức $i-1$ là 2^{i-2} , mặt khác theo định nghĩa mỗi nút trên cây nhị phân có không quá 2 con

$$\Rightarrow \text{số lượng nút lớn nhất trên mức } i \leq 2 * \underbrace{\text{số lượng nút trên mức } i-1}_{\frac{2 * 2^{i-2}}{2^{i-1}}}$$

Tính chất của cây nhị phân

Bố đề 1:

- 2) $\max \# \text{ số nút} = 2^{\text{depth(tree)}} - 1$: cây nhị phân với chiều cao k có số nút nhiều nhất là $2^k - 1$, $k \geq 1$.

Chứng minh:

Theo 1) có: Mức i của cây nhị phân có số nút nhiều nhất là 2^{i-1} ($i \geq 1$)

Cây nhị phân chiều cao $k \rightarrow$ có k mức

\rightarrow Tổng số nút trên cây nhều nhất chỉ có thể là

$$\sum_{i=1}^k 2^{i-1} = 2^k - 1.$$

Tính chất của cây nhị phân

Bố đề 1:

3) Cây nhị phân có n nút: $\text{depth}(\text{cây}) \geq \lceil \log_2(n) \rceil$

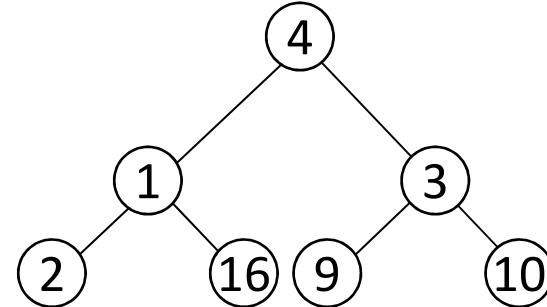
Chứng minh:

- Cây nhị phân n nút có chiều cao thấp nhất k khi số lượng nút ở các mức $i = 1, 2, \dots, k$ đều là lớn nhất có thể được. Từ đó ta có:

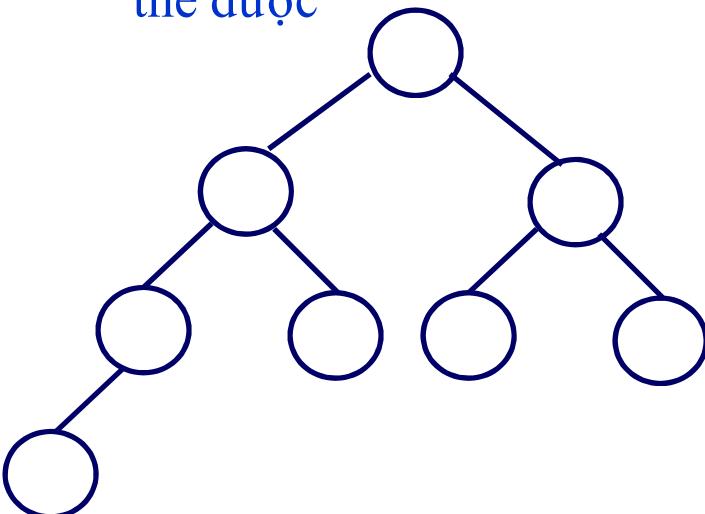
$$n = \sum_{i=1}^k 2^{i-1} = 2^k - 1, \text{ suy ra } 2^k = n + 1, \text{ hay } k = \lceil \log_2(n+1) \rceil.$$

Cây nhị phân **đầy đủ** (Full binary tree) và Cây nhị phân **hoàn chỉnh** (Complete binary tree)

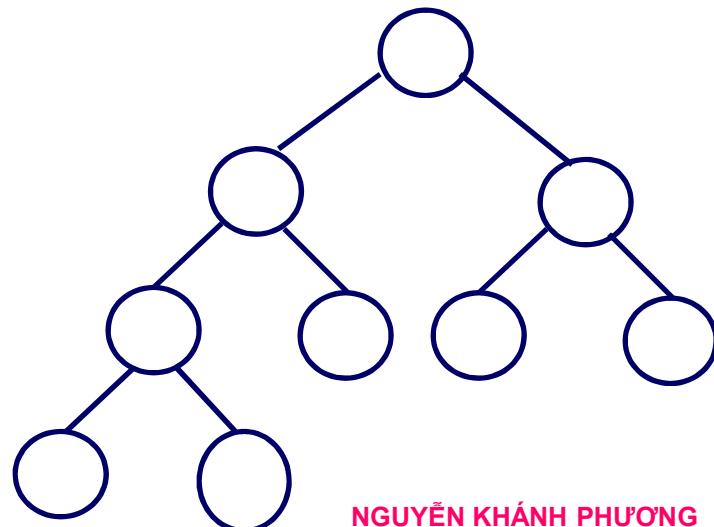
- Cây nhị phân **đầy đủ**: là cây nhị phân thỏa mãn
 - Mỗi nút trong đều có 2 con,
 - Mọi nút lá đều có cùng độ sâu



- Cây nhị phân **hoàn chỉnh**: là cây nhị phân thỏa mãn
 - Tất cả các mức (có thể không kể đến mức sâu nhất) đều đầy
 - Nếu mức sâu nhất không đầy, thì tất cả các nút ở mức này là lệch sang trái nhất có thể được



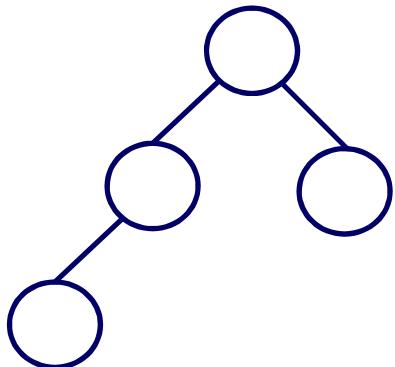
Complete binary tree



Tìm cây nhị phân đầy đủ / cây nhị phân hoàn chỉnh

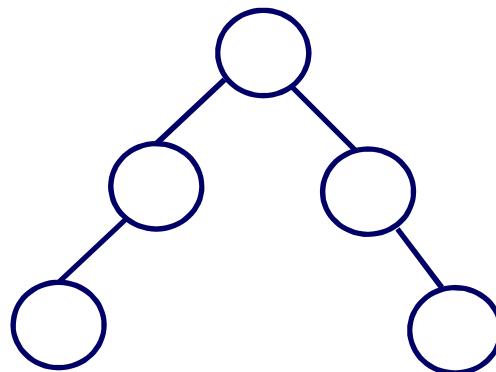
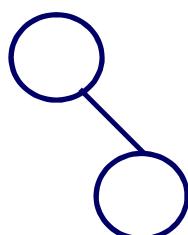
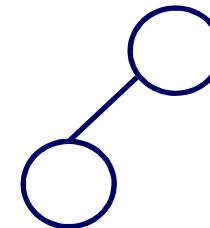
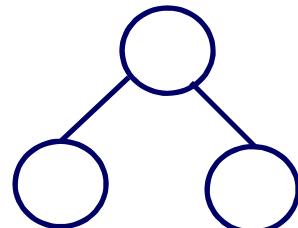
Cây nhị phân **đầy đủ**: là cây nhị phân

- Mỗi nút trong đều có 2 con,
- Mọi nút lá đều có cùng độ sâu



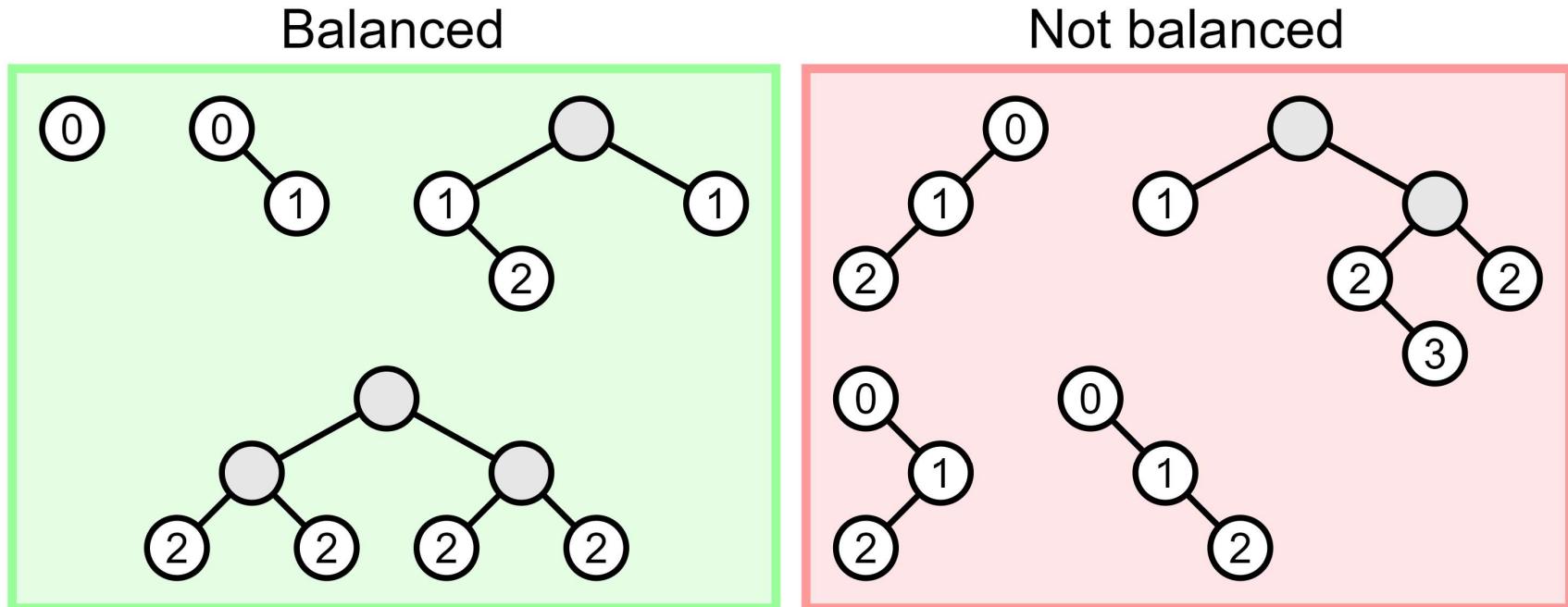
Cây nhị phân **hoàn chỉnh**: là cây nhị phân độ sâu n thỏa mãn

- là cây nhị phân đầy đủ nếu không tính đến các nút ở độ sâu n , và
- tất cả các nút ở độ sâu n là lệch sang trái nhất có thể được.



Cây nhị phân cân đối (balanced binary tree)

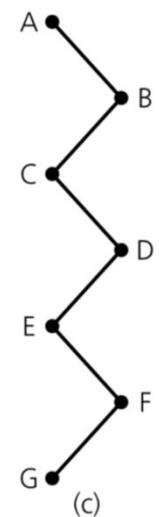
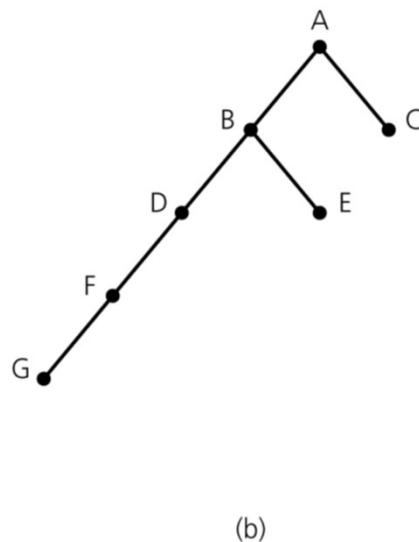
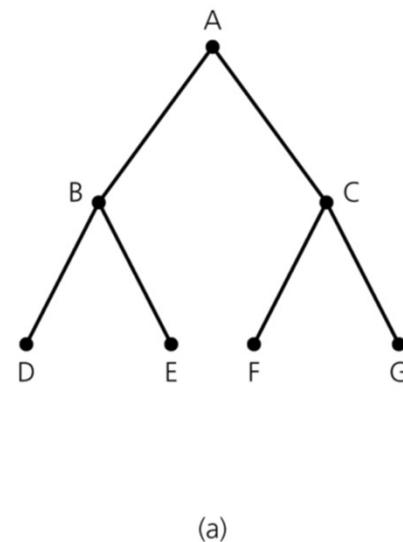
- **Định nghĩa.** Cây nhị phân được gọi là *cân đối* (*balanced*) nếu chiều cao của cây con trái và chiều cao của cây con phải của mọi nút chênh lệch nhau không quá 1 đơn vị.



Cây nhị phân cân đối (balanced binary tree)

- **Định nghĩa.** Cây nhị phân được gọi là *cân đối* (*balanced*) nếu chiều cao của cây con trái và chiều cao của cây con phải của mọi nút chênh lệch nhau không quá 1 đơn vị.
- Nhận xét:
 - Nếu cây nhị phân là đầy đủ thì nó là hoàn chỉnh
 - Nếu cây nhị phân là hoàn chỉnh thì nó là cân đối

Ví dụ:



1. Cây nào là đầy đủ?
2. Cây nào là hoàn chỉnh?
3. Cây nào là cân đối?

True or False?

- a) Nếu cây nhị phân là đầy đủ thì nó là cân đối
- b) Nếu cây nhị phân là hoàn chỉnh thì nó là đầy đủ
- c) Nếu cây nhị phân là cân đối thì nó là hoàn chỉnh
- d) Nếu cây nhị phân là cân đối thì nó là đầy đủ

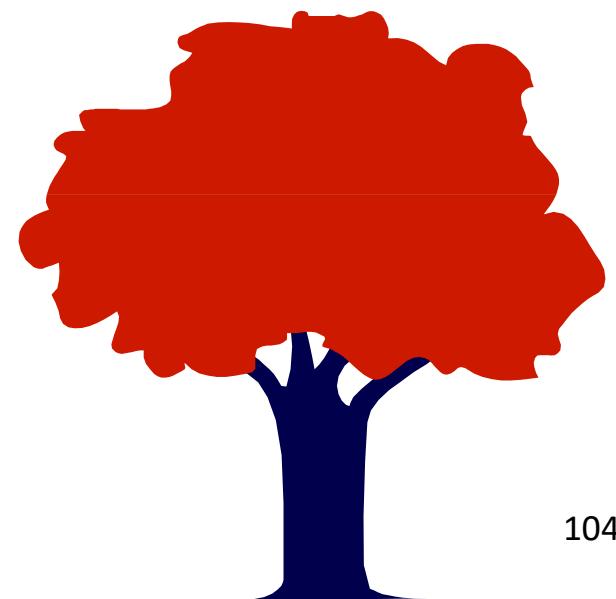
4. Cây nhị phân (Binary tree)

4.1. Định nghĩa và tính chất

4.2. Biểu diễn cây nhị phân

4.3. Duyệt cây nhị phân

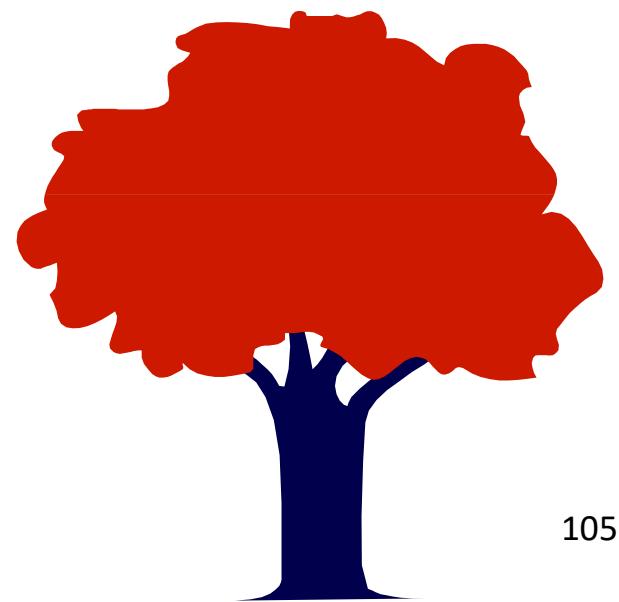
4.4. Một số ứng dụng



4.2. Biểu diễn cây nhị phân

4.2.1. Mảng

4.2.2. Con trỏ



Mảng 1 chiều biểu diễn cây nhị phân

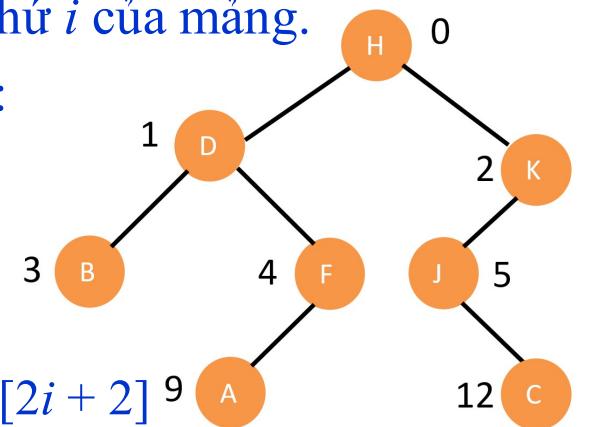
Các nút trên cây được đánh chỉ số như sau:

- Chỉ số 0: nút gốc,
- Tất cả các nút còn lại được đánh theo thứ tự từ trái sang phải, từ trên xuống dưới.
Các nút rỗng cũng được đánh số.

Khi đó, nút được đánh chỉ số i sẽ được lưu vào thành phần thứ i của mảng.

→ Mảng 1 chiều **A** dùng để biểu diễn cây nhị phân như sau:

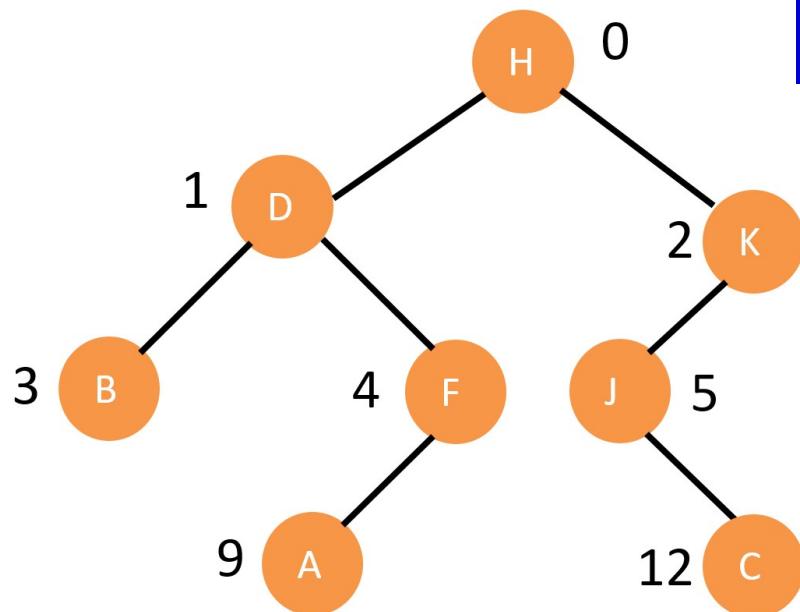
- Gốc của cây là $A[0]$
- Với mỗi nút tại $A[i]$:
 - Nút cha của nó là $A[(i-1)/2]$
 - Nút con trái của nó là $A[2i + 1]$ và con phải của nó là $A[2i + 2]$
 - Nếu nút không có con nào, thì các giá trị null sẽ được lưu tại các thành phần tương ứng trên mảng.



H	D	K	B	F	J	NULL	NULL	NULL	A	NUL	NUL	c	NULL	25	26	30		
A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	25	26	30

Con phải của K
Con trái của B
Con phải của B
Con trái của J
Con trái của C
Con phải của C

Mảng 1 chiều biểu diễn cây nhị phân



Depth/level = 1

Max #nút = $2^0=1$

$l = 2$

Max #nút = $2^1=2$

$l = 3$

Max #nút = $2^2=4$

$l = 4$

Max #nút = $2^3=8$

Max #núts = $2^4=16$

$$\sum \text{max#nút} = 31$$

Với mỗi nút tại $A[i]$:

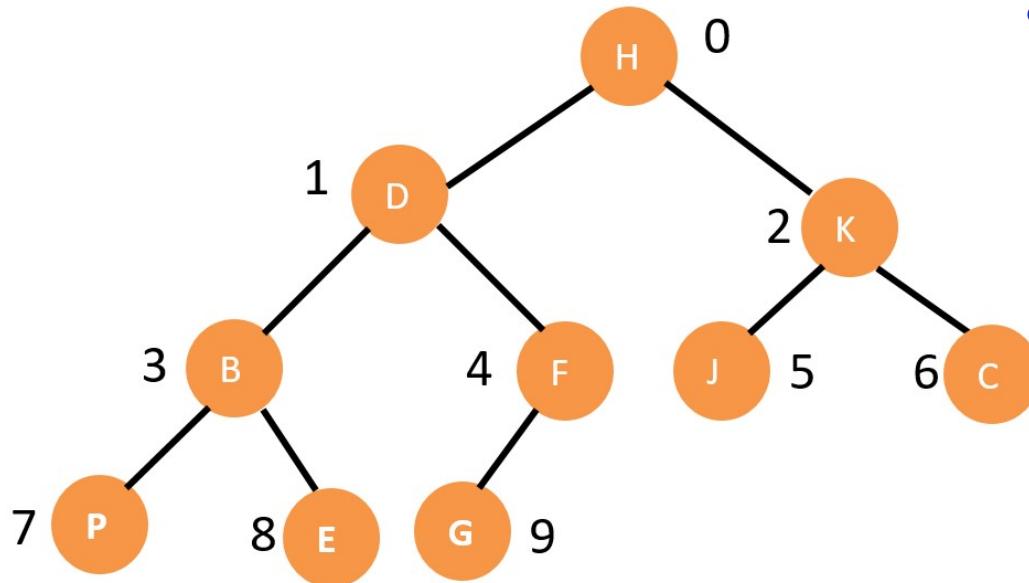
- Nút cha của nó là $A[(i-1)/2]$
- Nút con trái của nó là $A[2i + 1]$ và con phải của nó là $A[2i + 2]$

	H	D	K	B	F	J	NUL	NUL	NUL	A	NUL	NUL	C	NUL	25	26	30
A	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14			

Annotations below the array:

- Right child of K (points to index 6)
- Left child of B (points to index 7)
- Right child of B (points to index 8)
- left child of J (points to index 11)
- Left child of C (points to index 12)
- Right child of C (points to index 13)

Mảng 1 chiều biểu diễn cây nhị phân hoàn chỉnh



Cây nhị phân **hoàn chỉnh**: là cây nhị phân thỏa mãn

- Tất cả các mức (có thể không kề đến mức sâu nhất) đều đầy
- Nếu mức sâu nhất không đầy, thì tất cả các nút ở mức này là lệch sang trái nhất có thể được

Với mỗi nút tại $A[i]$:

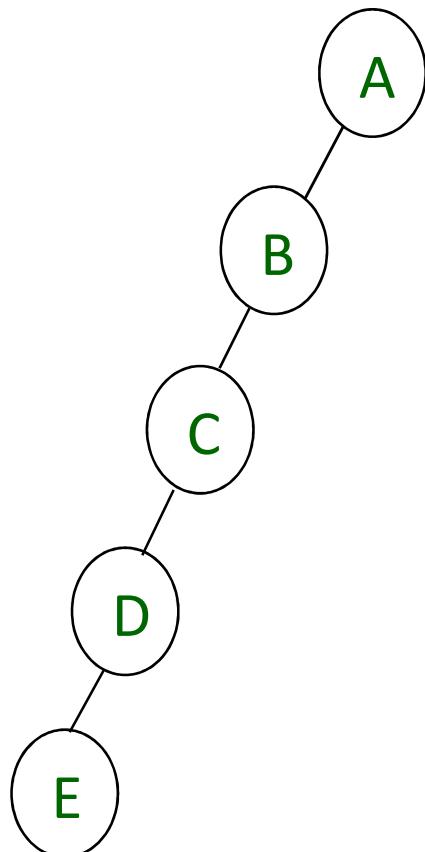
- Nút cha của nó là $A[(i-1)/2]$
- Nút con trái của nó là $A[2i + 1]$ và con phải của nó là $A[2i + 2]$

H	D	K	B	F	J	C	P	E	G
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

Sử dụng mảng 1 chiều biểu diễn cây nhị phân hoàn chỉnh n nút:

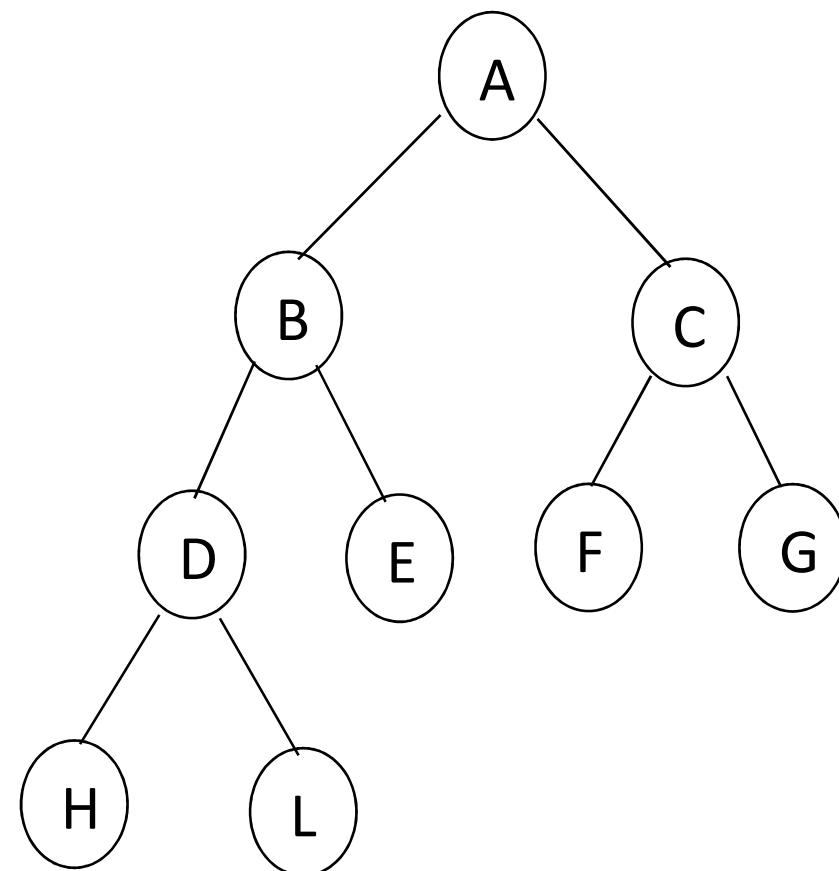
- Ta chỉ cần mảng 1 chiều gồm n phần tử
- Kiểm tra $A[i]$ có phải là nút lá hay không ?

Mảng 1 chiều biểu diễn cây nhị phân



[0]	A
[1]	B
[2]	NULL
[3]	C
[4]	NULL
[5]	NULL
[6]	NULL
[7]	D
[8]	NULL
[9]	E
[15]	F

(1) Lãng phí bộ nhớ
(2) thêm/xóa phức tạp

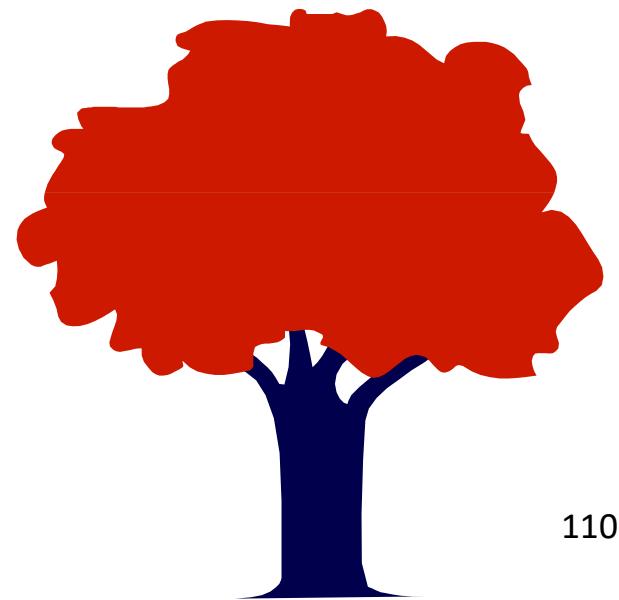


[0]	A
[1]	B
[2]	C
[3]	D
[4]	E
[5]	F
[6]	G
[7]	H
[8]	L

4.2. Biểu diễn cây nhị phân

4.2.1. Mảng

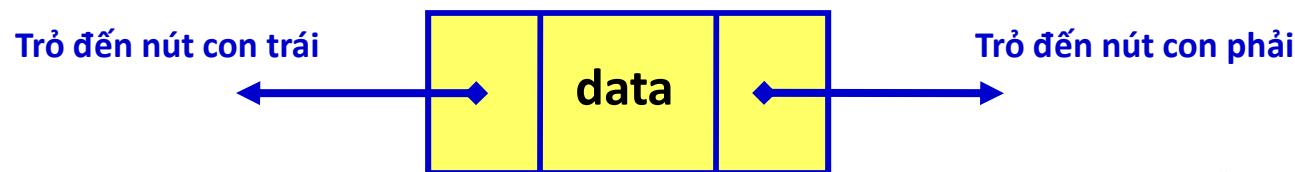
4.2.2. Con trỏ



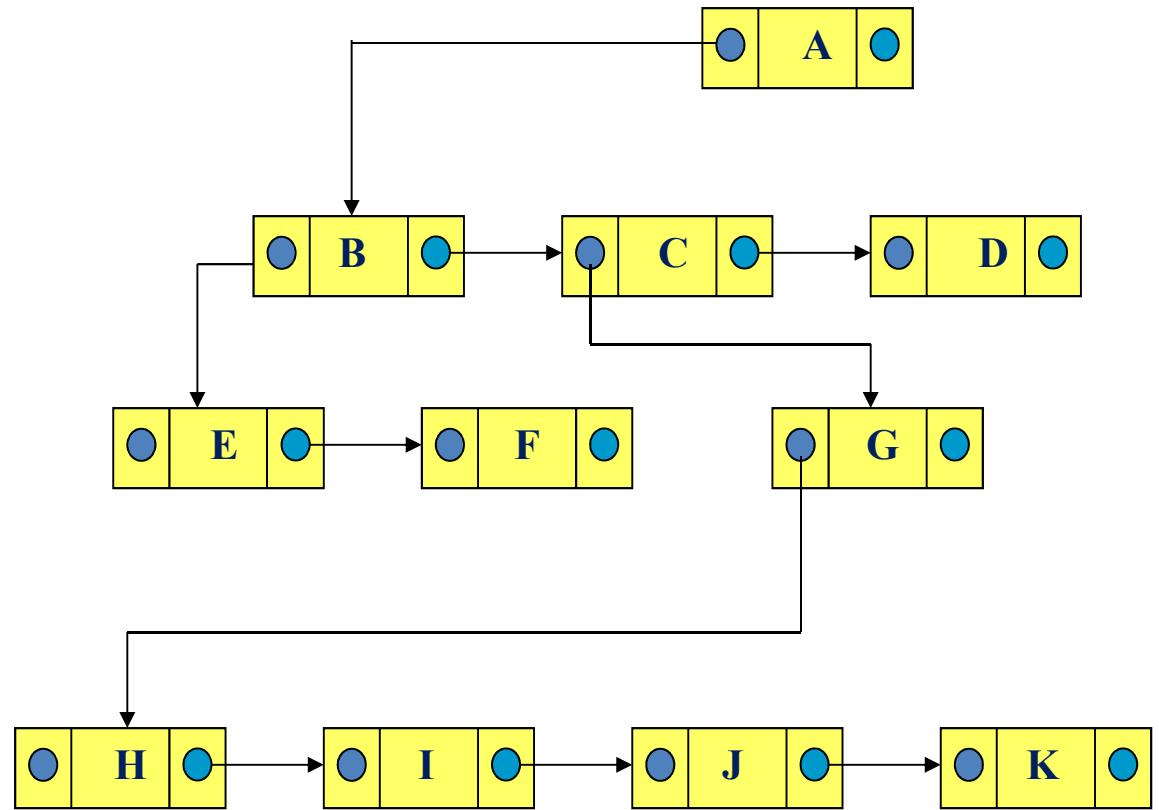
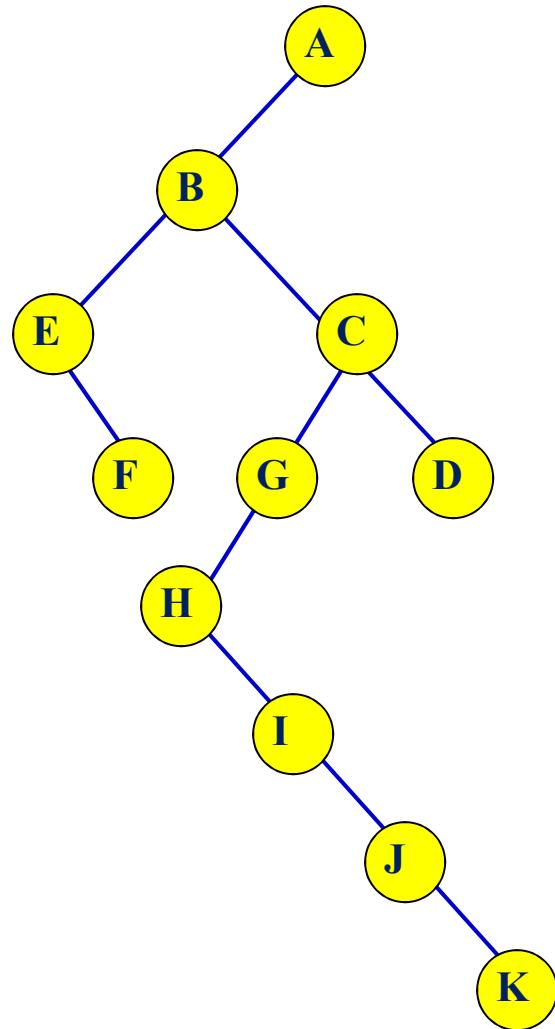
Sử dụng con trỏ để biểu diễn cây nhị phân

- Mỗi nút của cây sẽ có con trỏ trỏ đến con trái và con trỏ trỏ đến con phải.
- Nếu nút không có con trái hoặc con phải, thì con trỏ trỏ đầy con tương ứng sẽ chứa giá trị NULL.
- Nút lá con trỏ trái và con trỏ phải đều = NULL vì nó không có con nào.
- Cấu trúc một nút trên cây nhị phân được định nghĩa trên ngôn ngữ C như sau:

```
typedef struct
{
    DataType data; /*data: dữ liệu của nút; DataType: int, char,
double...*/
    struct node *left ; /* trỏ đến nút con trái */
    struct node *right; /* trỏ đến nút con phải */
}node;
```



Sử dụng con trỏ để biểu diễn cây nhị phân: Ví dụ



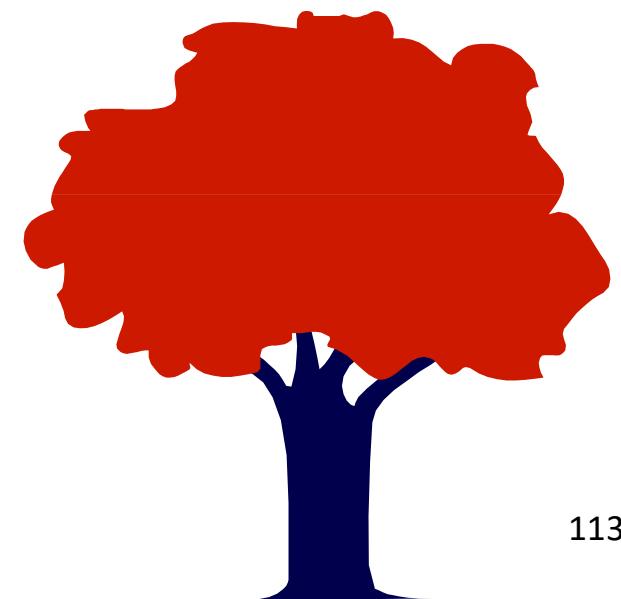
4. Cây nhị phân (Binary tree)

4.1. Định nghĩa và tính chất

4.2. Biểu diễn cây nhị phân

4.3. Duyệt cây nhị phân

4.4. Một số ứng dụng

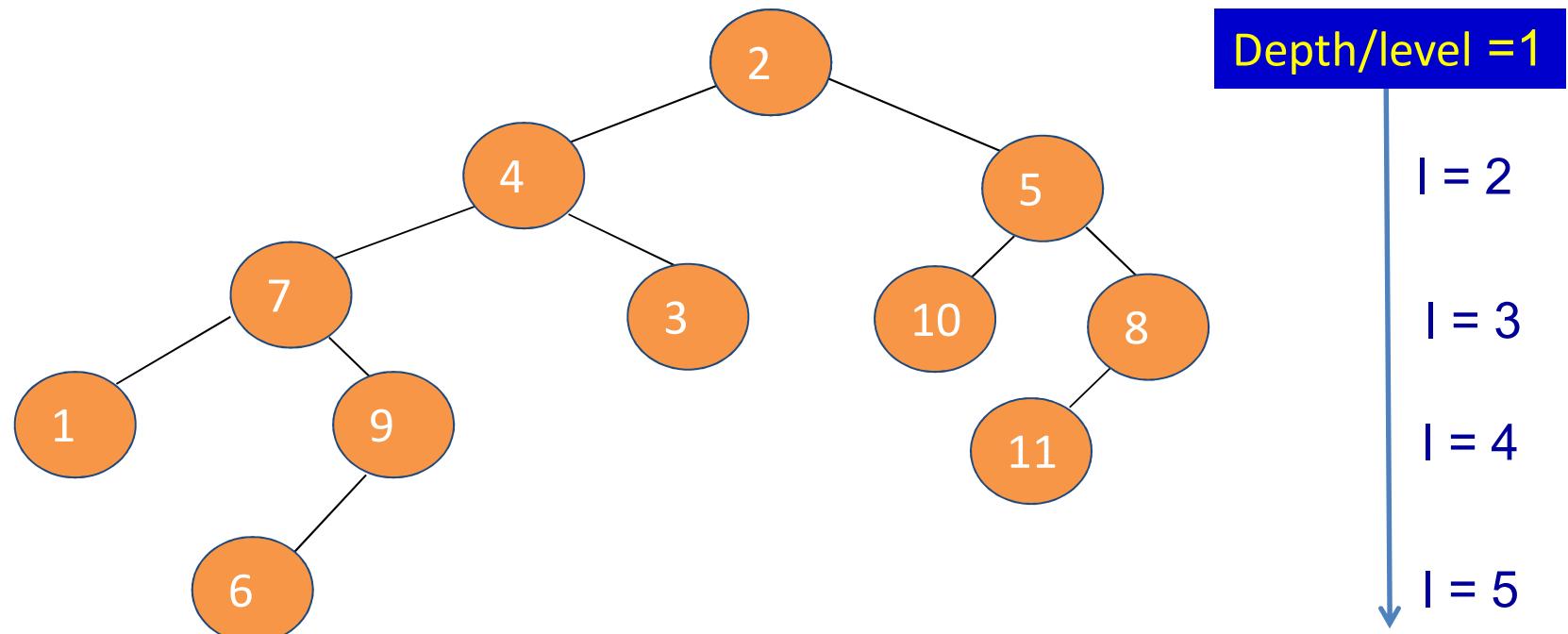


4.3. Duyệt cây nhị phân

- Duyệt cây tức là mỗi nút trên cây đều được thăm và thăm đúng 1 lần
- Có hai cách duyệt cây thông dụng
 - Duyệt theo chiều rộng (Breadth First Search)
 - Duyệt theo chiều sâu (Depth First Search)

4.3. Duyệt cây nhị phân: theo chiều rộng

- Duyệt cây theo chiều rộng: thăm nút gốc trên mức 1, rồi thăm đến các nút trên mức 2, rồi đến các nút trên mức 3, ...
- Các nút trên cùng 1 mức được thăm theo thứ tự từ trái sang phải



- `Breadth_First_Search(2)`: 2, 4, 5, 7, 3, 10, 8, 1, 9, 11, 6

4.3. Duyệt cây nhị phân: theo chiều sâu

- Duyệt cây theo chiều sâu: lần lượt thăm các nút theo từng nhánh
- Có ba loại duyệt cây theo chiều sâu
 - Thứ tự giữa Inorder
 - Thứ tự trước Preorder
 - Thứ tự sau Postorder
- Mỗi loại duyệt đều có ứng dụng thực tế tương ứng với nó

4.3. Duyệt cây nhị phân: theo chiều sâu

- **Thứ tự trước Preorder NLR**

- Thăm nút (Visit a **node**),
- Thăm cây con trái theo thứ tự trước (Visit **left subtree in preorder**),
- Thăm cây con phải theo thứ tự trước (Visit **right subtree in preorder**)

- **Thứ tự giữa Inorder LNR**

- Thăm cây con trái theo thứ tự giữa (Visit **left subtree in inorder**),
- Thăm nút (Visit a **node**),
- Thăm cây con phải theo thứ tự giữa (Visit **right subtree in inorder**)

- **Thứ tự sau Postorder LRN**

- Thăm cây con trái theo thứ tự sau (visit **left subtree in postorder**),
- Thăm cây con phải theo thứ tự sau (isit **right subtree in postorder**),
- Thăm nút (Visit a **node**)

Duyệt thứ tự trước (Preorder traversal – NLR)

- Thăm nút (Visit the node).
- Duyệt cây con trái (Traverse the left subtree).
- Duyệt cây con phải (Traverse the right subtree).

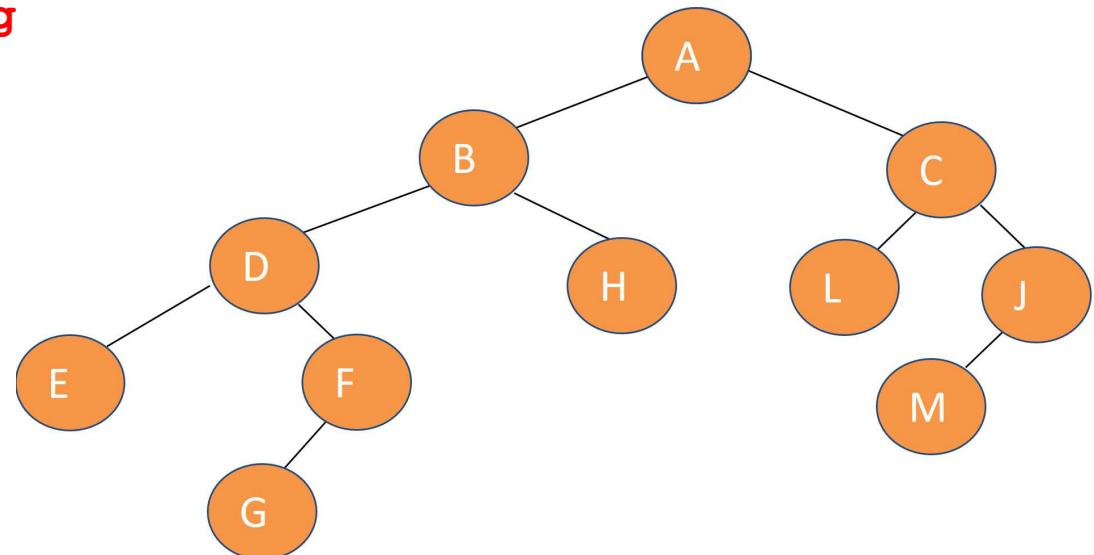
```
void printPreorder(node *root)
{
    if( root != NULL )
    {
        printf("%s ", root->word);
        printPreorder(root->left);
        printPreorder(root->right);
    }
}
```

Gọi: printPreorder (A) ;

A B D E F G H C L J M

Cây con trái

Cây con phải



Duyệt thứ tự giữa (Inorder traversal – LNR)

- Duyệt cây con trái (Traverse the left subtree)
- Thăm nút (Visit the node)
- Duyệt cây con phải (Traverse the right subtree)

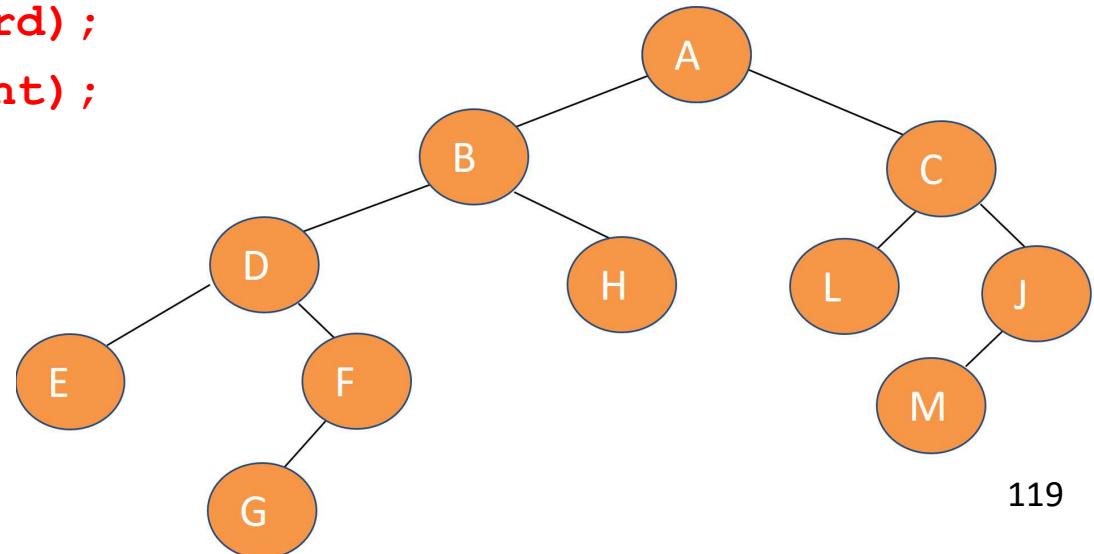
```
void printInorder(node *tree)
{
    if( tree != NULL )
    {
        printInorder(tree->left);
        printf("%s ", tree->word);
        printInorder(tree->right);
    }
}
```

Gọi: printInorder(A) ;

E D G F B H A L C M J

Cây con trái

Cây con phải



Duyệt thứ tự sau (Postorder traversal – LRN)

- Duyệt cây con trái (Traverse the left subtree)
- Duyệt cây con phải (Traverse the right subtree)
- Thăm nút (Visit the node)

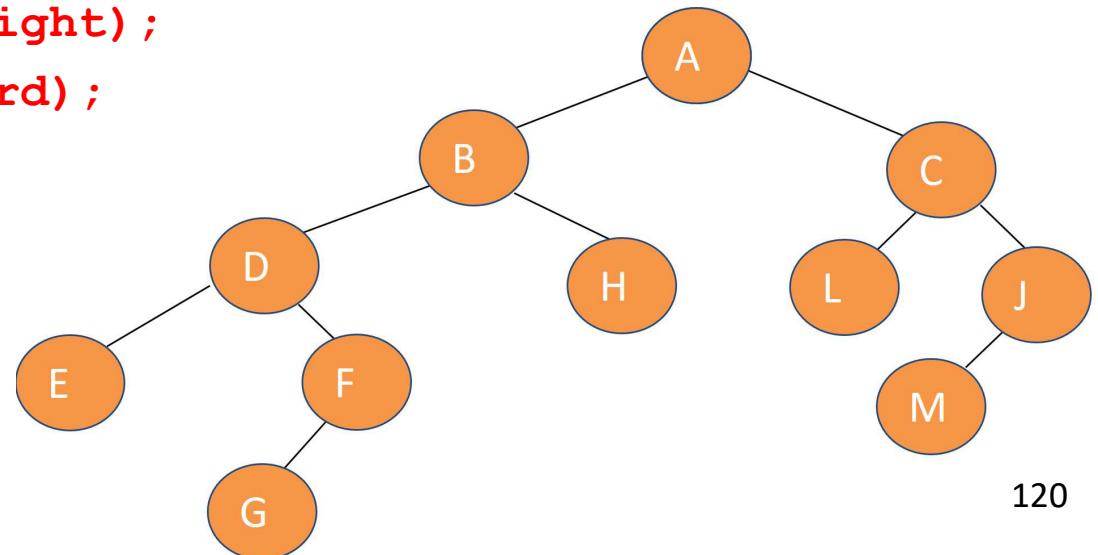
```
void printPostorder(node *tree)
{
    if( tree != NULL )
    {
        printPostorder(tree->left);
        printPostorder(tree->right);
        printf("%s ", tree->word);
    }
}
```

Gọi: `printPostorder(A);`

E G F D H B L M J C A

Cây con trái

Cây con phải



Các thao tác cơ bản trên cây nhị phân

```
typedef struct
{
    char word[20]; // dữ liệu của nút
    struct node * left;
    struct node * right;
}node;

node* makeTreeNode(char *word);
node *insertNode(node* root, char *word, bool LEFT);
int countNodes(node *root);
int depth(node *root);
void freeTree(node *root);
void printPreorder(node *root);
void printPostorder(node *root);
void printInorder(node *root);
```

Các thao tác cơ bản trên cây nhị phân: MakeNode

```
node* makeTreeNode(char *word);
```

- Tham số đầu vào: dữ liệu của nút mới
- Đầu ra: hàm trả về con trỏ trỏ tới nút mới được tạo (tức là địa chỉ của nút được tạo)

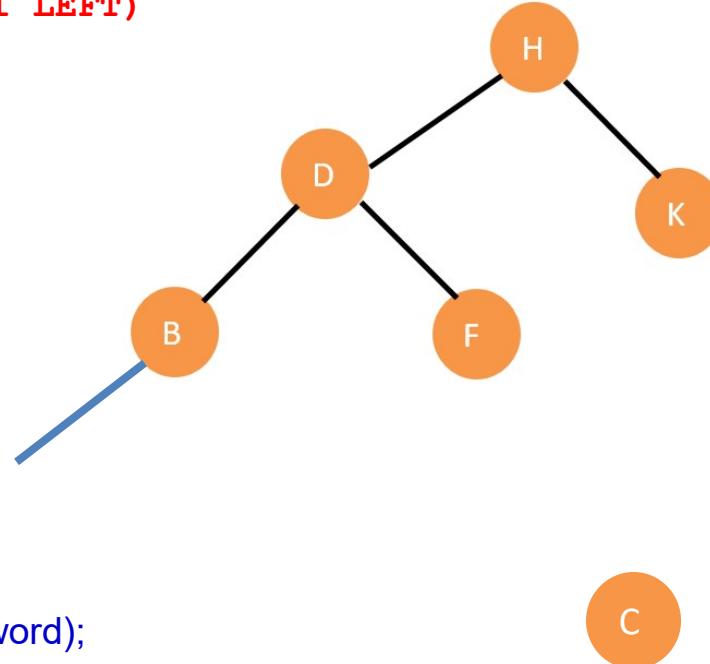
Các bước phải thực hiện:

- Cấp phát bộ nhớ cho nút mới. Kiểm tra cấp phát bộ nhớ có thành công hay không?
 - Nếu Yes: gán dữ liệu cho nút mới = word
 - con trỏ trỏ tới con trái = NULL
 - con trỏ trỏ tới con phải = NULL
- Trả về con trỏ trỏ tới nút mới (trả về địa chỉ của nút mới)

```
node* makeTreeNode(char *word)
{
    node* newNode = NULL;
    newNode = (node*)malloc(sizeof(node));
    if (newNode == NULL){ //Memory allocation is unsuccessful
        printf("Out of memory\n");
        exit(1);
    }
    else {
        strcpy(newNode->word, word);
        newNode->left = NULL;
        newNode->right= NULL;
    }
    return newNode;
}
```

Các thao tác cơ bản trên cây nhị phân: thêm một nút mới vào cây

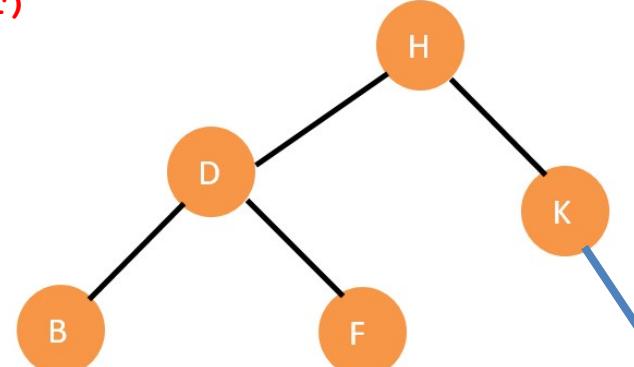
```
node *insertNode(node *tree, char *word, bool LEFT)
{
    node *newNode, *p;
    newNode = makeTreeNode(word);
    if ( tree == NULL ) return newNode;
    if (LEFT) //thêm nút vào vị trí trái nhất trên cây
    {
        p = tree;
        while (p->left !=NULL) p = p->left;
        p->left = newNode;
        printf("Nút %s là con trái của nút %s \n", word, (*p).word);
    }
    else { //thêm nút vào vị trí phải nhất trên cây
        p = tree;
        while (p->right !=NULL) p = p->right;
        p->right = newNode;
        printf("Nút %s là con phải của nút %s \n", word, (*p).word);
    }
    return tree;
}
```



Gọi: RandomInsert (H, "C", 1);

Các thao tác cơ bản trên cây nhị phân: thêm một nút mới vào cây

```
node *insertNode(node *tree, char *word, bool LEFT)
{
    node *newNode, *p;
    newNode = makeTreeNode(word);
    if ( tree == NULL ) return newNode;
    if (LEFT) //thêm nút vào vị trí trái nhất trên cây
    {
        p = tree;
        while (p->left !=NULL) p = p->left;
        p->left = newNode;
        printf("Nút %s là con trái của nút %s \n", word, (*p).word);
    }
    else { //thêm nút vào vị trí phải nhất trên cây
        p = tree;
        while (p->right !=NULL) p = p->right;
        p->right = newNode;
        printf("Nút %s là con phải của nút %s \n", word, (*p).word);
    }
    return tree;
}
```



Call: RandomInsert (H, "C", 0);

Các thao tác cơ bản trên cây nhị phân: tính chiều sâu của cây

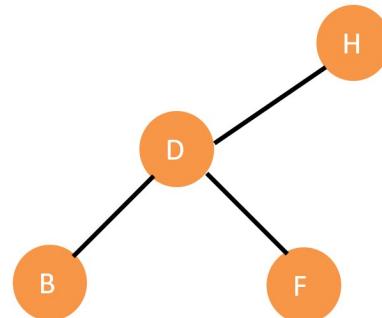
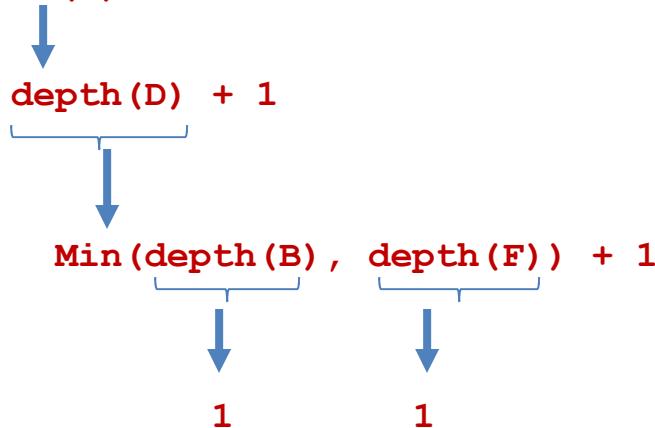
```
int depth(node *root) { /* hàm tính chiều sâu của cây */
    if( root == NULL ) return 0;
    // Trường hợp cơ sở : nút lá. Khi đó cây chỉ có 1 nút => chiều sâu = 1.
    if (root->left == NULL && root->right == NULL) return 1;

    // Nếu cây con trái NULL, gọi đệ quy trên cây con phải
    if (root->left==NULL) return depth(root->right) + 1;

    // Nếu cây con phải NULL, gọi đệ quy trên cây con trái
    if (root->right==NULL) return depth(root->left) + 1;

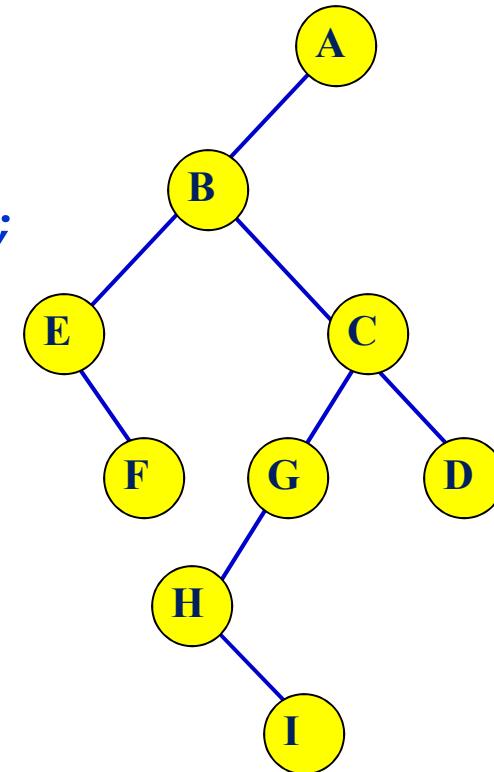
    return min(depth(root->left), depth(root->right)) + 1;
}
```

Gọi **depth(H)**:



Các thao tác cơ bản trên cây nhị phân: đếm số nút có trên cây

```
int countNodes(node *root) {  
    /* hàm đếm số nút trên cây*/  
    if( root == NULL ) return 0;  
    else {  
        int ld = countNodes(root->left);  
        int rd = countNodes(root->right);  
        return 1+ld+rd;  
    }  
}
```



#nút trên cây = 1 + #nút trên cây con trái + #nút trên cây con phải

Gọi `countNodes(A)`;

Các thao tác cơ bản trên cây nhị phân: xóa cây

```
void freeTree(node *root)
{
    if( root == NULL ) return;
    freeTree(root->left);
    freeTree(root->right);
    free(root);
}
```

Program in C

```
/* The program for testing binary tree traversal */
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
typedef struct {
    char word[20];
    struct node * left;
    struct node *right;
} node;
node *makeTreeNode(char *word);
node *insertNode(node* tree,char *word);
void freeTree(node *tree);
void printPreorder(node *tree);
void printPostorder(node *tree);
void printInorder(node *tree);
int countNodes(node *tree);
int depth(node *tree);
```

Program in C

```
void main() {  
    node *randomTree=NULL;  
    char word[20] = "a";  
    while( strcmp(word,"~") !=0 )  
    {    printf("\nEnter item (~ to finish): ");  
        scanf("%s", word);  
        if (strcmp(word,"~")==0) printf("Finish to input data for node... \n");  
        else randomTree = insertNode(randomTree,word, rand()%2);  
    }  
    printf("The tree in preorder:\n"); printPreorder(randomTree);  
    printf("The tree in postorder:\n"); printPostorder(randomTree);  
    printf("The tree in inorder:\n"); printInorder(randomTree);  
    printf("The number of nodes is: %d\n",countNodes(randomTree));  
    printf("The depth of the tree is: %d\n", depth(randomTree));  
    freeTree(randomTree);  
}
```

Bài tập

- Vẽ cây nhị phân có chiều cao = 3, kết quả duyệt thứ tự sau là (10, 30, 50, 20, 40, 70, 60)

Postorder: Left_Right_Root

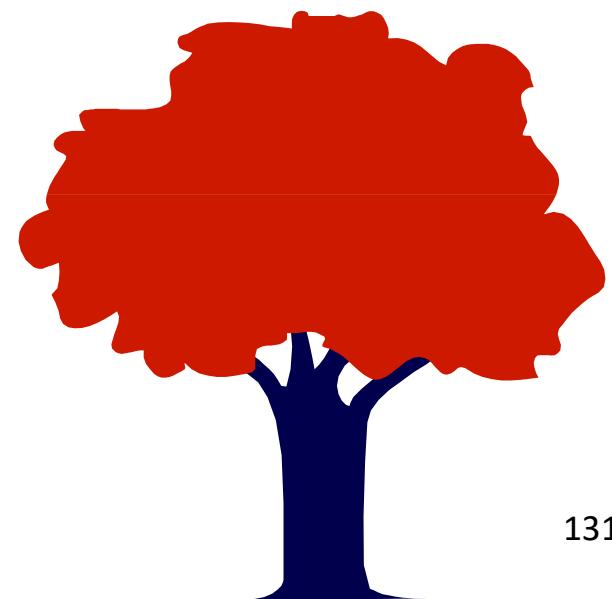
4. Cây nhị phân (Binary tree)

4.1. Định nghĩa và tính chất

4.2. Biểu diễn cây nhị phân

4.3. Duyệt cây nhị phân

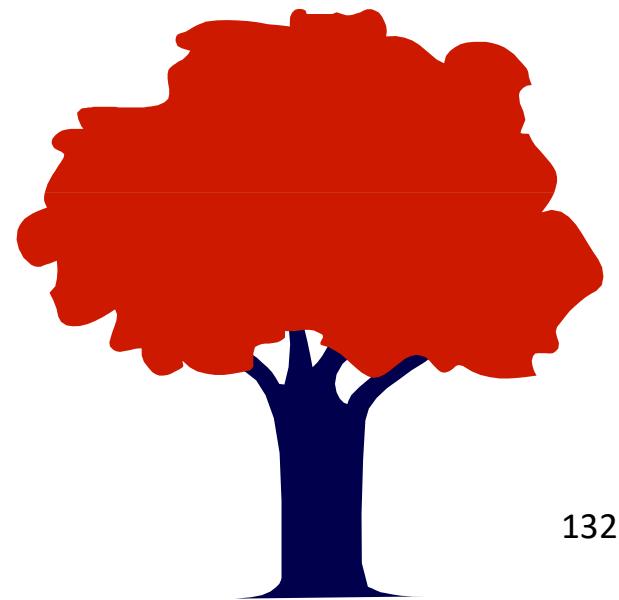
4.4. Một số ứng dụng



4.4. Một số ứng dụng

4.4.1. Biểu thức số học

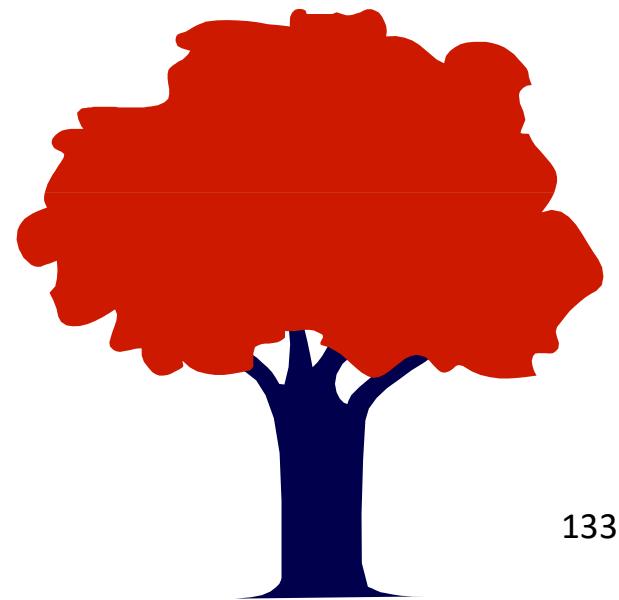
4.4.2. Mã Huffman



4.4. Một số ứng dụng

4.4.1. Biểu thức số học

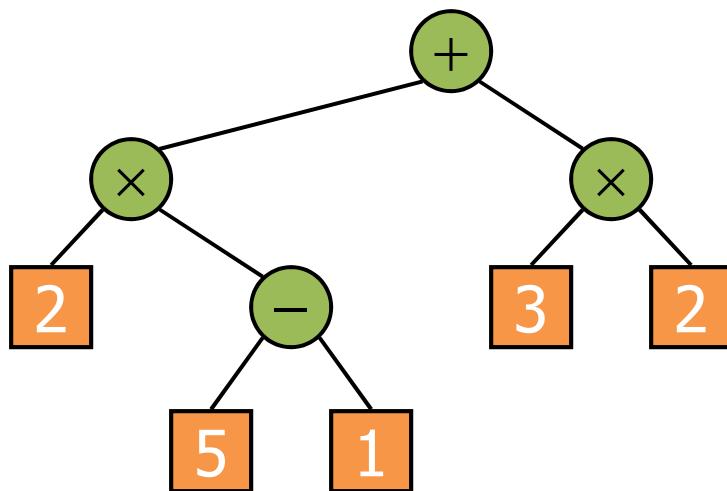
4.4.2. Mã Huffman



4.4.1. Biểu thức số học

- Cây biểu thức là cây nhị phân được sử dụng để biểu diễn biểu thức số học, trong đó:
 - Các nút trong: các phép toán
 - Các nút lá: các toán hạng

Ví dụ 1: cây biểu thức biểu diễn biểu thức $((2 \times (5 - 1)) + (3 \times 2))$



Duyệt cây:

1. Theo thứ tự trước (preorder traversal)

$+ \times 2 - 5 1 \times 3 2$

Cho ta: kí pháp tiền tố (prefix expression)

2. Theo thứ tự giữa (inorder traversal)

$2 \times 5 - 1 + 3 \times 2$

Cho ta: kí pháp trung tố (infix expression)

3. Theo thứ tự sau (postorder traversal)

$2 5 1 - \times 3 2 \times +$

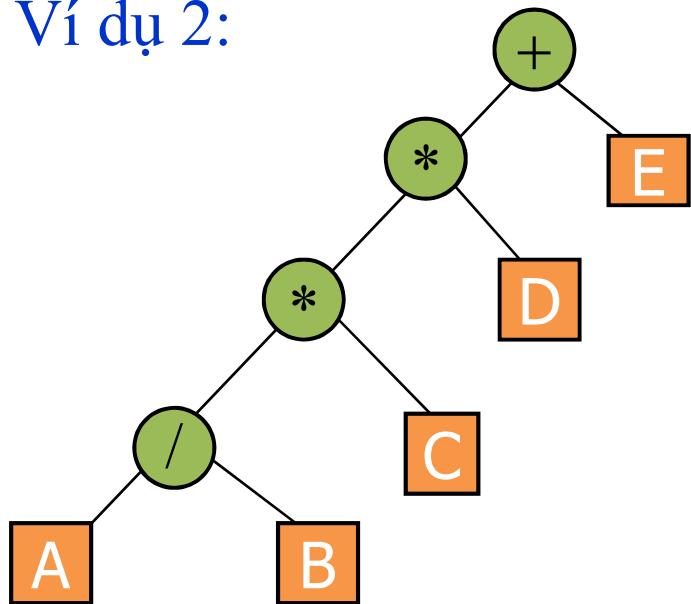
Cho ta: kí pháp hậu tố (postfix expression)

4. Theo mức (breath first traversal)

$+ \times x 2 - 3 2 5 1$

4.4.1. Biểu thức số học

Ví dụ 2:



Duyệt cây:

1. Theo thứ tự trước (preorder traversal)

+ * * / A B C D E

Cho ta: kí pháp tiền tố (prefix expression)

2. Theo thứ tự giữa (inorder traversal)

A / B * C * D + E

Cho ta: kí pháp trung tố (infix expression)

3. Theo thứ tự sau (postorder traversal)

A B / C * D * E +

Cho ta: kí pháp hậu tố (postfix expression)

4. Theo mức (breath first traversal)

+ * E * D / C A B

- Mức (độ sâu) của các nút trên cây cho biết trình tự thực hiện chúng (ta không cần sử dụng ngoặc để chỉ ra trình tự).
- Phép toán tại mức cao hơn của cây được thực hiện muộn hơn so với các mức dưới chúng.
- Phép toán ở gốc luôn được thực hiện sau cùng.

Kí pháp trung/hậu/tiền tố (Infix/Postfix/Prefix Expressions)

- **Trung tố**: là biểu thức trong đó các toán hạng đặt bao quanh các phép toán

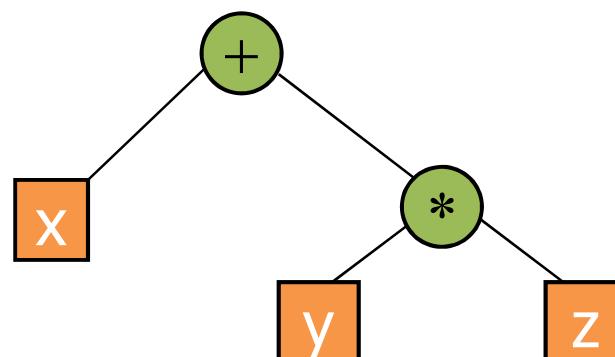
Ví dụ: $x+y$, $x+y*z$

- **Hậu tố** (còn gọi là Reverse Polish Notation): các phép toán được đặt sau các toán hạng

Ví dụ: $xy+$, $xyz*+$

- **Tiền tố** (còn gọi là Known as Polish notation): các phép toán được đặt trước các toán hạng

Ví dụ: $+xy$, $+x*yz$



Kí pháp trung tố (Infix Expressions)

- **Trung tố**: biểu thức trong đó các toán hạng đặt bao quanh các phép toán

Cách tính giá trị biểu thức :

$$a * b + c / d$$

- 1) Dựa trên thứ tự ưu tiên của các phép toán:

$$\text{utriên}(*) = \text{utriên}(/) > \text{utriên}(+) = \text{utriên}(-)$$

- 2) Khi một toán hạng đặt giữa hai phép toán, thì toán hạng đó sẽ thuộc về phép toán có thứ tự ưu tiên cao hơn.

→ Toán hạng b đặt giữa phép * và + ==> b thuộc về phép *

→ Toán hạng c đặt giữa phép + và / ==> c thuộc về phép /

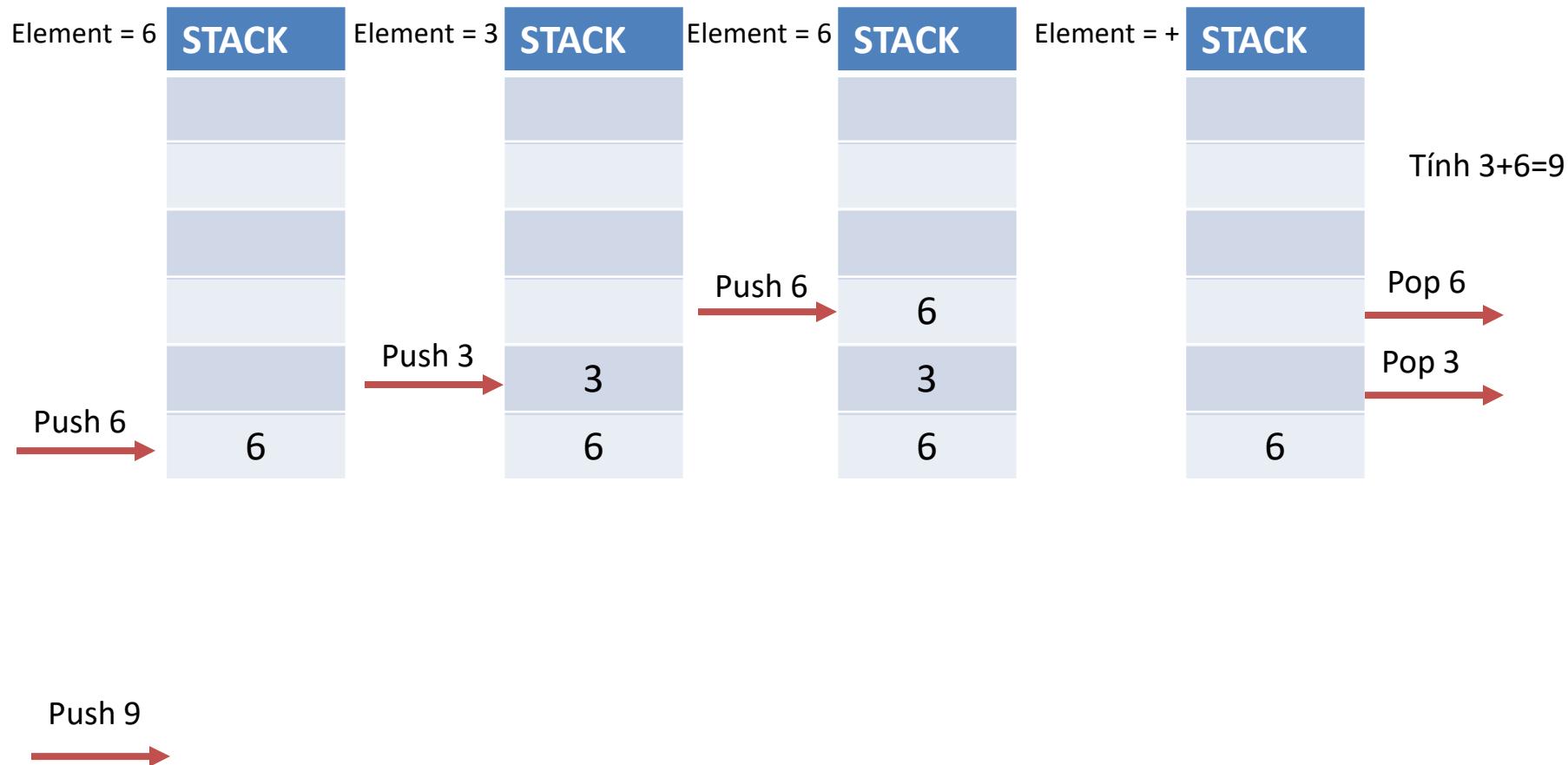
Do đó: $a * b + c / d = (a * b) + (c / d)$

Ví dụ 1: Tính giá trị kí pháp hậu tố: sử dụng stack

- Tính giá trị kí pháp hậu tố sử dụng stack:

Quá trình tính toán thực hiện trên stack lần lượt như sau:

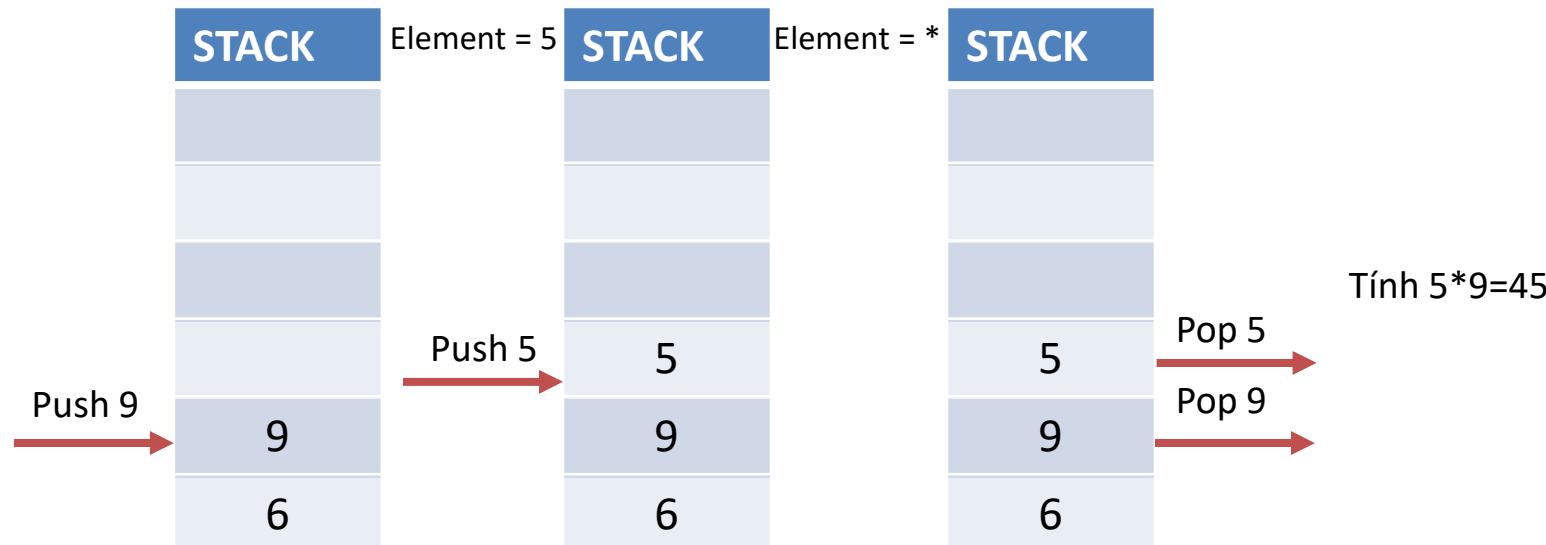
6 3 6 + 5 * 9 / -
→



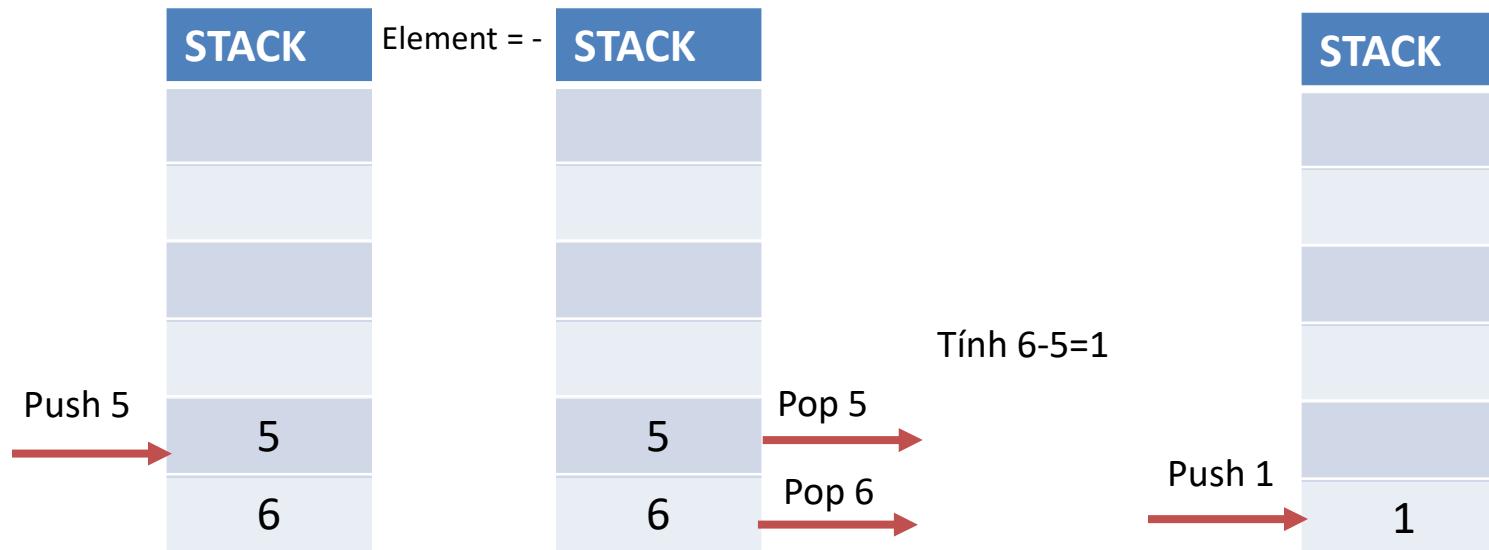
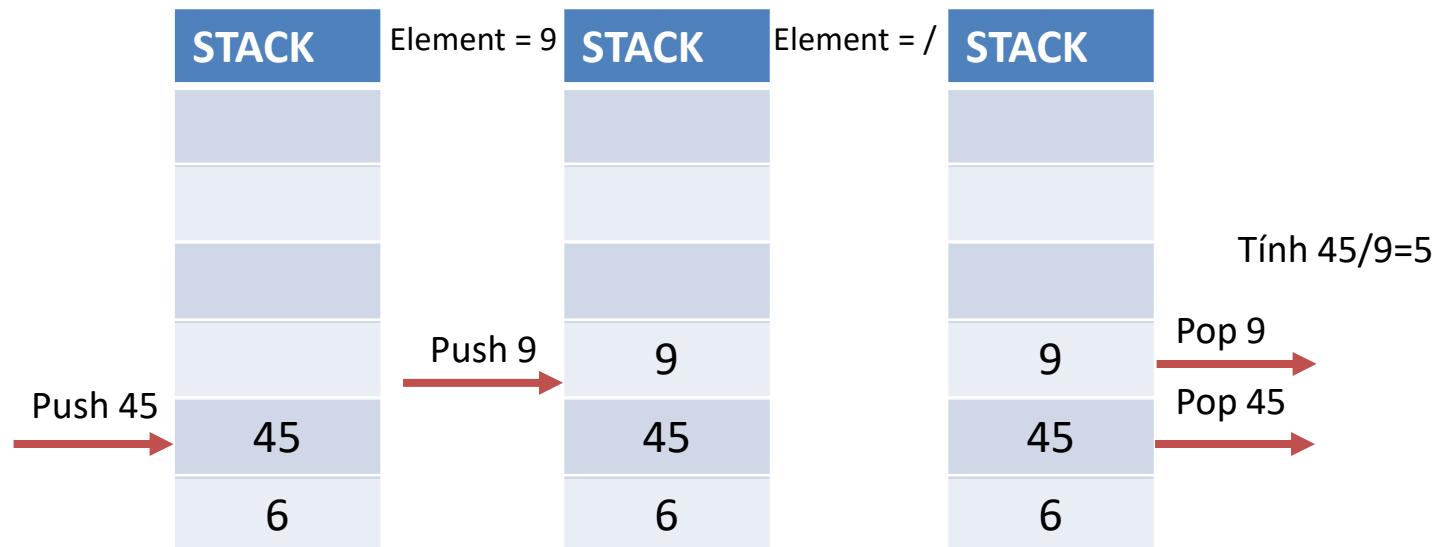
Ví dụ 1: Tính giá trị kí pháp hậu tố: sử dụng stack

- Evaluate the following postfix expression using stacks:
- The expression is evaluated by using stack as follows:

6 3 6 + 5 * 9 / -



Tính giá trị kí pháp hậu tố sử dụng stack: $6 \ 3 \ 6 + 5 * 9 / -$



Trạng thái cuối cùng của stack chứa kết quả của biểu thức

Tính giá trị kí pháp hậu tố sử dụng stack

```
evaluationPostfix(stack s, postfixExpression E)
{ //Biểu thức được lưu ở mảng E
    //Đọc các kí tự trong biểu thức từ trái sang phải:
    i = 0;
    while ( i < số_kí_tự_trong_biểu_thức)
    {
        if (E[i] là toán hạng)
            push E[i] vào stack
        else if (E[i] là phép toán)
        {
            1. Pop phần tử đầu stack và lưu vào biến operand1
            2. Pop phần tử đầu stack và lưu vào biến in operand2
            3. Tính: operand2 op operand1 {op = E[i]}, rồi push kết quả vào
                   stack
        }
        i = i + 1;
    } //end while
    Pop phần tử đầu stack, lưu vào biến Result;
    return Result; //chứa kết quả của biểu thức
}
```

Ví dụ 1:Tính kí pháp hậu tố:

636+5*9/-



Giá trị biểu thức = 1

	Kí tự của biểu thức	Thao tác thực hiện	Trạng thái Stack
1	6	Push 6 vào stack	6
2	3	Push 3 vào stack	6 3
3	6	Push 6 vào stack	6 3 6
4	+	Pop 6	6 3
5		Pop 3	6
6		Tính $3 + 6 = 9$	6
7		Push 9 vào stack	6 9
8	5	Push 5 vào stack	6 9 5
9	*	Pop 5	6 9
10		Pop 9	6
11		Tính $9 * 5 = 45$	6
12		Push 45 vào stack	6 45
13	9	Push 9 vào stack	6 45 9
14	/	Pop 9	6 45
15		Pop 45	6
16		Tính $45 / 9 = 5$	6
17		Push 5 vào stack	6 5
18	-	Pop 5	6
19		Pop 6	Empty
20		Tính $6 - 5 = 1$	Empty
21		Push 1 vào stack	1
22		Pop value = 1	Empty

Ví dụ 1: Tính giá trị kí pháp hậu tố: dùng stack

6 3 6 + 5 * 9 / -
 \u2193 \u2193 \u2193 \u2193 \u2193 \u2193 \u2193 \u2193
 3 + 6 = 9

6 9 5 * 9 / -
 \u2193 \u2193 \u2193 \u2193 \u2193 \u2193 \u2193
 9 * 5 = 45

6 45 9 / -
 \u2193 \u2193 \u2193 \u2193 \u2193
 45 / 9 = 5

6 5 -
 \u2193 \u2193 \u2193
 6 - 5 = 1

Đọc biểu thức từ trái sang phải

Tính giá trị biểu thức = 1

Ví dụ 2: Tính giá trị kí pháp tiền tố: sử dụng stack

$$+ \ - \ * \ 2 \ 3 \ 5 \ / \ \underbrace{* \ 2 \ 3}_{2*3=6} \ 2$$

$$+ \ - \ * \ 2 \ 3 \ 5 \ / \ \underbrace{6 \ 2}_{6/2=3}$$

$$+ \ - \ * \ \underbrace{2 \ 3}_{2*3=6} \ 5 \ 3$$

Đọc biểu thức từ phải sang trái

$$+ \ - \ \underbrace{6 \ 5}_{6-5=1} \ 3$$

$$+ \ \underbrace{1 \ 3}_{1+3=4}$$

Giá trị biểu thức = 4

Ví dụ 2: Tính giá trị kí pháp tiền tố:

$+-*235/*232$

	Kí tự của biểu thức	Thao tác thực hiện	Trạng thái stack		Kí tự của biểu thức	Thao tác thực hiện	Trạng thái Stack	
1	2	Push 2 to stack	2		15	*	Pop 2	3 5 3
2	3	Push 3 to stack	2 3		16		Pop 3	3 5
3	2	Push 2 to stack	2 3 2		17		Evaluate $2*3=6$	3 5
4	*	Pop 2	2 3		18		Push 6 to stack	3 5 6
5		Pop 3	2		19	-	Pop 6	3 5
6		Evaluate $2 * 3 = 6$	2		20		Pop 5	3
7		Push 6 to stack	2 6		21		Evaluate $6-5 = 1$	3
8	/	Pop 6 to stack	2		22		Push 1 to stack	3 1
9		Pop 2	Empty		23	+	Pop 1	3
10		Evaluate: $6/2=3$	Empty		24		Pop 3	Empty
11		Push 3 to stack	3		25		Evaluate $1 + 3 = 4$	Empty
12	5	Push 5 to stack	3 5		26		Push 4 to stack	4
13	3	Push 3 to stack	3 5 3		27		Pop value = 4	Empty
14	2	Push 2 to stack	3 5 3 2					

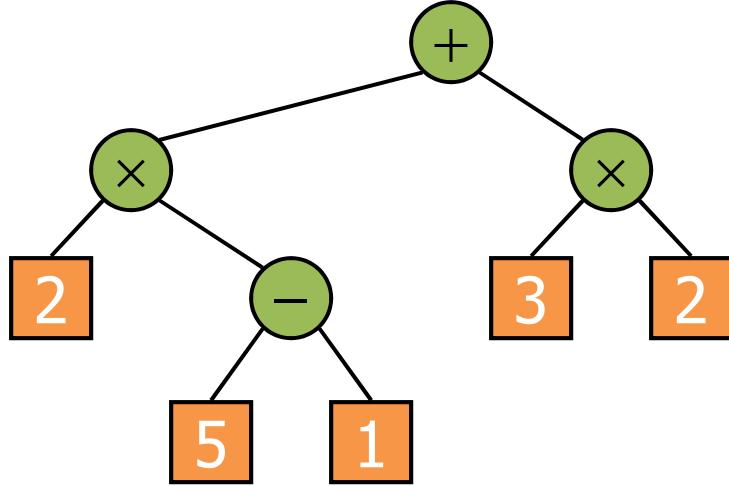
Giá trị biểu thức = 4

In biểu thức

Cho cây nhị phân có **con trỏ root** trỏ đến gốc của cây biểu diễn một biểu thức số học. Viết hàm in ra biểu thức đó.

- Duyệt cây theo thứ tự giữa:

- In kí tự “(“ trước khi thực hiện duyệt cây con trái
- In ra toán hạng hoặc phép toán có ở trong nút khi thực hiện thăm nút
- In kí tự “)” sau khi thăm xong cây con phải



Gọi `printTree(+);`

Biểu thức in ra là:

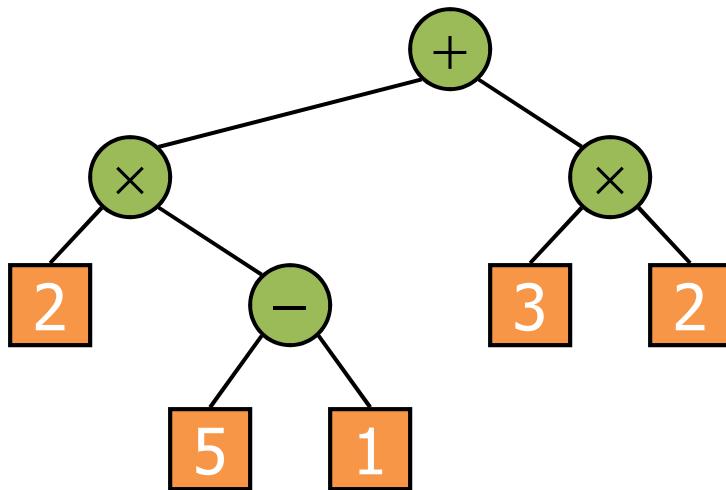
$((2 \times (5 - 1)) + (3 \times 2))$

```
void printTree(node *root)
{
    if (root.left != NULL)
    {
        printf("(");
        printTree (root.left);
    }
    print(root.data);
    if (root.right != NULL)
    {
        printTree (root.right);
        printf(")");
    }
}
```

Tính giá trị biểu thức số học

Cho cây nhị phân có con trỏ **root** trỏ vào gốc của cây biểu diễn một biểu thức ô học. Hãy viết hàm tính giá trị của biểu thức này.

- Duyệt cây theo thứ tự sau (Postorder traversal):
 - Tính giá trị các cây con một cách đệ quy
 - Thực hiện phép toán ở nút gốc sau khi các cây con trái và phải của nó đã được tính xong giá trị.



Gọi A = `evaluate(+) ;`

Giá trị của A là:

```
int evaluate (node *root)
{
    if (root.left == NULL)//external node
        return root.data;
    else { //internal node
        x = evaluate (root.left);
        y = evaluate (root.right);
        gọi OP là phép toán ở nút gốc root.data
        z = x OP y
        return z;
    }
}
```

Bài tập: Tính giá trị biểu thức sử dụng stack

Bài tập 1: tính giá trị các kí pháp hậu tố

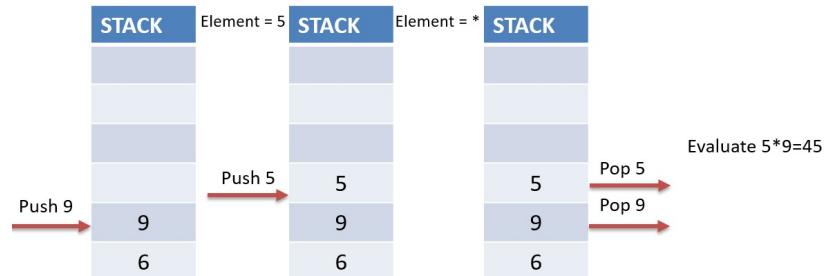
1) $2 \ 3 \ + \ 4 \ * \ 5 \ *$

2) $27 \ 3 \ 2 \ ^ \ / \ 3 \ 17 \ * \ + \ 27 \ 2 \ * \ -$

3) $7 \ 6 \ + \ 4 \ * \ 410 \ - \ 5 \ ^$

4) $7 \ 5 \ - \ 9 \ 2 \ / \ *$

Cách 1: Miêu tả bằng cách vẽ lần lượt trạng thái stack:



Bài tập 2: tính giá trị các kí pháp tiền tố

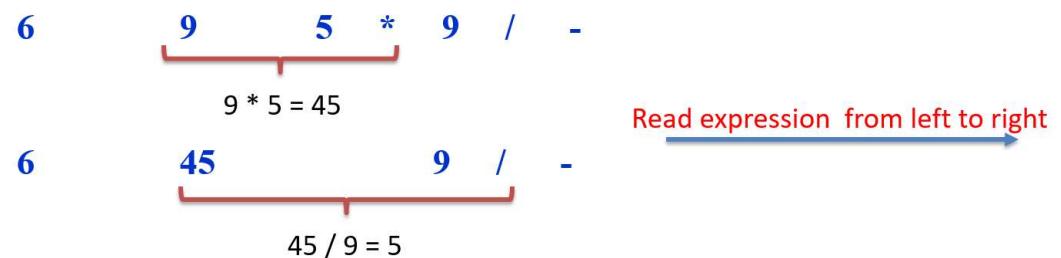
1) $- \ + \ 7 \ * \ 4 \ 5 \ + \ 2 \ 10$

2) $- \ * \ 9 \ + \ 2 \ 3 \ / \ 27 \ 3$

Cách 2: Miêu tả các bước của thuật toán bằng cách kẻ bảng:

	Element of expression	Action performed	Stack status
1	6	Push 6 to stack	6
2	3	Push 3 to stack	6 3
3	6	Push 6 to stack	6 3 6

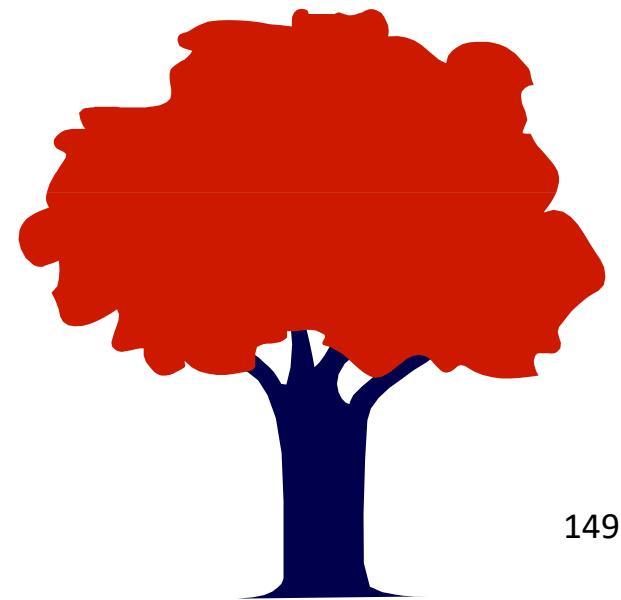
Cách 3:



4.4. Một số ứng dụng

4.4.1. Biểu thức số học

4.4.2. Mã Huffman



Bài tập

- Giả sử ta cần lưu trữ một file gồm 100,000 kí tự. File này chỉ chứa 6 kí tự, với tần suất xuất hiện như sau:

	a	b	c	d	e	f
Frequency	45000	13000	12000	16000	9000	5000

- Ta sẽ mã hóa mỗi kí tự bởi một xâu nhị phân (gọi là **codeword**). Do đó, ta cần tìm một dãy các bit nhị phân mã hóa file này sao cho số bit sử dụng là ít nhất có thể, tức là tìm cách mã hóa sao cho sử dụng bộ nhớ ít nhất.
- Có hai cách mã hóa:
 - Mã hóa với độ dài cố định (fixed-length code): các codeword đều có độ dài bằng nhau.
 - Mã hóa với độ dài biến đổi (variable-length code): các codeword có độ dài khác nhau.
- Ví dụ: nếu sử dụng mã với độ dài cố định, thì với file văn bản đã cho, với 6 kí tự, ta cần ít nhất 3 bit cho mỗi codeword (vì $2^2 = 4 < 6$, $2^3 = 8$)

	a	b	c	d	e	f
Frequency	45000	13000	12000	16000	9000	5000
A fixed-length	000	001	010	011	100	101
A variable-length	0	101	100	111	1101	1100

Bài tập

- Ví dụ: nếu sử dụng mã với độ dài cố định, thì với file văn bản đã cho, với 6 ký tự, ta cần ít nhất 3 bit cho mỗi codeword (vì $2^2 = 4 < 6$, $2^3 = 8$)

	a	b	c	d	e	f
Frequency	45000	13000	12000	16000	9000	5000
A fixed-length	000	001	010	011	100	101
A variable-length	0	101	100	111	1101	1100

- Mã hóa với độ dài cố định (fixed length-code) cần:

$$3 * 100000 = 300000 \text{ bits để lưu trữ file đã cho}$$

3 bit cho mỗi ký tự

Số lượng ký tự trong file

- Mã hóa với độ dài thay đổi (variable-length code) cần:

$$1 * 45000 + 3 * 13000 + 3 * 12000 + 3 * 16000 + 4 * 9000 + 4 * 5000 = 224000 \text{ bit}$$

#bit sử dụng để mã hóa ký tự "a"

#bit sử dụng để mã hóa ký tự "f"

→ Tiết kiệm bộ nhớ: 25%

Bài tập

Có thể mã hóa bằng 2 cách:

- **Mã hóa với độ dài cố định (fixed-length code)**: các codeword có độ dài bằng nhau
→ dễ mã hóa cũng như giải mã, nhưng lại đòi hỏi bộ nhớ lớn.
- **Mã hóa với độ dài biến đổi (variable-length code)**: các codeword có thể có độ dài khác nhau
 - Kí tự có tần suất xuất hiện nhiều nhất (ít nhất) sẽ được mã hóa bởi codeword có độ dài ngắn nhất (dài nhất).
 - Mỗi codeword phải được xác định duy nhất, không phụ thuộc vào độ dài của nó → không có codeword nào là đoạn đầu của một codeword khác (ví dụ: {a = 1, b = 110, c = 10, d = 111}, thì khi giải mã “1101111” ta thu được xâu ?)
→ **Mã phi tiền tố (Prefix code)**: là cách mã hóa mỗi ký tự c bởi một xâu nhị phân code(c) sao cho mã của một ký tự bất kì không là đoạn đầu của bất cứ mã của ký tự nào trong số các ký tự còn lại (ví dụ: {a = 0, b = 110, c = 10, d = 111} là mã phi tiền tố

	a	b	c	d	e	f
Frequency	45000	13000	12000	16000	9000	5000
A fixed-length	000	001	010	011	100	101
A variable-length	0	101	100	111	1101	1100

Bài toán mã hóa tối ưu (Optimum source coding problem)

- Mã hóa với độ dài biến đổi sử dụng 224000 bit thay vì 300000 bits nếu sử dụng mã hóa với độ dài cố định \rightarrow tiết kiệm bộ nhớ 25%. Câu hỏi: có cách mã hóa nào tốt hơn không? Hay đây đã là cách mã hóa tối ưu (cho chi phí nhỏ nhất)?

	a	b	c	d	e	f
Frequency	45000	13000	12000	16000	9000	5000
A fixed-length	000	001	010	011	100	101
A variable-length	0	101	100	111	1101	1100

- Bài toán mã hóa tối ưu: Cho bảng ký tự $A = \{a_1, \dots, a_n\}$ với tần suất xuất hiện $f(a_i)$, hãy tìm mã phi tiền tố C cho bảng ký tự A sao cho số lượng bit cần sử dụng là nhỏ nhất

$$B(C) = \sum_{a=1}^n f(a_i)L(c(a_i))$$

Cần để mã hóa file gồm $\sum_{a=1}^n f(a)$ ký tự, với:

- $c(a_i)$ là xâu nhị phân để mã hóa ký tự a_i ,
- $L(c(a_i))$ là độ dài của $c(a_i)$.

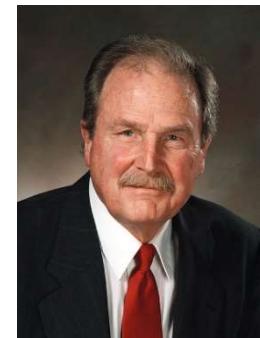
Bài toán mã hóa tối ưu (Optimum source coding problem)

- **Bài toán mã hóa tối ưu:** Cho bảng kí tự $A = \{a_1, \dots, a_n\}$ với tần suất xuất hiện $f(a_i)$, hãy tìm mã phi tiền tố C cho bảng kí tự A sao cho số lượng bit cần sử dụng là nhỏ nhất

$$B(C) = \sum_{a=1}^n f(a_i)L(c(a_i))$$

Cần để mã hóa file gồm $\sum_{a=1}^n f(a)$ kí tự, với:

- $c(a_i)$ là xâu nhị phân để mã hóa kí tự a_i ,
- $L(c(a_i))$ là độ dài của $c(a_i)$.



David A. Huffman
1925-1999

Huffman đã đề xuất một thuật toán tham lam để giải quyết bài toán mã hóa tối ưu, tức là xây dựng được một mã phi tiền tố với tổng số bit sử dụng là ít nhất. Mã đó được gọi là **mã Huffman**.

4.4.2 Mã Huffman

Thuật toán:

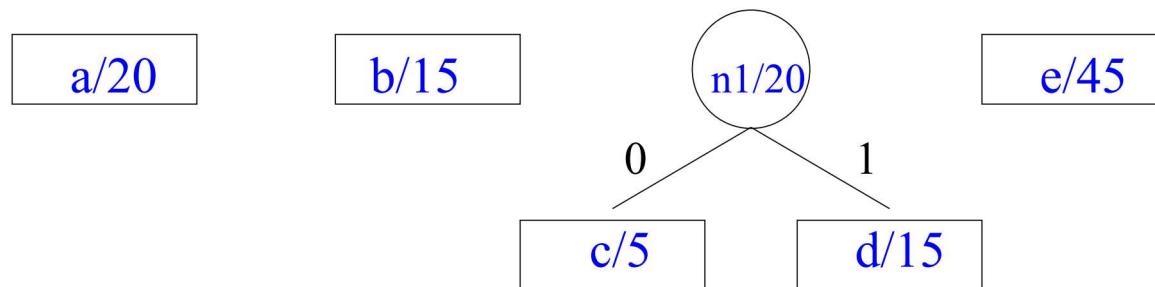
- Bước 1: Lấy 2 ký tự có tần suất xuất hiện nhỏ nhất x, y từ bảng kí tự A, rồi xây dựng cây con gồm 2 nút lá chứa 2 ký tự này (ý tưởng tham lam). Gán nhãn cho nút gốc của cây này là z.
- Bước 2: Đặt tần suất xuất hiện $f(z) = f(x) + f(y)$. Xóa x, y và thêm z tạo nên bảng kí tự mới: $A' = A \cup \{z\} - \{x, y\}$, khi đó $|A'| = |A| - 1$

Lặp lại 2 bước trên, đặt tên là **ghép**, với bảng kí tự mới A' cho đến khi bảng kí tự chỉ còn gồm 1 ký tự.

Cây thu được chính là mã Huffman

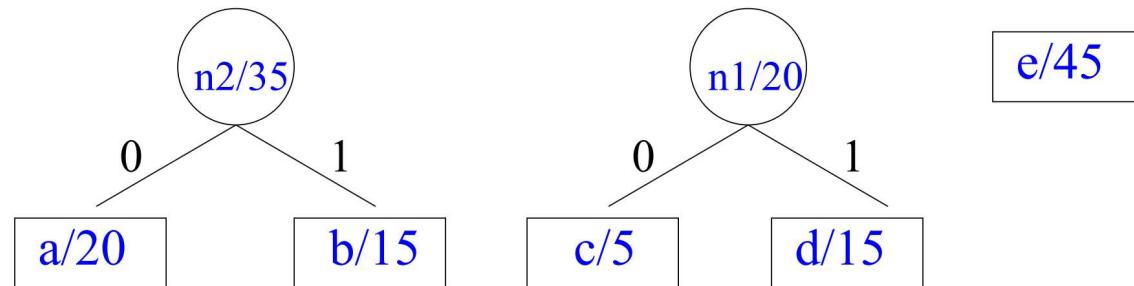
4.4.2 Mã Huffman

- Cho $A = \{a / 20, b / 15, c / 5, d / 15, e / 45\}$ là bảng kí tự cùng tần suất xuất hiện của các kí tự.
- Bước thứ 1: mã Huffman sẽ **ghép** c và d



Bảng kí tự mới $A_1 = \{a / 20, b / 15, n1 / 20, e / 45\}$

- Thuật toán tiếp tục ghép a và b (hoặc có thể ghép b và n1)

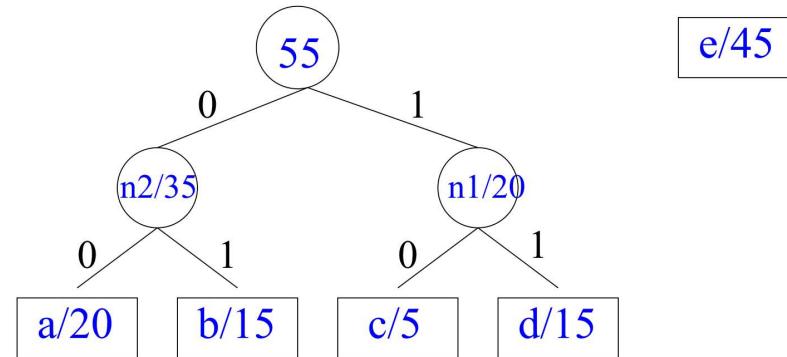


Bảng kí tự mới $A_2 = \{n2 / 35, n1 / 20, e / 45\}$

4.4.2 Mã Huffman

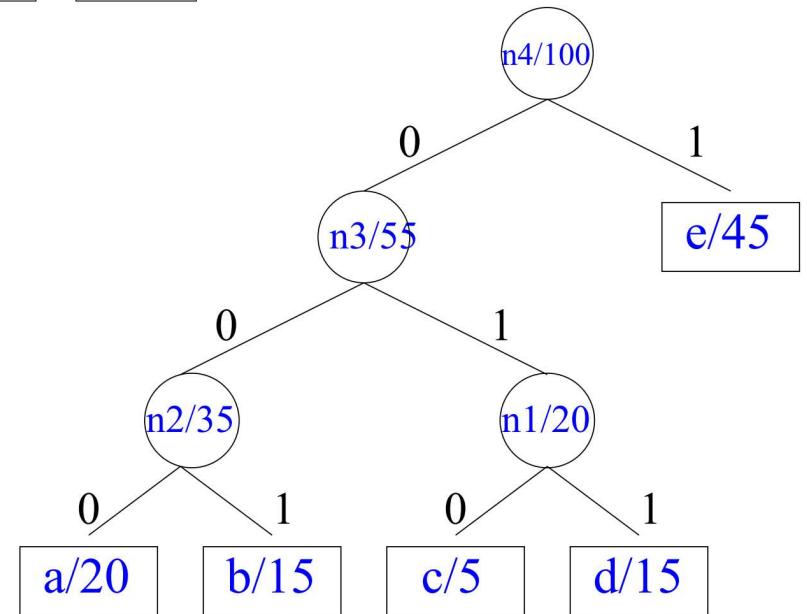
Bảng kí tự mới $A_2 = \{n2 / 35, n1 / 20, e / 45\}$

Ghép n1 và n2:



Bảng kí tự mới $A_3 = \{n3 / 55, e / 45\}$

Ghép e và n3, và thuật toán kết thúc



Mã Huffman thu được:

a = 000, b = 001, c = 010, d = 011, e = 1

Là mã phi tiền tố tối ưu

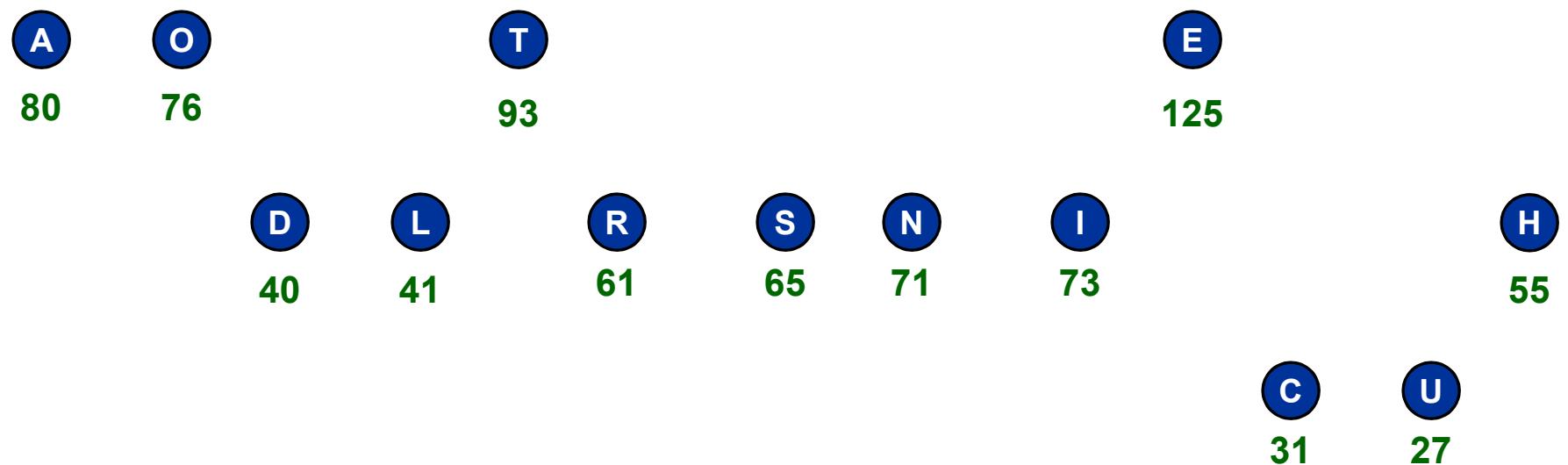
(cho tổng số bit sử dụng ít nhất)

4.4.2 Mã Huffman

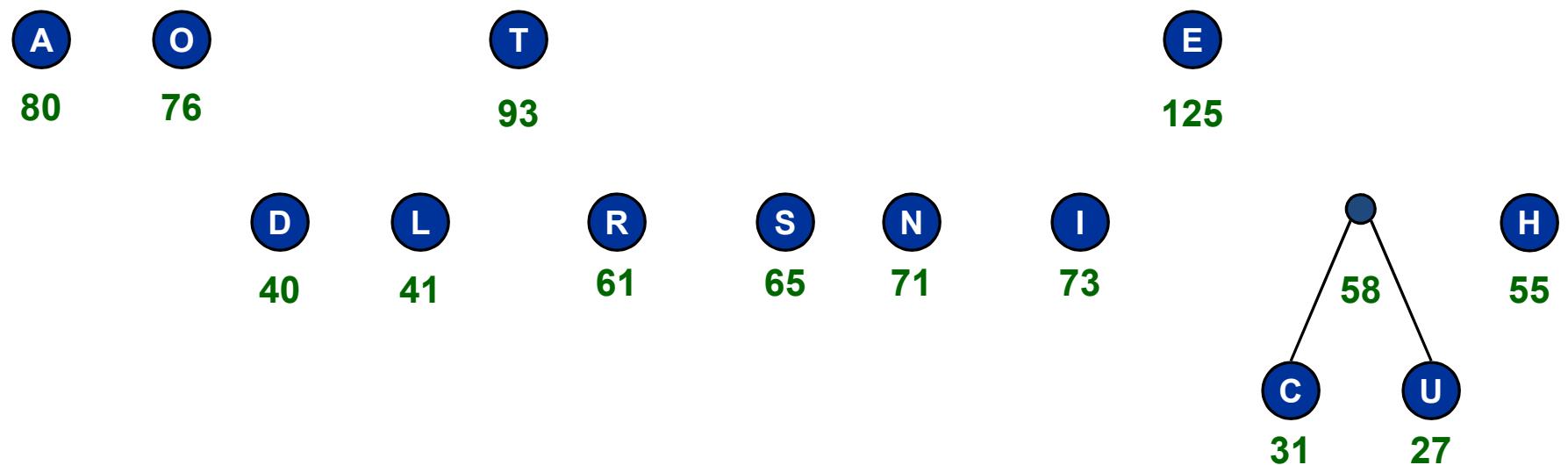
- Tần suất của các ký tự trong văn bản:

Char	Freq
E	125
T	93
A	80
O	76
I	72
N	71
S	65
R	61
H	55
L	41
D	40
C	31
U	27

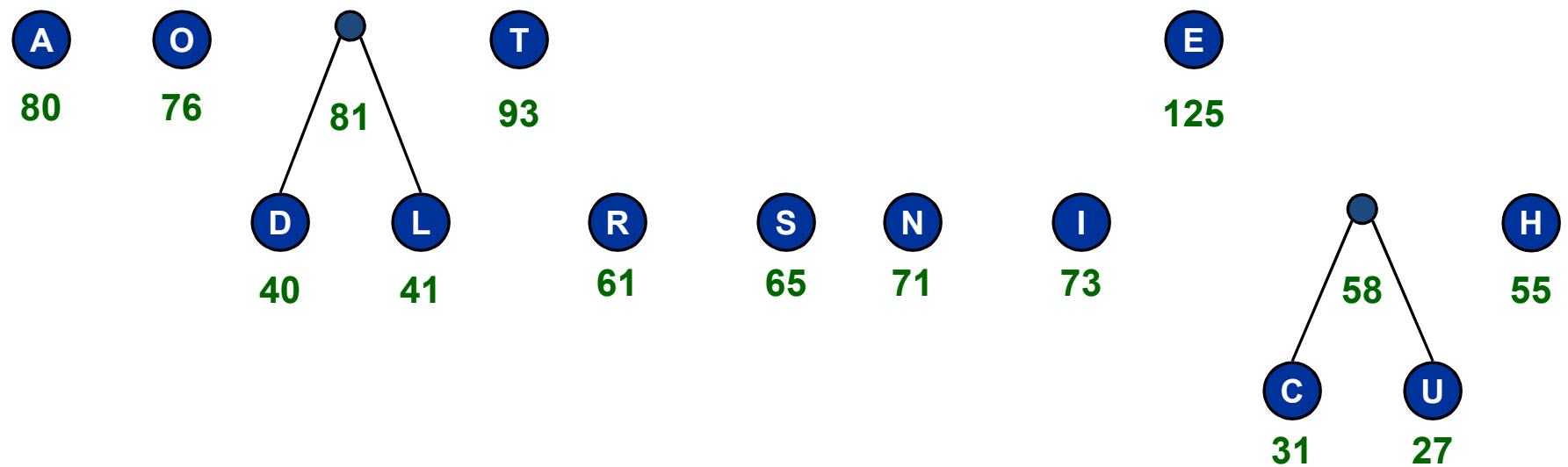
4.4.2 Mā Huffman



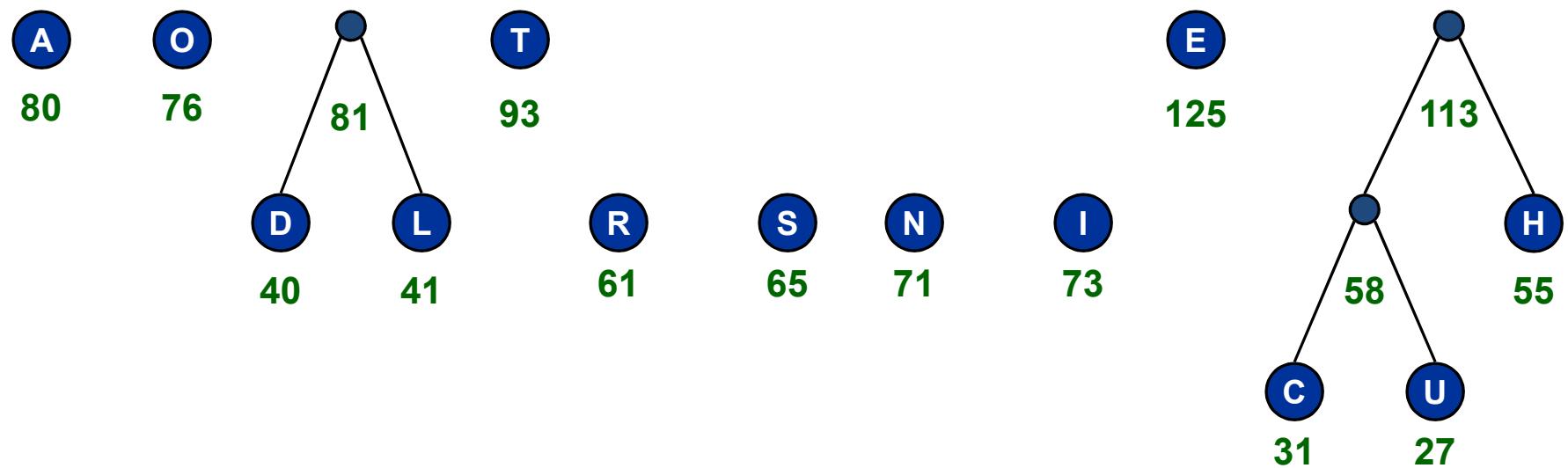
4.4.2 Mă Huffman



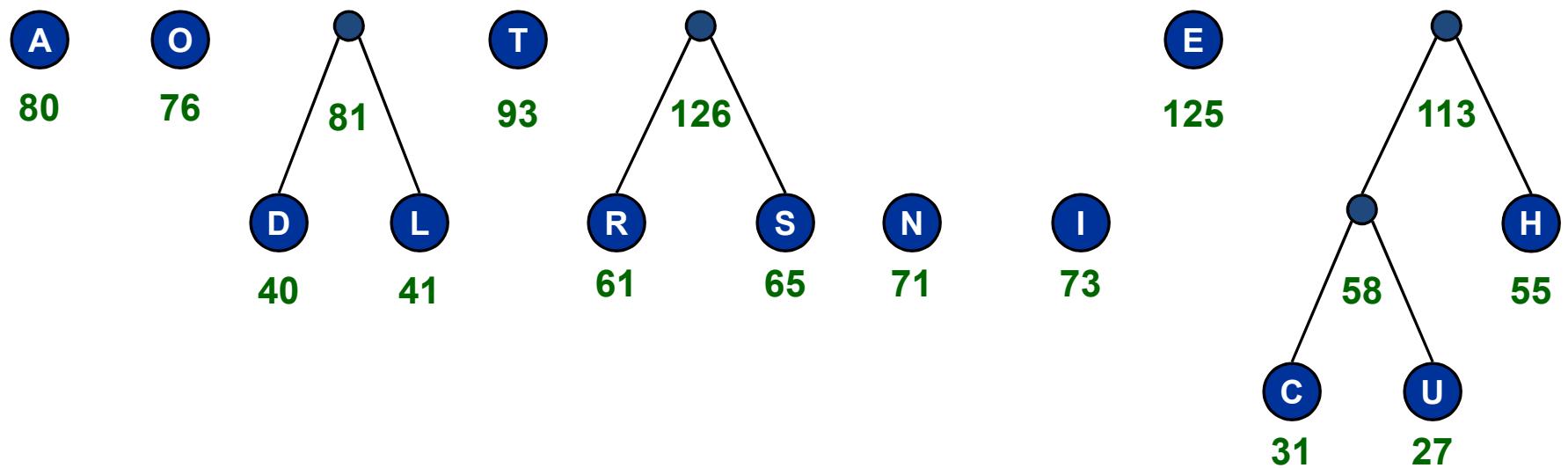
4.4.2 Mă Huffman



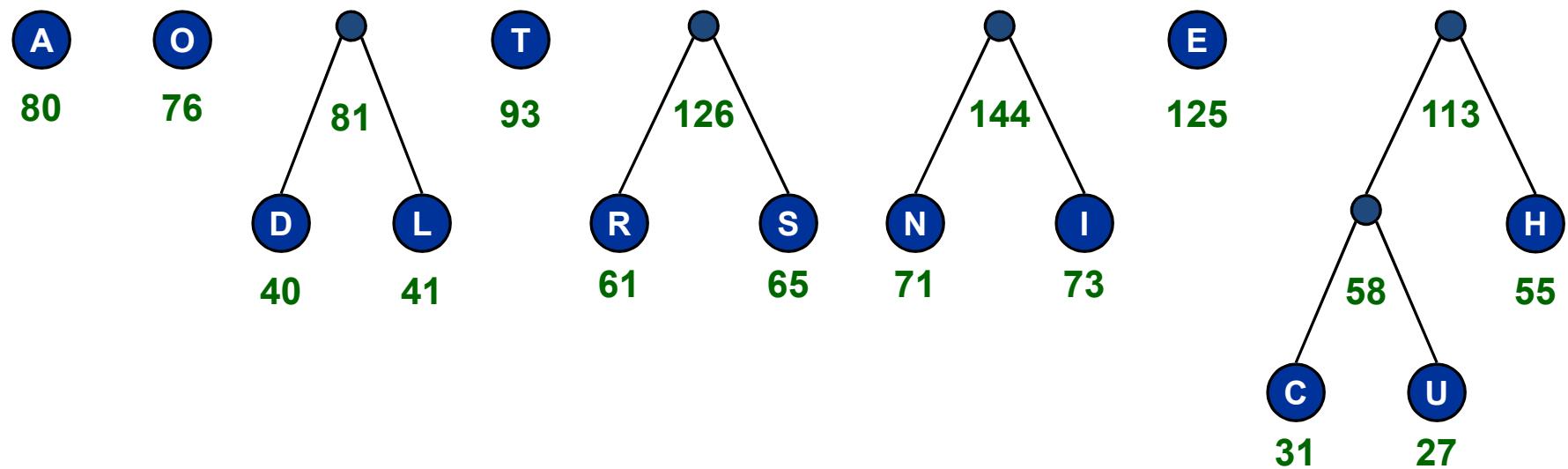
4.4.2 Mă Huffman



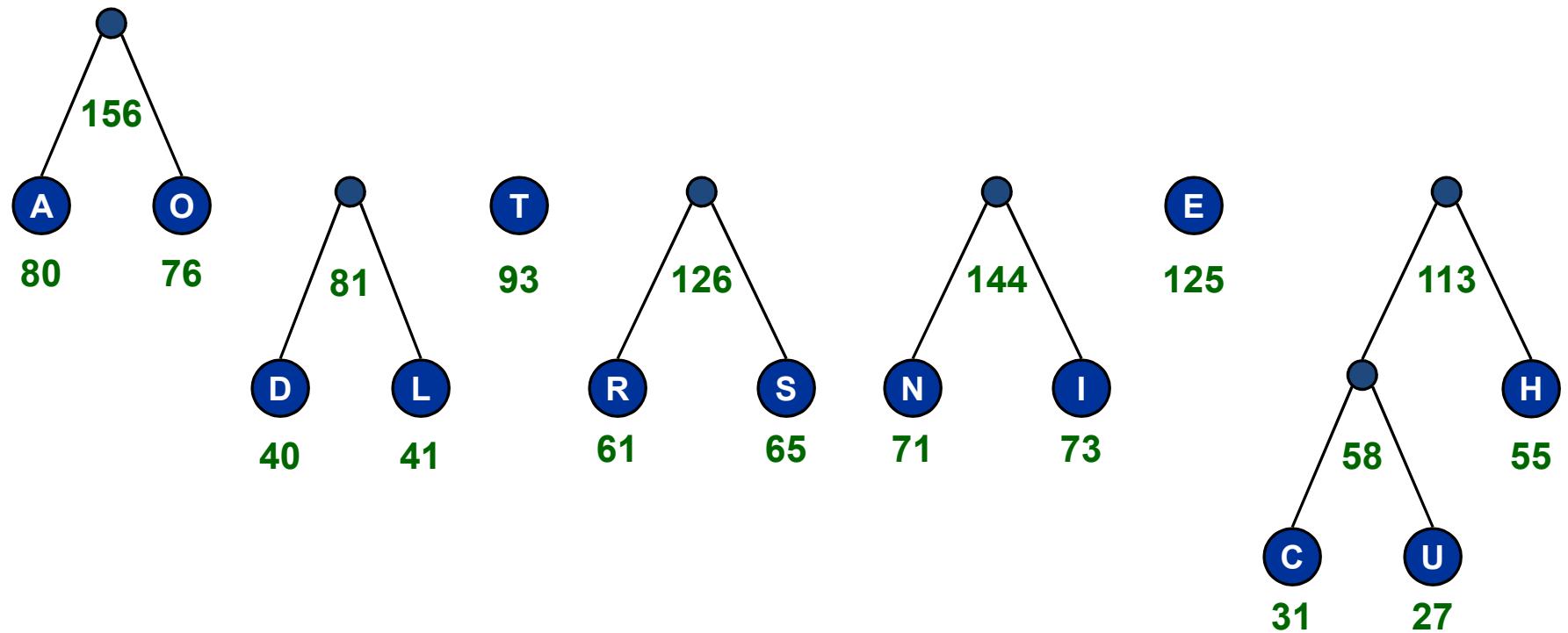
4.4.2 Mă Huffman



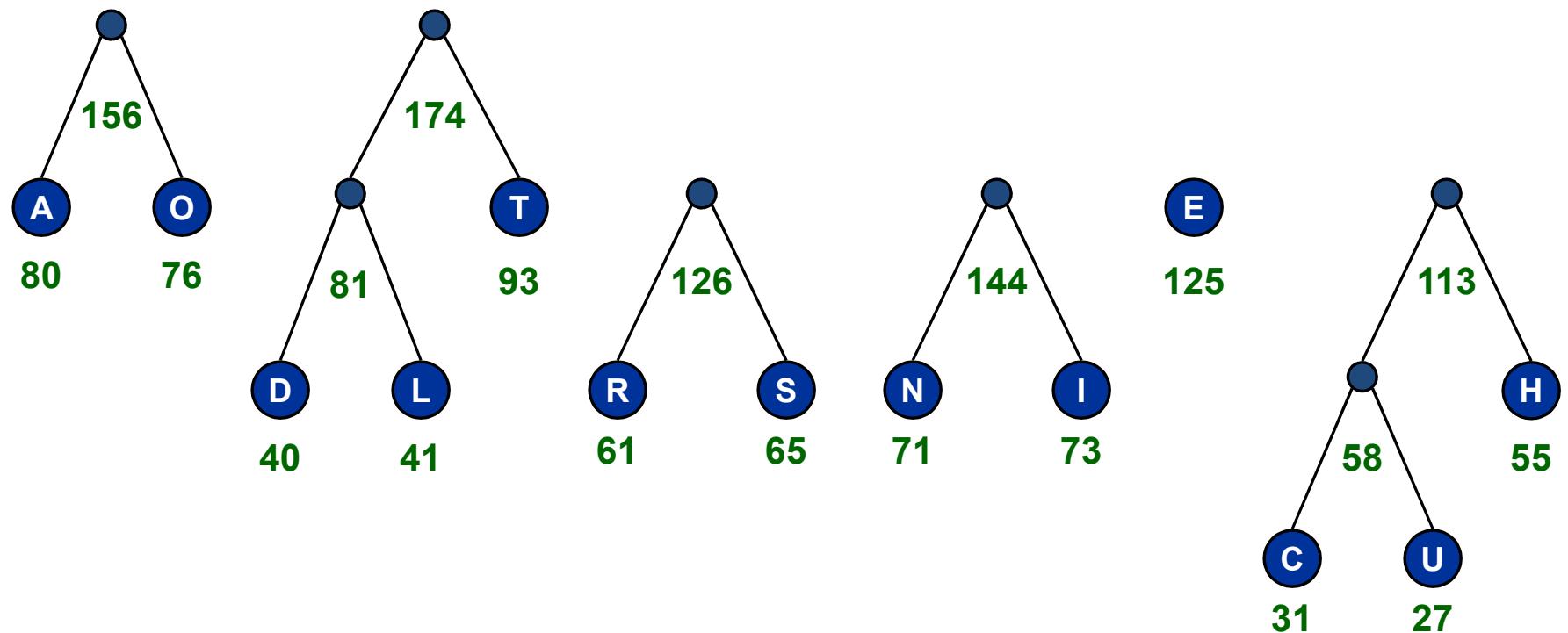
4.4.2 Mă Huffman



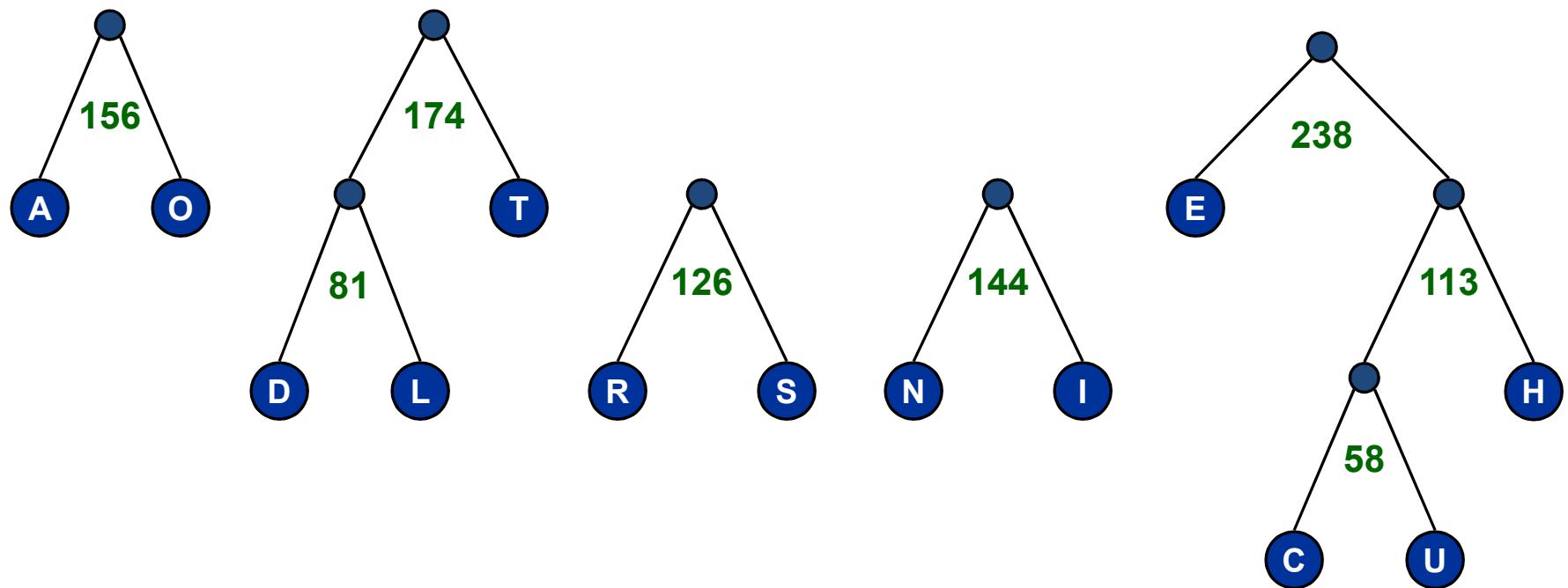
4.4.2 Mă Huffman



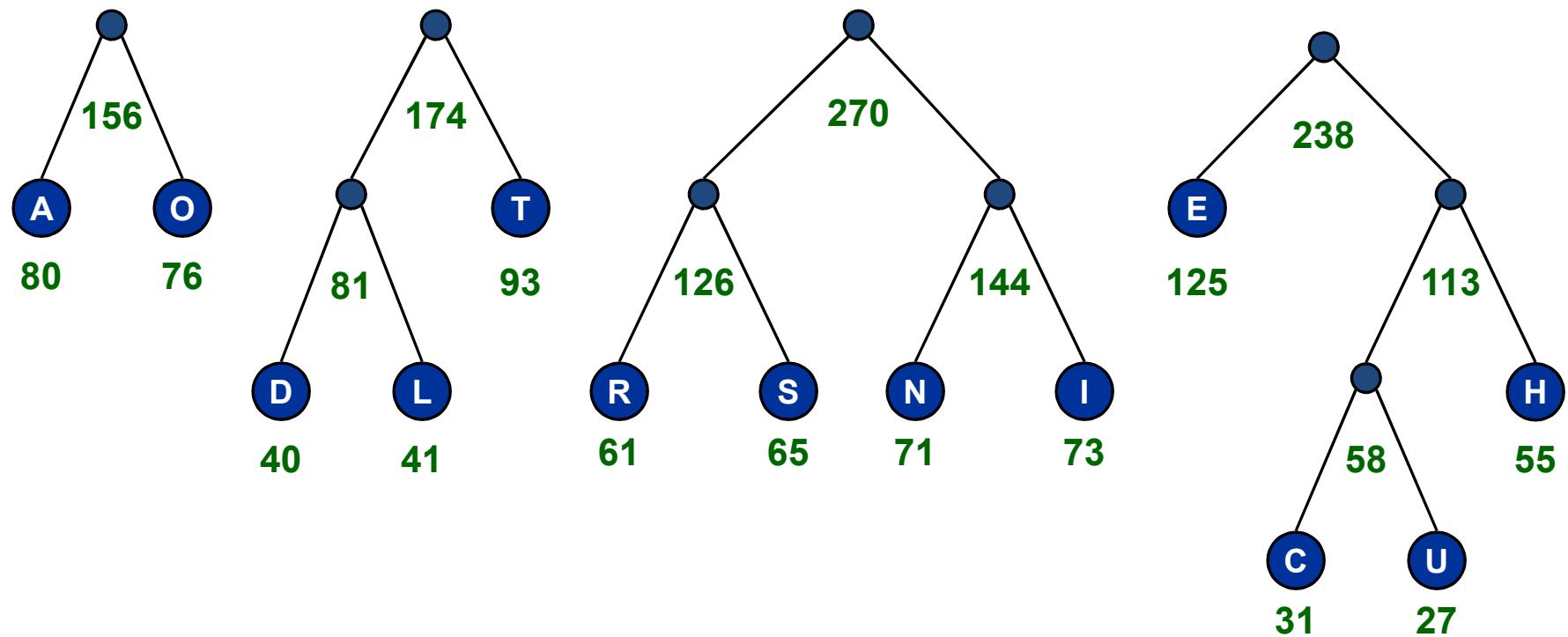
4.4.2 Mă Huffman



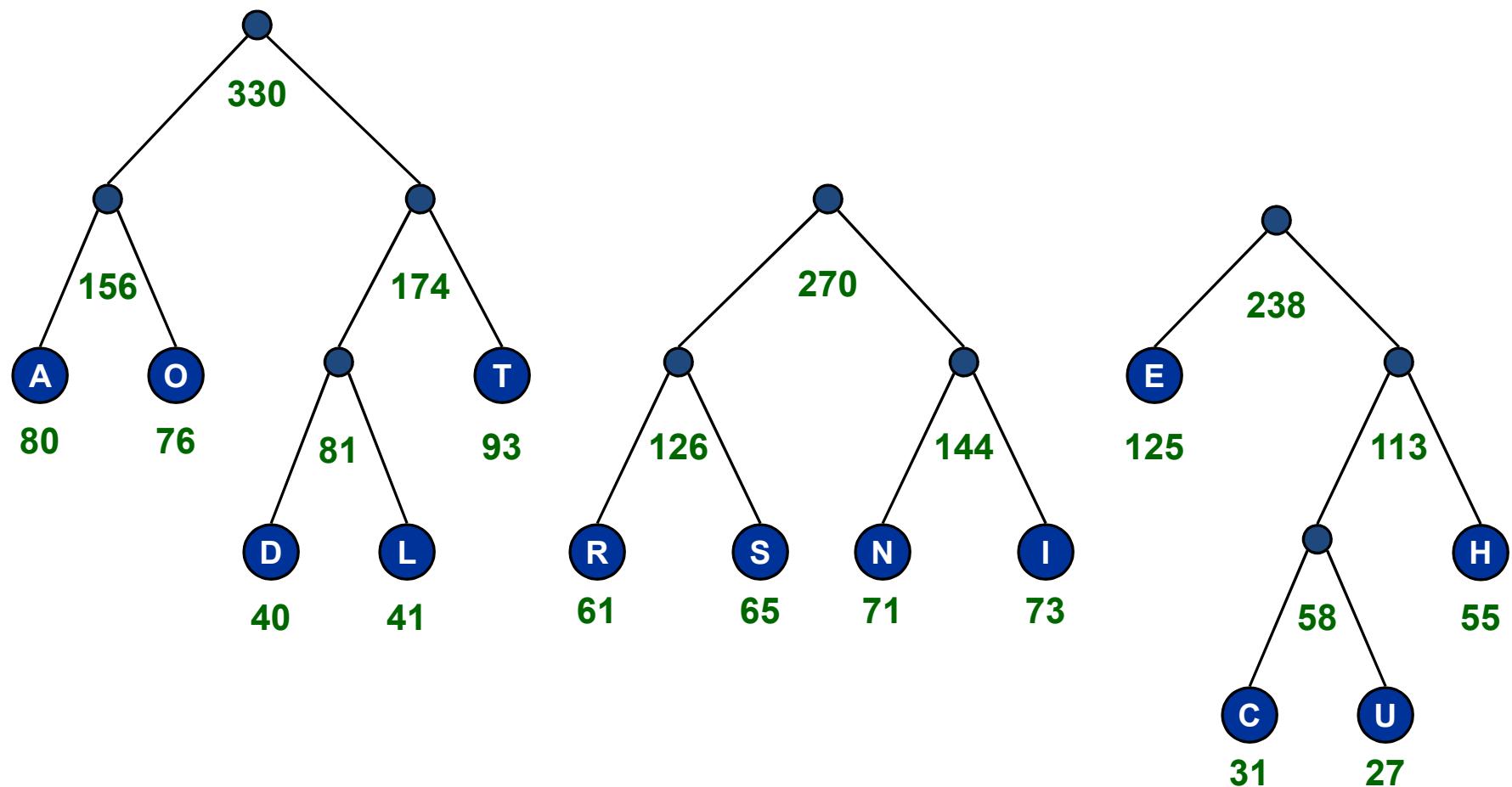
4.4.2 Mă Huffman



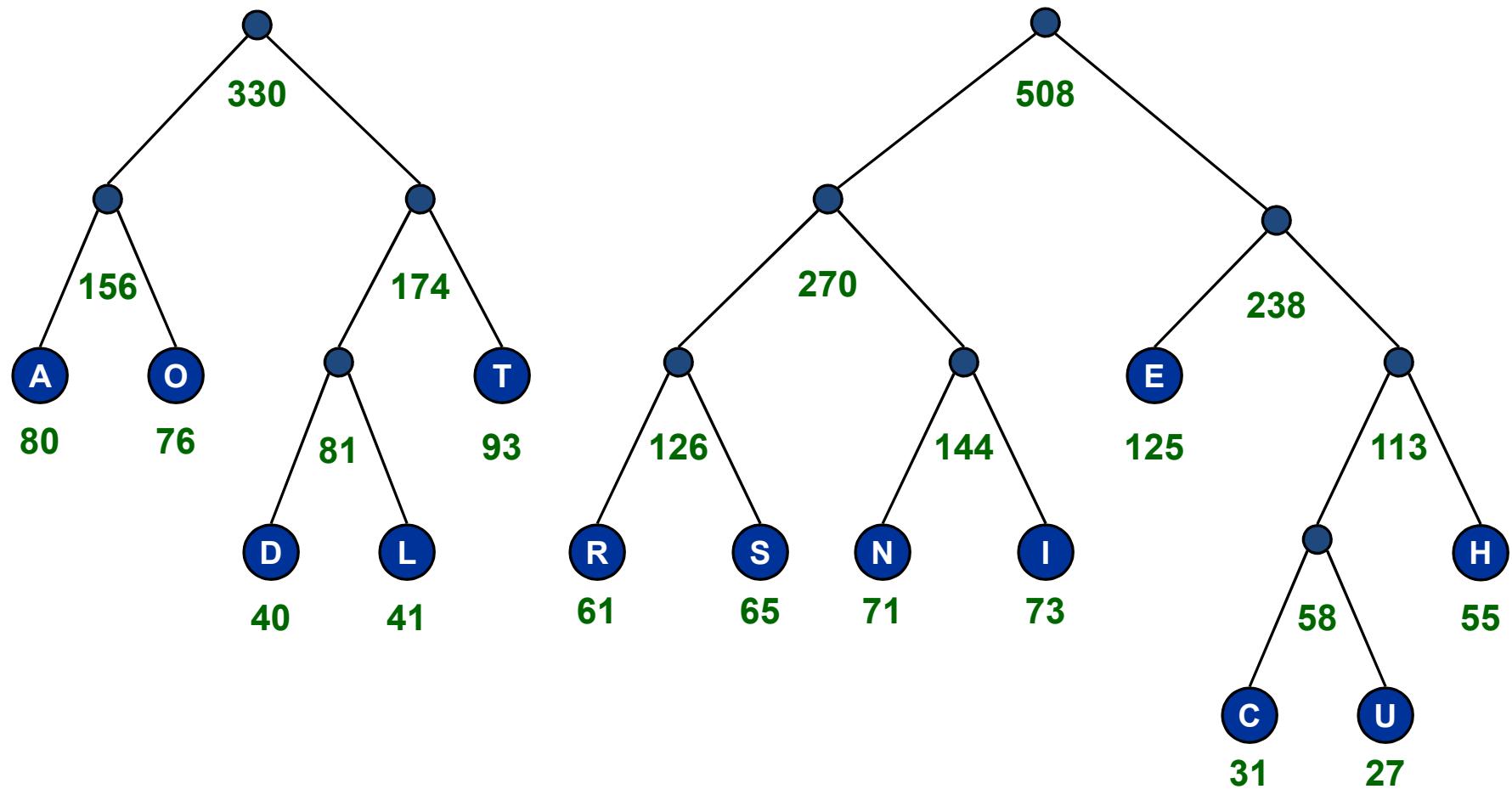
4.4.2 Mă Huffman



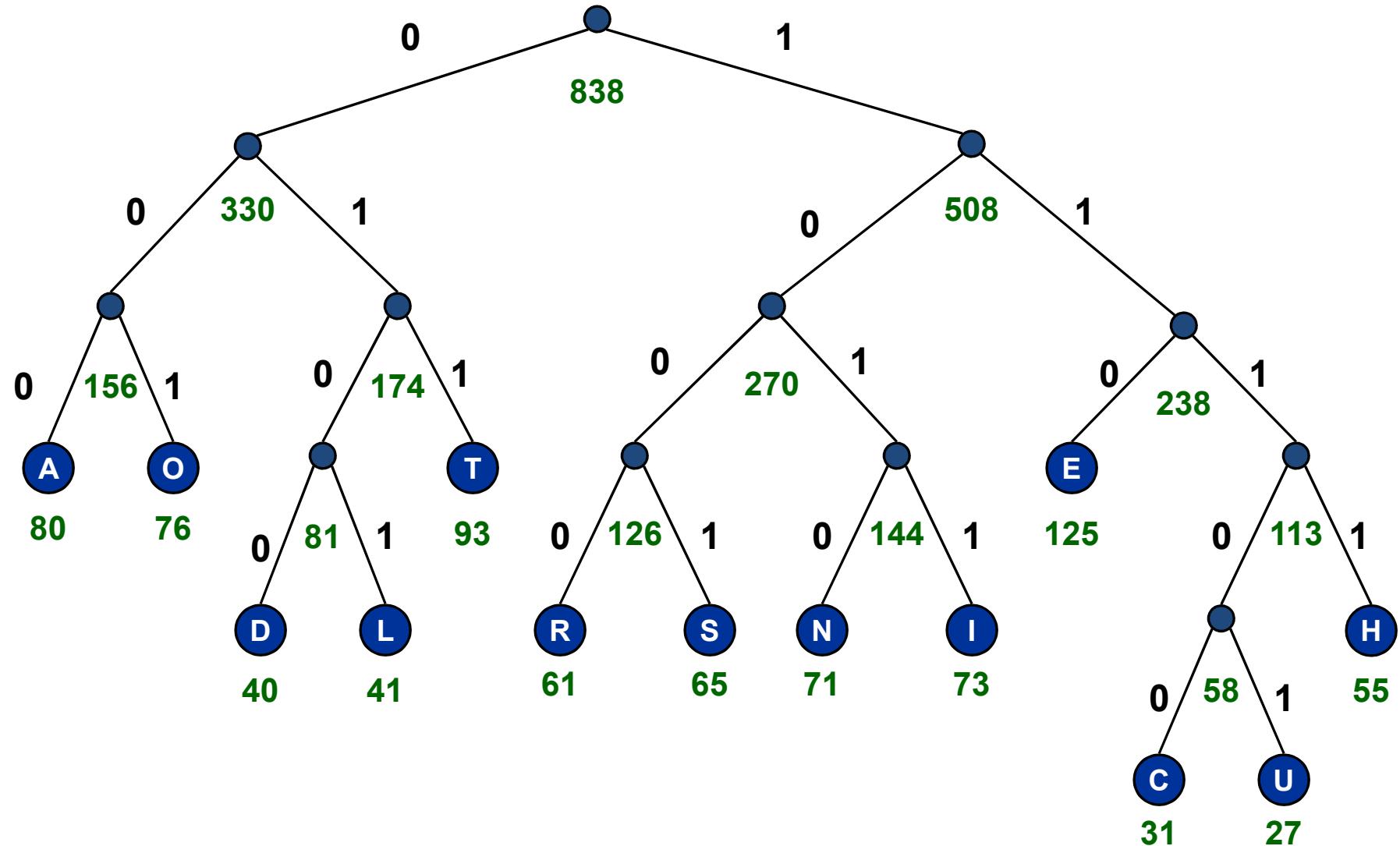
4.4.2 Mă Huffman



4.4.2 Mă Huffman



4.4.2 Mă Huffman



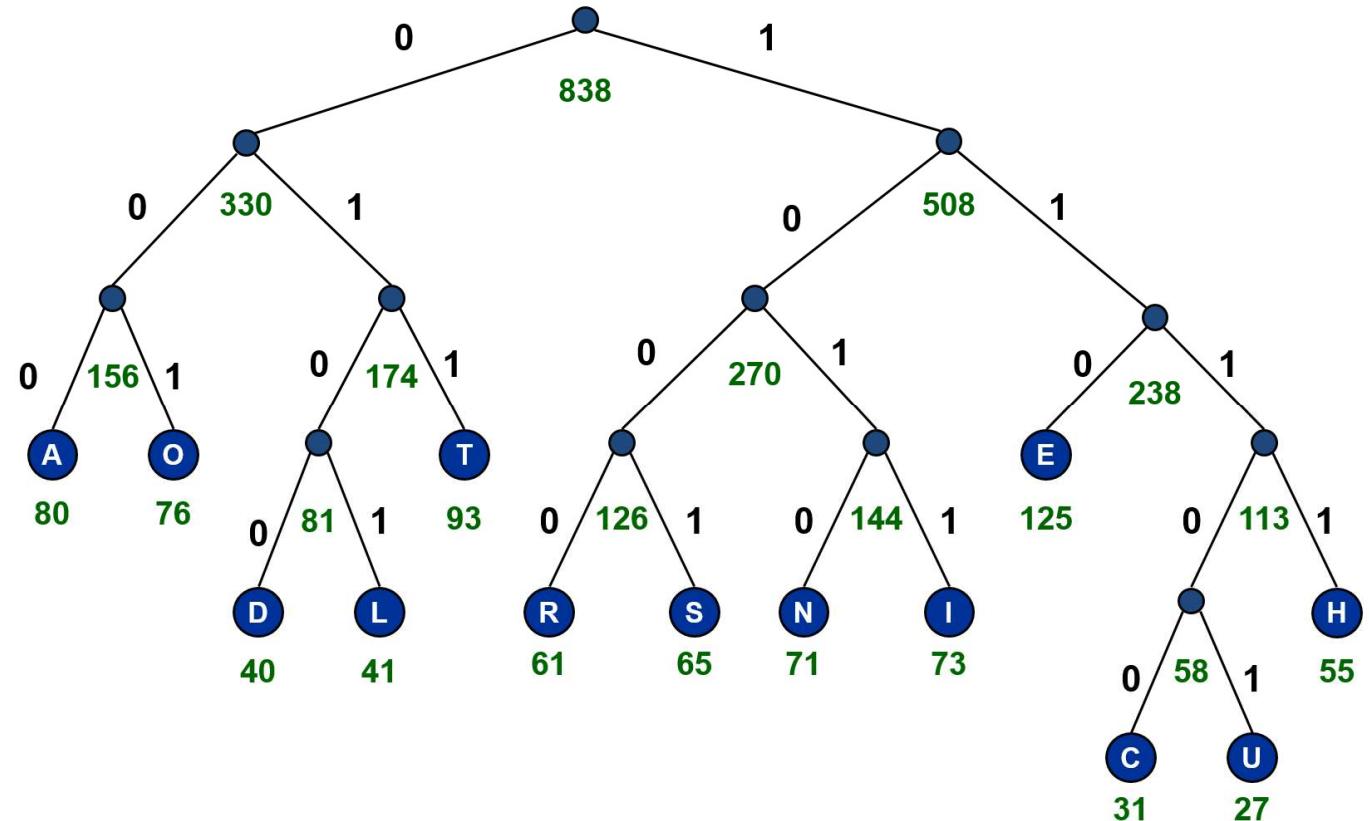
4.4.2 Mã Huffman

Char	Frequency	Fixed-length code	Huffman code
E	125	0000	110
T	93	0001	011
A	80	0010	000
O	76	0011	001
I	73	0100	1011
N	71	0101	1010
S	65	0110	1001
R	61	0111	1000
H	55	1000	1111
L	41	1001	0101
D	40	1010	0100
C	31	1011	11100
U	27	1100	11101
Sum	838	3352	3036

- Sử dụng mã Huffman: tiết kiệm bộ nhớ 10%
- Mã Huffman là ví dụ đơn giản của **mã hóa dữ liệu**

4.4.2 Mã Huffman: giải mã

- Thuật toán giải mã:
 - Đọc file nén & xây dựng cây nhị phân
 - Sử dụng cây nhị phân để giải mã
 - Đi từ nút gốc đến nút lá
- Ví dụ: Giải mã “11100000011”



4.4.2 Mã Huffman: giải mã

```
procedure Huffman_Decode(B)
//B là xâu mã hóa văn bản theo mã Huffman
begin
    <Khởi tạo P là gốc của cây Huffman>
    while <chưa đạt đến kết thúc của B> do
        begin
            x ← bit tiếp theo trong xâu B;
            If x = 0 then P ← Con trái của P
            Else                  P ← Con phải của P
            If (P là nút lá) then
                begin
                    <Hiển thị kí hiệu tương ứng với nút lá>
                    <Đặt lại P là gốc của cây Huffman>
                end;
            end;
        end;
```