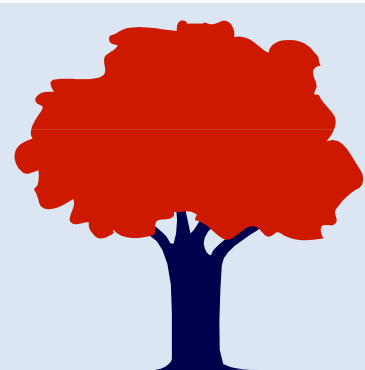
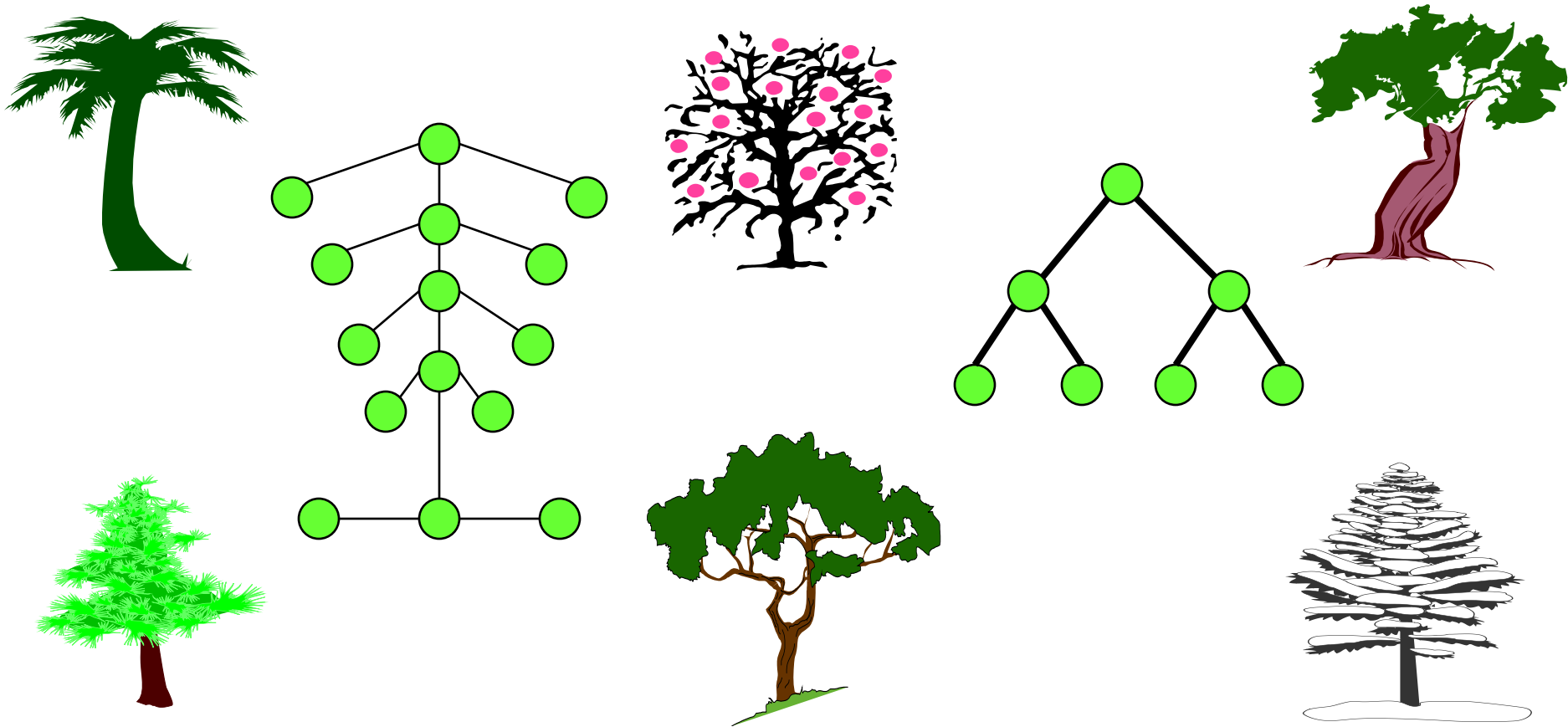


Chương 4

CÂY





Nội dung

4.1. Định nghĩa và các khái niệm

4.2. Cây nhị phân

4.3. Các ứng dụng

4.1. Định nghĩa và khái niệm

4.1.1. Định nghĩa

4.1.2. Các thuật ngữ

4.1.3. Cây có thứ tự

4.1.4. Cây có nhãn

4.1.5. ADT cây

4.1.1. Định nghĩa cây

Cây bao gồm các nút, có một nút đặc biệt được gọi là gốc (root) và các cạnh nối các nút.

Định nghĩa cây:

Bước cơ sở: Một nút r là cây và r được gọi là gốc của cây này.

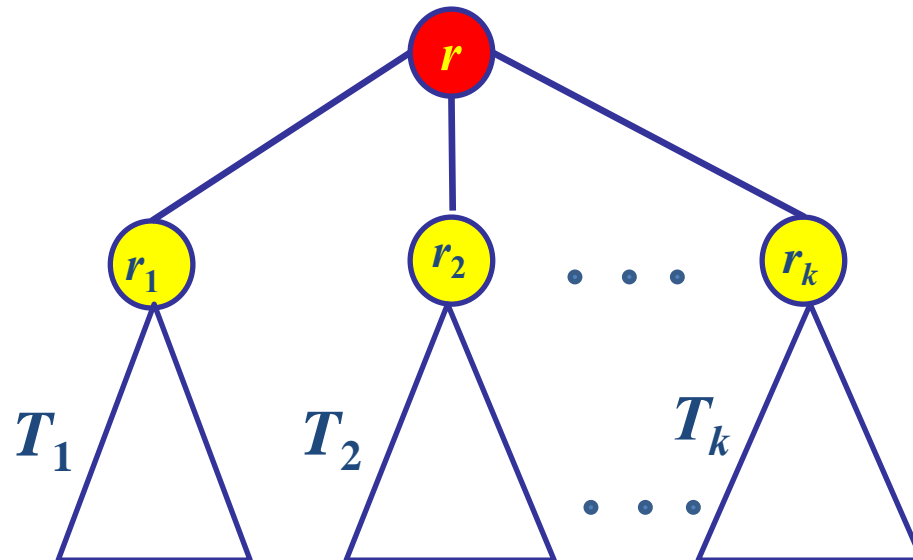
Bước đệ quy: Giả sử T_1, T_2, \dots, T_k là các cây với gốc là r_1, r_2, \dots, r_k .

Xây dựng cây mới bằng cách đặt r làm cha (parent) của các nút r_1, r_2, \dots, r_k .

Trong cây này r là gốc và 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 .

Chú ý: Nhiều khi để phù hợp ta cần định nghĩa cây rỗng (null tree) là cây không có nút nào cả.

Cấu trúc đệ quy của cây

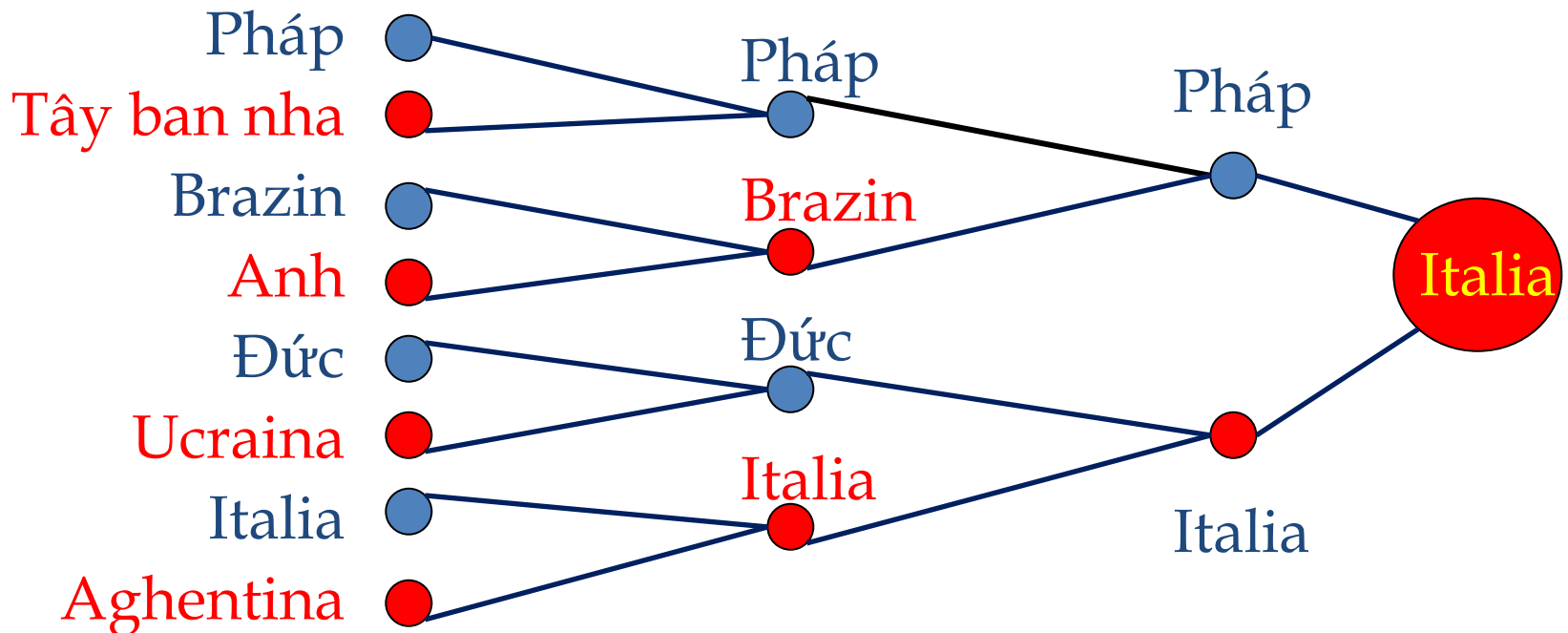


Cây trong thực tế ứng dụng

- Biểu đồ lịch thi đấu
- Cây gia phả
- Biểu đồ phân cấp quản lý hành chính.
- Cây thư mục
- Cấu trúc của một quyển sách
- Cây biểu thức
- Cây phân hoạch tập hợp
- ...

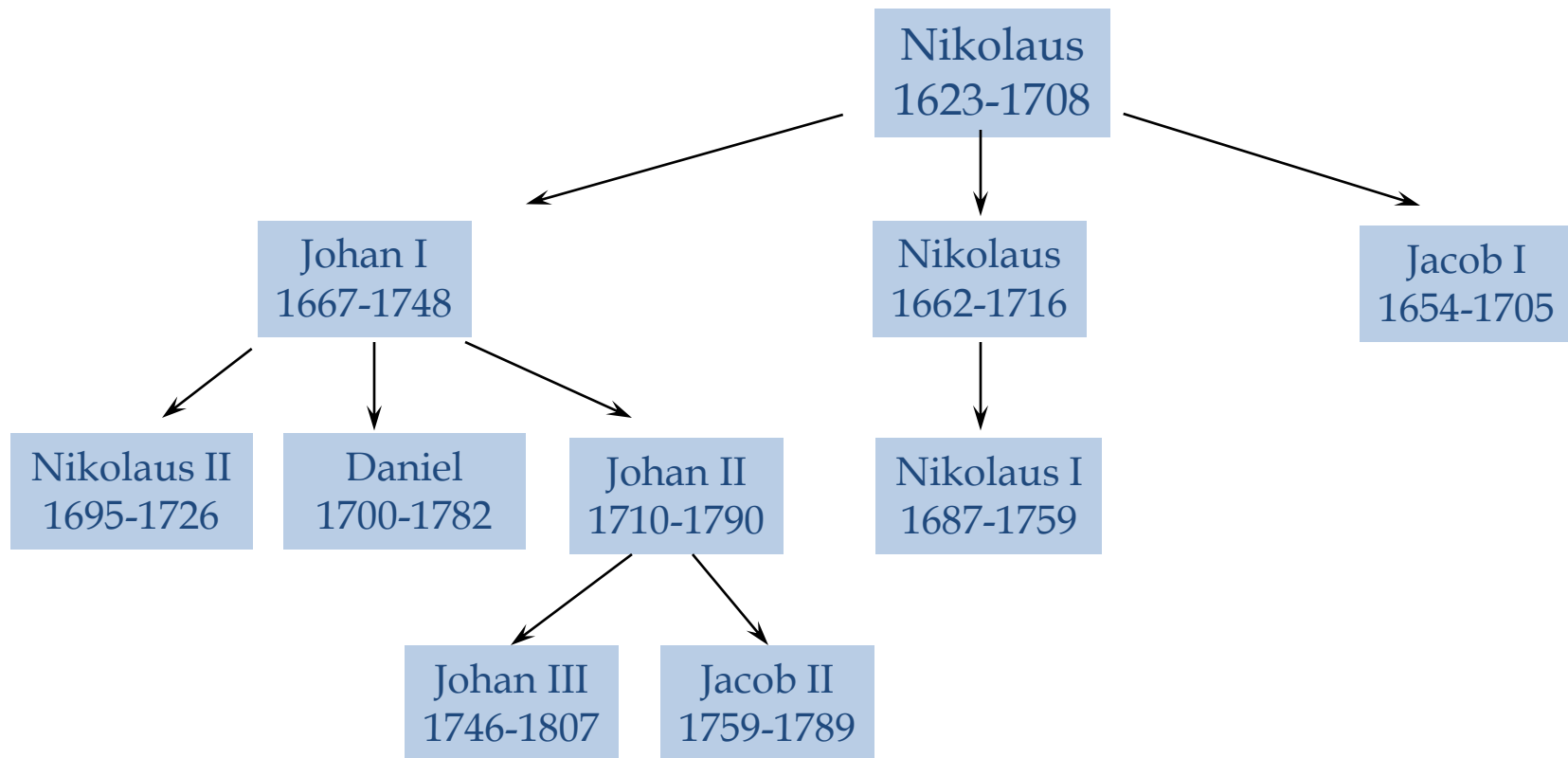
Cây lịch thi đấu

Diễn tả lịch thi đấu của các giải thể thao theo thể thức đấu loại trực tiếp, chẳng hạn vòng 2 của World Cup

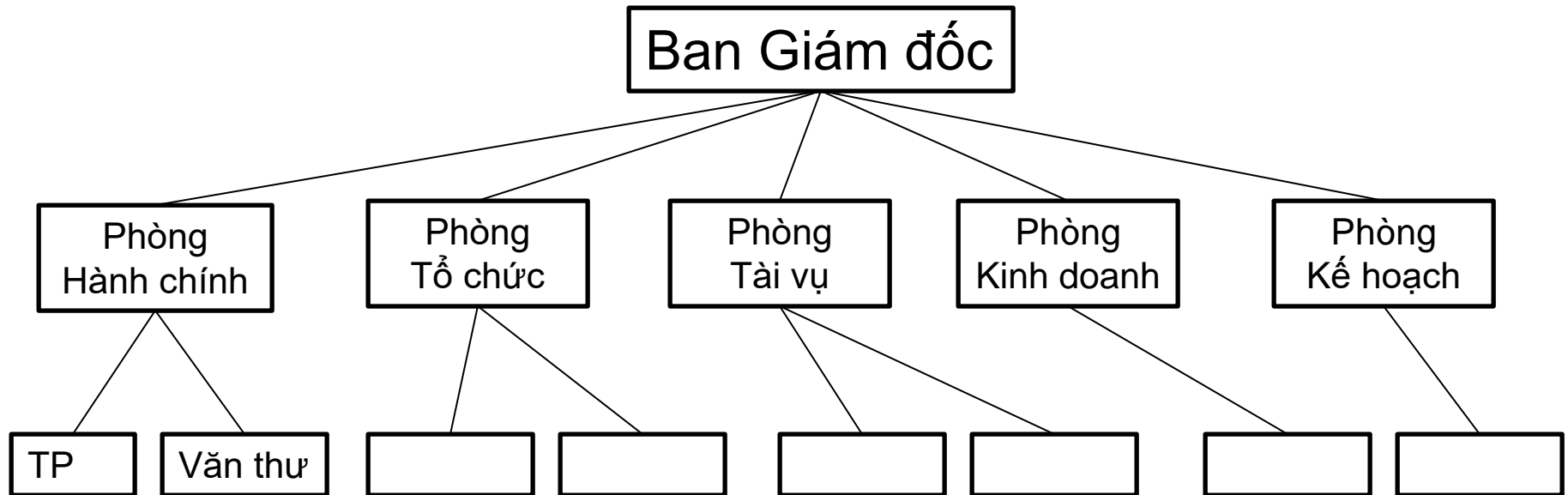


Cây gia phả

Cây gia phả của các nhà toán dòng họ Bernoulli



Cây phân cấp quản lý hành chính



Cây thư mục



Cây mục lục 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

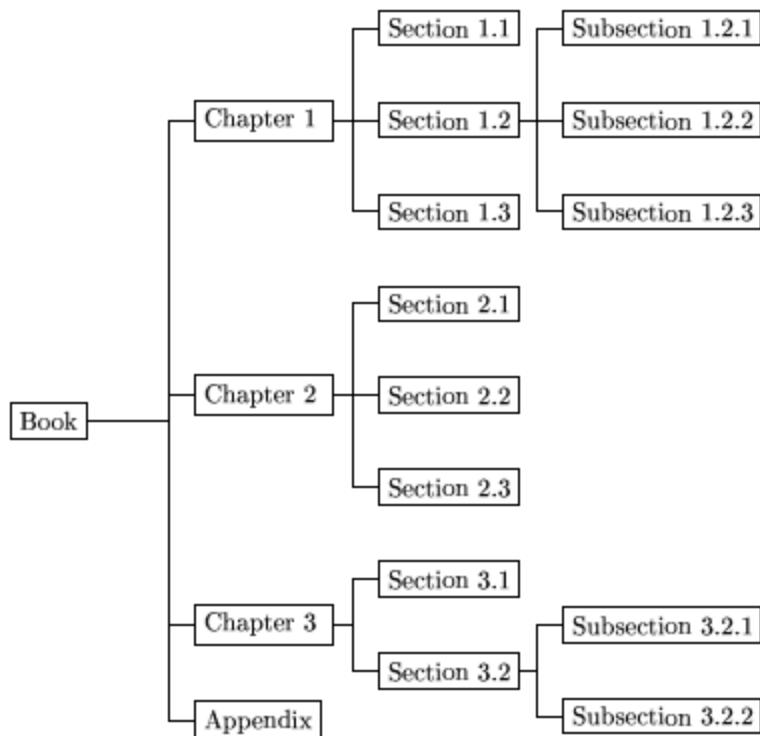
Section 3.1

Section 3.2

Subsection 3.2.1

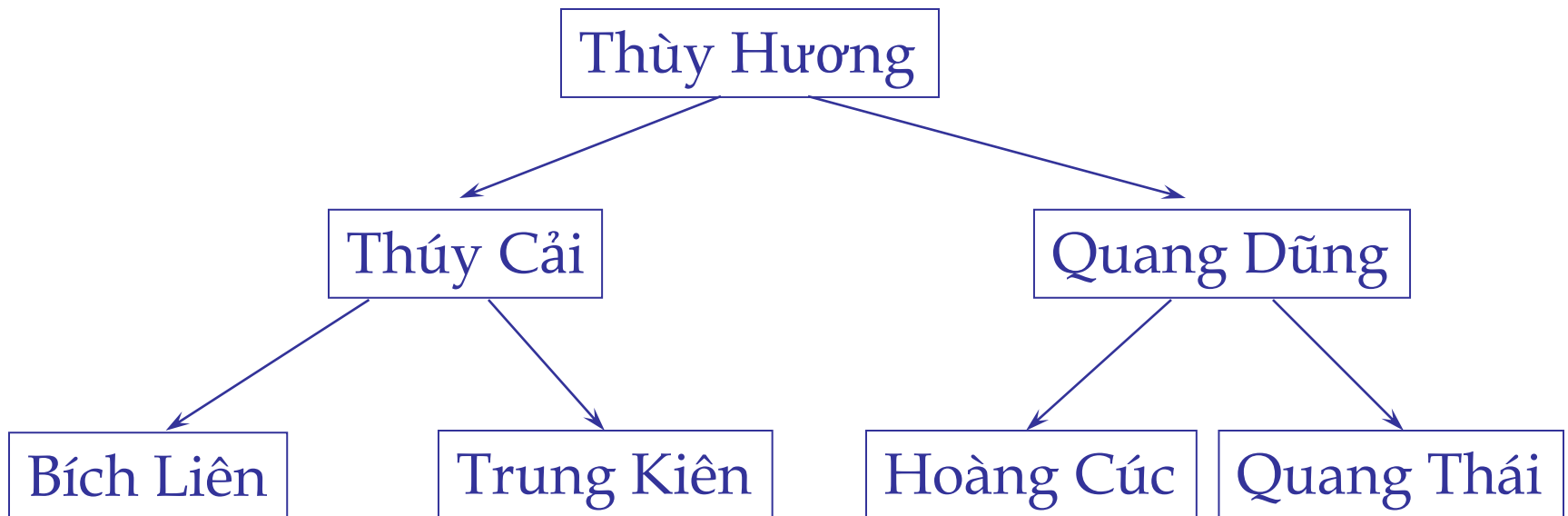
Subsection 3.2.2

Appendix

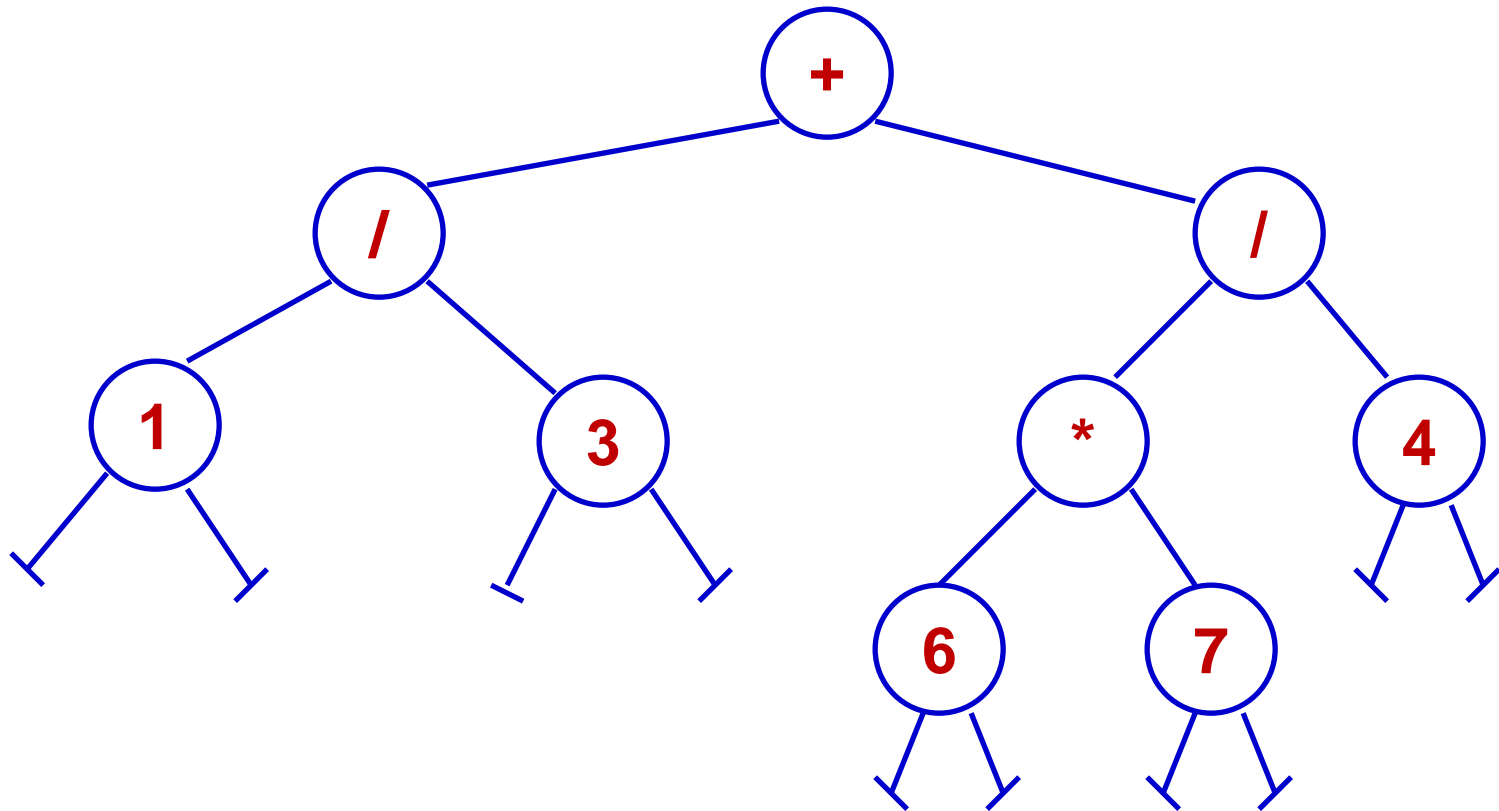


Cây gia phả ngược (Ancestor Tree)

Cây phả hệ ngược: mỗi người đều có bố mẹ. Cây này là cây nhị phân (*binary tree*).

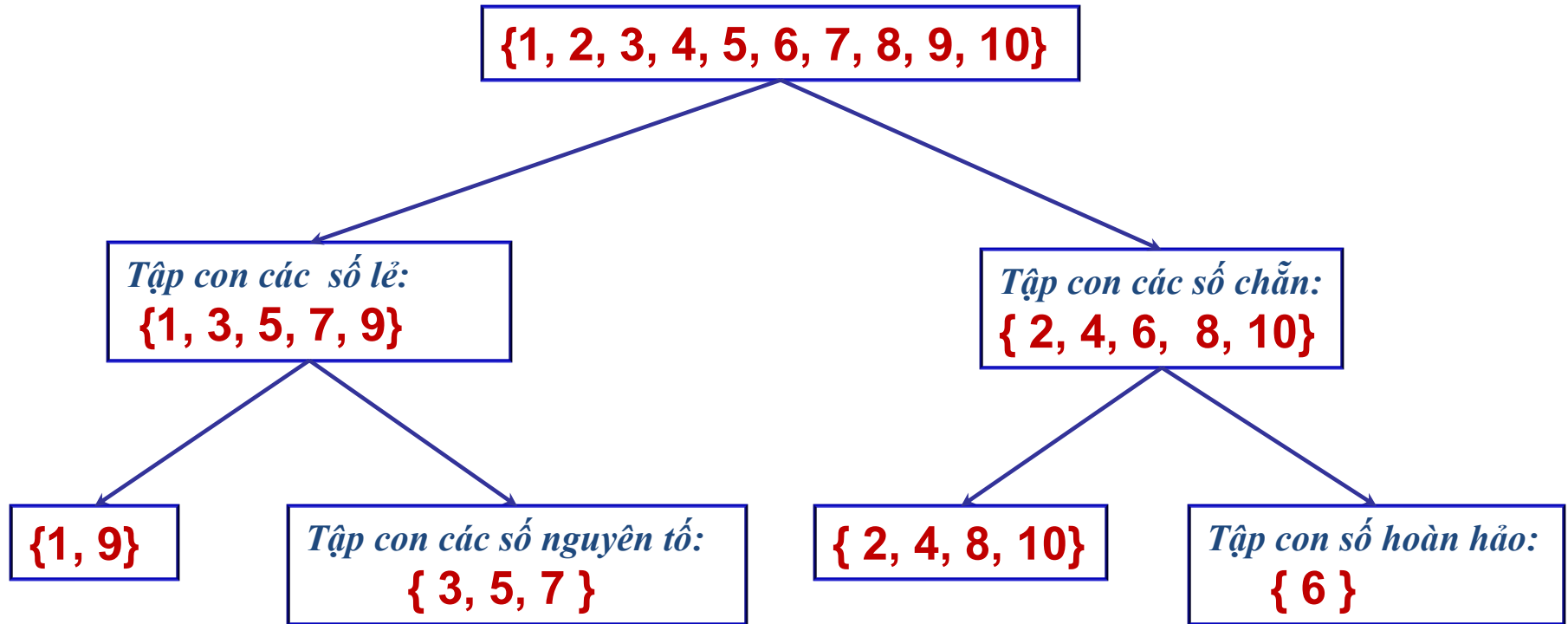


Cây biểu thức (Expression Tree)



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

Cây phân hoạch tập hợp



4.1. Định nghĩa và khái niệm

4.1.1. Định nghĩa

4.1.2. Các thuật ngữ

4.1.3. Cây có thứ tự

4.1.4. Cây có nhãn

4.1.5. ADT cây

4.1.2. Các thuật ngữ chính

- 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

Các thuật ngữ chính

- Nếu n_1, n_2, \dots, n_k là dãy nút trên cây sao cho n_i là cha của n_{i+1} với $1 \leq i < k$, thì dãy này được gọi là **đường đi** (*path*) từ nút n_1 tới nút n_k .
- **Độ dài** (*length*) của đường đi là bằng số lượng nút trên đường đi trừ bớt 1.
- Như vậy đường đi độ dài 0 là đường đi từ một nút đến chính nó.

Các thuật ngữ chính (tiếp 1)

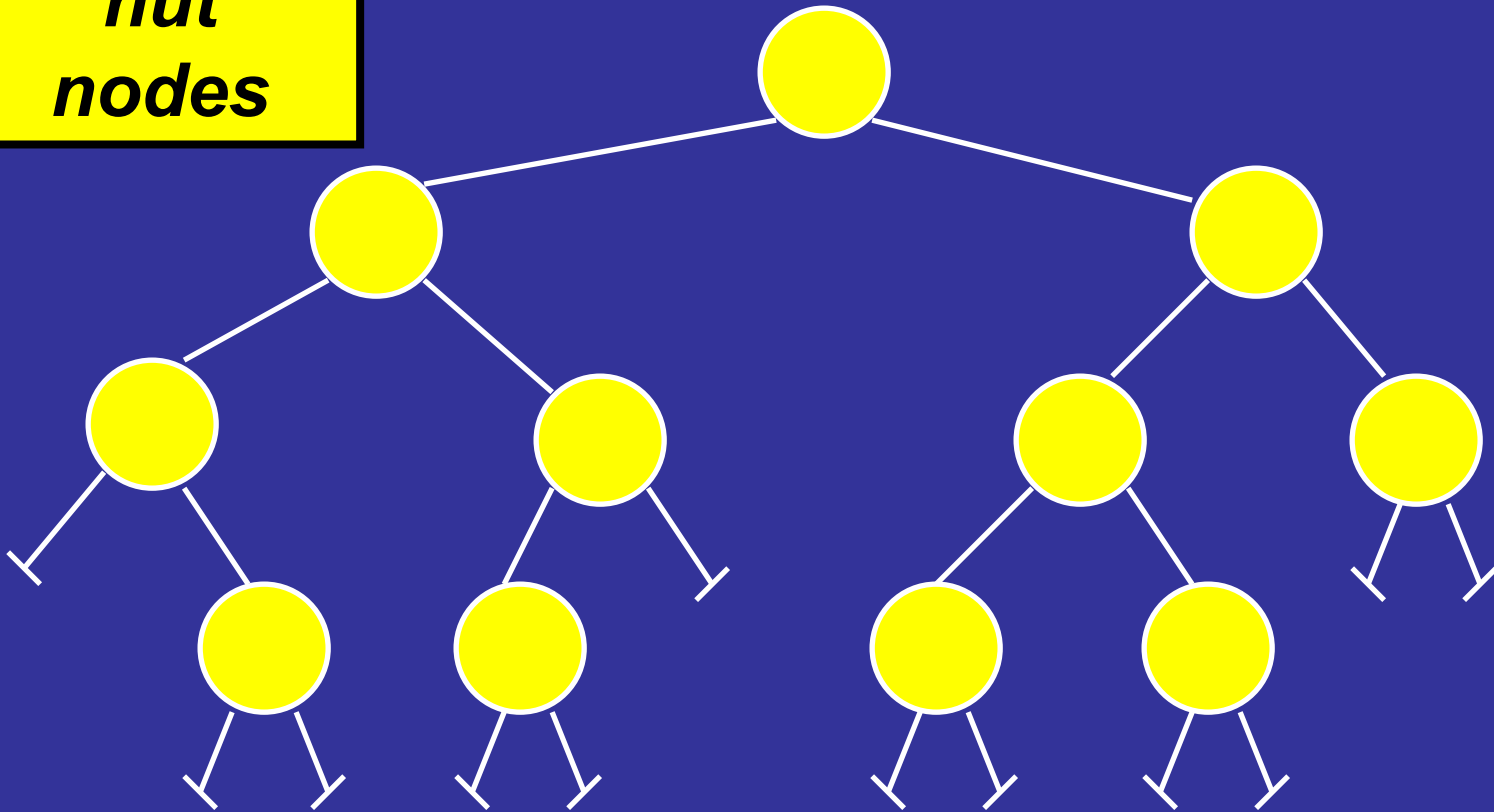
- Nếu có đường đi từ nút a tới nút b , thì a được gọi là **tổ tiên** (*ancestor*) của b , còn b được gọi là **hậu duệ** (*descendant*) của a .
- Tổ tiên (hậu duệ) của một nút khác với chính nó được gọi là tổ tiên (hậu duệ) chính thường (*proper*).
- Trong cây, gốc là nút không có tổ tiên chính thường và mọi nút khác trên cây đều là hậu duệ chính thường của nó.
- Một nút không có hậu duệ chính thường được gọi là **lá** (*leaf*).
- Các nút có cùng cha được gọi là **anh em** (*sibling*).

Các thuật ngữ chính (tiếp 2)

- **Cây con** (*subtree*) của một cây là một nút cùng với tất cả các hậu duệ của nó.
- **Chiều cao** (*height*) của nút trên cây là bằng độ dài của đường đi dài nhất từ nút đó đến lá cộng 1.
- Chiều cao của cây (*height of a tree*) là chiều cao của gốc.
- **Độ sâu/mức** (*depth/level*) của nút là bằng 1 cộng với độ dài của đường đi duy nhất từ gốc đến nó.

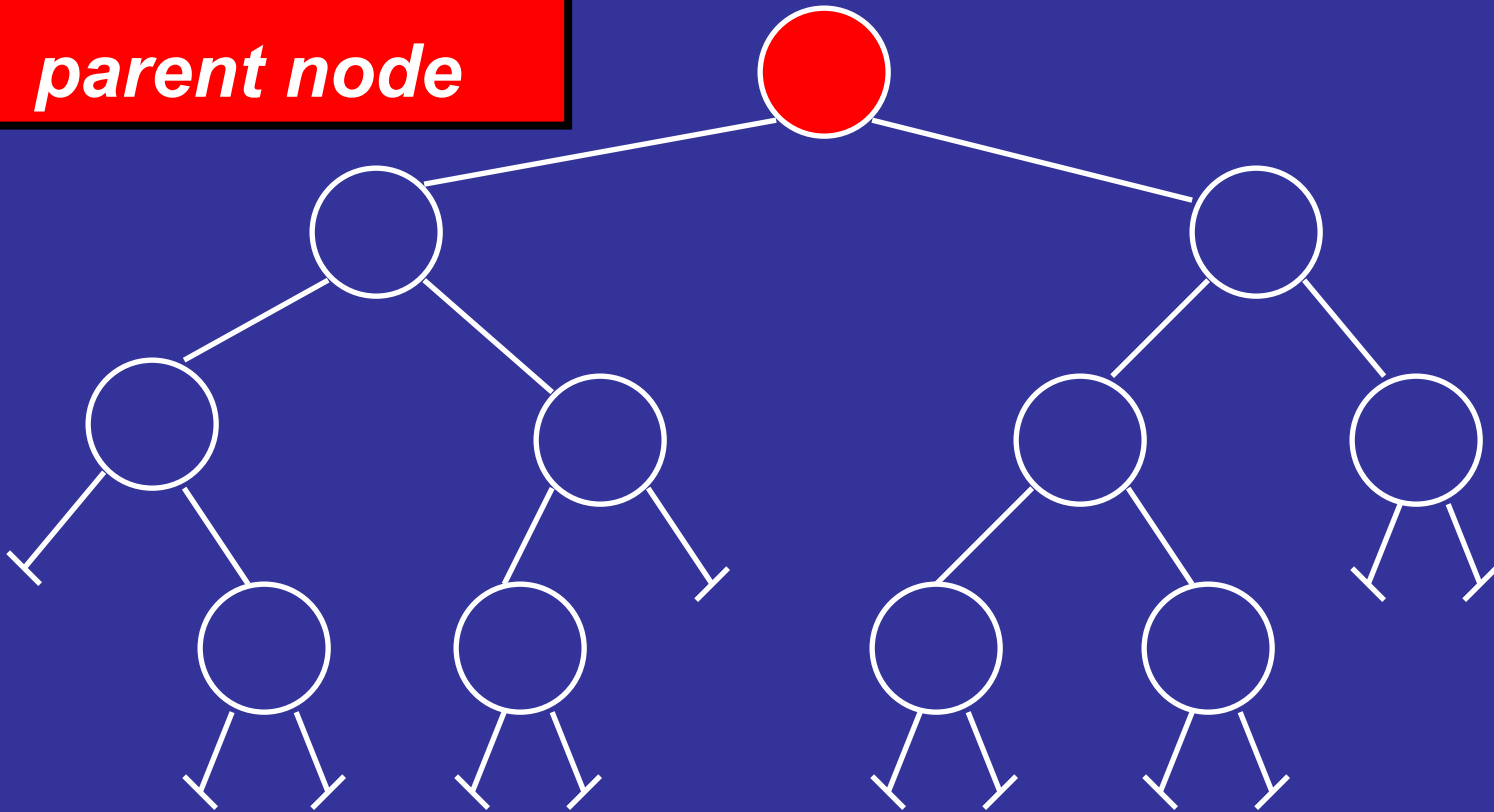
Thuật ngữ

nút
nodes

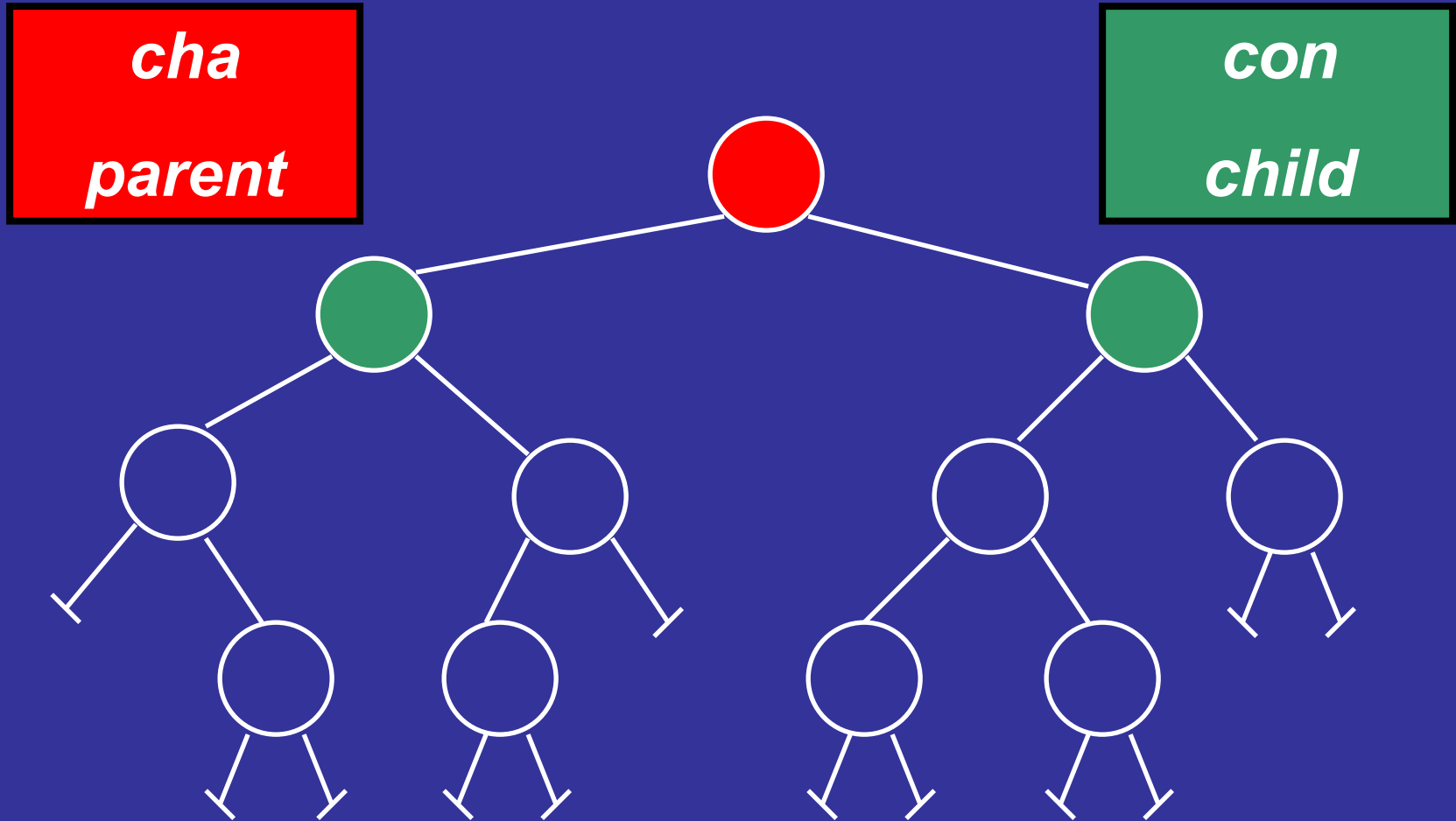


Thuật ngữ

Nút cha
parent node



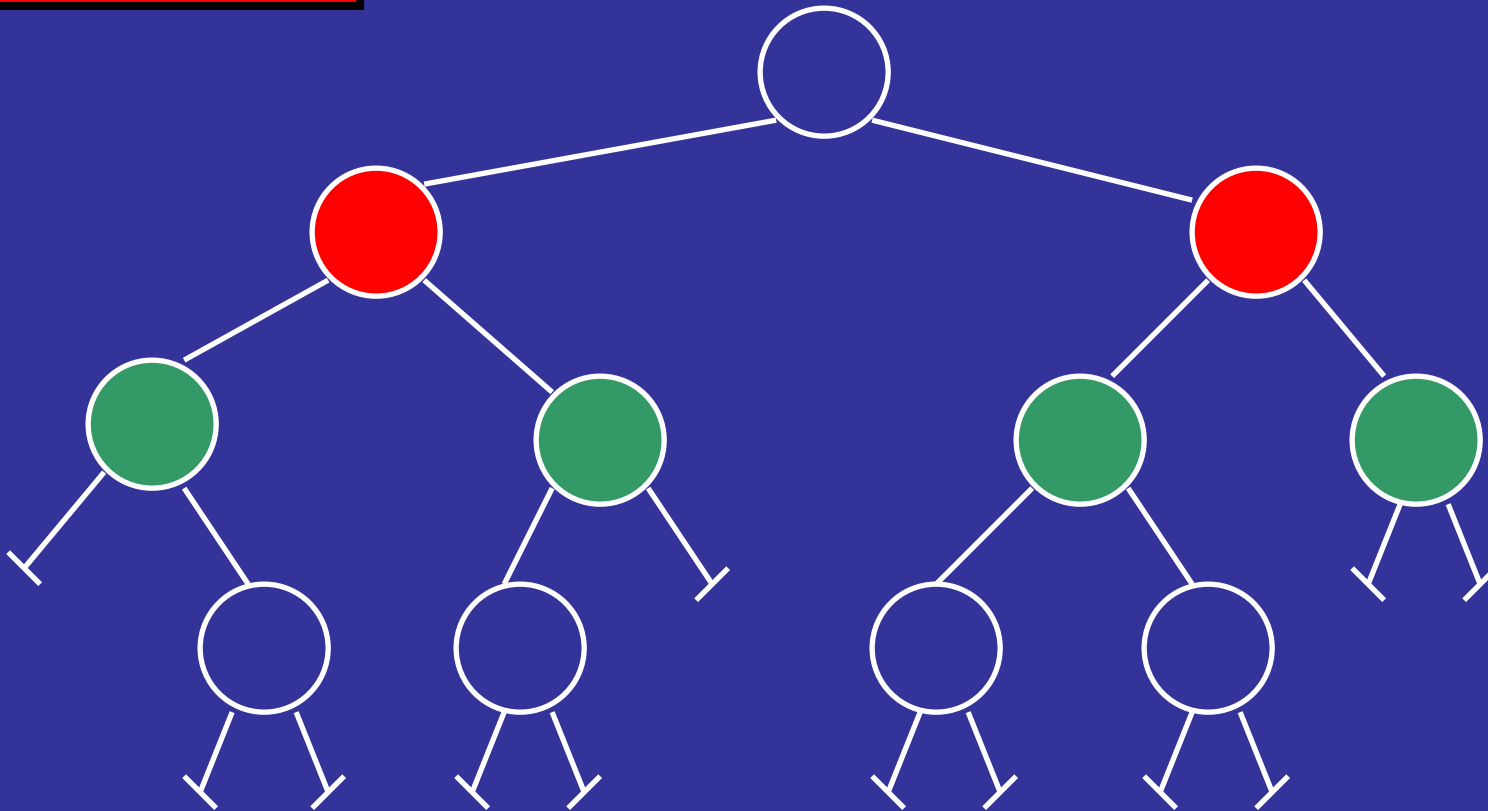
Thuật ngữ



Thuật ngữ

cha

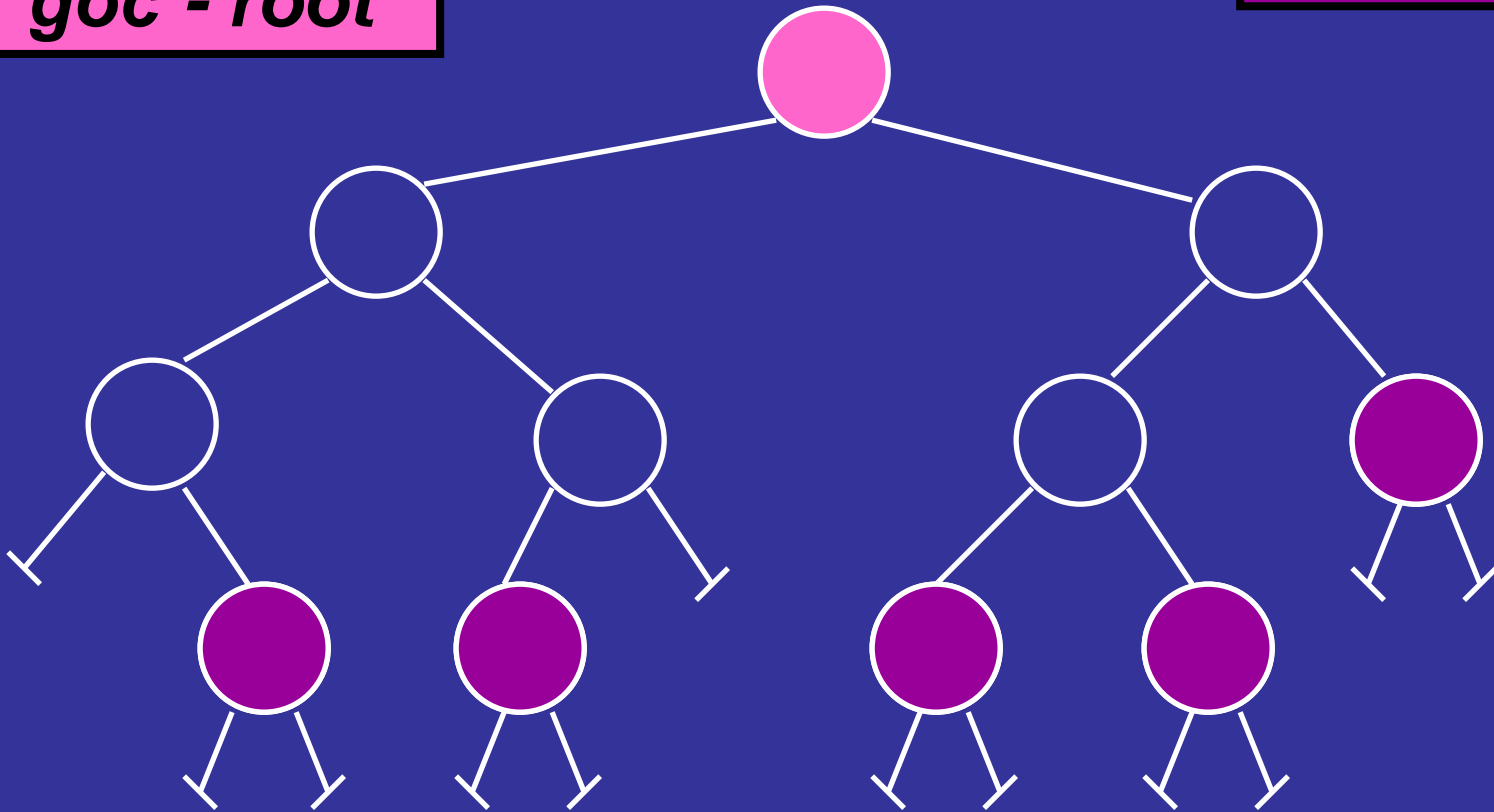
con



Thuật ngữ

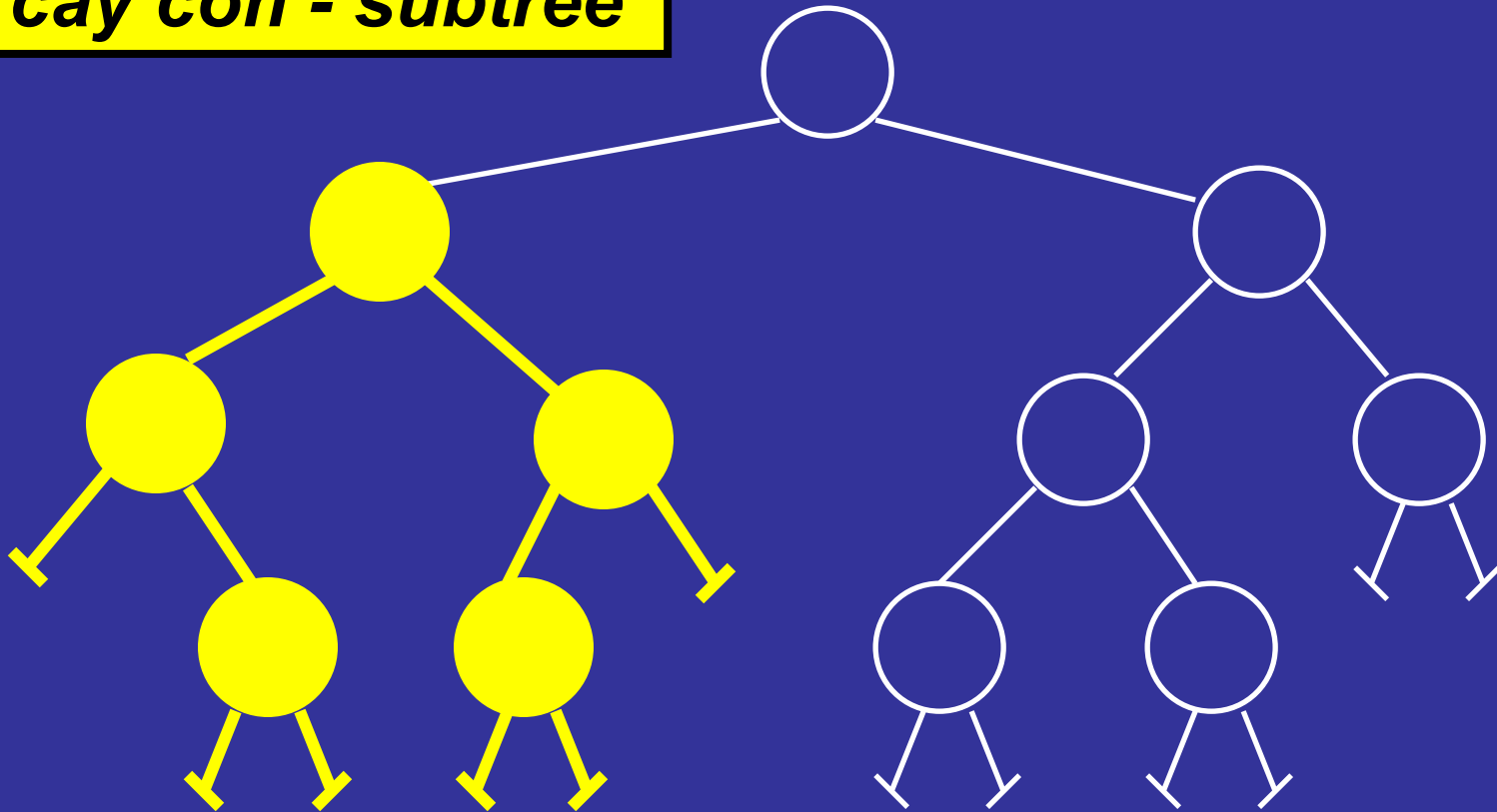
gốc - root

lá - leaf



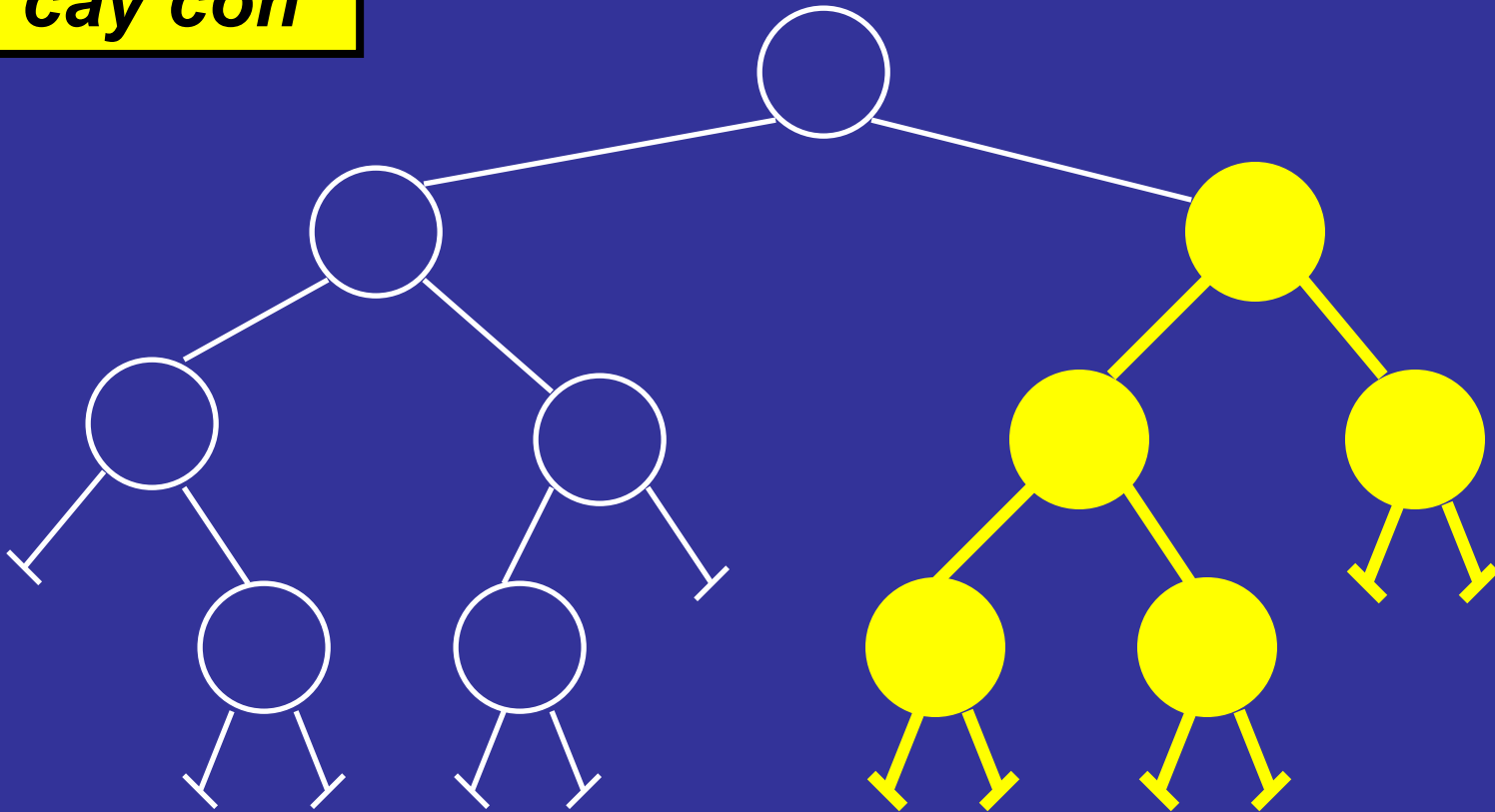
Thuật ngữ

cây con - subtree



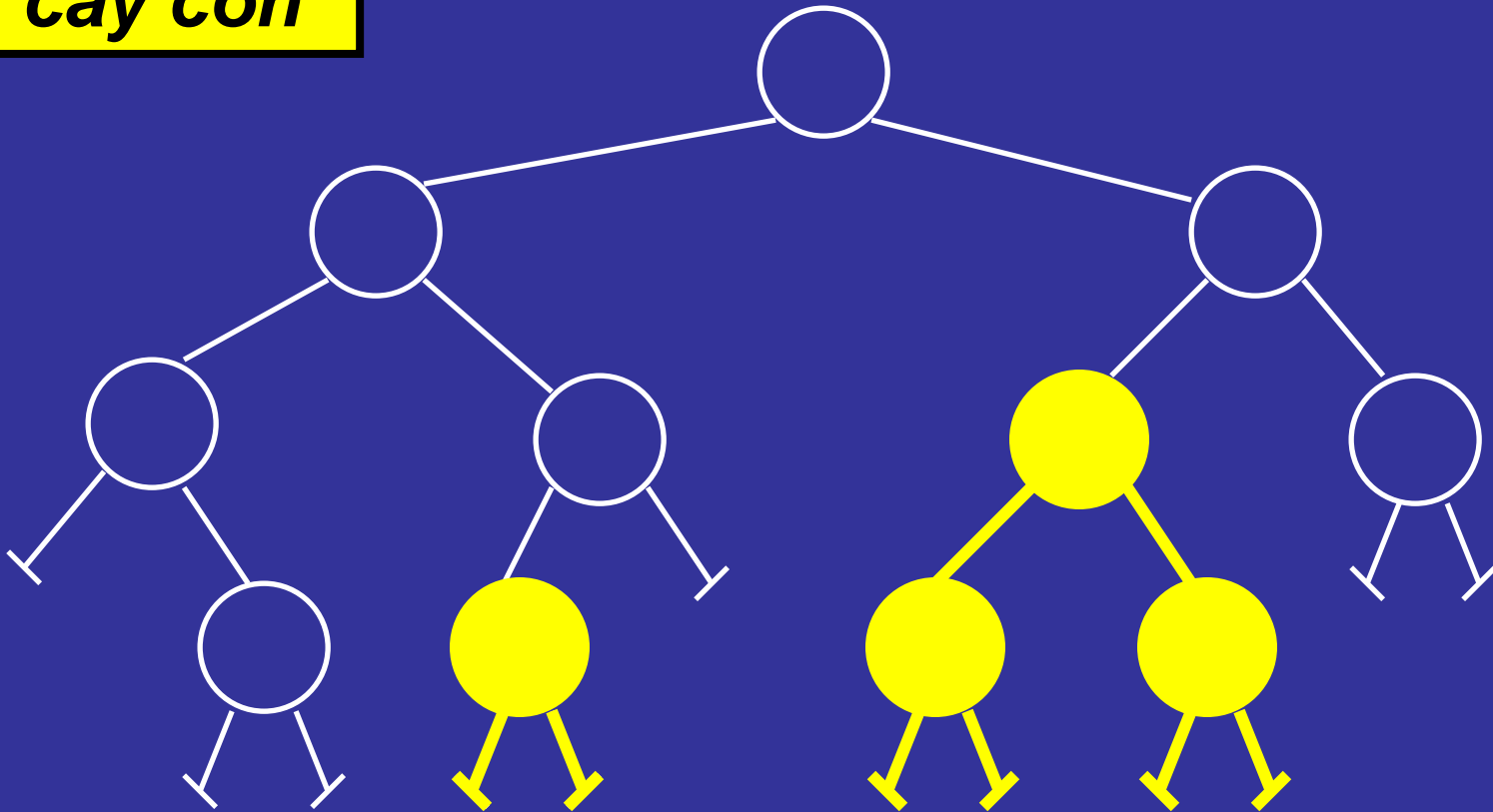
Thuật ngữ

cây con

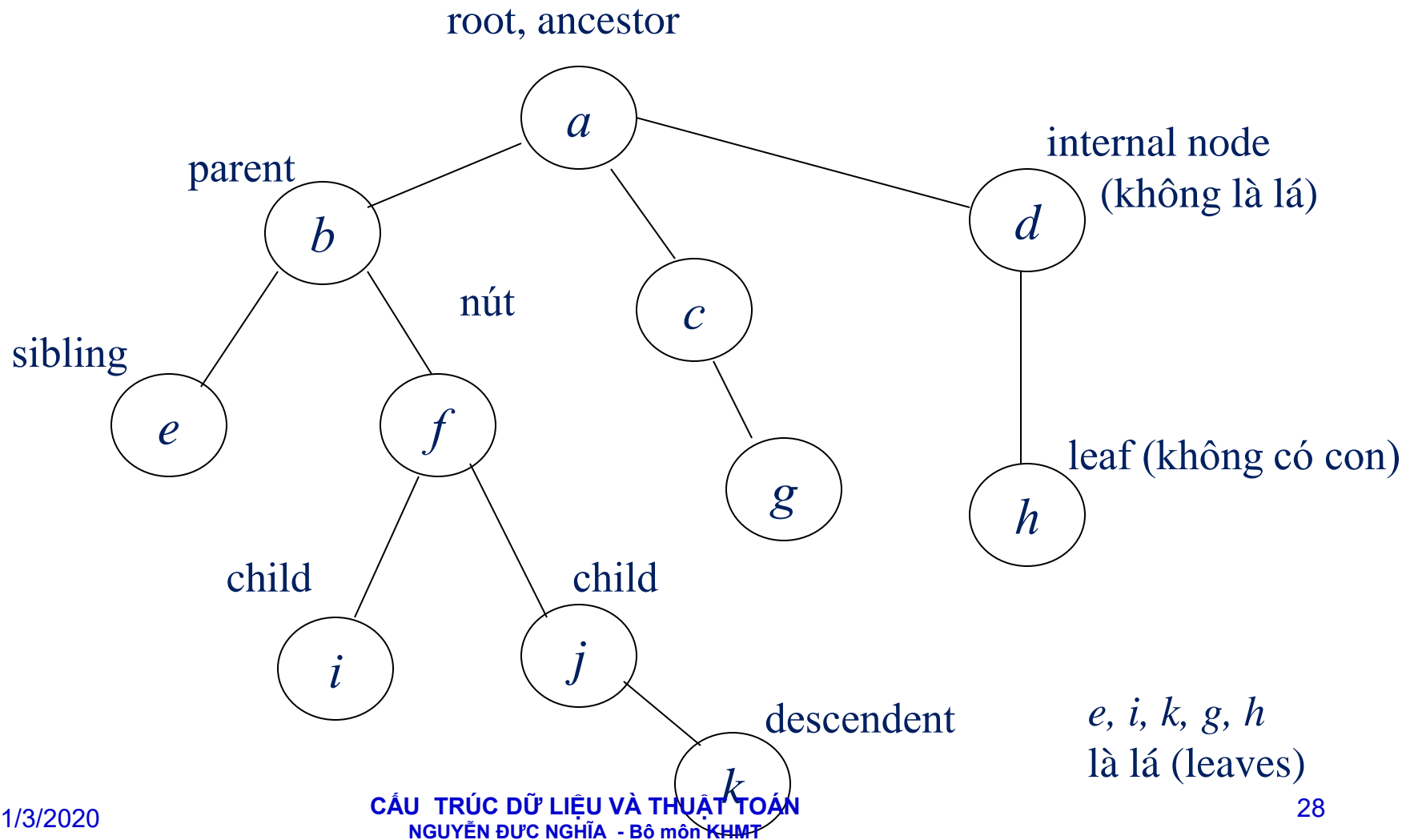


Thuật ngữ

cây con



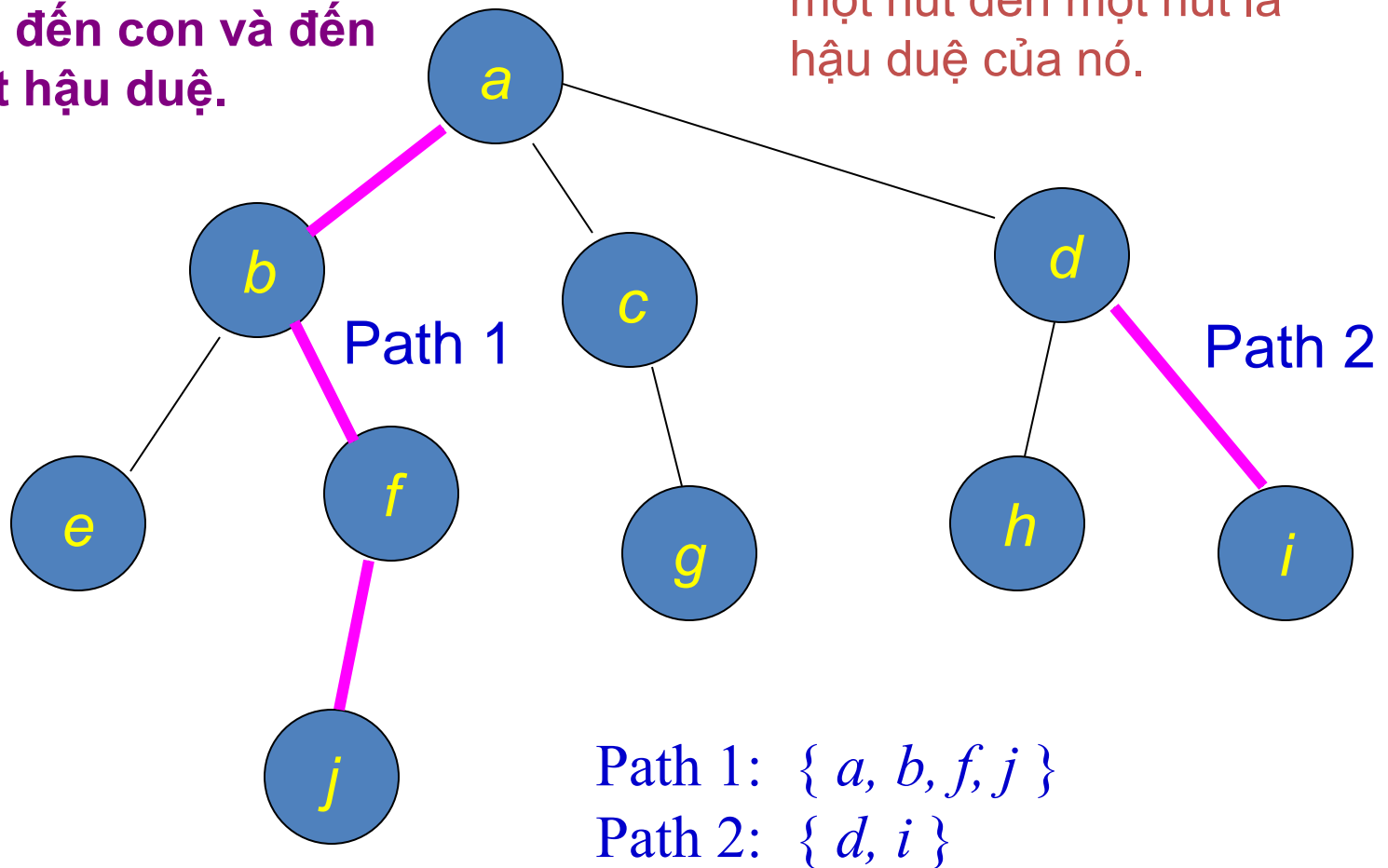
Các thuật ngữ với cây có gốc



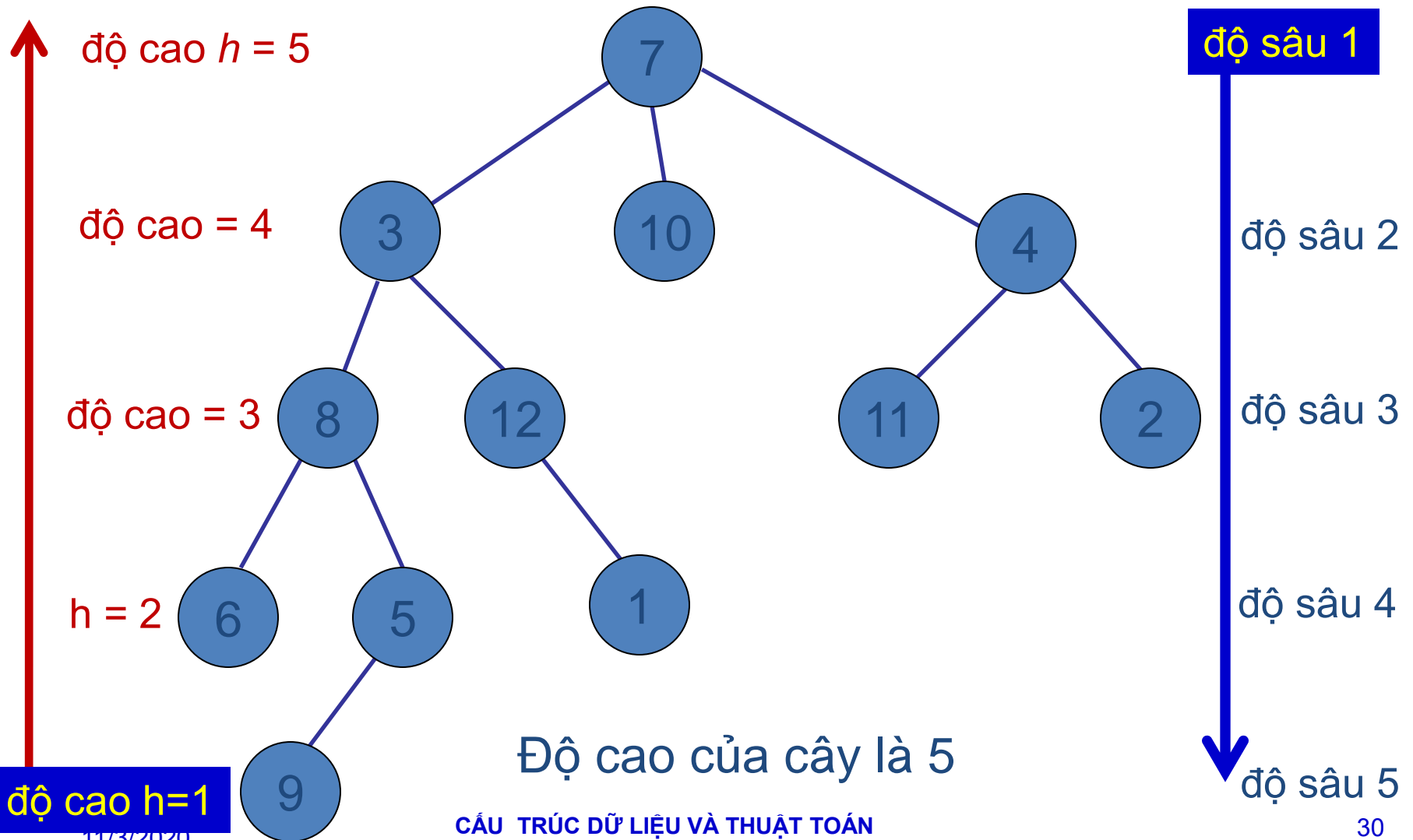
Đường đi trên cây

Từ cha đến con và đến các nút hậu duệ.

Có duy nhất 1 đường đi từ một nút đến một nút là hậu duệ của nó.



Độ cao (height) và độ sâu/mức (depth/level)



How We View a Tree



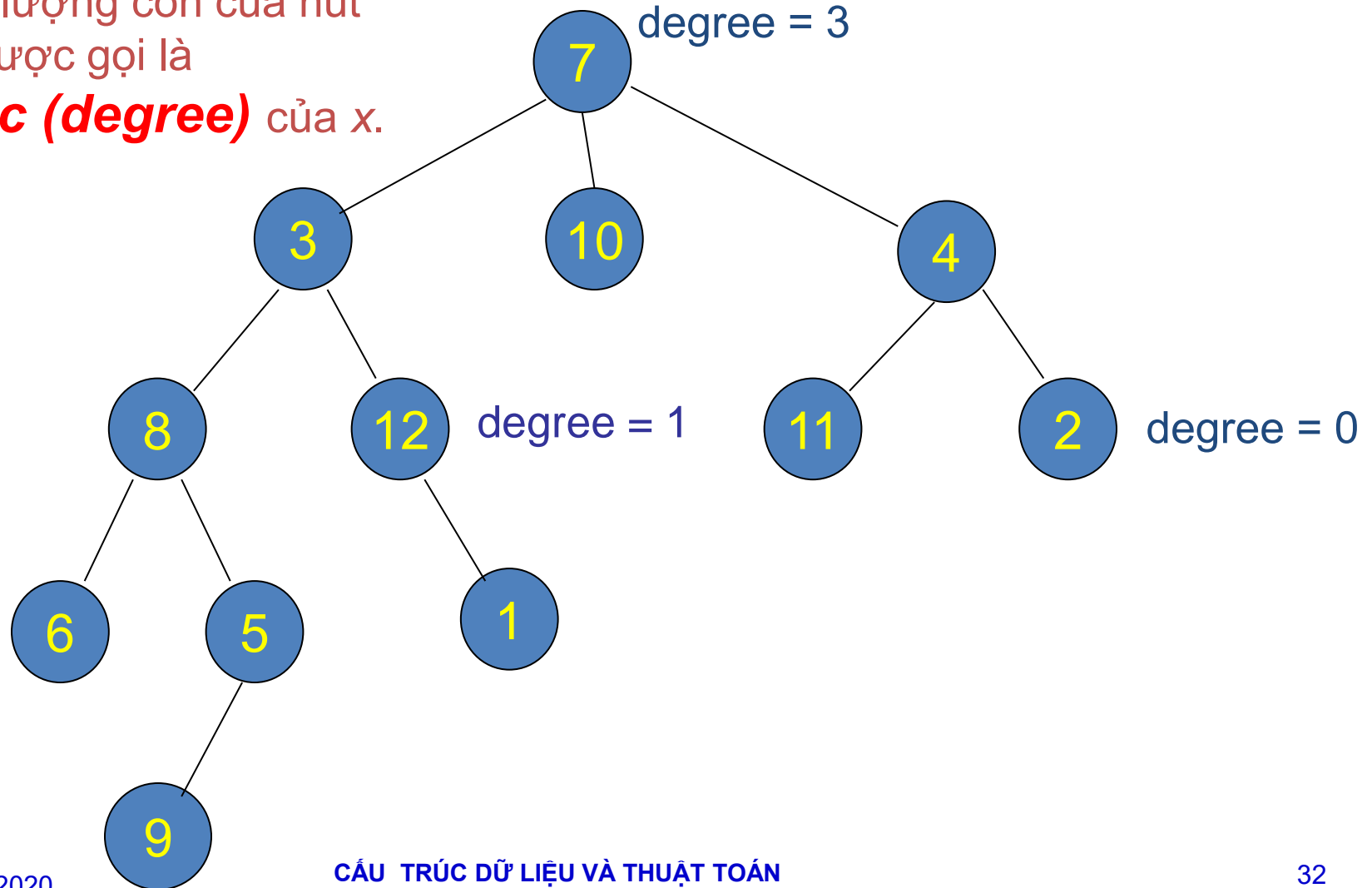
Nature Lovers View



Computer Scientists View

Bậc (Degree)

Số lượng con của nút x được gọi là **bậc (degree)** của x.



4.1. Định nghĩa và khái niệm

4.1.1. Định nghĩa

4.1.2. Các thuật ngữ

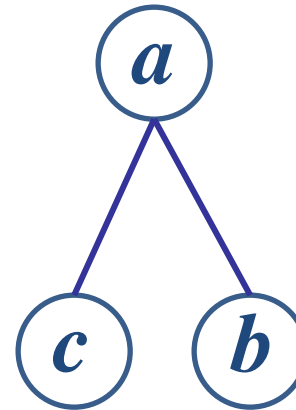
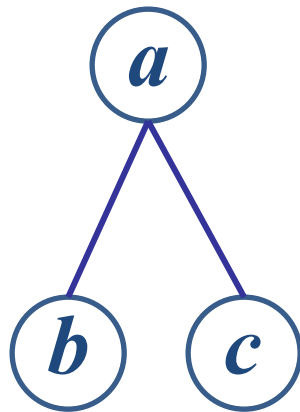
4.1.3. Cây có thứ tự

4.1.4. Cây có nhãn

4.1.5. ADT cây

4.1.3. Cây có thứ tự - Ordered Tree

- Thứ tự của các nút
- Các con của một nút thường được xếp thứ tự *từ trái sang phải*.

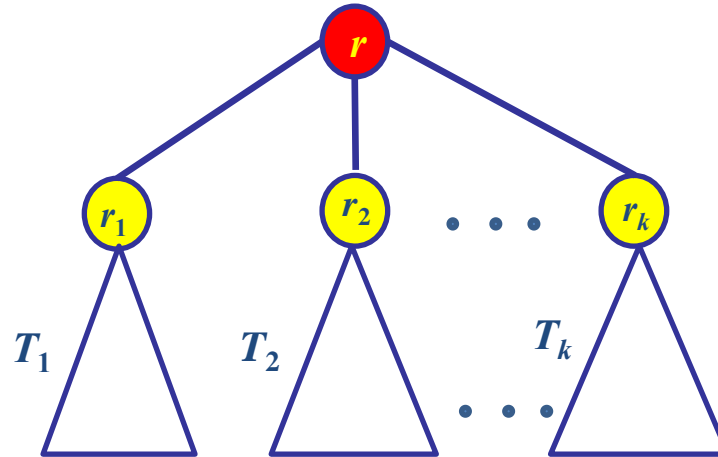


- Xét chủ yếu là *cây có thứ tự*
- Khi muốn khẳng định là không quan tâm đến thứ tự, phải nói rõ là *cây không có thứ tự*.

Xếp thứ tự các nút

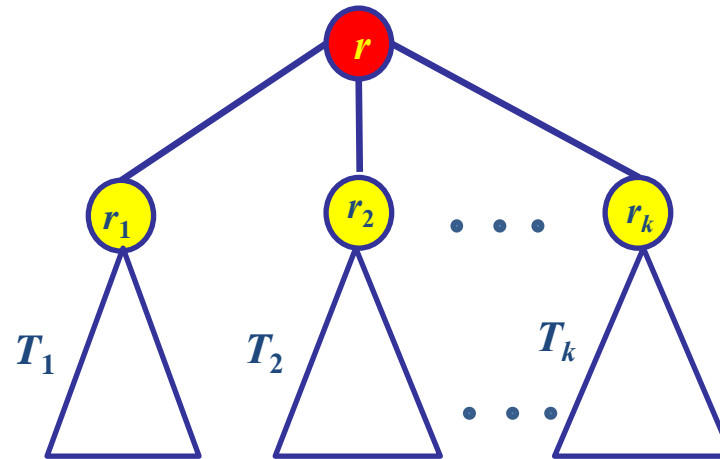
- Thứ tự trước, Thứ tự sau và Thứ tự giữa (Preorder, Postorder, và Inorder)
- Các thứ tự này được định nghĩa một cách đệ qui như sau
 - Nếu cây T là rỗng, thì danh sách rỗng là danh sách theo thứ tự trước, thứ tự sau và thứ tự giữa của cây T .
 - Nếu cây T có 1 nút, thì nút đó chính là danh sách theo thứ tự trước, thứ tự sau và thứ tự giữa của cây T .
 - Trái lại, giả sử T là cây có gốc r với các cây con là T_1, T_2, \dots, T_k .

Duyệt theo thứ tự trước - Preorder Traversal



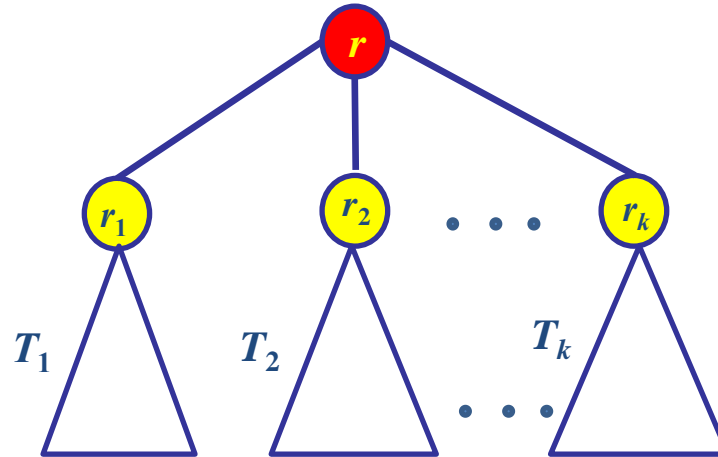
- **Thứ tự trước** (hay duyệt theo thứ tự trước - *preorder traversal*) của các nút của T là:
 - Gốc r của T ,
 - tiếp đến là các nút của T_1 theo thứ tự trước,
 - tiếp đến là các nút của T_2 theo thứ tự trước,
 - ...
 - và cuối cùng là các nút của T_k theo thứ tự trước.

Duyệt theo thứ tự sau - Postorder Traversal



- **Thứ tự sau** của các nút của cây T là:
 - Các nút của T_1 theo thứ tự sau,
 - tiếp đến là các nút của T_2 theo thứ tự sau,
 - ...
 - các nút của T_k theo thứ tự sau,
 - sau cùng là nút gốc r .

Duyệt theo thứ tự giữa - Inorder Traversal

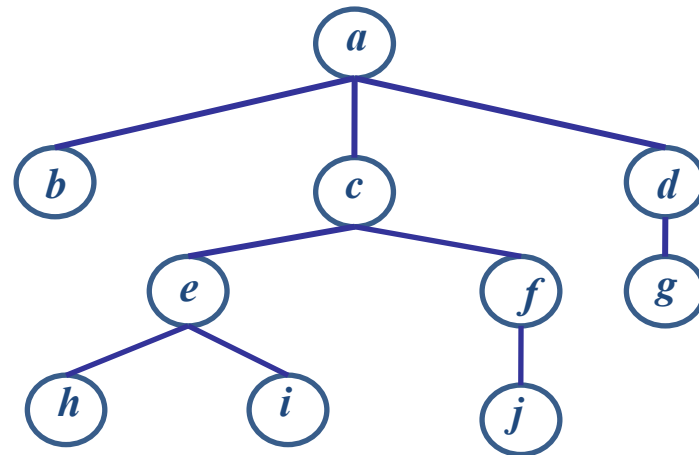


- **Thứ tự giữa** của các nút của cây T là:
 - Các nút của T_1 theo thứ tự giữa,
 - tiếp đến là nút gốc r ,
 - tiếp theo là các nút của T_2, \dots, T_k , mỗi nhóm nút được xếp theo thứ tự giữa.

Thuật toán duyệt theo thứ tự trước - Preorder Traversal

```
void PREORDER ( nodeT r )
```

```
{  
(1)  Đưa ra  $r$ ;  
(2)  for (mỗi con  $c$  của  $r$ , nếu có, theo thứ tự từ trái sang) do  
      PREORDER( $c$ );  
}
```

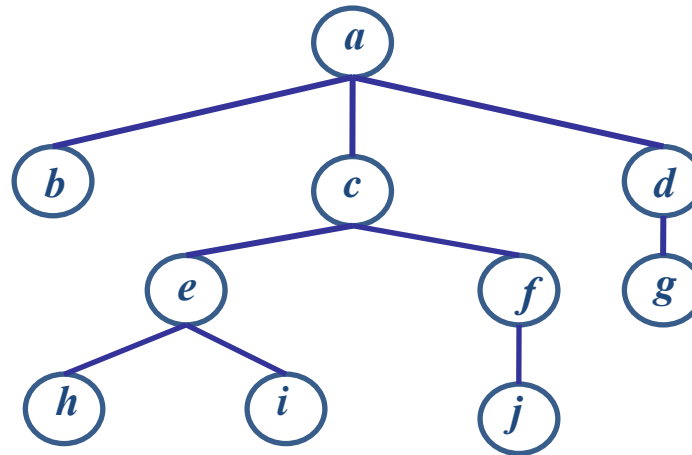


- **Ví dụ:** Thứ tự trước của các đỉnh của cây trên hình vẽ là $a, b, c, e, h, i, f, j, d, g$

Thuật toán duyệt theo thứ tự sau - Postorder Traversal

- Thuật toán duyệt theo thứ tự sau thu được bằng cách đảo ngược hai thao tác (1) và (2) trong PREORDER:

```
void POSTORDER ( nodeT r )  
{  
    for (mỗi con  $c$  của  $r$ , nếu có, theo thứ tự từ trái sang) do  
        POSTORDER( $c$ )  
    Đưa ra  $r$ ;  
}
```

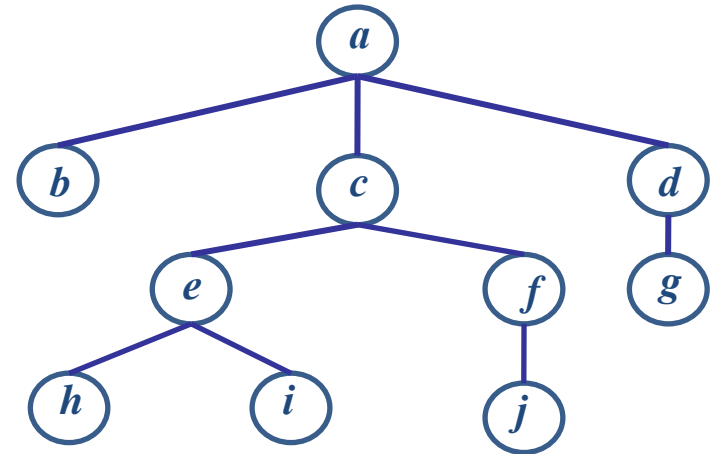


- Ví dụ:** Dãy các đỉnh được liệt kê theo *thứ tự sau* của cây trong hình vẽ là:
 $b, h, i, e, j, f, c, g, d, a$

Thuật toán duyệt theo thứ tự giữa - Inorder Traversal

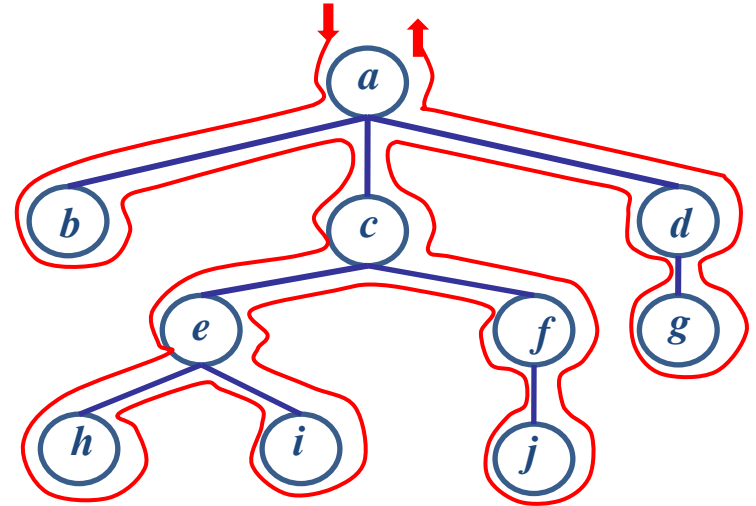
```
void INORDER (nodeT r )
```

```
{  
    if ( r là lá ) Đưa ra r;  
    else  
    {  
        INORDER(con trái nhất của r);  
        Đưa ra r;  
        for (mỗi con c của r, ngoại trừ con trái nhất, theo thứ tự từ trái sang) do  
            INORDER(c);  
    }  
}
```



- Ví dụ:** Dãy các đỉnh của cây trong hình vẽ được liệt kê theo thứ tự giữa:
b, a, h, e, i, c, j, f, g, d

Xếp thứ tự các nút

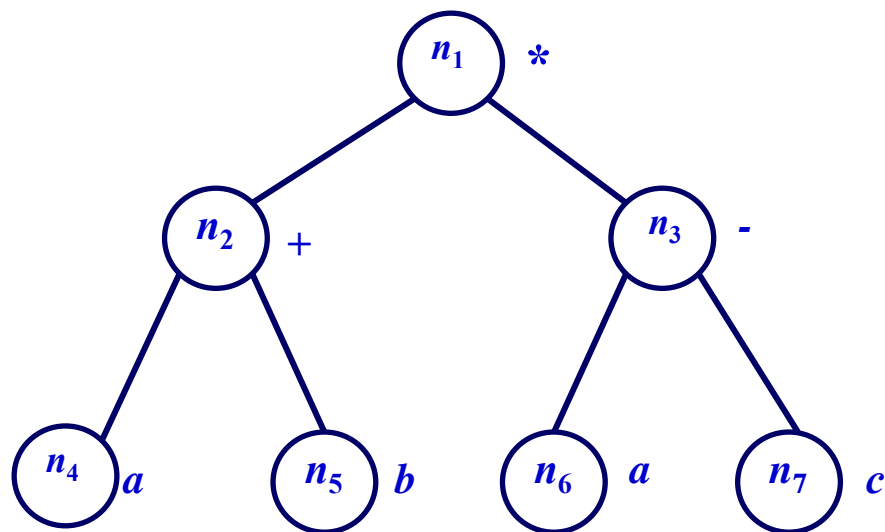


- Thứ tự trước: đưa ra nút mỗi khi đi qua nó.
- Thứ tự sau: đưa ra nút khi qua nó ở lần cuối trước khi quay về cha của nó.
- Thứ tự giữa: đưa ra lá ngay khi đi qua nó, còn những nút trong được đưa ra khi lần thứ hai được đi qua.
- Chú ý: các lá được xếp thứ tự từ trái sang phải như nhau trong cả 3 cách sắp xếp.

4.1.4. Cây có nhãn (Labeled Tree)

- Mỗi nút của cây 1 nhãn (*label*) hoặc 1 giá trị.
- nhãn của nút không phải là tên gọi của nút mà là giá trị được cất giữ trong nó.
- **Ví dụ:** Xét cây có 7 nút n_1, \dots, n_7 . Ta gán nhãn cho các nút như sau:

- Nút n_1 có nhãn $*$;
- Nút n_2 có nhãn $+$;
- Nút n_3 có nhãn $-$;
- Nút n_4 có nhãn a ;
- Nút n_5 có nhãn b ;
- Nút n_6 có nhãn a ;
- Nút n_7 có nhãn c .



- Cây trong ví dụ vừa nêu có tên gọi là cây biểu thức $(a+b)*(a-c)$

Cây biểu thức (Expression Tree)

- Qui tắc để cây có nhãn biểu diễn một biểu thức:
 - Mỗi nút lá có **nhãn** là toán hạng và chỉ gồm 1 toán hạng đó.
 - Ví dụ nút n_4 biểu diễn biểu thức a .
 - Mỗi nút trong n được gán nhãn là phép toán.
 - Giả sử n có nhãn là phép toán 2 ngôi **q** (VD: + hoặc *), và con trái biểu diễn **biểu thức E_1** và con phải biểu diễn **biểu thức E_2** .
 - Khi đó n biểu diễn biểu thức $(E_1) \text{ q } (E_2)$.
 - *Có thể bỏ dấu ngoặc nếu như điều đó là không cần thiết.*

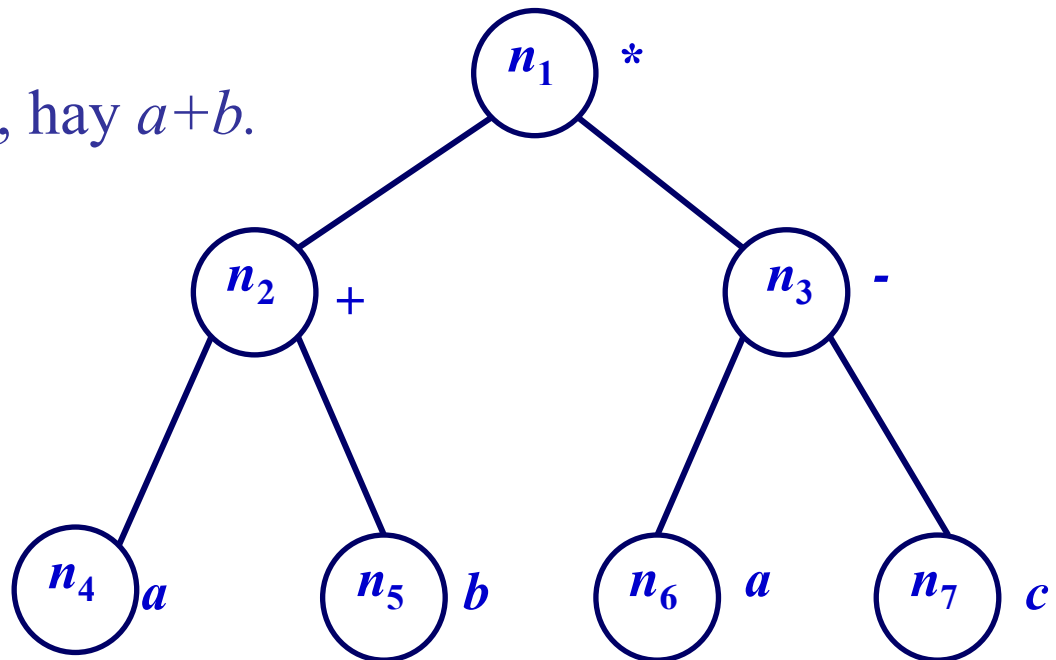
Cây biểu thức (Expression Tree)

- Ví dụ:*

- Nút n_2 chứa toán hạng $+$ và con trái và con phải của nó là a và b .

Vì thế

n_2 biểu diễn $(a) + (b)$, hay $a+b$.



4.1. Định nghĩa và khái niệm

4.1.1. Định nghĩa

4.1.2. Các thuật ngữ

4.1.3. Cây có thứ tự

4.1.4. Cây có nhãn

4.1.5. ADT cây

4.1.5. ADT Cây

- Cây dùng để thiết kế và cài đặt nhiều kiểu dữ liệu trừu tượng quan trọng khác như "Cây tìm kiếm nhị phân", "Tập hợp",....
- Có nhiều phép toán làm việc với cây.

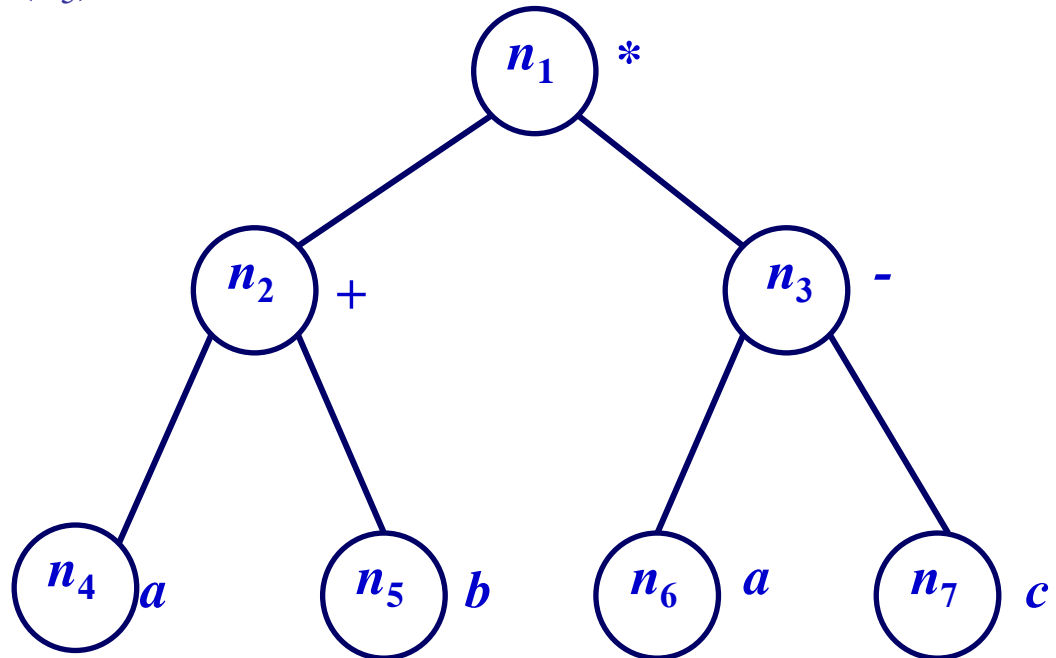
4.1.5. ADT Cây

- **parent(n, T).** Hàm này trả lại cha của nút n trong cây T .
 - Nếu n là gốc (nó không có cha), trả lại Λ .
 - Theo nghĩa này, Λ là nút rỗng ("null node") dùng để báo hiệu rằng chúng ta sẽ dời khỏi cây.
- **leftmost_child(n, T)** trả lại con trái nhất của nút n trong cây T , và trả lại Λ nếu n là lá (không có con).
- **right_sibling(n, T)** trả lại em bên phải của nút n trong cây T (được định nghĩa như là nút m có cùng cha là p giống như n sao cho m nằm sát bên phải của n trong danh sách sắp thứ tự các con của p).

Cây biểu thức (Expression Tree)

Ví dụ:

- $\text{leftmost_child}(n_2) = n_4$;
- $\text{right_sibling}(n_4) = n_5$,
- $\text{RIGHT_SIBLING}(n_5) = \Lambda$.



4.1.5. ADT Cây

- **label(n, T)** trả lại nhãn của nút n trong cây T . Tuy nhiên, ta không đòi hỏi cây nào cũng có nhãn.
- **createi(v, T_1, T_2, \dots, T_i)** là họ các hàm, mỗi hàm cho một giá trị của $i = 0, 1, 2, \dots$ createi tạo một nút mới r với nhãn v và gán cho nó i con, với các con là các gốc của cây T_1, T_2, \dots, T_i , theo thứ tự từ trái sang. Trả lại cây với gốc r . Chú ý, nếu $i = 0$, thì r vừa là lá vừa là gốc.
- **root(T)** trả lại nút là gốc của cây T , hoặc Λ nếu T là cây rỗng.
- **makenull(T)** biến T thành cây rỗng.

Biểu diễn cây

- Có nhiều cách biểu diễn cây. Ta giới thiệu qua về ba cách biểu diễn cơ bản:
 - dùng mảng (Array Representation)
 - danh sách các con (Lists of Children)
 - dùng con trái và em phải (The Leftmost-Child, Right-Sibling Representation)

Biểu diễn cây dùng mảng

- Giả sử T là cây với các nút đặt tên là $1, 2, \dots, n$.
- Để biểu diễn T cài đặt thao tác hỗ trợ **parent**
 - dùng danh sách tuyến tính A trong đó mỗi phần tử $A[i]$ chứa con trỏ đến cha của nút i .
 - Riêng gốc của T có thể phân biệt bởi con trỏ rỗng.
- Đặt $A[i] = j$ nếu nút j là cha của nút i ,
 - $A[i] = 0$ nếu nút i là gốc.
- Cách biểu diễn này dựa trên cơ sở là mỗi nút của cây (ngoại trừ gốc) đều có duy nhất một cha.

Biểu diễn cây dùng mảng (tiếp)

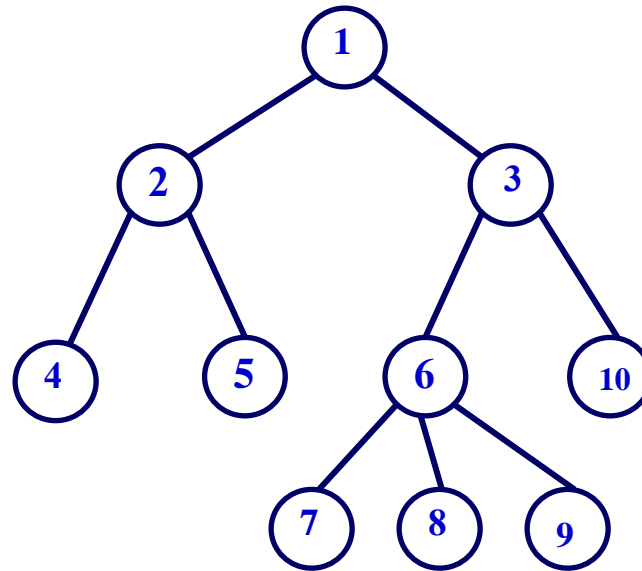
- Với cách biểu diễn này cha của một nút có thể xác định trong thời gian hằng số.
- Đường đi từ một nút đến tổ tiên của chúng (kể cả đến gốc) có thể xác định dễ dàng:

$n \leftarrow \text{parent}(n) \leftarrow \text{parent}(\text{parent}(n)) \leftarrow \dots$

- Có thể đưa thêm vào mảng $L[i]$ để hỗ trợ việc ghi nhận nhãn cho các nút,
 - hoặc biến mỗi phần tử $A[i]$ thành bản ghi gồm 2 trường: biến nguyên ghi nhận cha và nhãn.

Biểu diễn cây dùng mảng

- Ví dụ



A

0	1	1	2	2	3	6	6	6	3
---	---	---	---	---	---	---	---	---	---

Biểu diễn cây dùng mảng

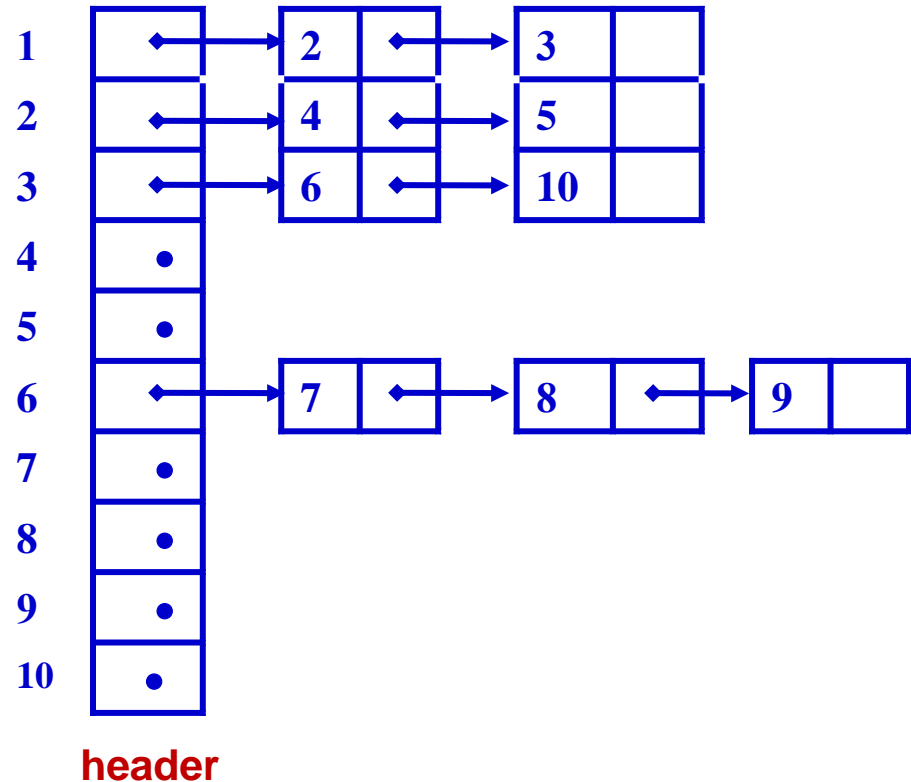
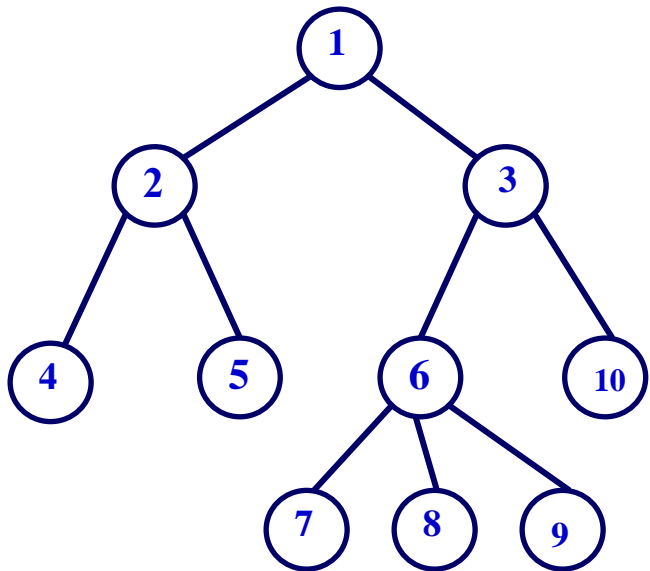
Hạn chế:

- Cách dùng con trỏ cha không thích hợp cho các thao tác với con.
- Cho nút n , ta sẽ **mất nhiều thời gian** để xác định các con của n , hoặc chiều cao của n .
- Không cho ta thứ tự của các nút con.
 - phép toán như **leftmost_child** và **right_sibling** là không xác định được.
- \Rightarrow cách biểu diễn này chỉ dùng trong một số trường hợp nhất định.

Danh sách các con (Lists of Children)

- Trong cách biểu diễn này, với mỗi nút của cây ta cất giữ 1 danh sách các con của nó.
- Danh sách con có thể biểu diễn bởi một trong những cách biểu diễn danh sách đã trình bày trong chương trước.
- Tuy nhiên, để ý rằng số lượng con của các nút là rất khác nhau, nên danh sách móc nối thường là lựa chọn thích hợp nhất.

Danh sách các con (Lists of Children)



- Có mảng con trỏ đến đầu các danh sách con của các nút 1, 2, ..., 10: $header[i]$ trỏ đến danh sách con của nút i .

Danh sách các con (Lists of Children)

- **Ví dụ:** Có thể sử dụng mô tả sau đây để biểu diễn cây

```
typedef ? NodeT; /* dấu ? cần thay bởi định nghĩa kiểu phù hợp */
typedef ? ListT; /* dấu ? cần thay bởi định nghĩa kiểu danh sách phù hợp */
typedef ? position;
typedef struct
{
    ListT header[maxNodes];
    labeltype labels[maxNodes];
    NodeT root;
} TreeT;
```

- Ta giả thiết rằng gốc của cây được cất giữ trong trường *root* và 0 để thể hiện nút rỗng.

Cài đặt leftmost_child

- Dưới đây là minh họa cài đặt phép toán **leftmost_child**. Việc cài đặt các phép toán còn lại được coi là bài tập.

```
NodeT leftmost_child (NodeT n, TreeT T)
/*  trả lại con trái nhất của nút n trong cây T */
{
    ListT L;  /* danh sách các con của n */
    L = T.header[n];
    if (empty(L))    /*  n là lá */
        return(0);
    else return(retrieve ( first(L), L));
}
```

Dùng con trái và em kế cận phải

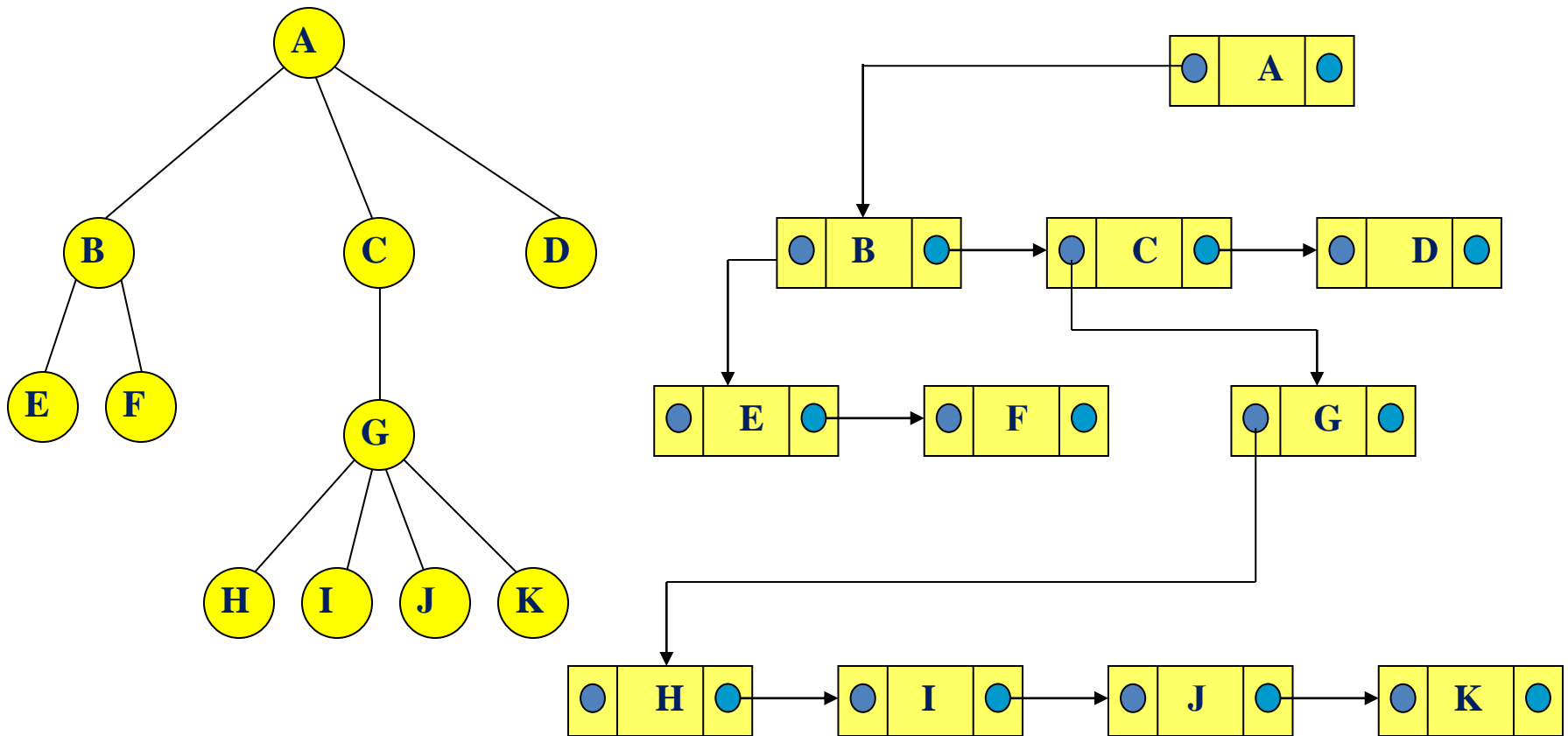
(The Leftmost-Child, Right-Sibling Representation)

- Rõ ràng, mỗi một nút của cây chỉ có thể có:
 - hoặc là không có con, hoặc có đúng một nút con cực trái (con cả);
 - hoặc là không có em kế cận phải, hoặc có đúng một nút em kế cận phải (right-sibling).
- Vì vậy để biểu diễn cây ta có thể lưu trữ thông tin về con cực trái và em kế cận phải của mỗi nút. Ta có thể sử dụng mô tả sau:

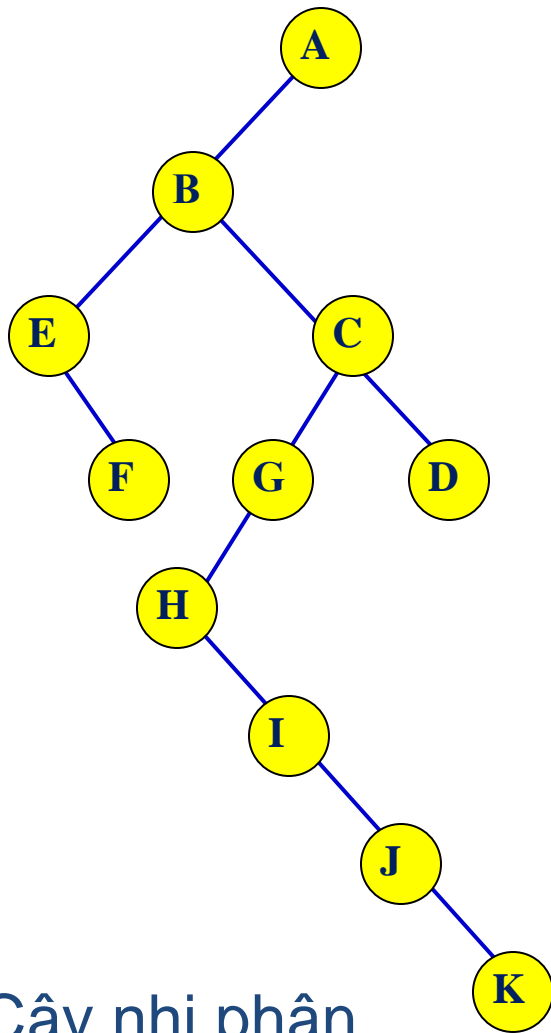
struct Tnode

```
{  
    char word[20];           // Dữ liệu cất giữ ở nút  
    struct Tnode *leftmost_child;  
    struct Tnode *right_sibling;  
};  
typedef struct Tnode treeNode;  
treeNode Root;
```

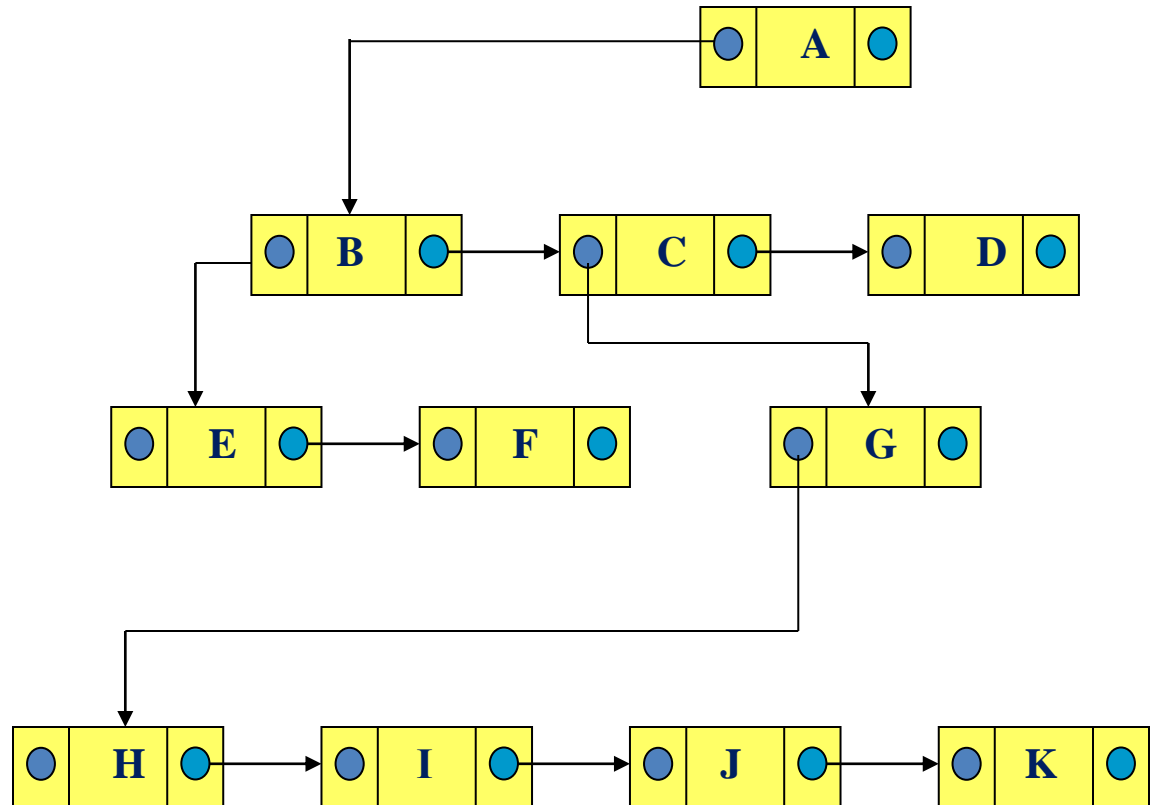
Biểu diễn cây bởi con trái và em kề cận phải



Biểu diễn cây tổng quát bởi cây nhị phân (con trái và con phải=em kề cận phải)



Cây nhị phân



Dùng con trái và em kế cận phải (The Leftmost-Child, Right-Sibling Representation)

- Với cách biểu diễn này, các thao tác cơ bản dễ dàng cài đặt. Duy chỉ có thao tác **parent** là đòi hỏi phải duyệt danh sách nên không hiệu quả.
- Trong trường hợp phép toán này phải dùng thường xuyên, người ta chấp nhận bổ sung thêm 1 trường nữa vào bản ghi để lưu cha của nút.

4.2. CÂY NHỊ PHÂN

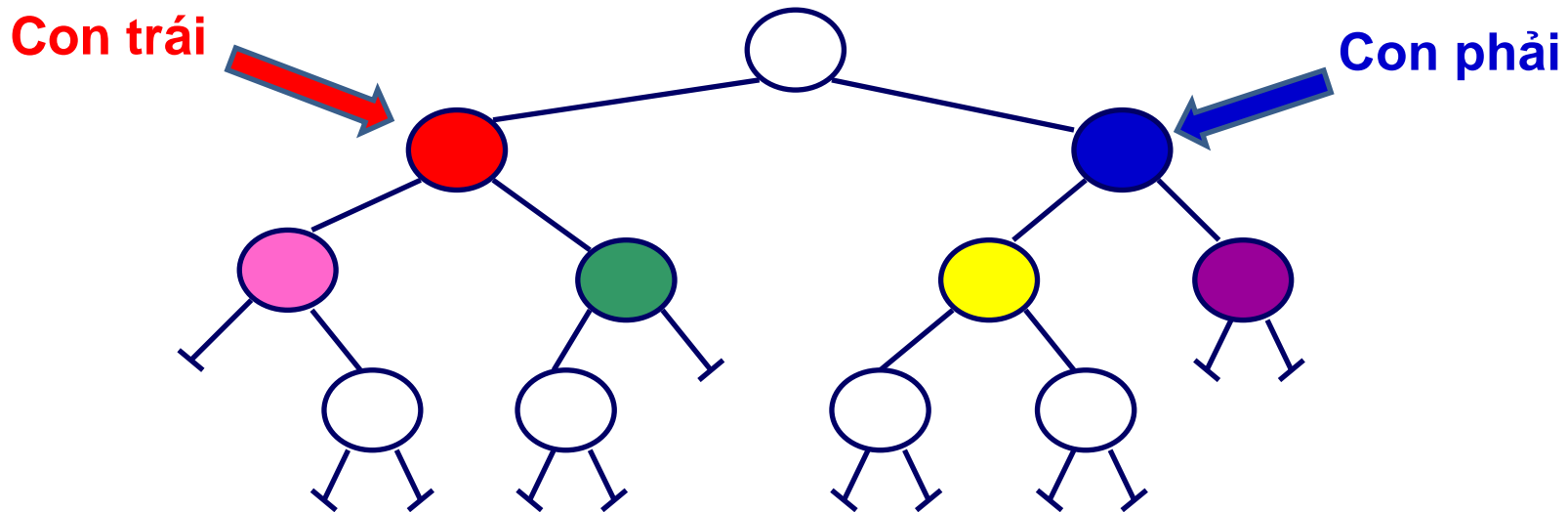
4.2.1. Định nghĩa và tính chất

4.2.2. Biểu diễn cây nhị phân

4.2.3. Duyệt cây nhị phân

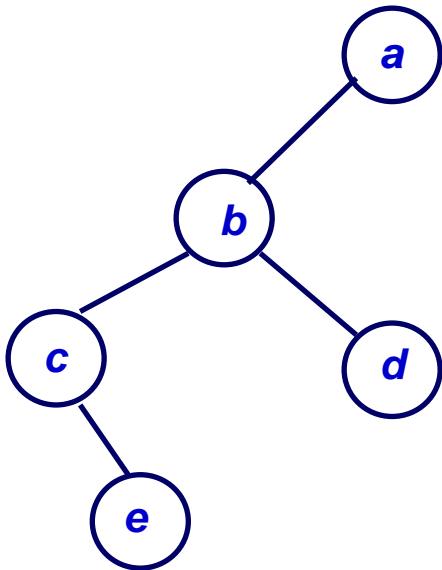
4.2.1. Định nghĩa cây nhị phân - Binary Tree

- **Định nghĩa.** Cây nhị phân là cây mà mỗi nút có nhiều nhất là hai con.
 - Vì mỗi nút chỉ có không quá hai con, nên ta sẽ gọi chúng là *con trái* và *con phải* (*left and right child*).
 - Như vậy mỗi nút của cây nhị phân hoặc là **không có con**, hoặc **chỉ có con trái**, hoặc **chỉ có con phải**, hoặc có cả con trái và con phải.

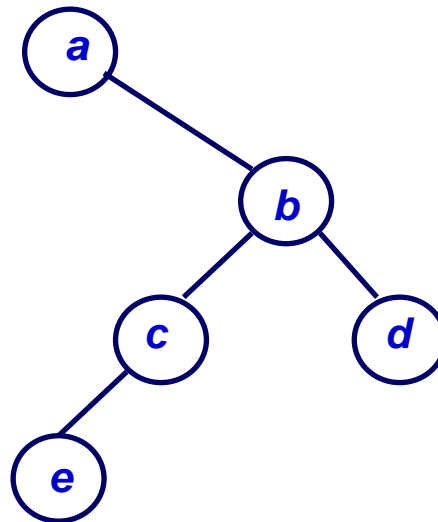


Chú ý

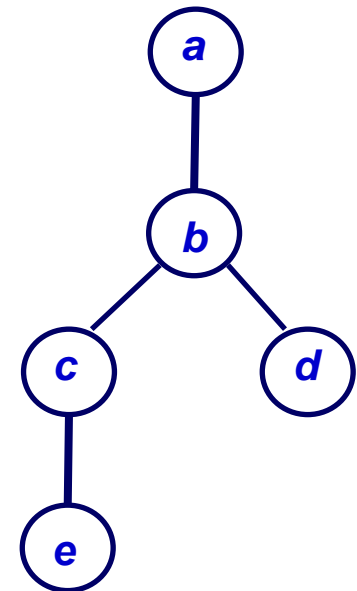
- Vì ta phân biệt con trái và con phải, nên khái niệm cây nhị phân là không trùng với cây có thứ tự định nghĩa ở 4.1.
- Ví dụ:**



Cây nhị phân T_1



Cây nhị phân T_2



Cây tổng quát

- Vì thế, chúng ta sẽ không so sánh cây nhị phân với cây tổng quát*

Tính chất của cây nhị phân

- **Bổ đề 1.**

- (i) Số đỉnh lớn nhất ở trên mức i của cây nhị phân là 2^{i-1} , $i \geq 1$.
- (ii) Một cây nhị phân với chiều cao k có không quá $2^k - 1$ nút, $k \geq 1$.
- (iii) Một cây nhị phân có n nút có chiều cao tối thiểu là $\lceil \log_2(n+1) \rceil$.

- **Chứng minh:**

- (i) Bằng qui nạp theo i .
 - **Cơ sở:** Gốc là nút duy nhất trên mức $i=1$. Như vậy số đỉnh 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 đỉnh lớn nhất trên mức j là 2^{j-1} . Do số đỉnh trên mức $i-1$ là 2^{i-2} , mặt khác theo định nghĩa mỗi đỉnh trên cây nhị phân có không quá 2 con, ta suy ra số lượng nút lớn nhất trên mức i là không vượt quá 2 lần số lượng nút trên mức $i-1$, nghĩa là không vượt quá $2 \cdot 2^{i-2} = 2^{i-1}$.

Tính chất của cây nhị phân

- (ii) Số lượng nút lớn nhất của cây nhị phân chiều cao k là không vượt quá tổng số lượng nút lớn nhất trên các mức $i = 1, 2, \dots, k$, theo bổ đề 1, số này là không vượt quá

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

- (iii) 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:

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

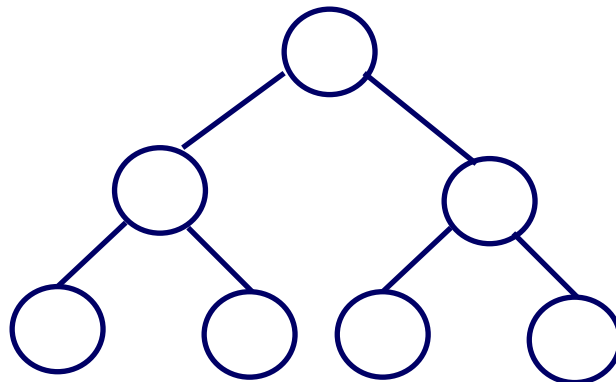
Cây nhị phân đầy đủ (full binary tree)

Định nghĩa. Cây nhị phân đầy đủ (Full Binary Trees) là cây nhị phân thoả mãn

- mỗi nút lá đều có cùng độ sâu và
- các nút trong có đúng 2 con.

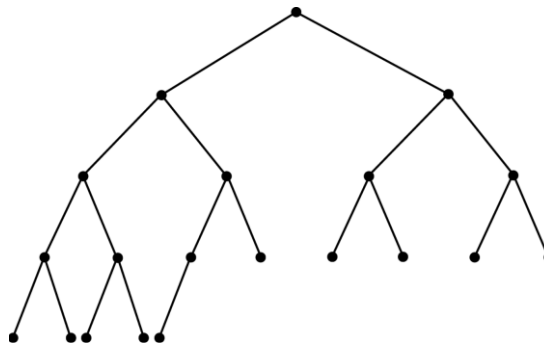
Bổ đề. *Cây nhị phân đầy đủ với độ sâu n có $2^n - 1$ nút.*

Chứng minh. Suy trực tiếp từ bổ đề 1.



Cây nhị phân hoàn chỉnh (Complete Binary Trees)

- **Định nghĩa. Cây nhị phân hoàn chỉnh (Complete Binary Trees):** Cây nhị phân độ sâu n thoả 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.
- **Bổ đề 3.** Cây nhị phân hoàn chỉnh độ sâu n có số lượng nút nằm trong khoảng từ 2^{n-1} đến $2^n - 1$.
- **Chứng minh.** Suy trực tiếp từ định nghĩa và bổ đề 1.

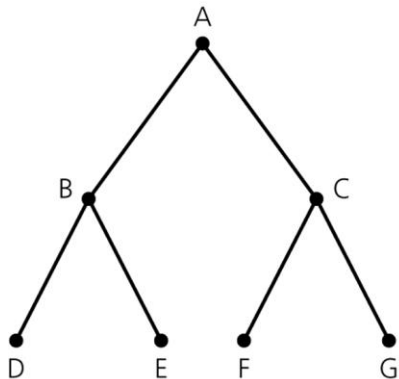


Ví dụ. Cây nhị phân hoàn chỉnh

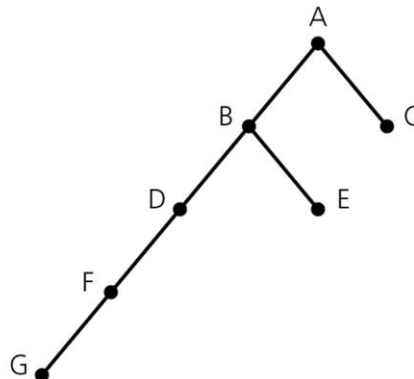
Cây nhị phân cân đối (balanced binary tree)

- **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 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

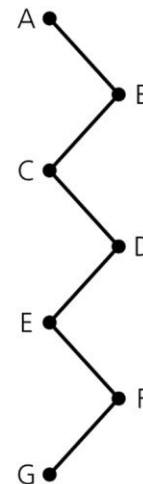
Ví dụ:



(a)



(b)



(c)

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

4.2. CÂY NHỊ PHÂN

4.2.1. Định nghĩa và tính chất

4.2.2. Biểu diễn cây nhị phân

4.2.3. Duyệt cây nhị phân

4.2.2. Biểu diễn cây nhị phân

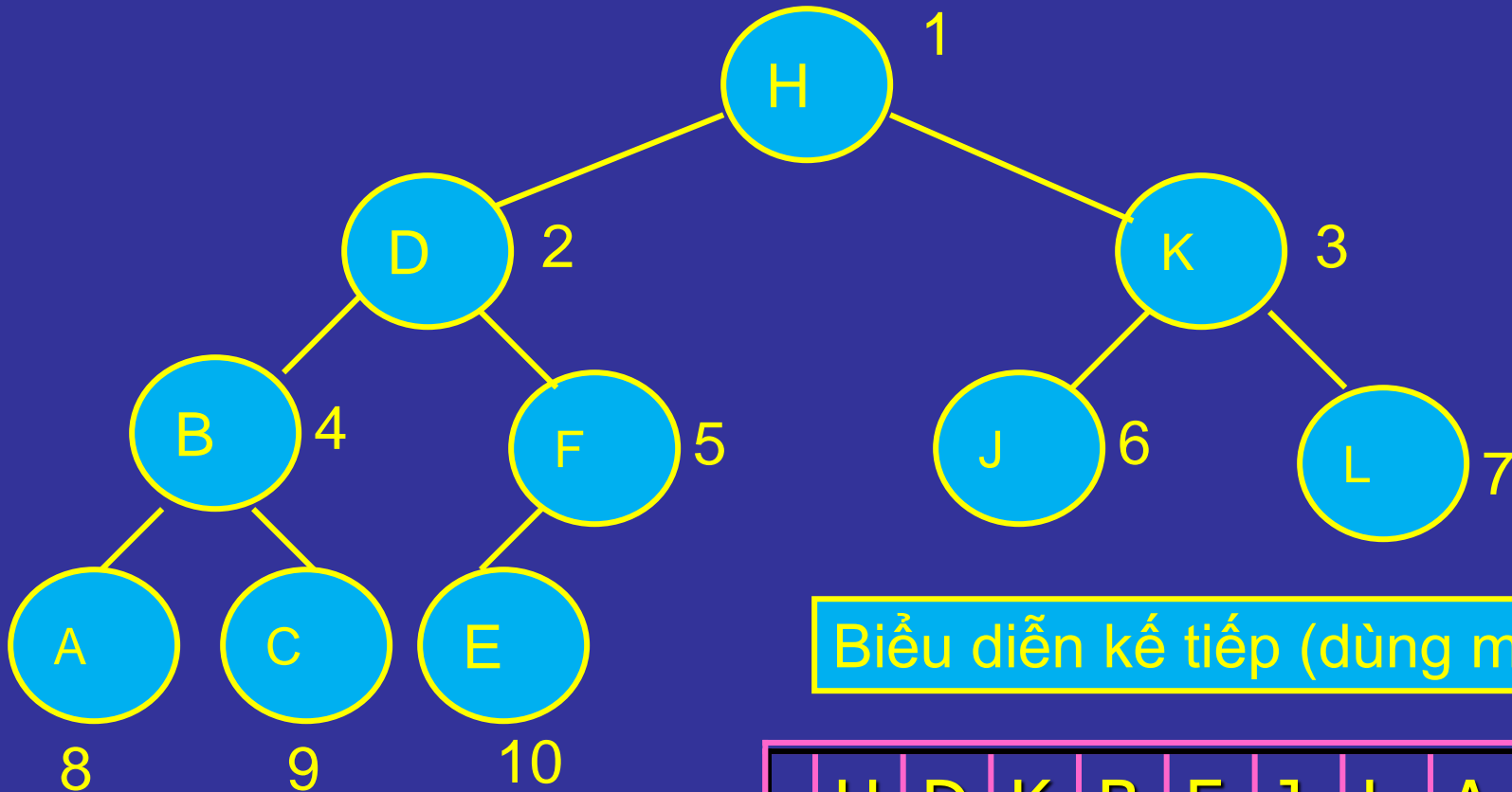
- Ta xét 2 phương pháp:
 - Biểu diễn sử dụng mảng
 - Biểu diễn sử dụng con trỏ

4.2.2. Biểu diễn cây nhị phân

- Biểu diễn sử dụng mảng:
 - Tương tự như trong cách biểu diễn cây tổng quát.
 - Trong trường hợp cây nhị phân hoàn chỉnh, có thể cài đặt nhiều phép toán với cây rất hiệu quả.
- Xét cây nhị phân hoàn chỉnh T có n nút, trong đó mỗi nút chứa một giá trị.
- Gán tên cho các nút của cây nhị phân hoàn chỉnh T từ trên xuống dưới và từ trái qua phải bằng các số $1, 2, \dots, n$.
- Đặt tương ứng cây T với **mảng A** trong đó phần tử thứ i của A là giá trị cất giữ trong nút thứ i của cây T , $i = 1, 2, \dots, n$.

Biểu diễn mảng của cây nhị phân hoàn chỉnh

Complete Binary Tree



Biểu diễn kế tiếp (dùng mảng)

	H	D	K	B	F	J	L	A	C	E
0	1	2	3	4	5	6	7	8	9	10

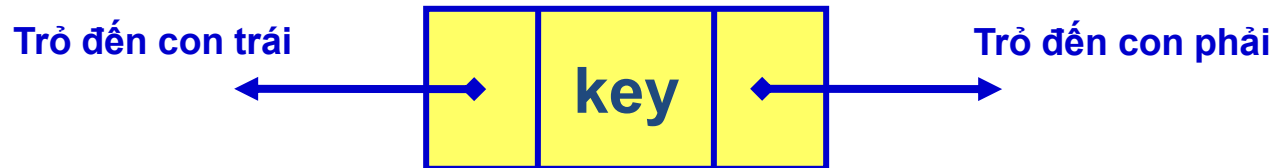
Cây nhị phân hoàn chỉnh - Complete Binary Tree

	H	D	K	B	F	J	L	A	C	E
0	1	2	3	4	5	6	7	8	9	10

Để tìm	Sử dụng	Hạn chế
Con trái của $A[i]$	$A[2*i]$	$2*i \leq n$
Con phải của $A[i]$	$A[2*i + 1]$	$2*i + 1 \leq n$
Cha của $A[i]$	$A[i/2]$	$i > 1$
Gốc	$A[1]$	A khác rỗng
Thử $A[i]$ là lá?	True	$2*i > n$

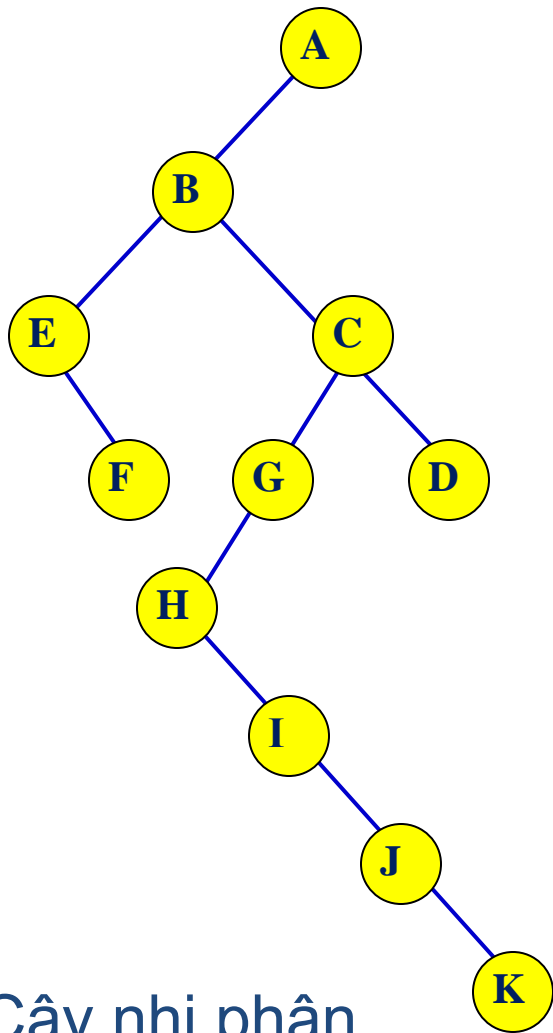
Biểu diễn cây nhị phân dùng con trỏ

Mỗi nút của cây sẽ có con trỏ đến con trái và con trỏ đến con phải:

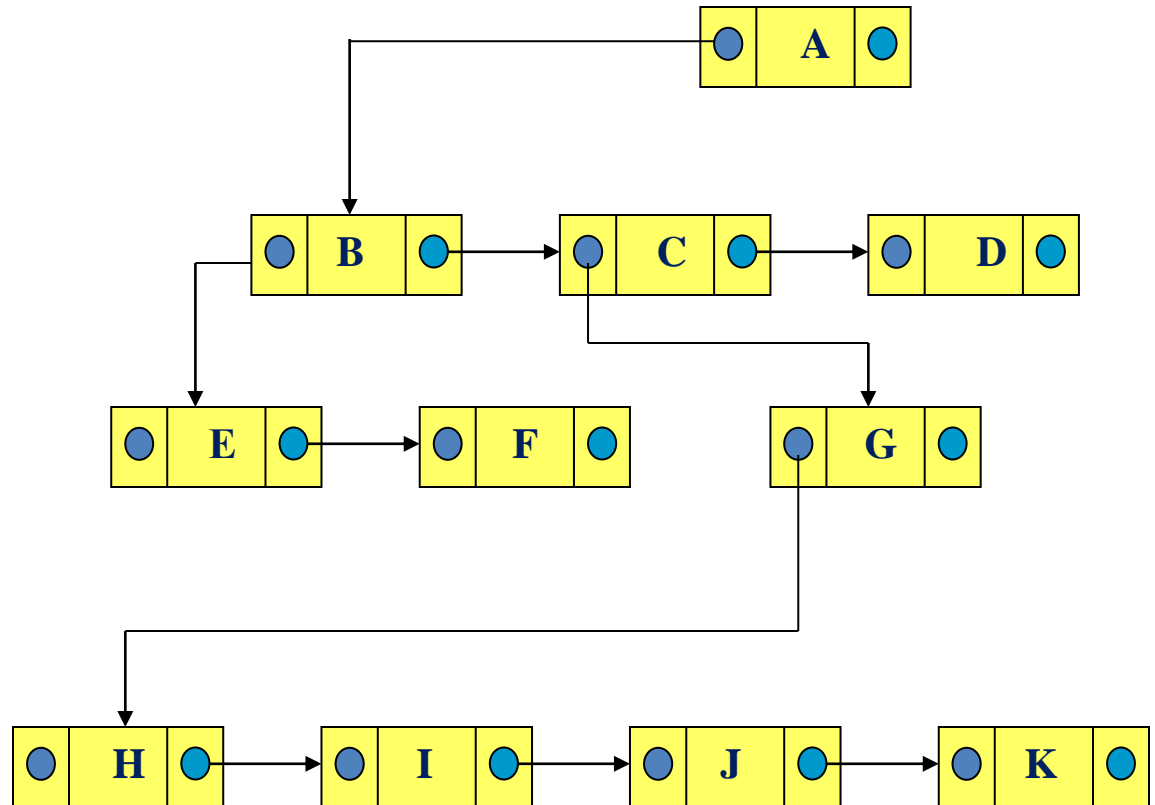


```
struct Tnode {  
    DataType Item; // DataType - kiểu dữ liệu của phần tử  
    struct Tnode *left;  
    struct Tnode *right;  
};  
typedef struct Tnode treeNode;
```

Ví dụ



Cây nhị phân



Các phép toán cơ bản

```
struct Tnode
{ char word[20]; // Dữ liệu của nút
  struct Tnode * left;
  struct Tnode *right;
};
typedef struct Tnode treeNode;

treeNode* makeTreeNode(char *word);
treeNode *RandomInsert(treeNode* tree,char *word);
void freeTree(treeNode *tree);
void printPreorder(treeNode *tree);
void printPostorder(treeNode *tree);
void printInorder(treeNode *tree);
int countNodes(treeNode *tree);
int depth(treeNode *tree);
```

MakeNode

- Thông số: Dữ liệu của nút cần bổ sung
- Các bước:
 - phân bổ bộ nhớ cho nút mới
 - kiểm tra cấp phát bộ nhớ có thành công?
 - nếu đúng, đưa item vào nút mới
 - đặt con trỏ trái và phải bằng NULL
- trả lại: con trỏ đến (là địa chỉ của) nút mới

Cài đặt MakeNode

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

Cài đặt hàm tính số nút và độ sâu của cây

```
int countNodes(treeNode *tree) {  
    /* the function counts the number of nodes of a tree*/  
    if( tree == NULL ) return 0;  
    else {  
        int ld = countNodes(tree->left);  
        int rd = countNodes(tree->right);  
        return 1+ld+rd;  
    }  
}
```

```
int depth(treeNode *tree) {  
    /* the function computes the depth of a tree */  
    if( tree == NULL ) return 0;  
    int ld = depth(tree->left);  
    int rd = depth(tree->right);  
    /* if (ld > rd) return 1+ld; else return 1+rd; */  
    return 1 + (ld > rd ? ld : rd);  
}
```

Loại bỏ cây

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

Duyệt cây nhị phân

- *Thứ tự trước (Preorder)* **NLR**
 - Thăm nút (Visit a node),
 - Thăm cây con trái theo thứ tự trước (Visit left subtree),
 - Thăm cây con phải theo thứ tự trước (Visit right subtree)
- *Thứ tự giữa (Inorder)* **LNR**
 - Thăm cây con trái theo thứ tự giữa (Visit left subtree),
 - Thăm nút (Visit a node),
 - Thăm cây con phải theo thứ tự giữa (Visit right subtree)
- *Thứ tự sau (Postorder)* **LRN**
 - Thăm cây con trái theo thứ tự sau (Visit left subtree),
 - Thăm cây con phải theo thứ tự sau (Visit right subtree)
 - Thăm nút (Visit a node),

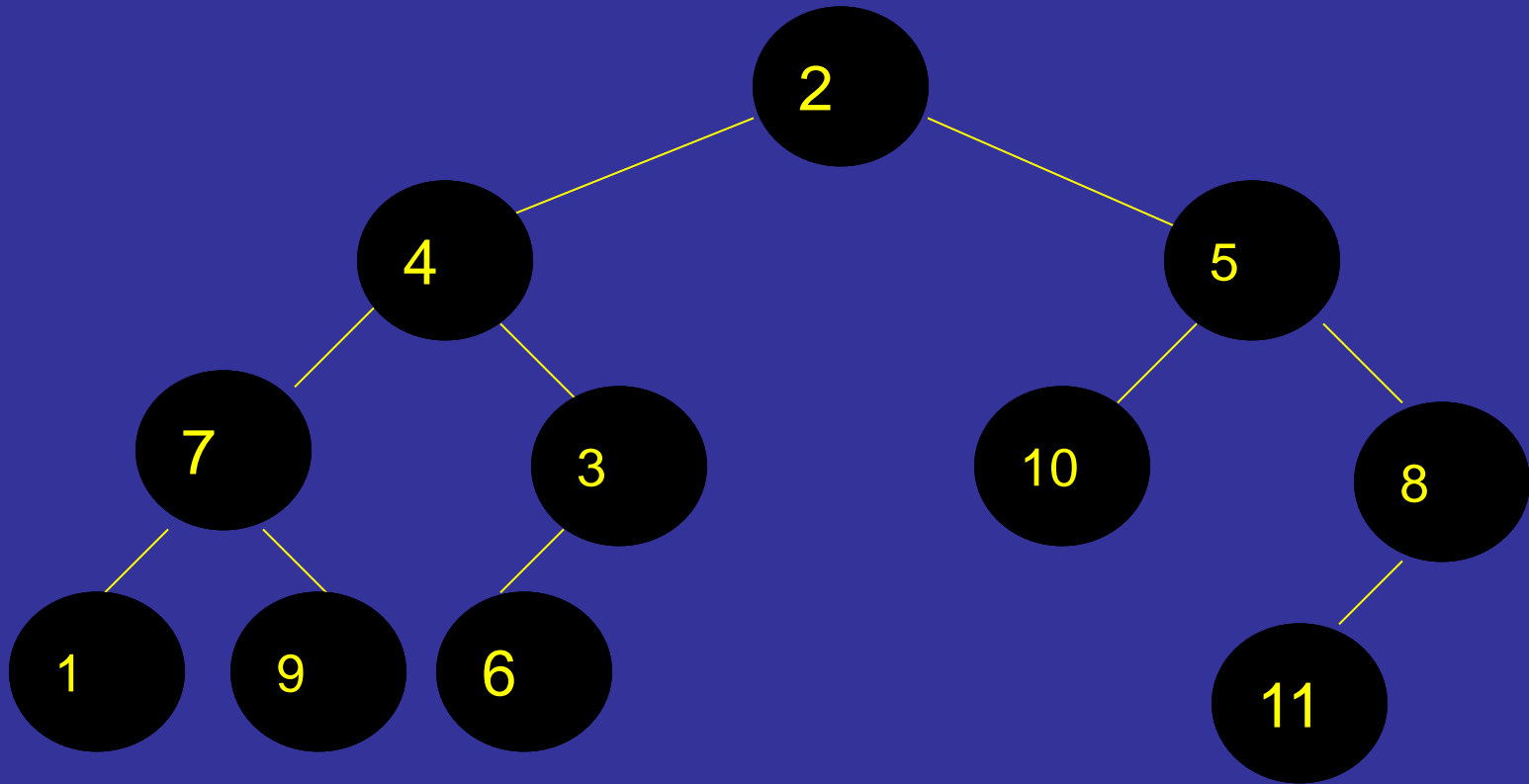
Duyệt theo thứ tự trước - NLR

Preorder Traversal

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

```
void printPreorder(treeNode *tree)
{
    if( tree != NULL )
    {
        printf("%s\n", tree->word);
        printPreorder(tree->left);
        printPreorder(tree->right);
    }
}
```

Preorder Traversal



2, 4, 7, 1, 9, 3, 6, 5, 10, 8, 11

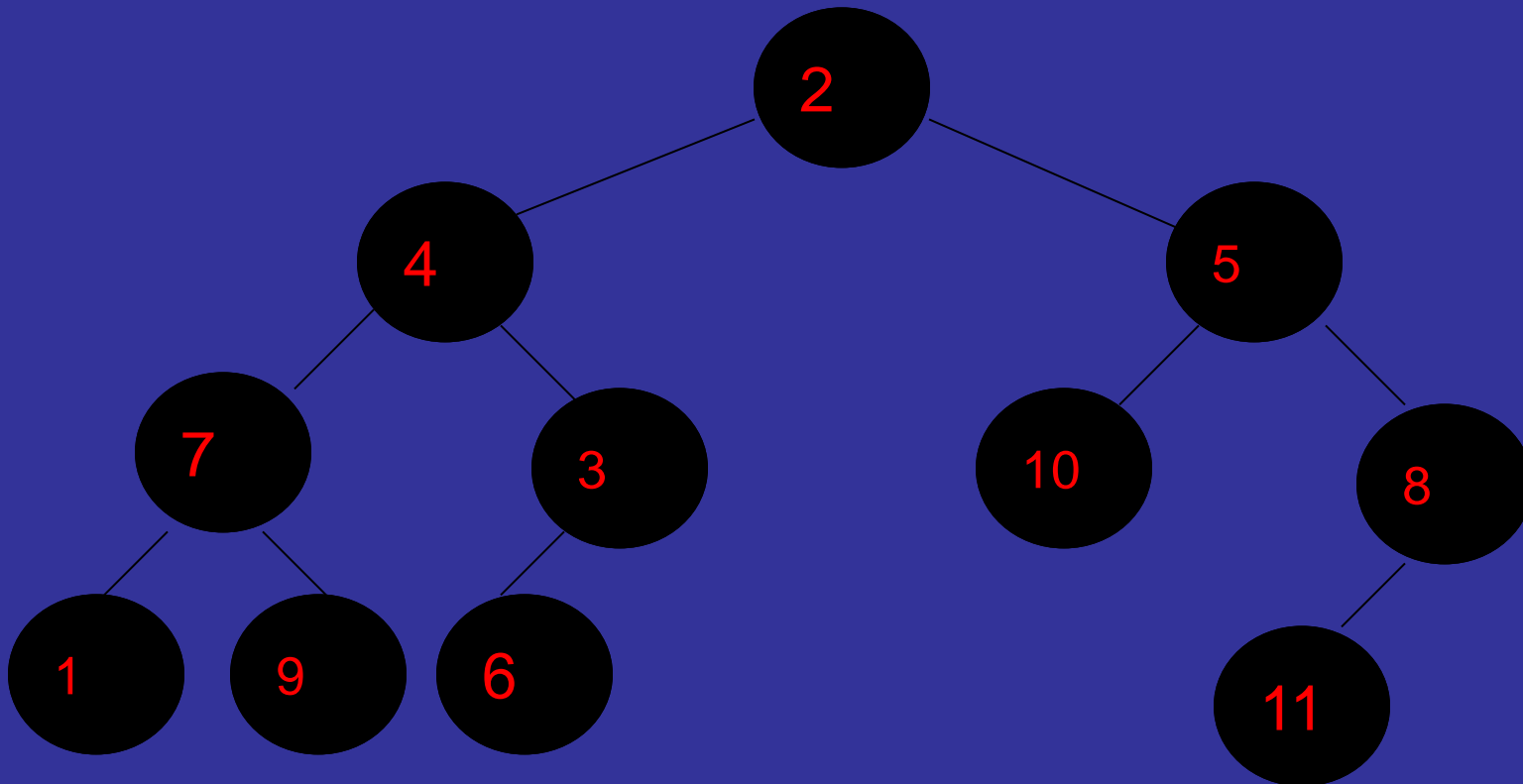
Duyệt theo thứ tự giữa - LNR

Inorder Traversal

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

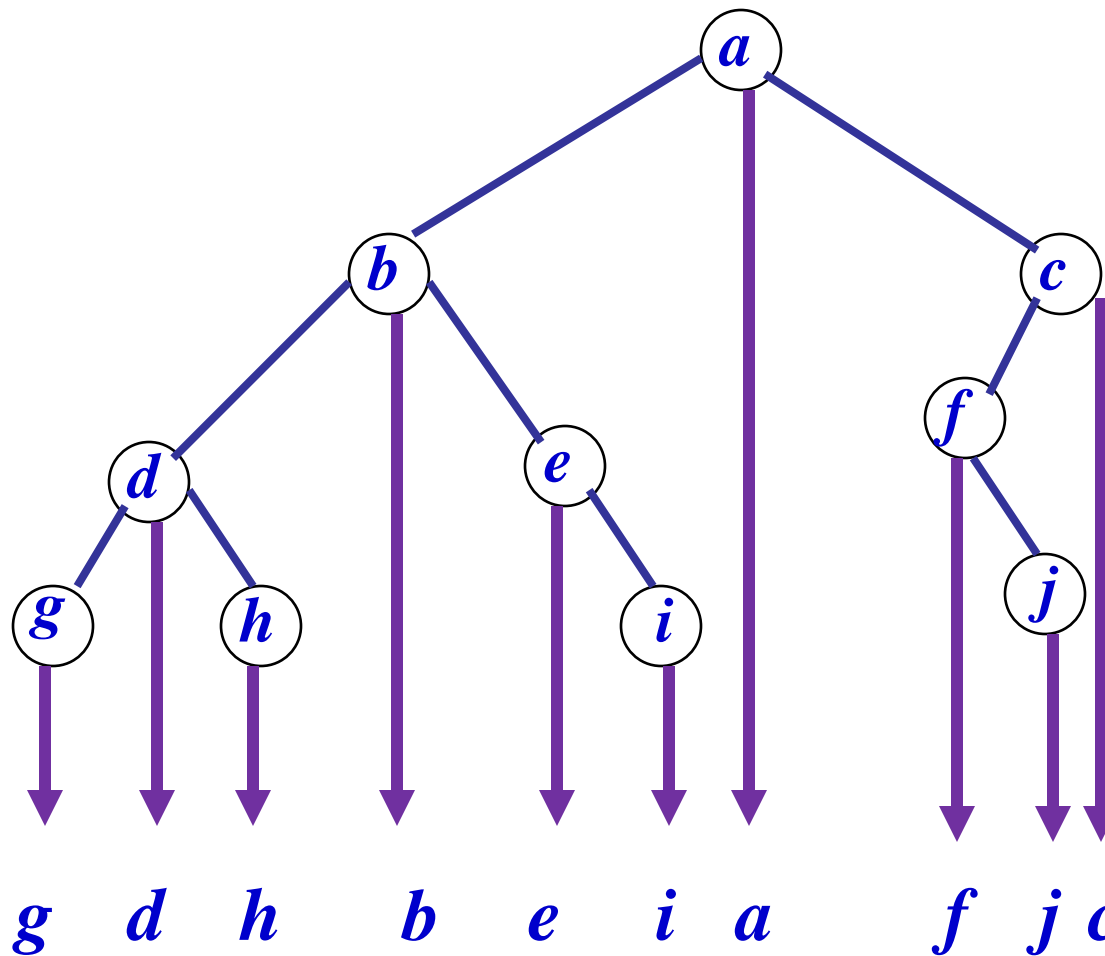
```
void printInorder(treeNode *tree)
{
    if( tree != NULL )
    {
        printInorder(tree->left);
        printf("%s\n", tree->word);
        printInorder(tree->right);
    }
}
```

Inorder Traversal



1, 7, 9, 4, 6, 3, 2, 10, 5, 11, 8

Thứ tự giữa thu được bằng phép chiếu Inorder By Projection



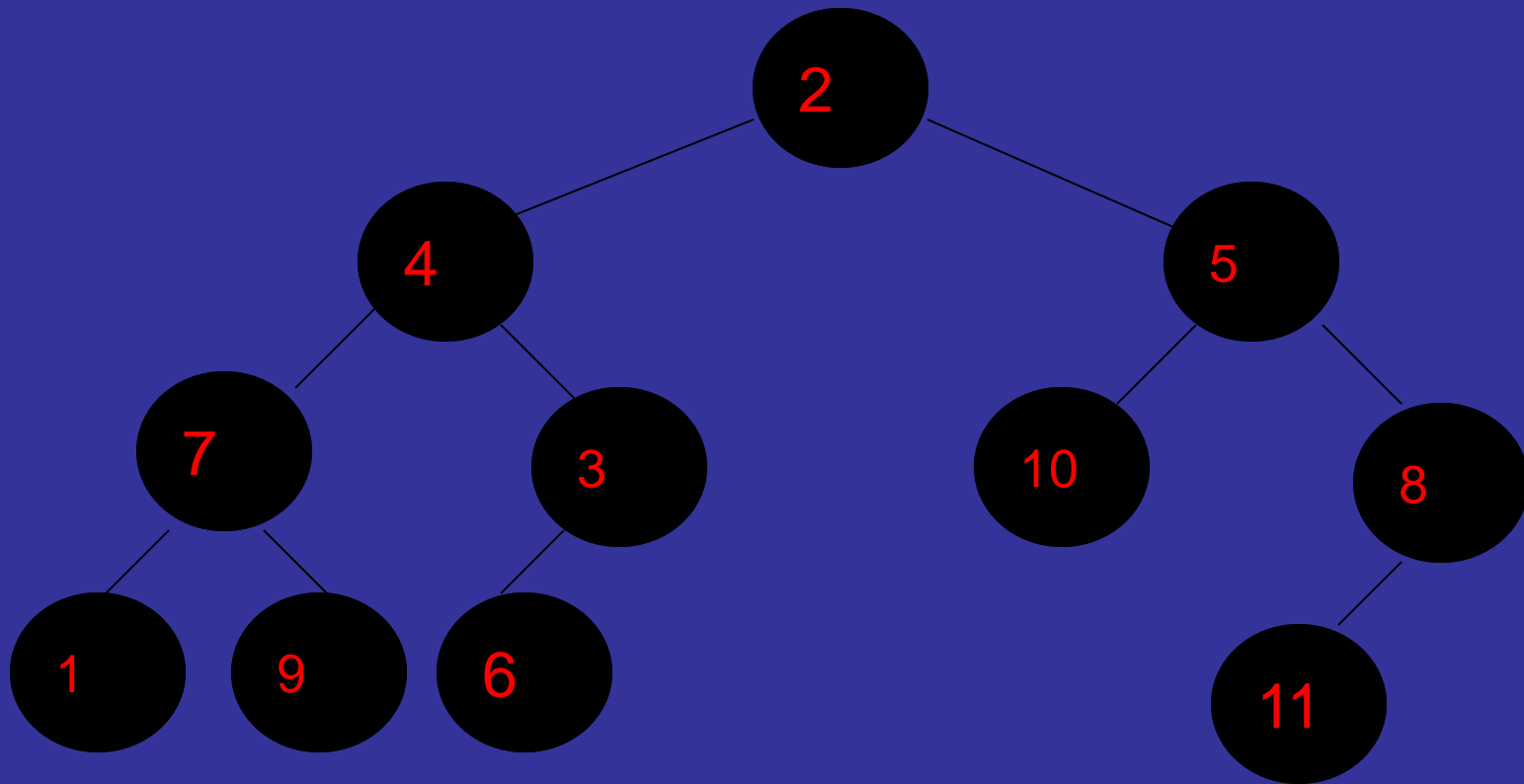
Duyệt theo thứ tự sau - LRN

Postorder Traversal

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

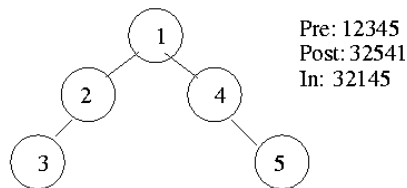
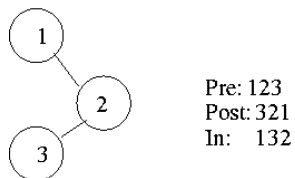
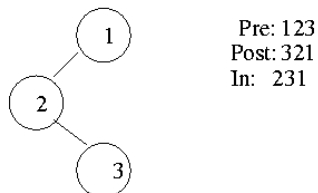
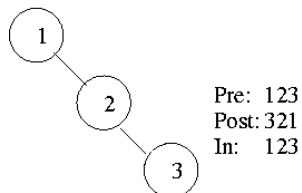
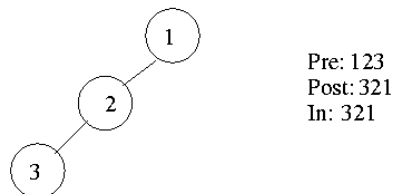
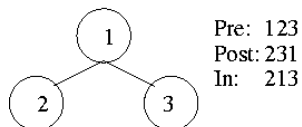
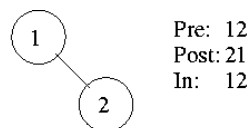
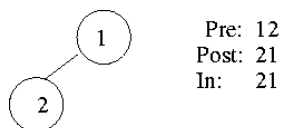
```
void printPostorder (treeNode *tree)
{
    if( tree != NULL )
    {
        printPostorder (tree->left) ;
        printPostorder (tree->right) ;
        printf ("%s\n", tree->word) ;
    }
}
```

Postorder Traversal



1, 9, 7, 6, 3, 4, 10, 11, 8, 5, 2

Ví dụ: Duyệt cây nhị phân



- Chú ý:** Ta có thể xây dựng được cây nhị phân mà thứ tự trước và thứ tự giữa *hoặc* thứ tự sau và thứ tự giữa là như nhau; nhưng không thể có cây nhị phân mà thứ tự trước và thứ tự sau là như nhau.

Chương trình minh họa

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

Chương trình minh họa

```
int main() {
    treeNode *randomTree=NULL;
    char word[20] = "a";
    while( strcmp(word,"~")!=0)
    {   printf("\nEnter item (~ to finish): ");
        scanf("%s", word);
        if (strcmp(word,"~")==0) printf("Cham dut nhap thong tin nut... \n");
        else randomTree=RandomInsert(randomTree,word);
    }
    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);
    getch(); return 0;
}
```

Gắn ngẫu nhiên nút mới vào cây

```
treeNode *RandomInsert(treeNode *tree,char *word){
    treeNode *newNode, *p;
    newNode = makeTreeNode(word);
    if ( tree == NULL )        return newNode;
    if ( rand()%2 ==0 ){
        p=tree;
        while (p->left !=NULL) p=p->left;
        p->left=newNode;
        printf("Node %s is left child of %s \n",word,(*p).word); }
    else {
        p=tree;
        while (p->right !=NULL) p=p->right;
        p->right=newNode;
        printf("Node %s is right child of %s \n",word,(*p).word);
    }
    return tree;
}
```

Chương trình minh họa

```
/******
The program for testing binary tree traversal
******/
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
#include <process.h>

struct Tnode
{
    char word[20];
    struct Tnode * left;
    struct Tnode *right;
};

typedef struct Tnode treeNode;

treeNode* makeTreeNode(char *word);
treeNode *RandomInsert(treeNode* tree,char *word);
void freeTree(treeNode *tree);
void printPreorder(treeNode *tree);
void printPostorder(treeNode *tree);
void printInorder(treeNode *tree);
int countNodes(treeNode *tree);
int depth(treeNode *tree);

int main()
{
    treeNode *randomTree=NULL;
    char word[20] = "a";
    while( strcmp(word,"~")!=0)
    { printf("\nEnter item (~ to finish): ");
      scanf("%s", word);
      if ( strcmp(word,"~")==0 )
          printf("Cham dut nhap thong tin nut... \n");
      else randomTree=RandomInsert(randomTree,word);
    }

    printf("\nThe tree in preorder:\n");
    printPreorder(randomTree);
    printf("*****\n");

    printf("\nThe tree in postorder:\n");
    printPostorder(randomTree);
    printf("*****\n");

    printf("\nThe tree in inorder:\n");
    printInorder(randomTree);
    printf("*****\n");

    printf("\nThe number of nodes is: %d\n",countNodes(randomTree));

    printf("\nThe depth of the tree is: %d\n", depth(randomTree));

    freeTree(randomTree);

    getch();
}
```


4.3. Các ví dụ ứng dụng

4.3.1. Cây biểu thức

4.3.2. Cây mã Huffman

4.3.1. Cây biểu thức

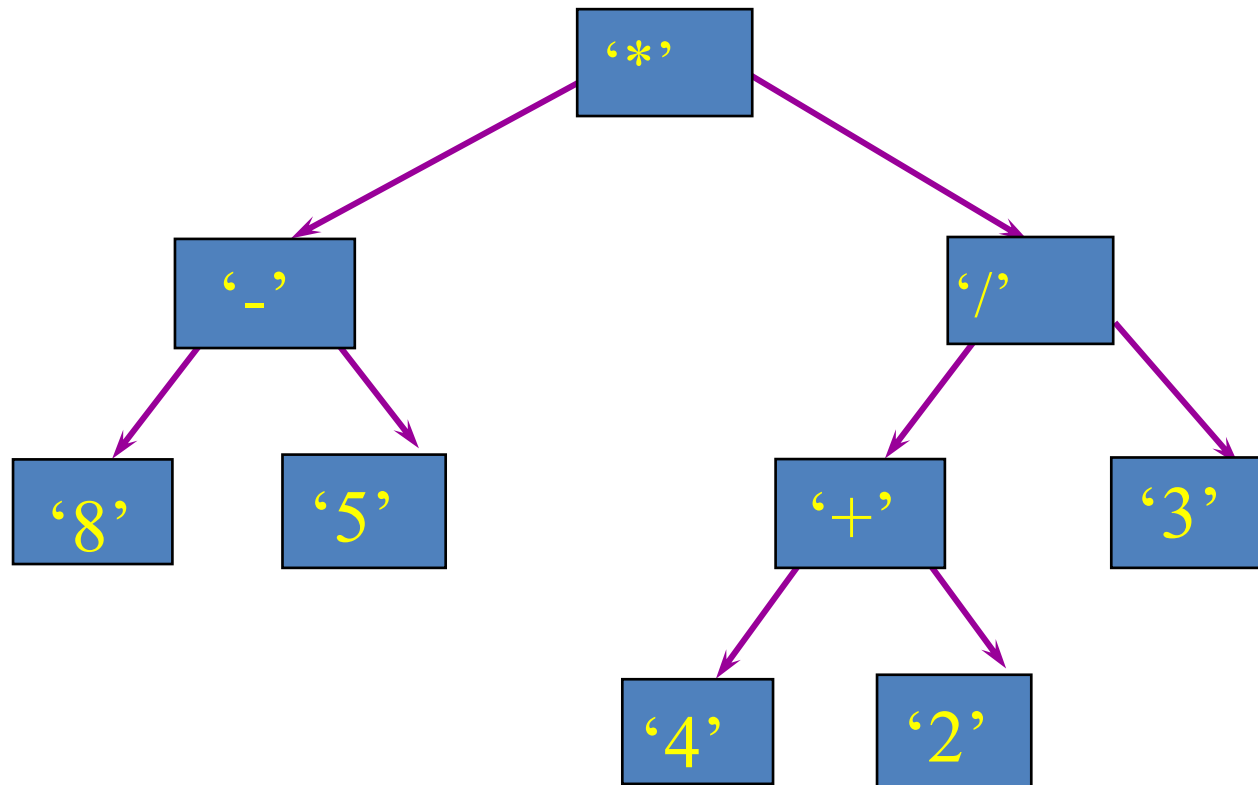
An application of binary trees:
Binary Expression Trees

Khái niệm

Cây biểu thức là cây nhị phân trong đó:

1. Mỗi **nút lá** chứa một toán hạng
2. Mỗi **nút trong** chứa một phép toán hai ngôi
3. Các cây con trái và phải của nút phép toán biểu diễn các **biểu thức con (subexpressions)** cần được thực hiện **trước khi** thực hiện phép toán ở gốc của các cây con.

Ví dụ: Cây nhị phân 4 mức

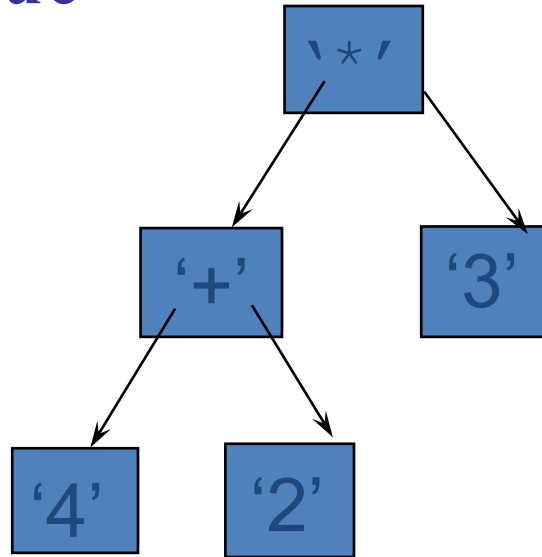


Các mức thể hiện mức độ ưu tiên

- 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.

Ví dụ:

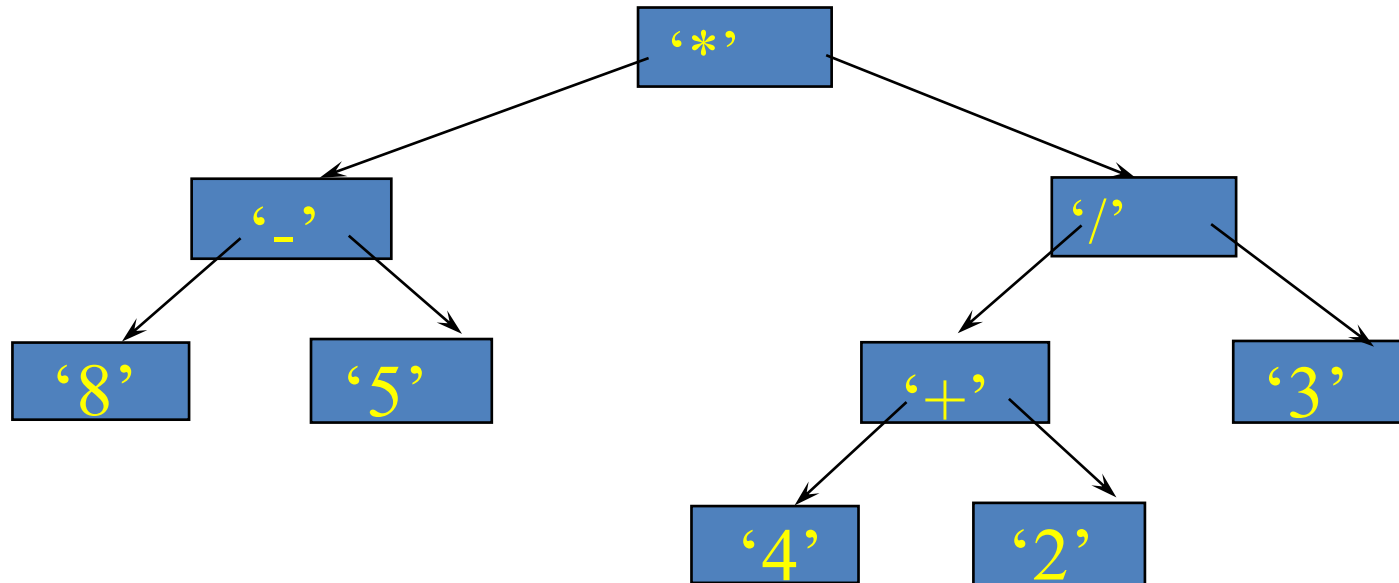
Xét Cây biểu thức



Cây này có giá trị nào?

$$(4 + 2) * 3 = 18$$

Sử dụng cây biểu thức ta có thể đưa ra biểu thức trong ký pháp trung tố, tiền tố và hậu tố (infix, prefix, postfix)



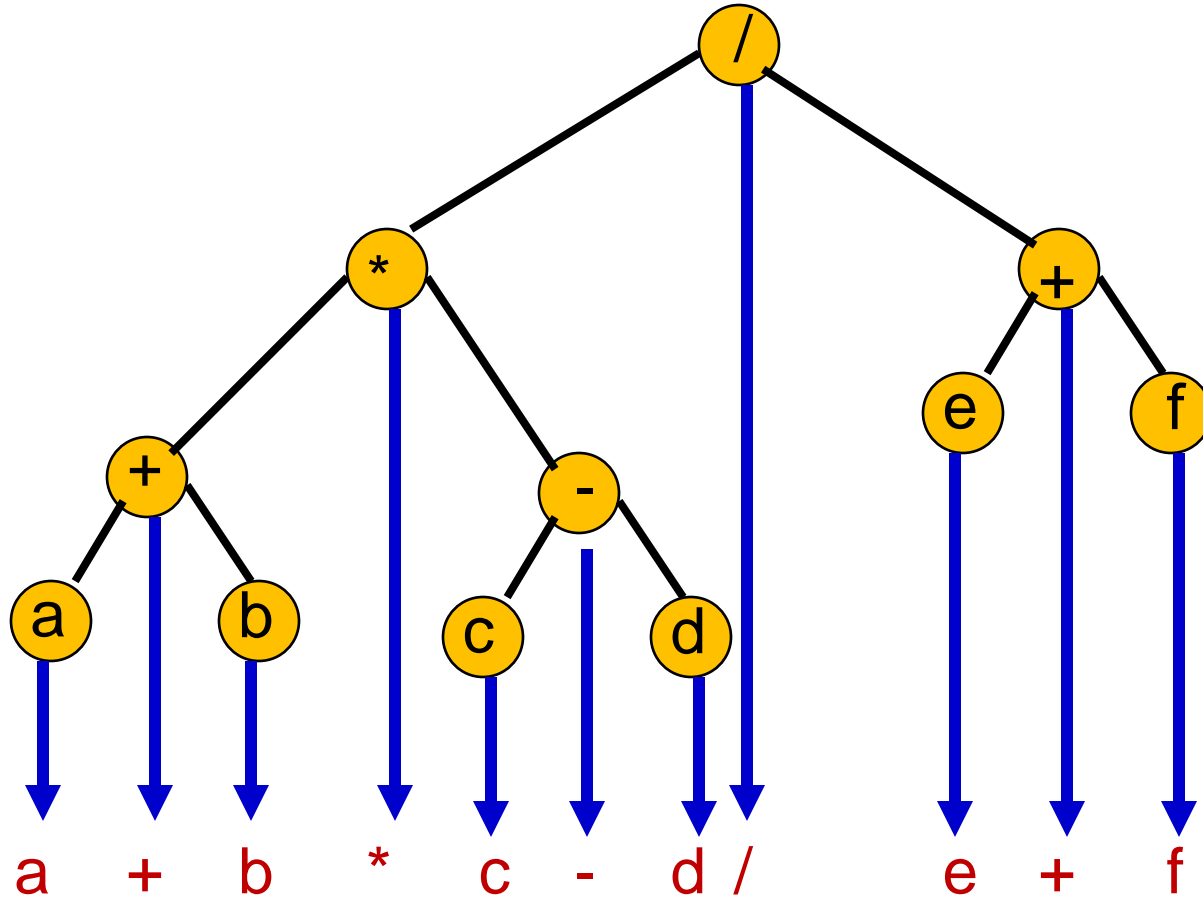
Infix: $((8 - 5) * ((4 + 2) / 3))$

Prefix: $* - 8 5 / + 4 2 3$

Postfix: $8 5 - 4 2 + 3 / *$

Thứ tự giữa của cây biểu thức

Inorder Of Expression Tree

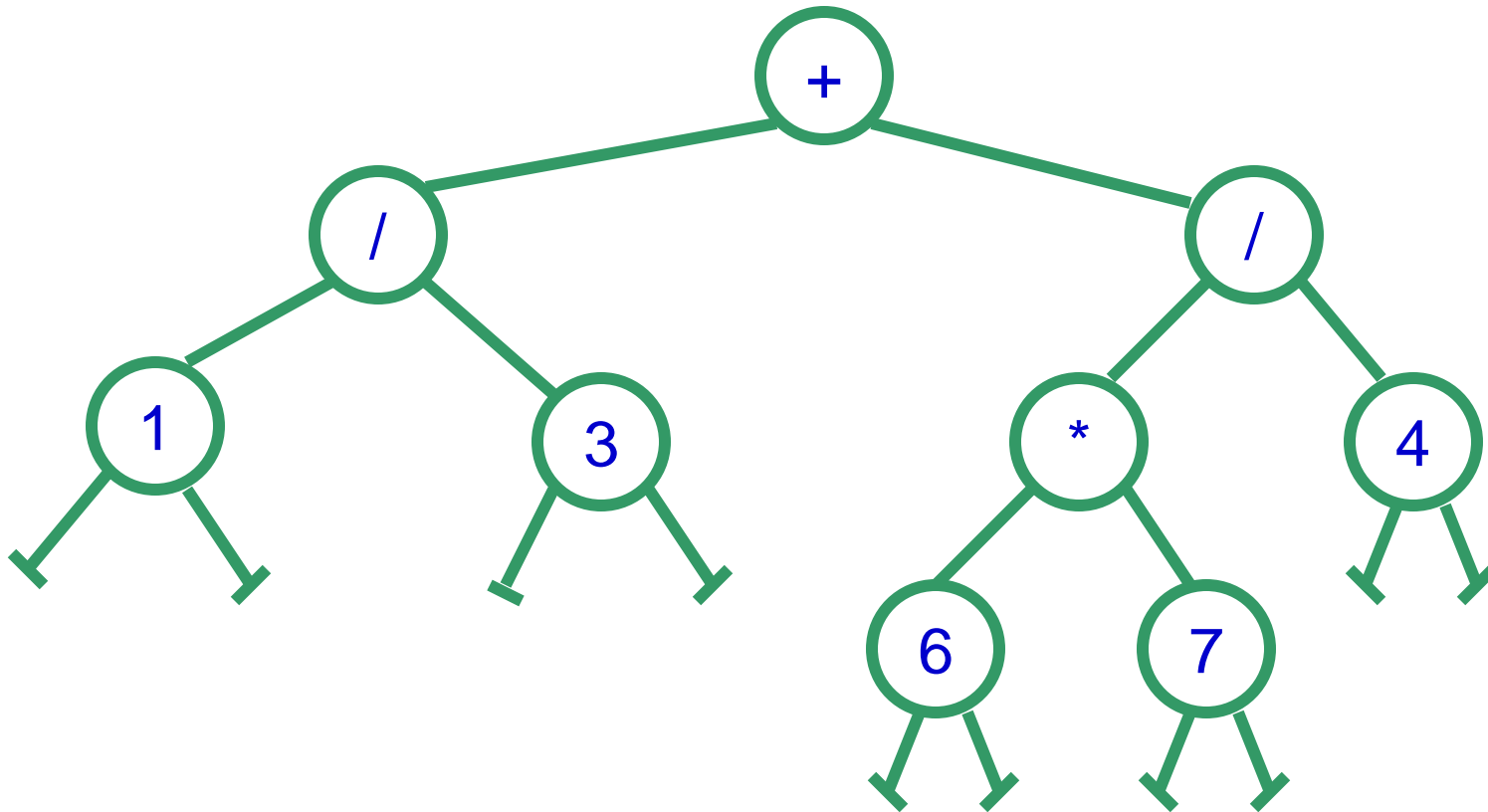


Cho ta biểu thức trung tố (thiếu dấu ngoặc)!

Các ký pháp

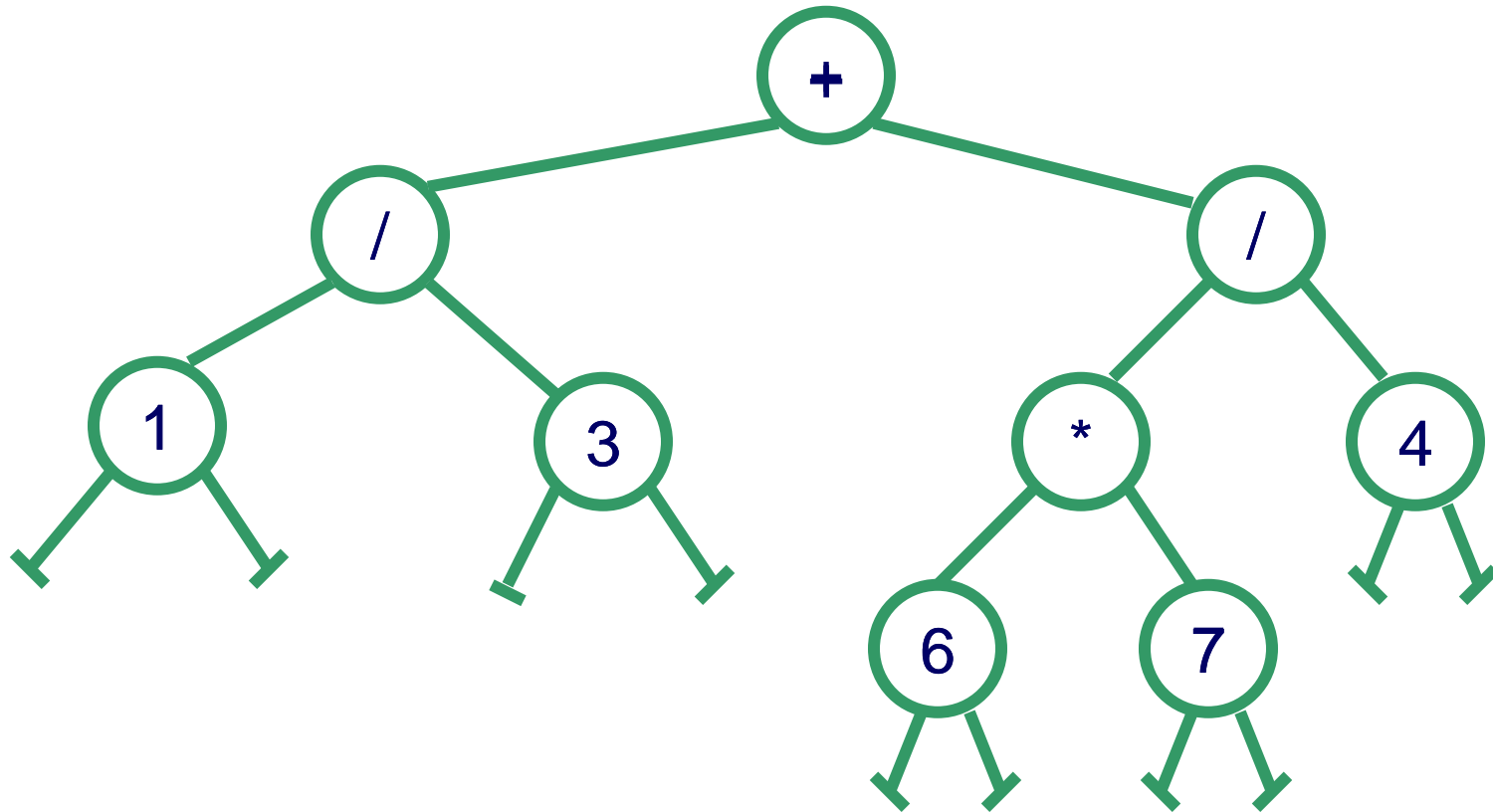
- Duyệt cây biểu thức theo thứ tự trước (Preorder)
 - Cho ta ký pháp tiền tố (Prefix Notation)
- Duyệt cây biểu thức theo thứ tự giữa (Inorder)
 - Cho ta ký pháp trung tố (Infix Notation)
- Duyệt cây biểu thức theo thứ tự sau (Postorder)
 - Cho ta ký pháp hậu tố (Postfix Notation)

Ví dụ: Infix Notation



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

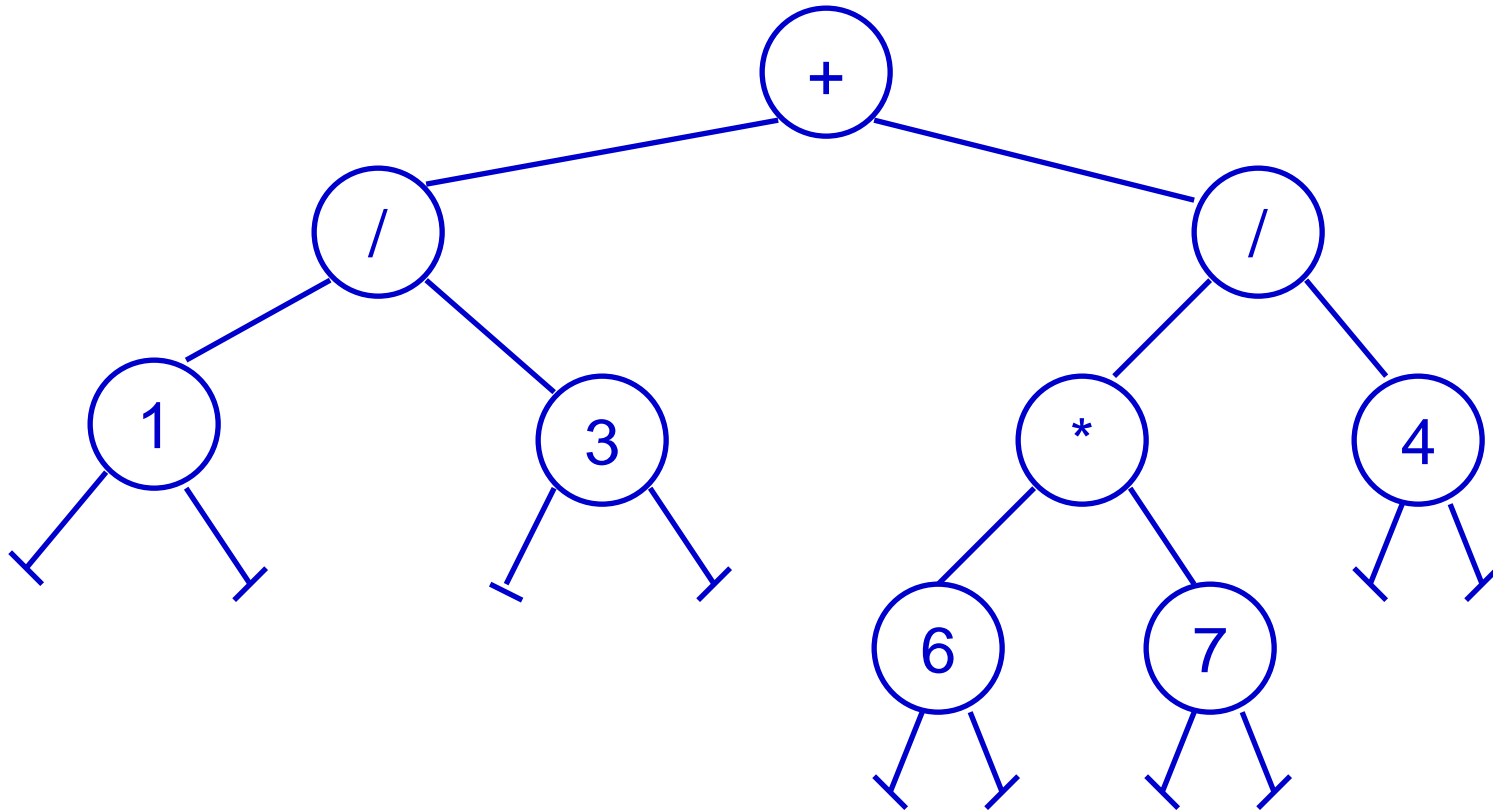
Ví dụ: Postfix Notation



Còn gọi là: Reverse Polish Notation



Ví dụ: Prefix



4.3. Các ví dụ ứng dụng

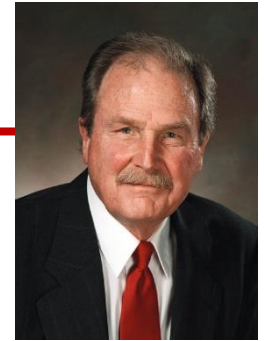
4.3.1. Cây biểu thức

4.3.2. Cây mã Huffman

4.3.3. Mã Huffman

An application of binary trees:
Huffman Code

Mã Huffman



David A. Huffman
1925-1999

- Giả sử có một văn bản trên bảng chữ cái C . Với mỗi chữ cái $c \in C$ ta biết tần suất xuất hiện của nó trong văn bản là $f(c)$. Cần tìm cách mã hoá văn bản sử dụng ít bộ nhớ nhất.
 - Mã hóa với độ dài cố định (fixed length code). Dữ liệu càng nhiều độ dài mã càng lớn, nhưng lại không hiệu quả.
 - Mã phi tiền tố (prefix free code) là cách mã hóa, mọi ký tự c bởi 1 chuỗi nhị phân $\text{code}(c)$ sao cho mã của 1 ký tự bất kỳ không là tiền tố của bất cứ mã của ký tự nào trong số các ký tự còn lại. Tuy nhiên việc mã hóa và giải mã phức tạp hơn nhưng thông tin tiết kiệm được.

Mã hoá độ dài cố định

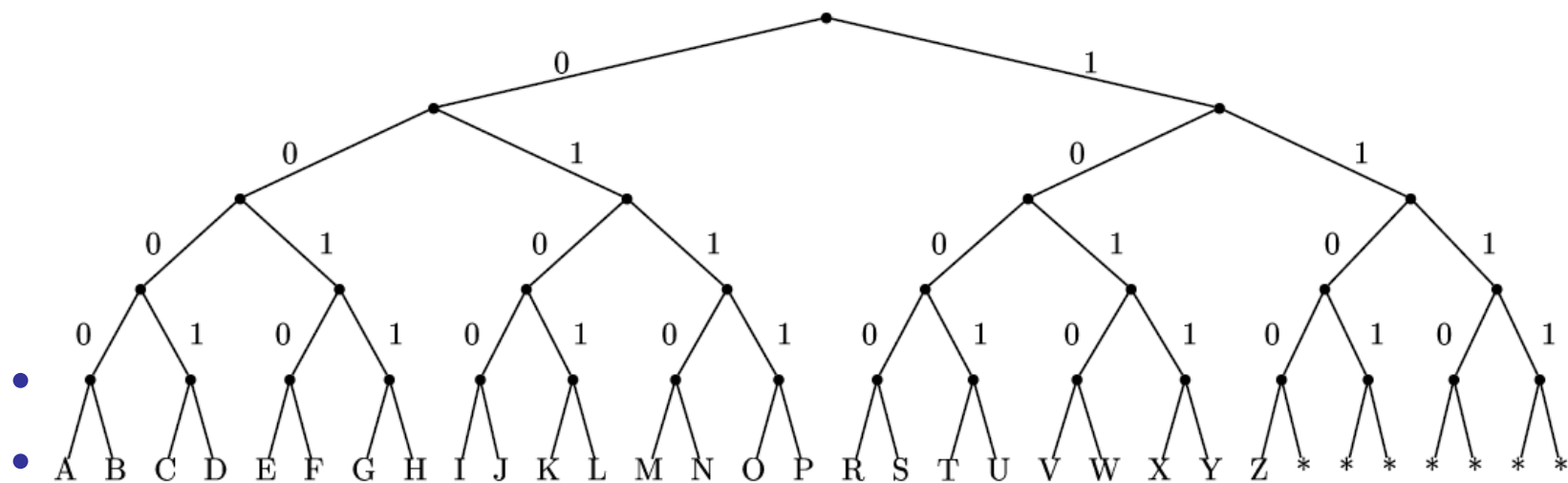
- Để mã hoá 26 chữ cái tiếng Anh bởi mã nhị phân độ dài cố định, độ dài của xâu tối thiểu là $\lceil \log_2 26 \rceil = 5$ bit.

00001	A	01000	H	01111	O	10110	V
00010	B	01001	I	10000	P	10111	W
00011	C	01010	J	10001	Q	11000	X
00100	D	01011	K	10010	R	11001	Y
00101	E	01100	L	10011	S	11010	Z
00110	F	01101	M	10100	T		
00111	G	01110	N	10101	U		

- Các xâu từ 11011 đến 11111 không sử dụng

Cây mã hoá độ dài cố định

- Mã hoá độ dài cố định là mã phi tiền tố



Morse Code

- Căn cứ vào thống kê tần suất của các chữ cái trong tiếng Anh

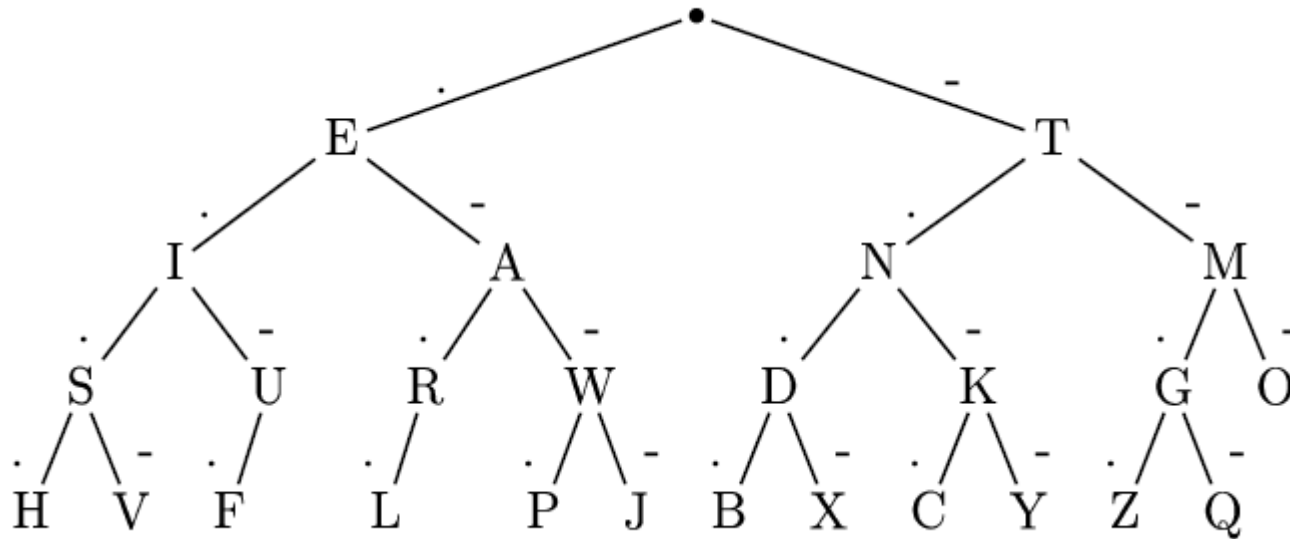
E	12,000
T	9,000
A, O, I, N, S	8,000
H	6,400
R	6,200
D	4,400
L	4,000
U	3,400

- Morse đề xuất cách mã hoá:

.	E	..	I	—	T	— —	M
.—	A	...	S	—.	N	.. —	U

Cây mã hoá Morse

- Mã hoá Morse không là phi tiền tố



- Giải mã “....-” ??? Chịu chết?
- Phải có dấu phân biệt các chữ cái: “..#..-” ➔ IU
- Cây không đầy đủ

Mã Huffman

- Mỗi mã phi tiền tố có thể biểu diễn bởi một cây nhị phân T mà mỗi lá của nó tương ứng với một chữ cái và cạnh của nó được gán cho một trong hai số 0 hoặc 1.
- Mã của một chữ cái c là 1 dãy nhị phân gồm các số gán cho các cạnh trên đường đi từ gốc đến lá tương ứng với c .

Mã Huffman

- **Bài toán:** Tìm cách mã hoá tối ưu, tức là tìm cây nhị phân T làm tối thiểu hoá tổng độ dài có trọng số

$$B(T) = \sum_{c \in C} f(c) \text{ depth}(c)$$

trong đó $\text{depth}(c)$ là độ dài đường đi từ gốc đến lá tương ứng với ký tự c .

Mã Huffman

- Ý tưởng thuật toán:
- Chữ cái có tần suất nhỏ hơn cần được gán cho lá có khoảng cách đến gốc là lớn hơn; chữ cái có tần suất xuất hiện lớn hơn cần được gán cho nút gần gốc hơn

Mã Huffman: Thuật toán

```
procedure Huffman(C, f);  
begin  
  n ← |C|;  
  Q ← C;  
  for i:=1 to n-1 do  
    begin  
      x, y ← 2 chữ cái có tần suất nhỏ nhất trong Q; (* Thao tác 1 *)  
      Tạo nút p với hai con x, y;  
      f(p) := f(x) + f(y);  
      Q ← Q \ {x, y} ∪ {p} (* Thao tác 2 *)  
    end;  
  end;  
end;
```

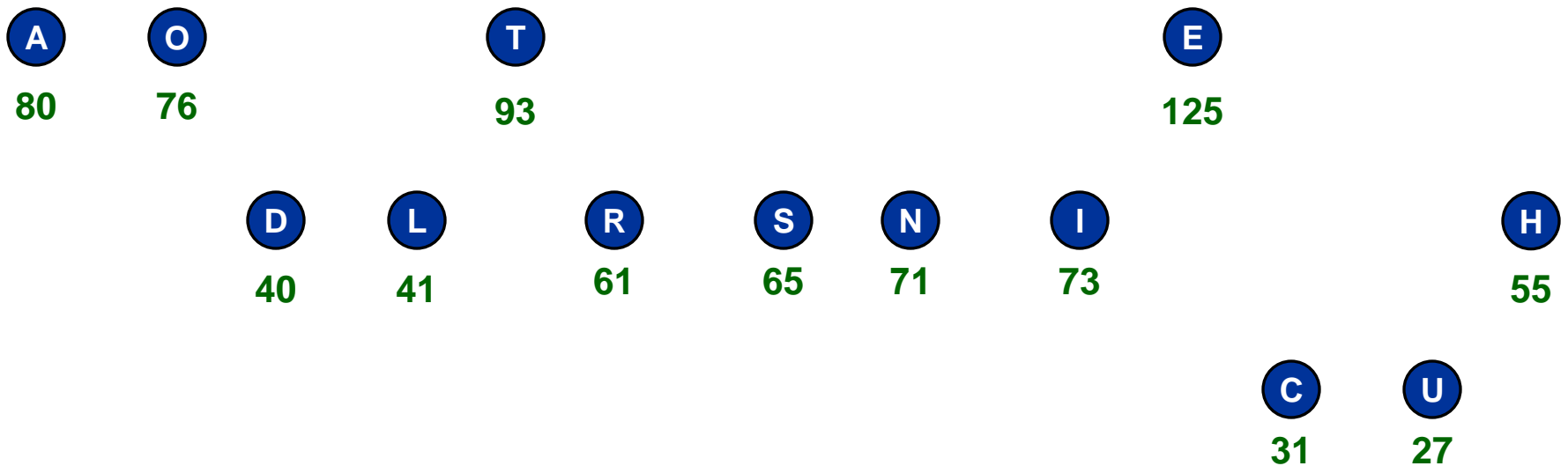
- Mã xây dựng theo thuật toán Huffman thường được gọi là mã Huffman (Huffman Code).
- Có thể cài đặt với thời gian $O(n \log n)$ sử dụng **priority queue**

Xây dựng mã Huffman

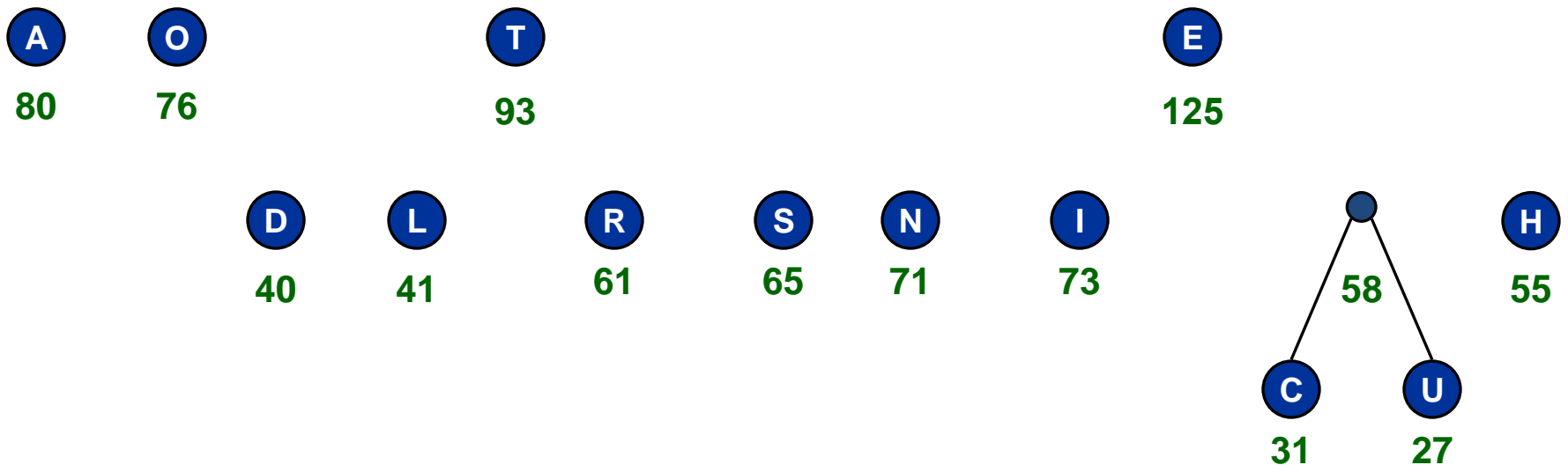
- Tần suất của các ký tự trong văn bản.

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

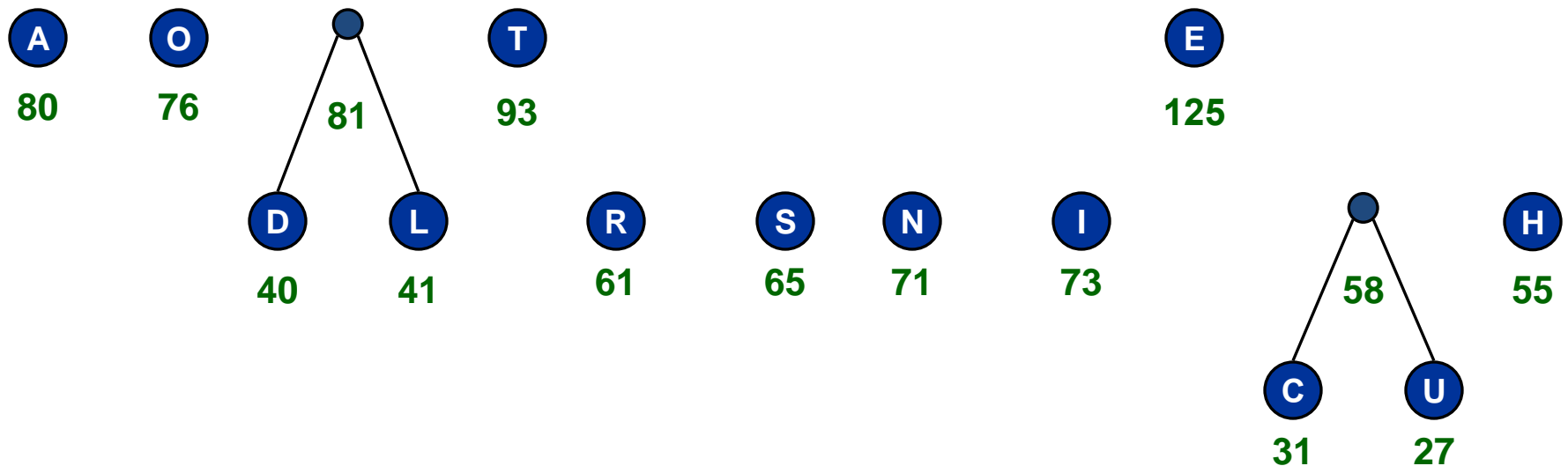
Xây dựng mã Huffman



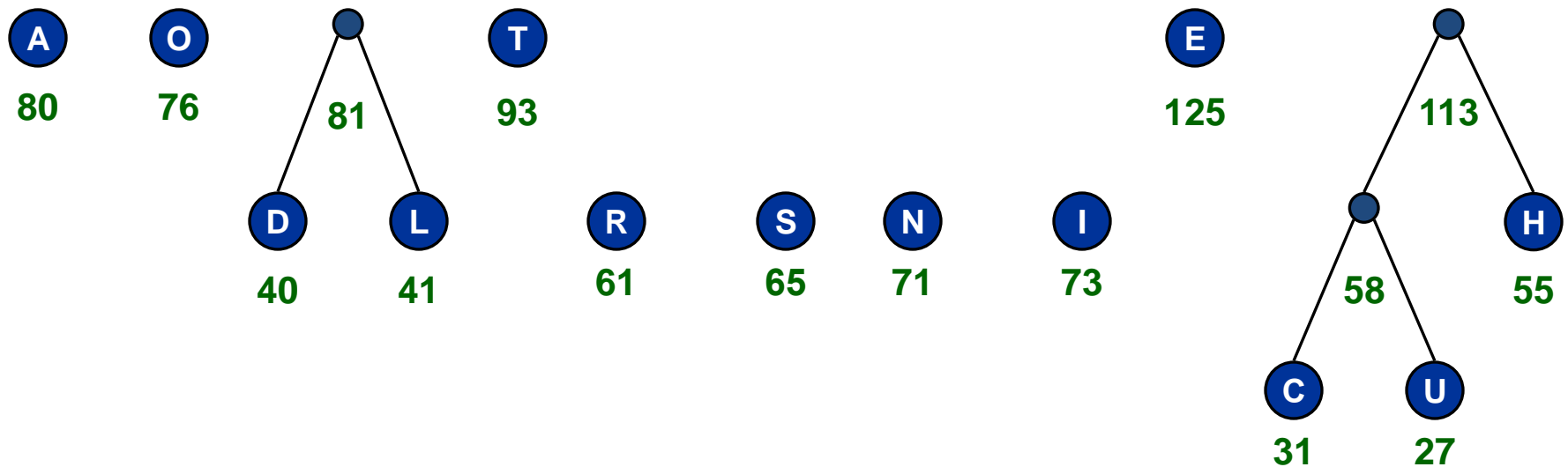
Xây dựng mã Huffman



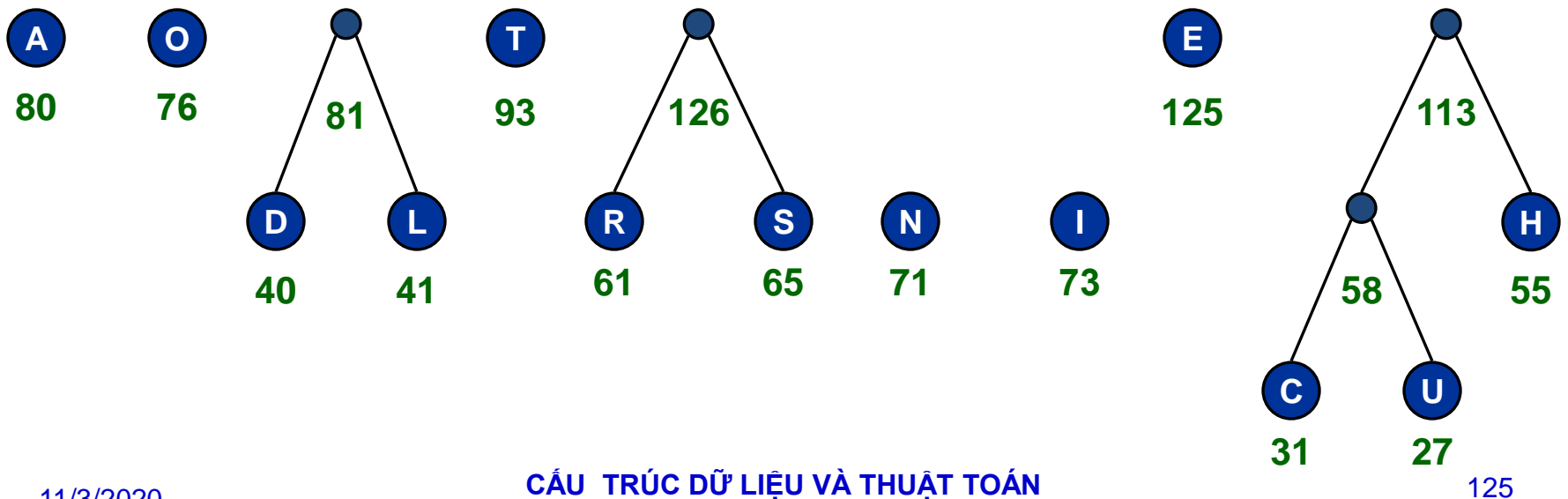
Xây dựng mã Huffman



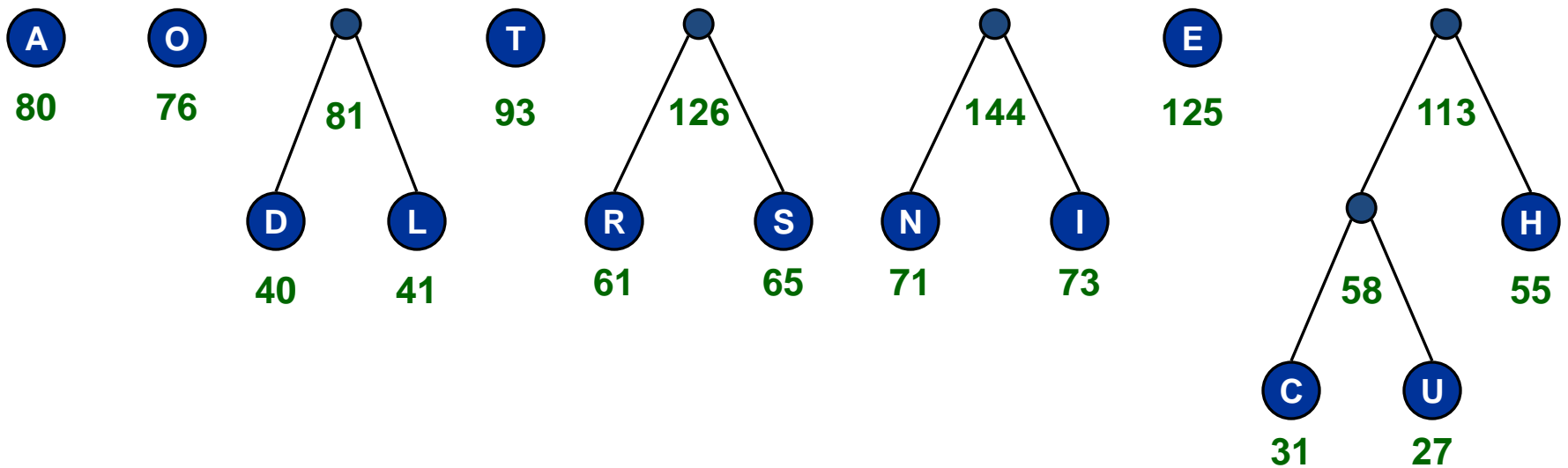
Xây dựng mã Huffman



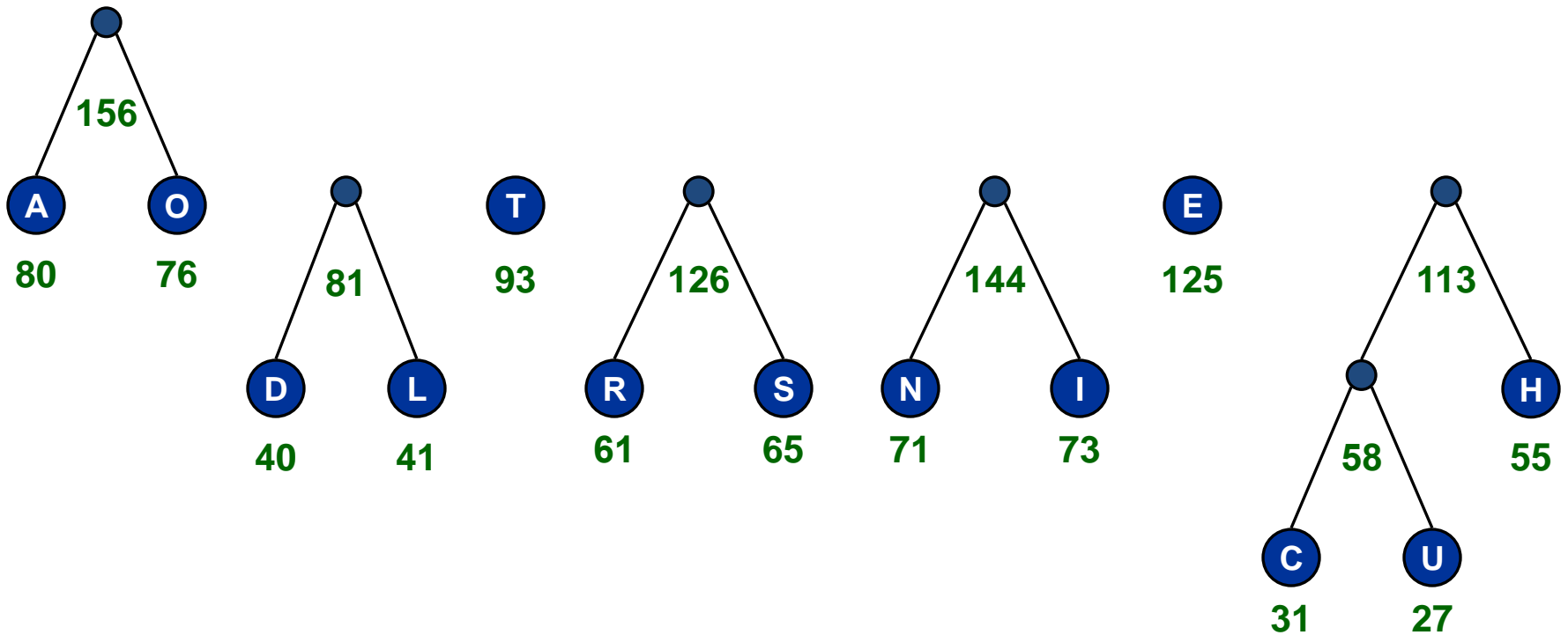
Xây dựng mã Huffman



Xây dựng mã Huffman

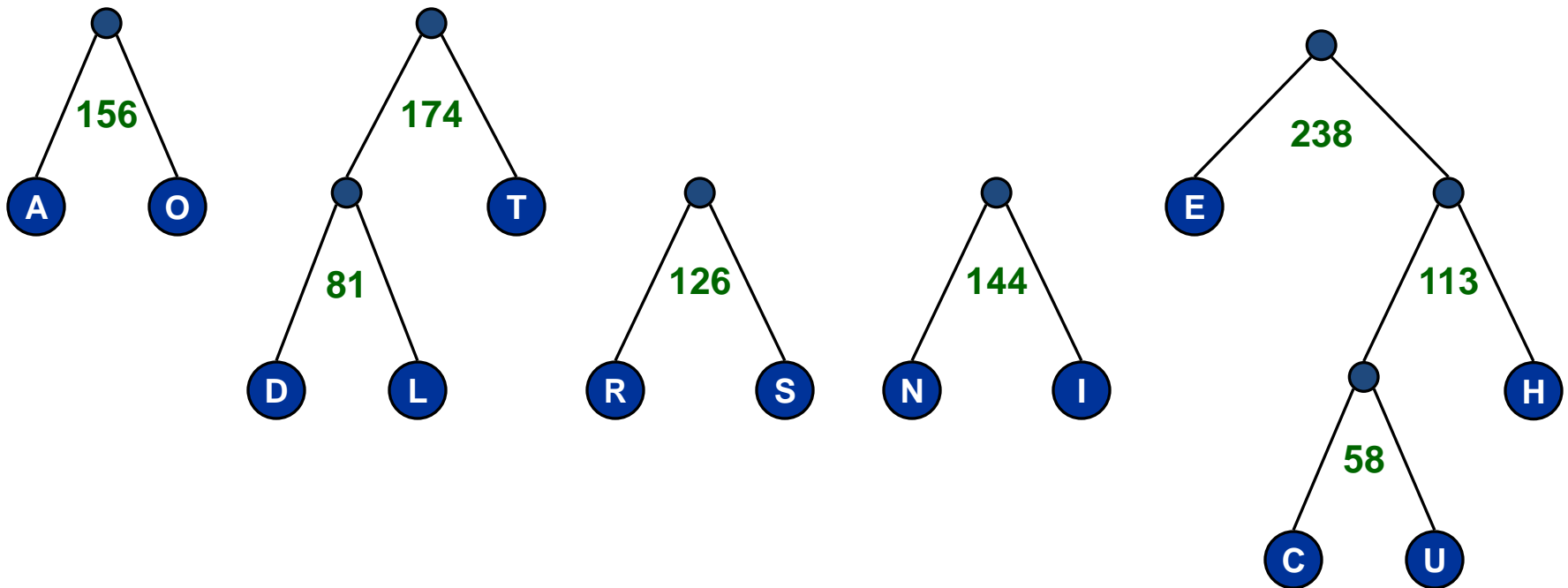


Xây dựng mã Huffman

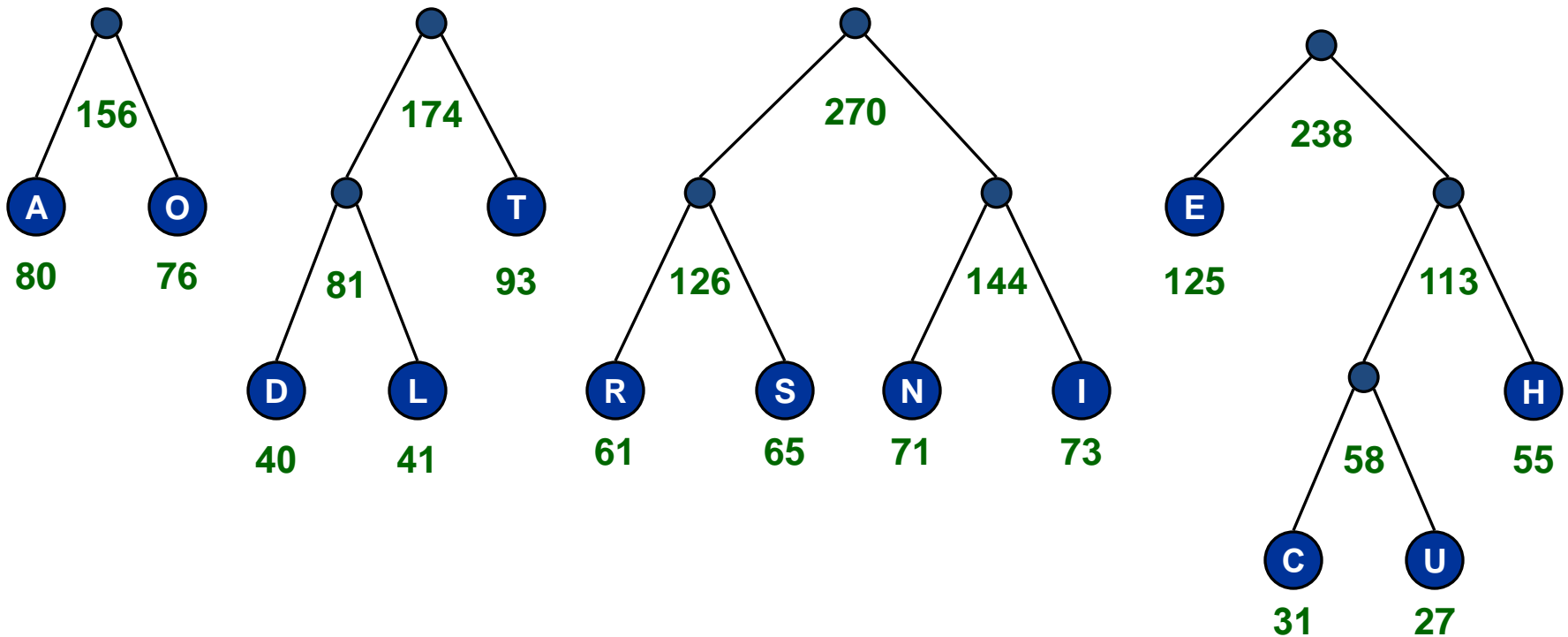




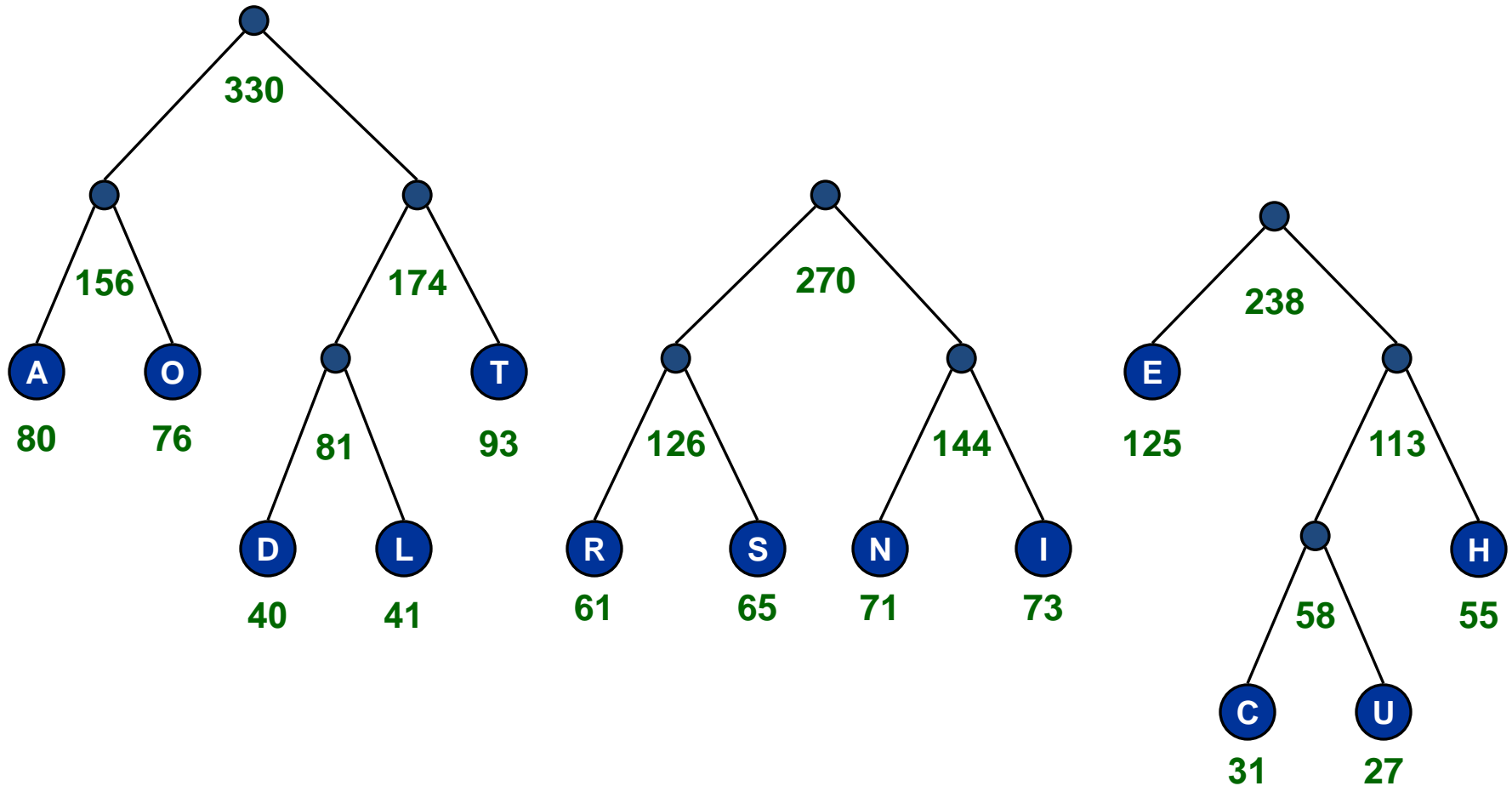
Xây dựng mã Huffman



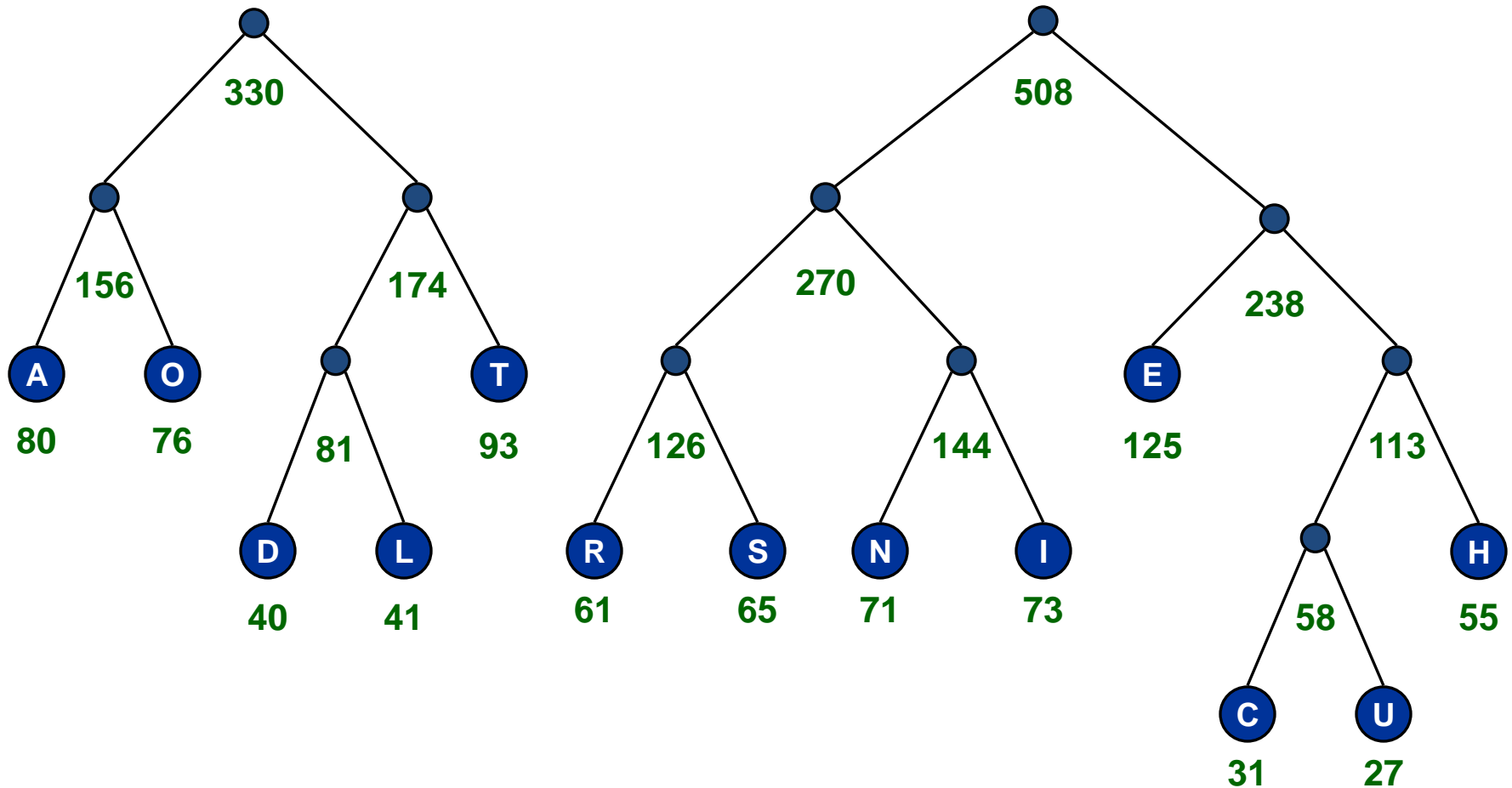
Xây dựng mã Huffman



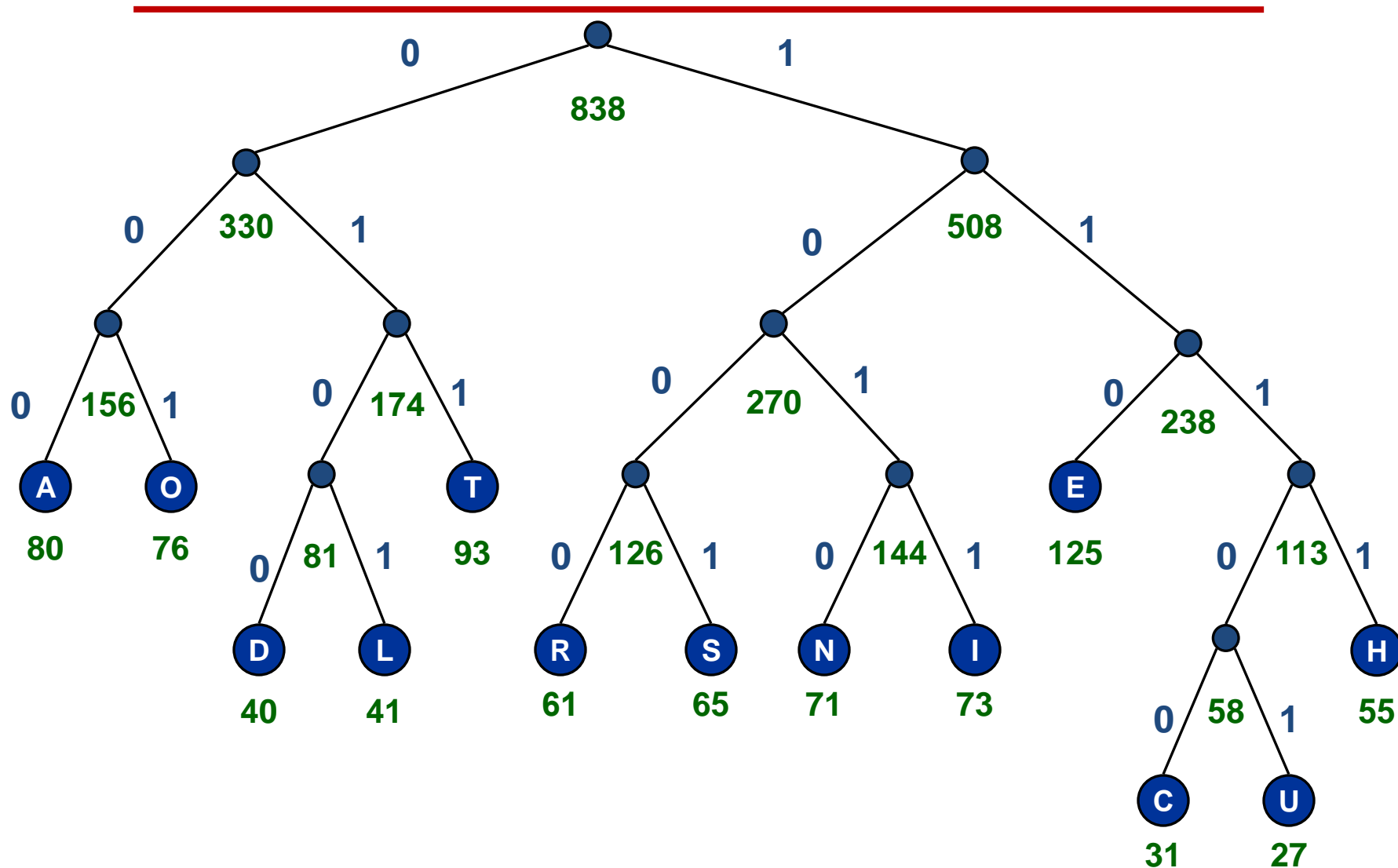
Xây dựng mã Huffman



Xây dựng mã Huffman



Xây dựng mã Huffman



Xây dựng mã Huffman

Char	Freq	Fixed	Huff
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
Tæng	838	3352	3036

Mã Huffman: Giải mã

```
procedure Huffman_Decode(B);  
(* B là xâu mã hóa văn bản theo mã hoá Huffman. *)  
begin  
  <Khởi ®éng P là gốc của cây Huffman>  
  While <cha ®¹t ®Ổn 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 nốt lá) then  
        begin  
          <Hiển thị ký hiệu tương ứng với nốt lá>  
          <Đặt lại P là gốc của cây Huffman>  
        end;  
      end  
    end;
```

QUESTIONS?