HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

Cây: định nghĩa và các khái niệm chung, cây nhị phân

ONE LOVE. ONE FUTURE.

MỤC TIÊU

Sau bài học này, người học có thể:

1. Hiểu được khái niệm cấu trúc dữ liệu cây và các khái niệm liên quan.

2. Cài đặt được cấu trúc dữ liệu cây.

NỘI DUNG TIẾP THEO

1. Tìm kiếm tuần tự

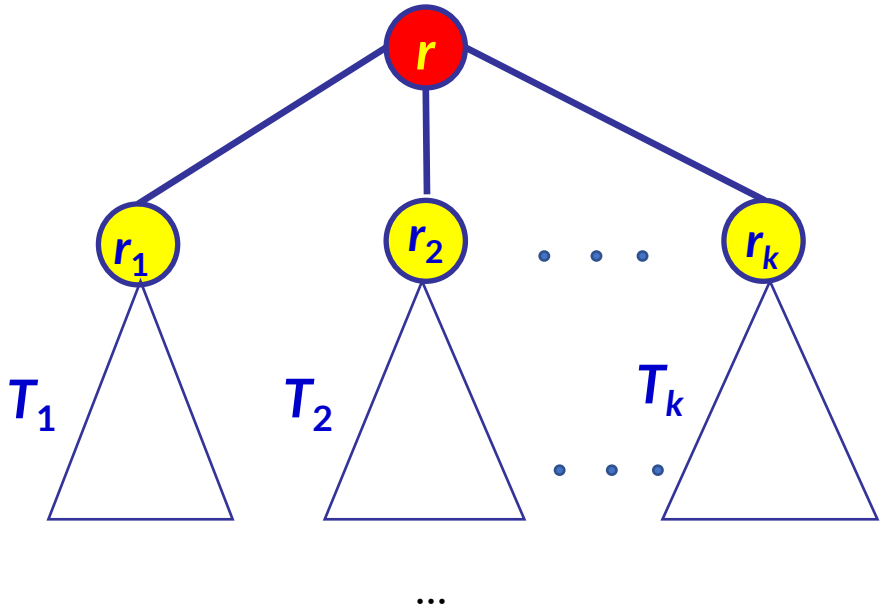
2. Cây nhị phân

1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

- 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 qui: Giả sử $T_1,T_2,...,T_k$ là các cây với gốc là $r_1,r_2,...,r_k$*
- *Xây dựng cây mới bằng cách đặt r làm cha của các nút $r_1,r_2,..., r_k$*
- *Trong cây này r là gốc và $T_1, T_2, . . . , T_k$ là các cây con của gốc r. Các nút $r_1, r_2, . . . , r_k$ được gọi là con của nút r*

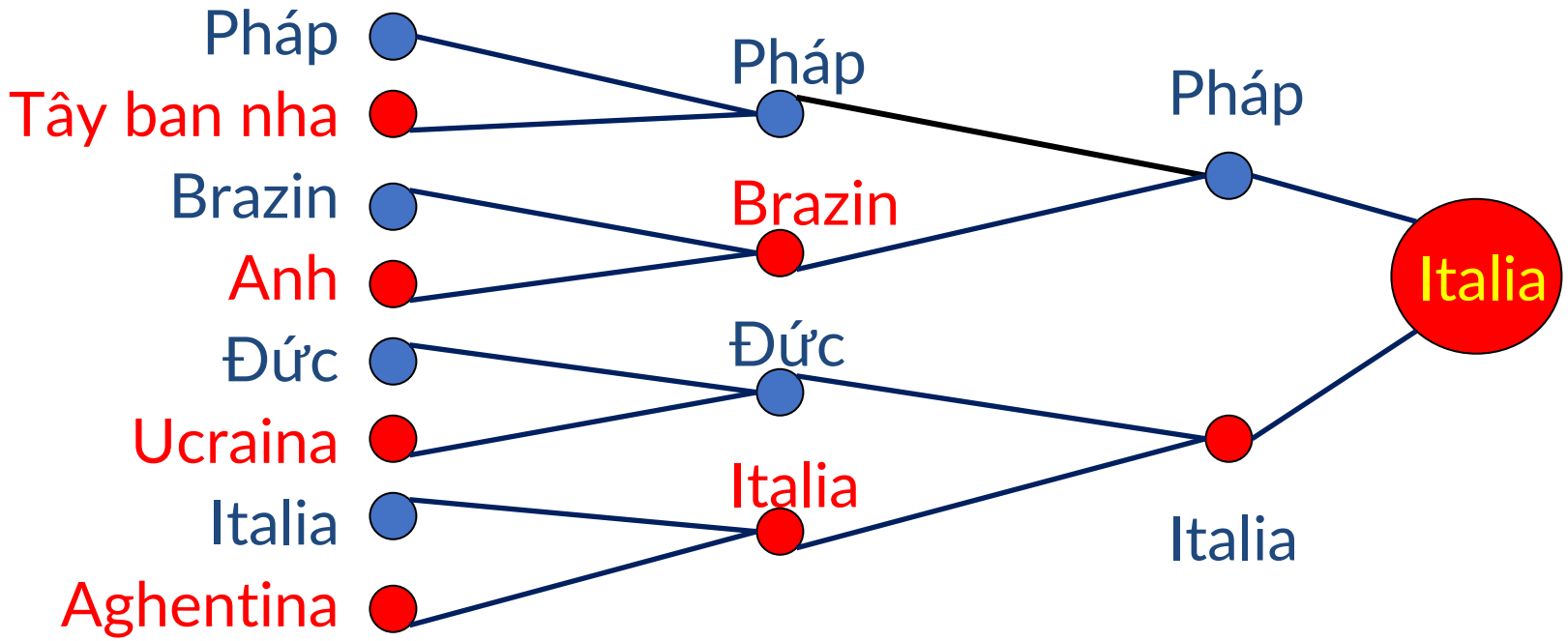


Định nghĩa cây

Chú ý: *Cây rỗng (null tree) là cây không có nút nào cả*

1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

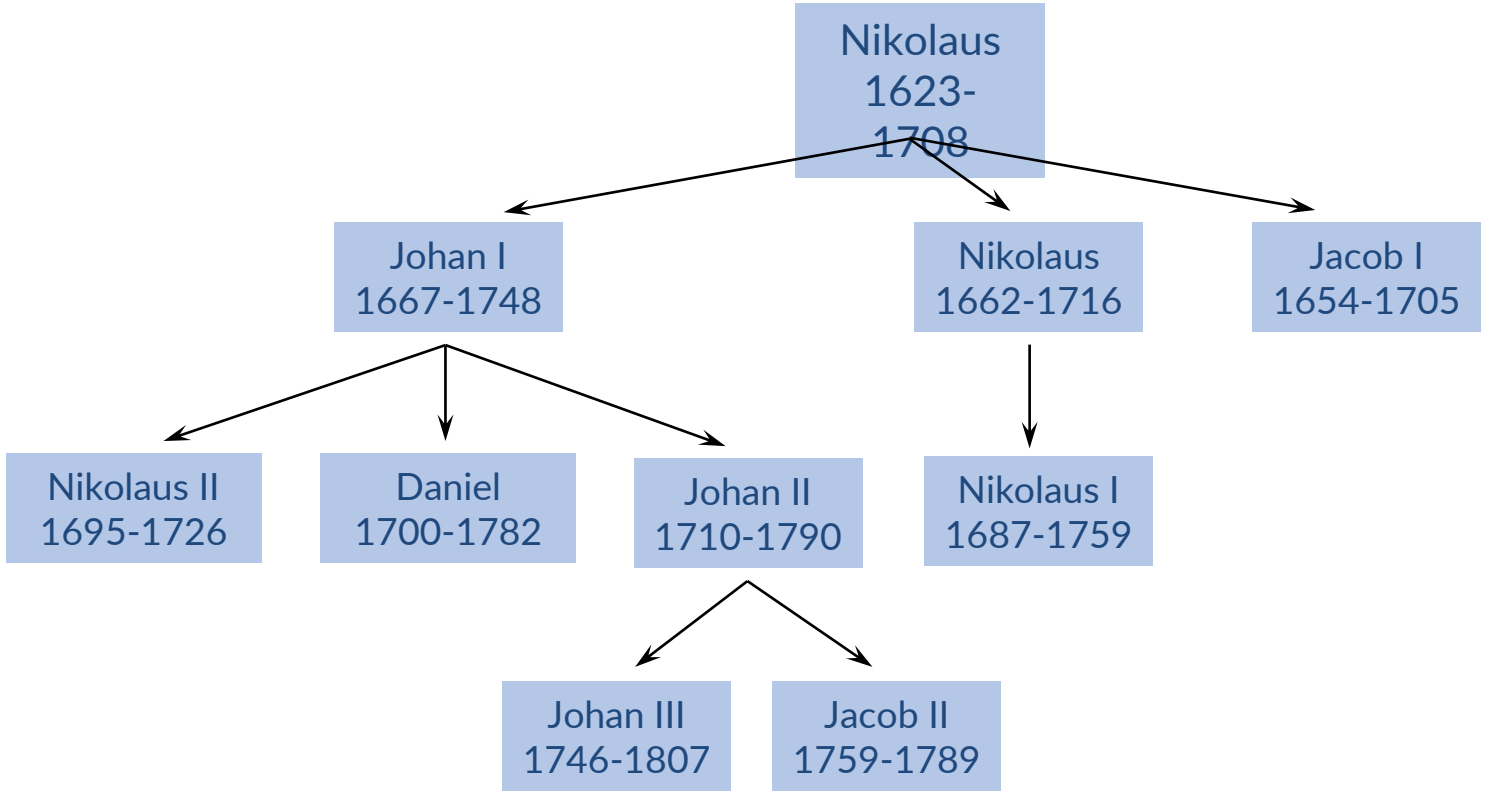
- 1.1. Định nghĩa cây
  - Ví dụ cây thực tế trong các ứng dụng



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

1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

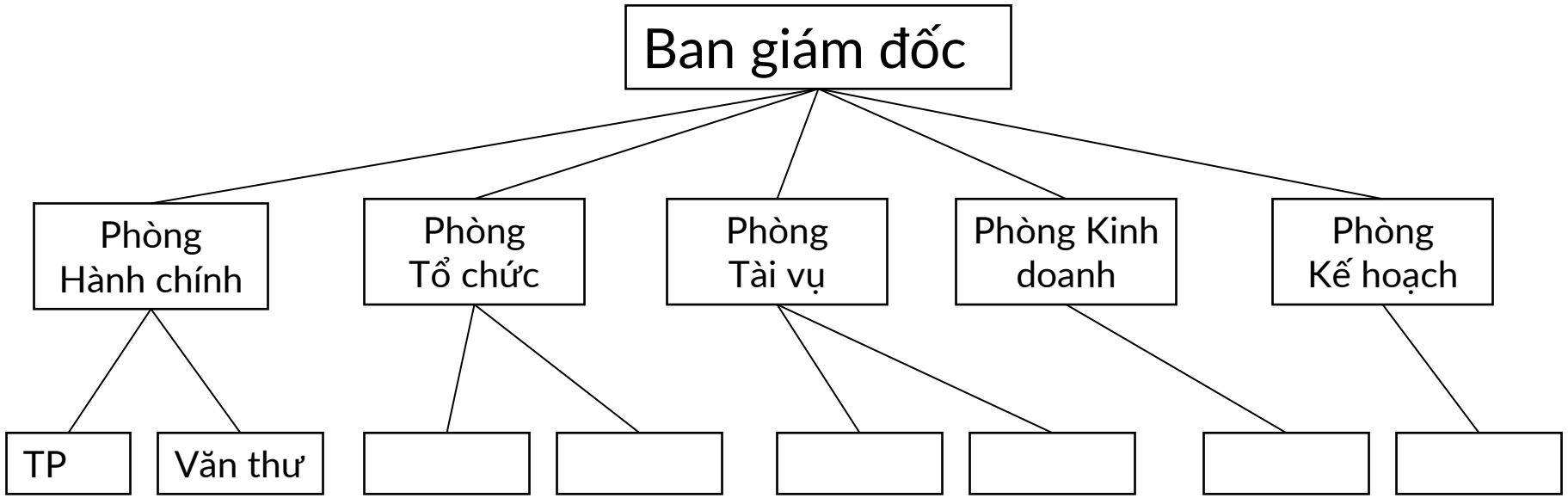
- 1.1. Định nghĩa cây
  - Ví dụ cây thực tế trong các ứng dụng



Cây gia phả của các nhà toán dòng họ Bernoulli

1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

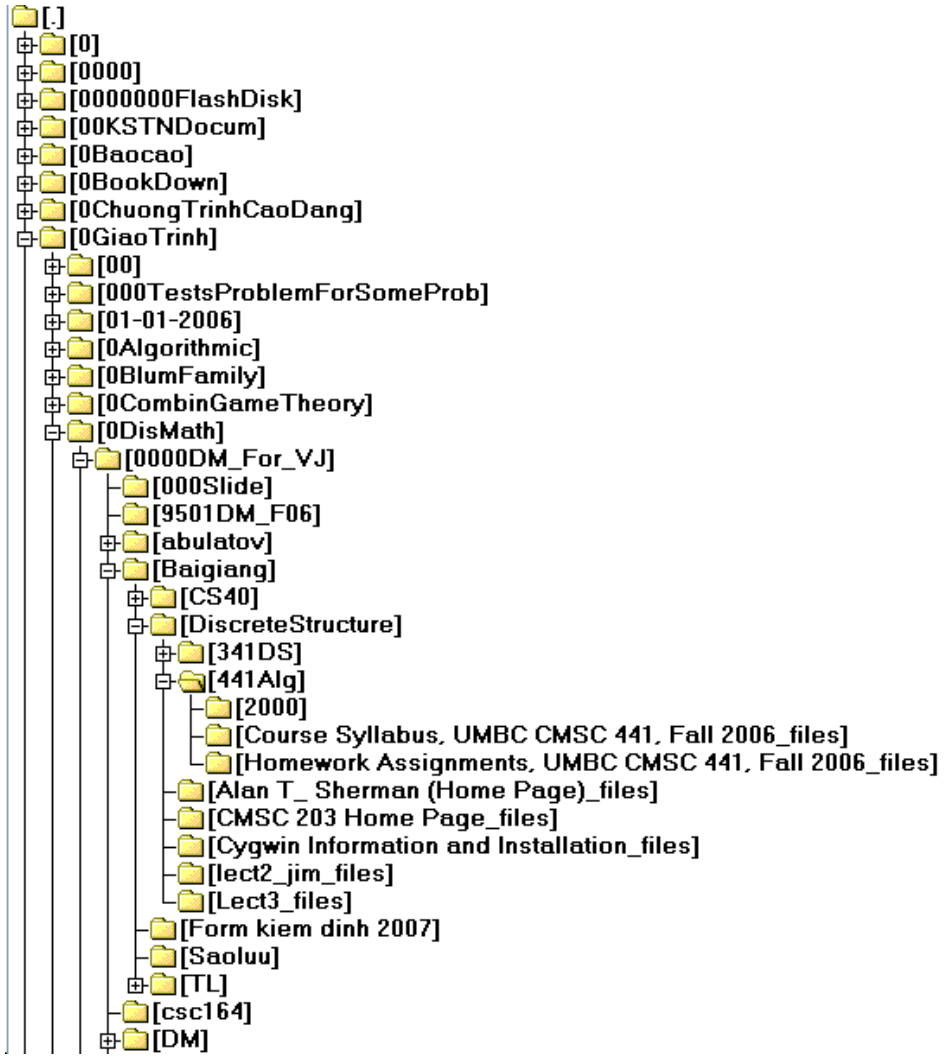
- 1.1. Định nghĩa cây
  - Ví dụ cây thực tế trong các ứng dụng



Cây phân cấp quản lý hành chính

1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

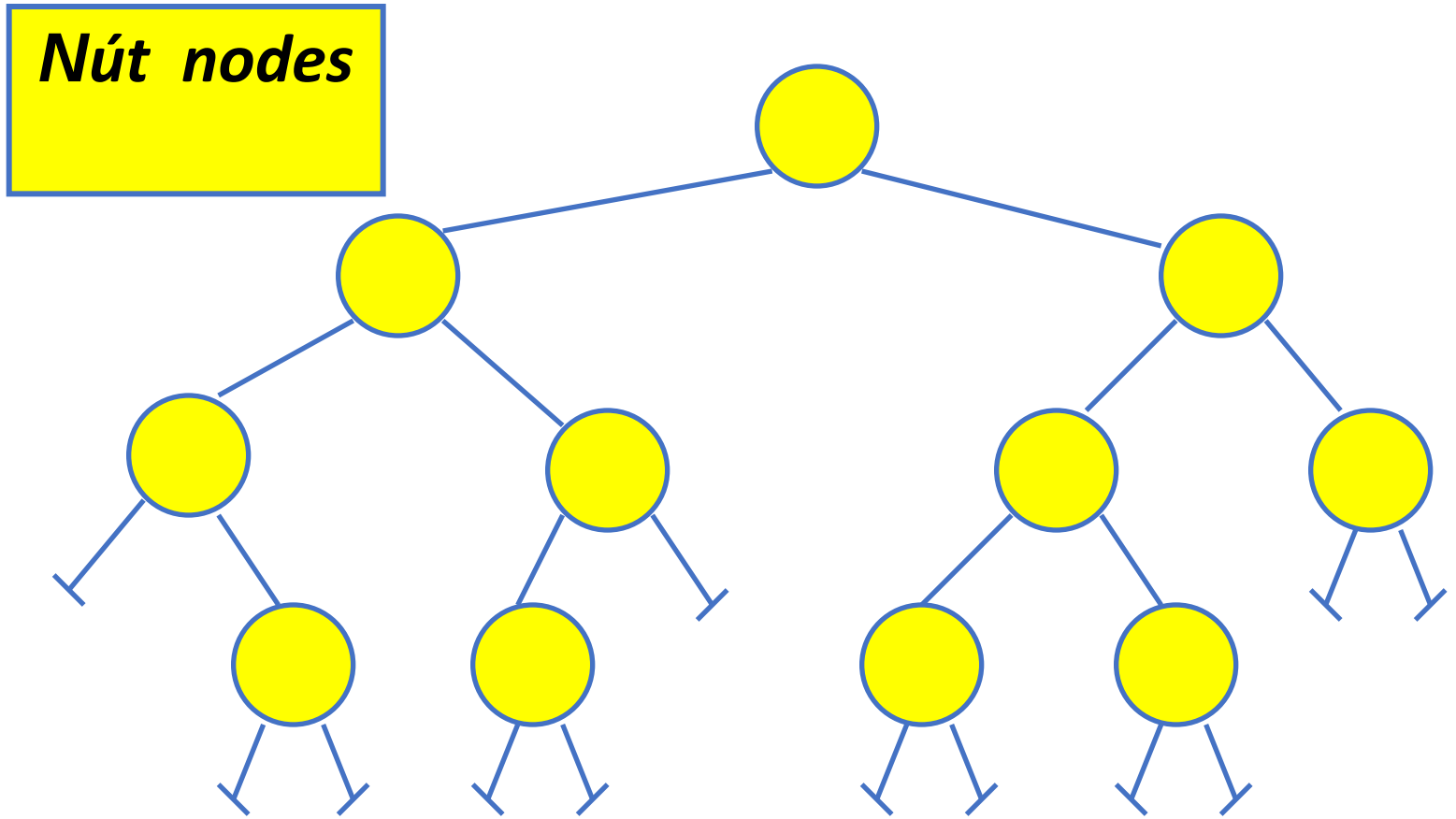
- 1.1. Định nghĩa cây
  - Ví dụ cây thực tế trong các ứng dụng



Cây thư mục

1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

- 1.2. Các thuật ngữ

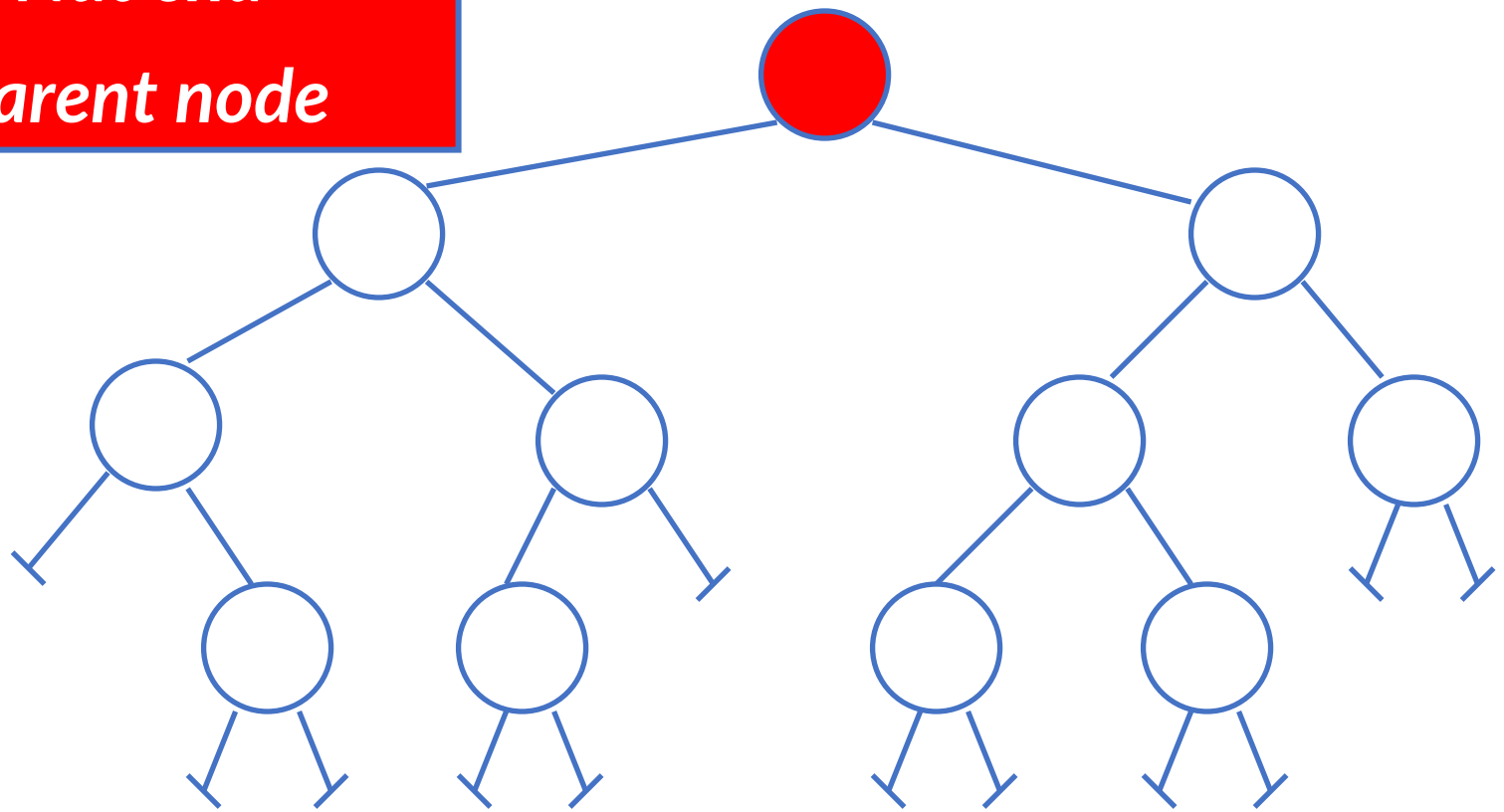


1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

- 1.2. Các thuật ngữ

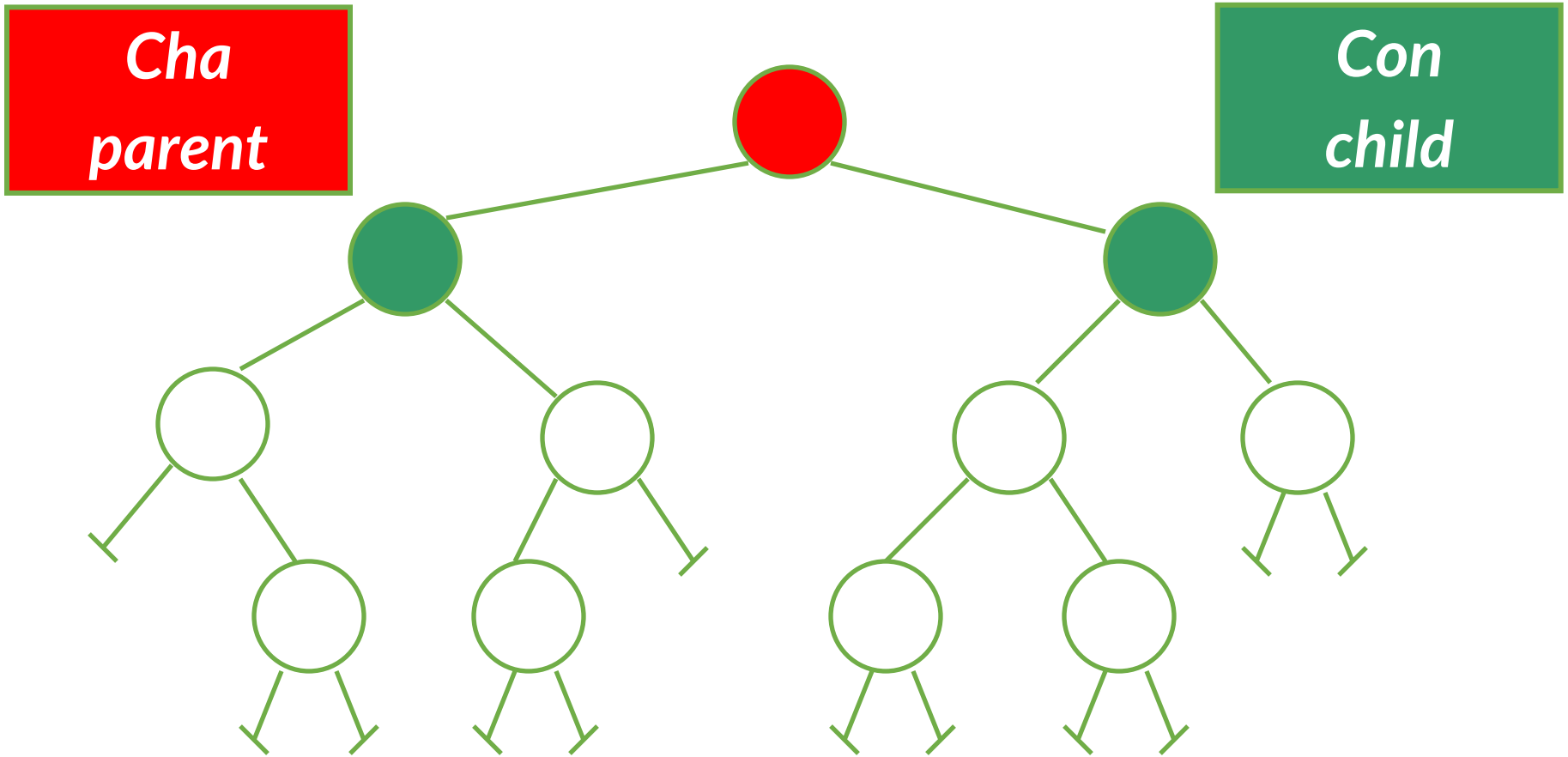
Nút cha

parent node



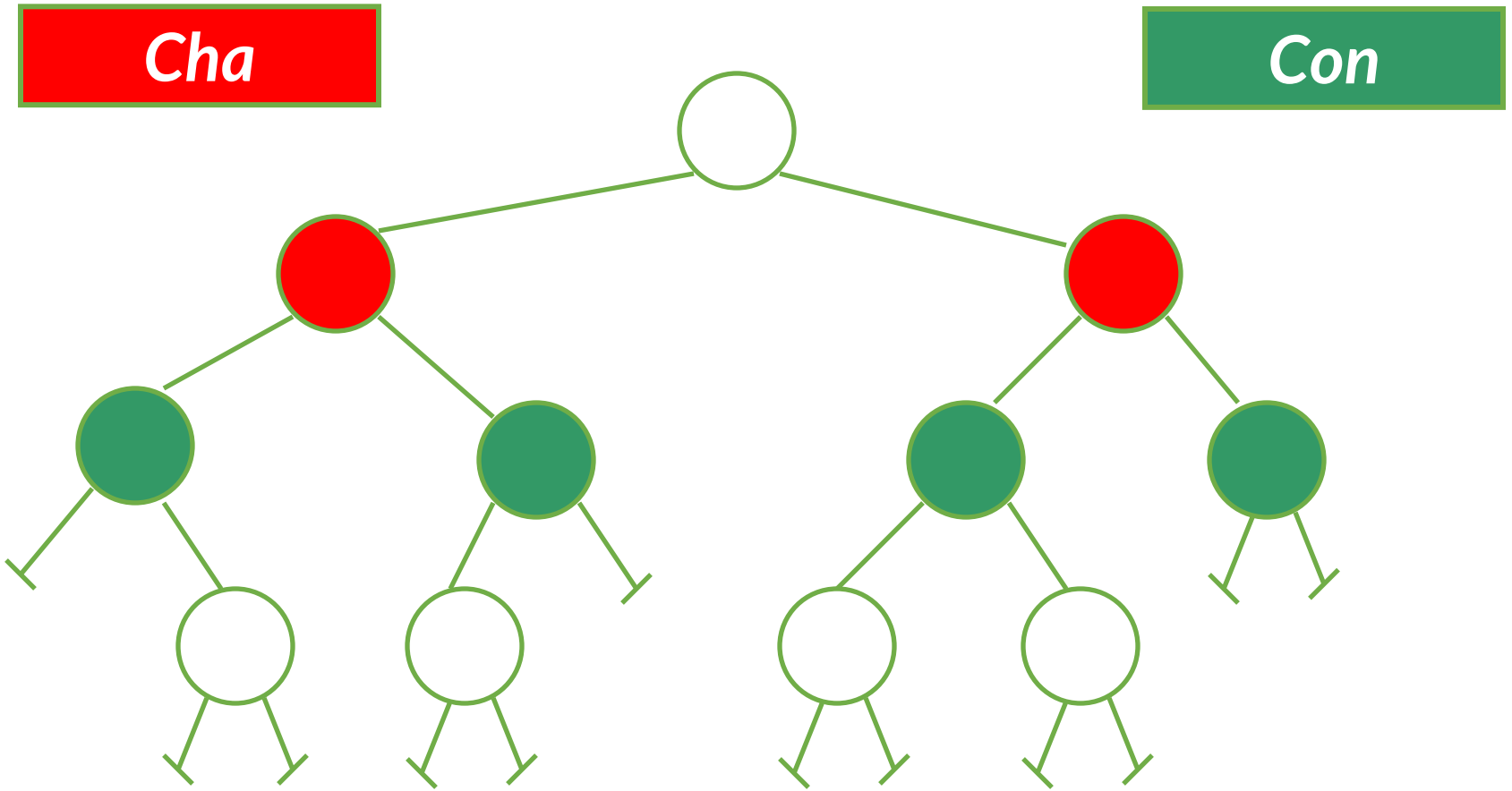
1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

- 1.2. Các thuật ngữ



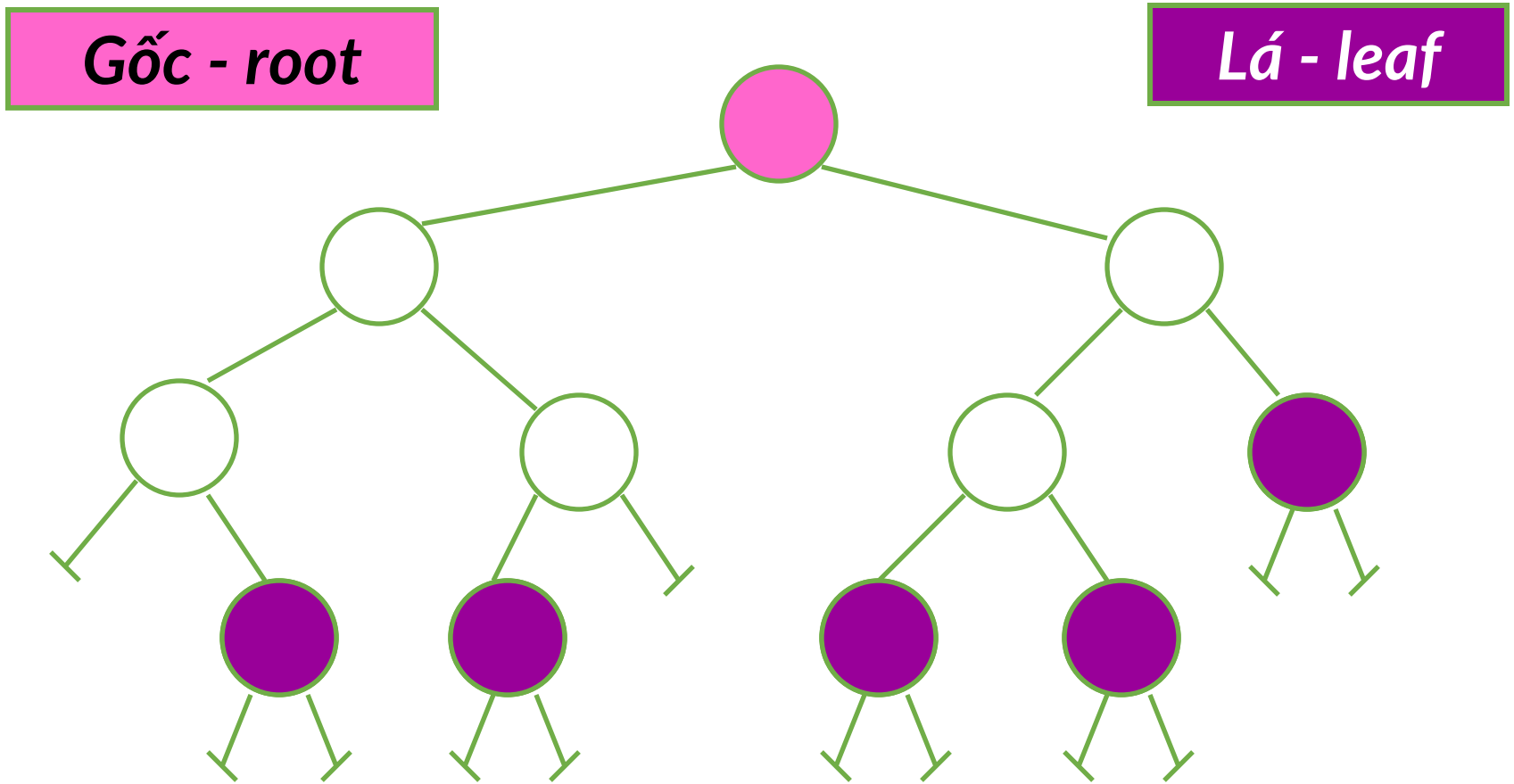
1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

- 1.2. Các thuật ngữ



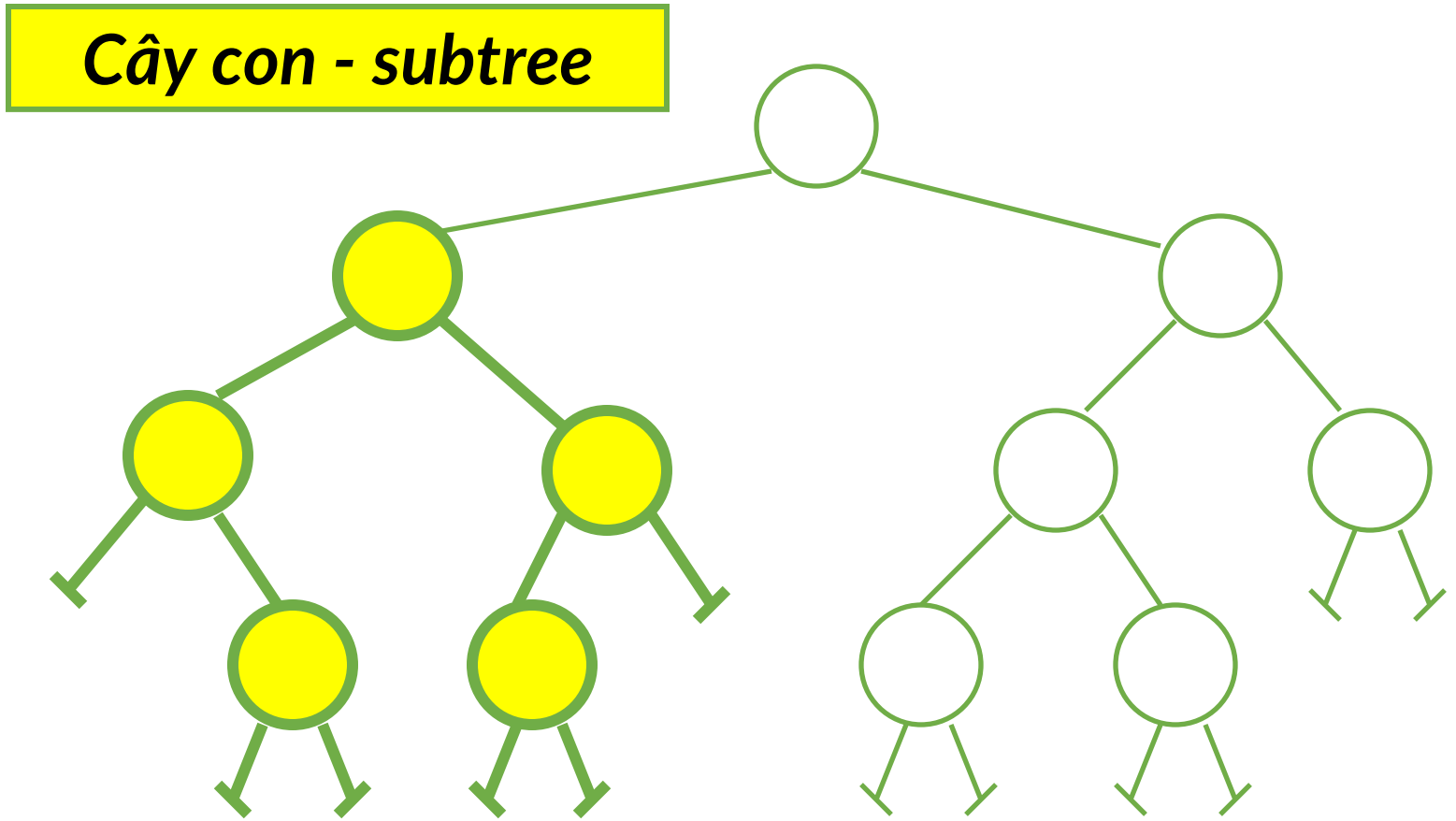
1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

- 1.2. Các thuật ngữ



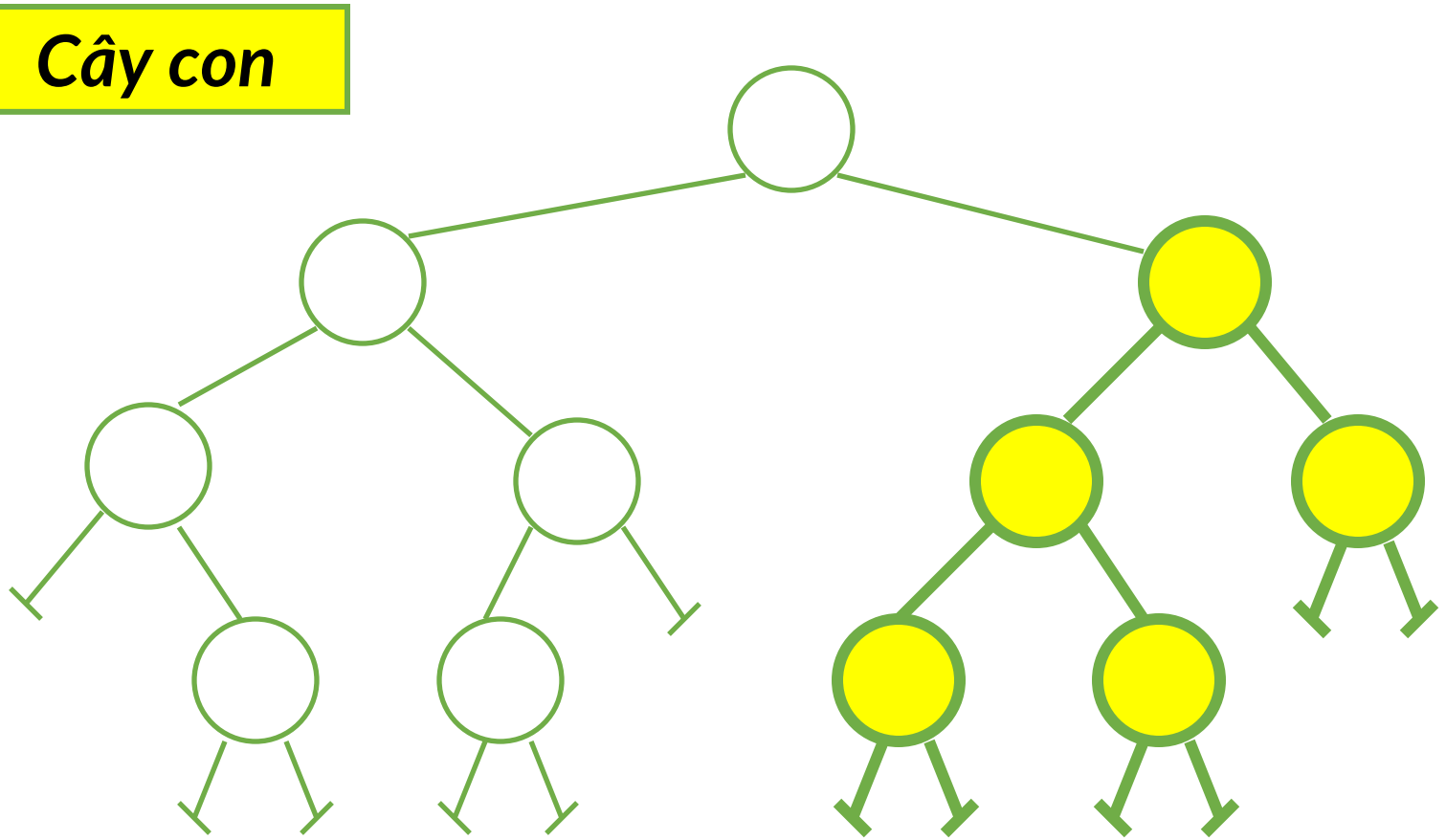
1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

- 1.2. Các thuật ngữ



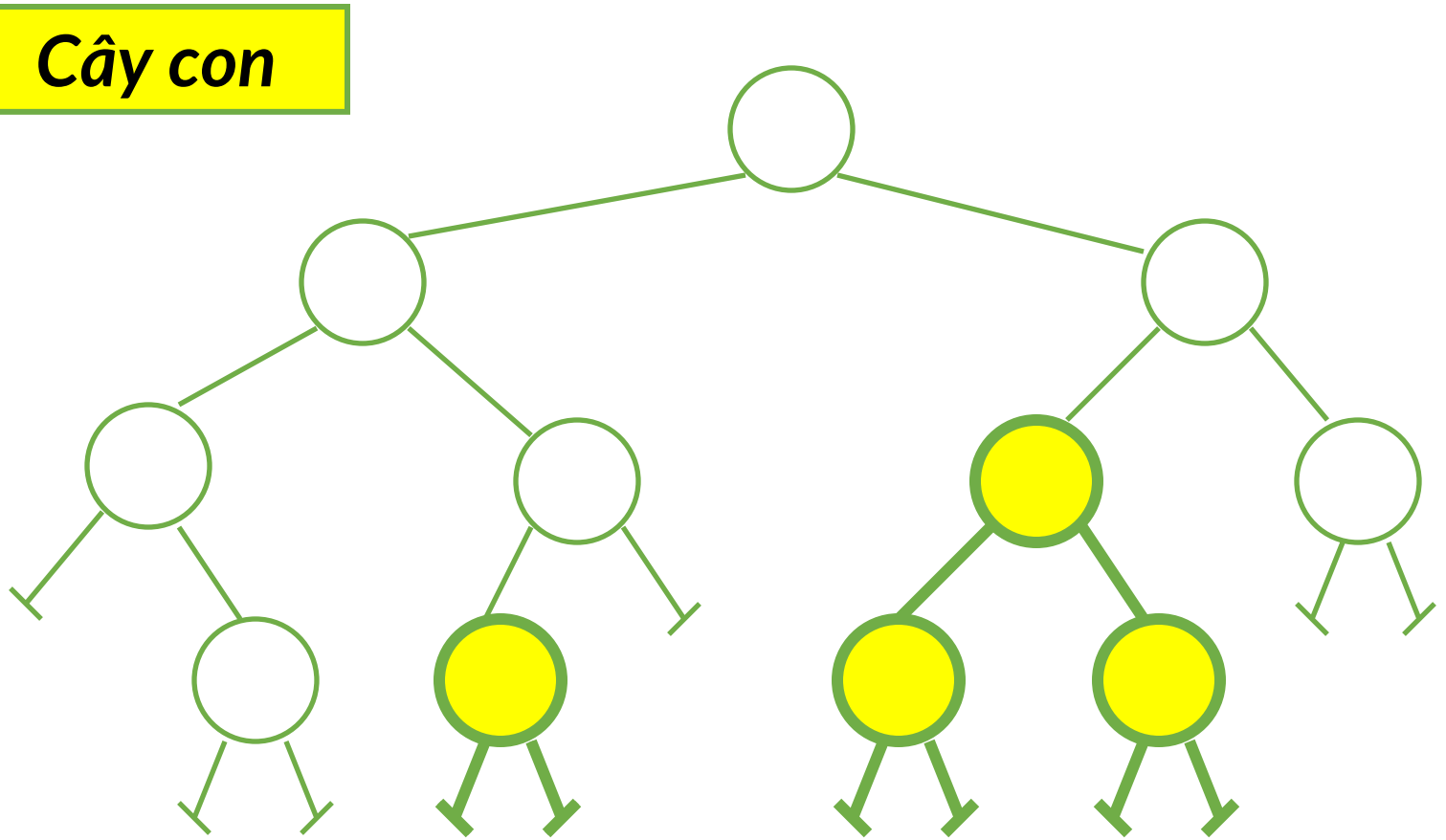
1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

- 1.2. Các thuật ngữ



1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

- 1.2. Các thuật ngữ

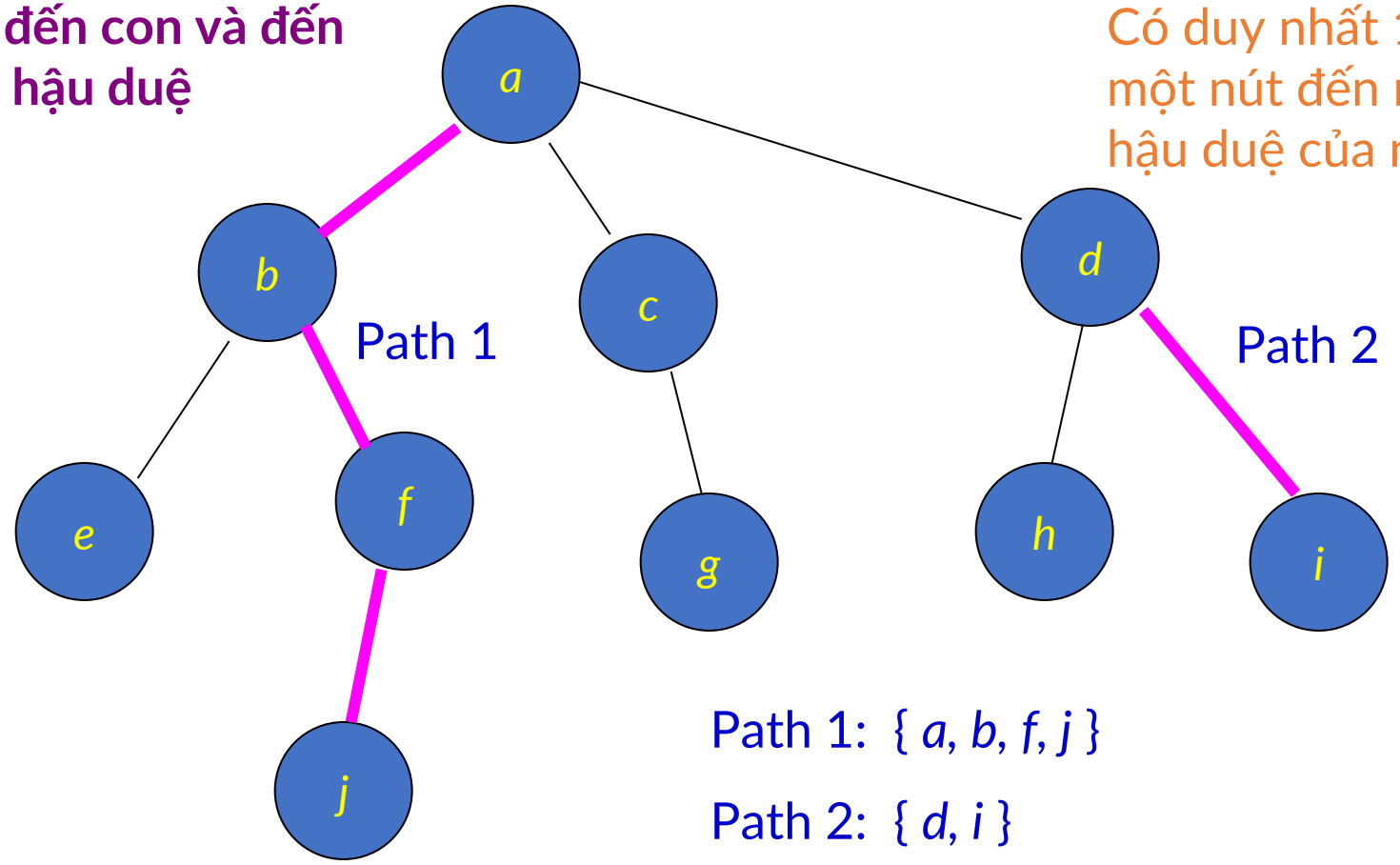


1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

- 1.2. Các thuật ngữ

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



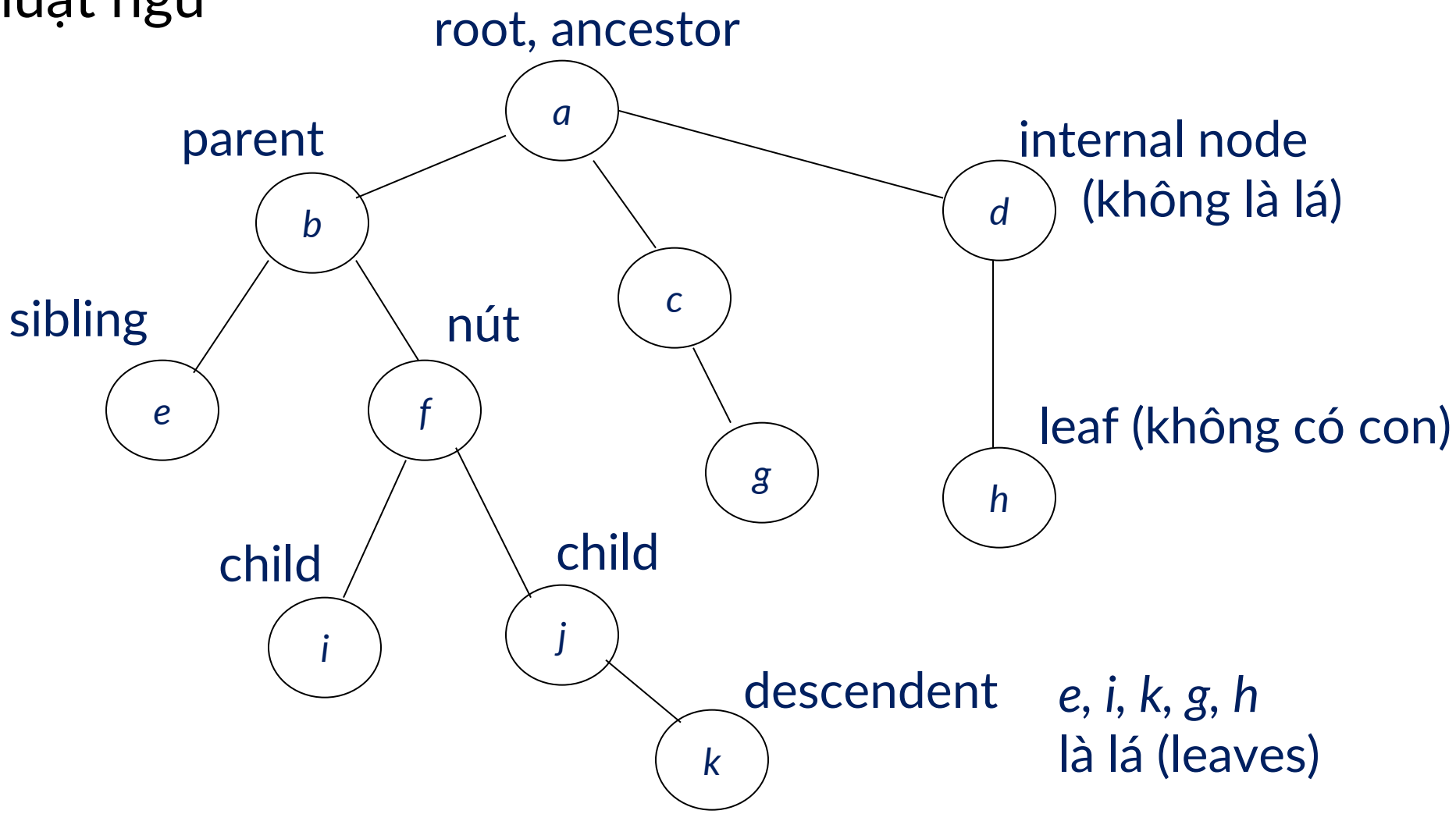
Path 1: { *a*, *b*, *f*, *j* }

Path 2: { *d*, *i* }

Đường đi trên cây

1. ĐỊNH NGHĨA VÀ CÁC KHÁI NIỆM CHUNG

- 1.2. Các thuật ngữ

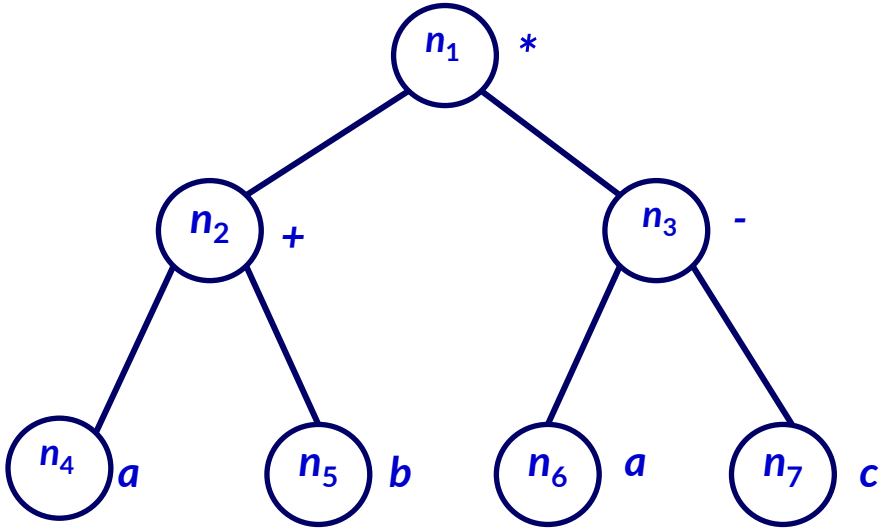


Các thuật ngữ với cây có gốc

- 1.2. Các thuật ngữ

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 n1, ..., n7. Gán nhãn cho các nút:
  - Nút n1 có nhãn *;
  - Nút n2 có nhãn +;
  - Nút n3 có nhãn -;
  - Nút n4 có nhãn a;
  - Nút n5 có nhãn b;
  - Nút n6 có nhãn a;
  - Nút n7 có nhãn c.



Cây biểu thức (a+b)(a-c)*

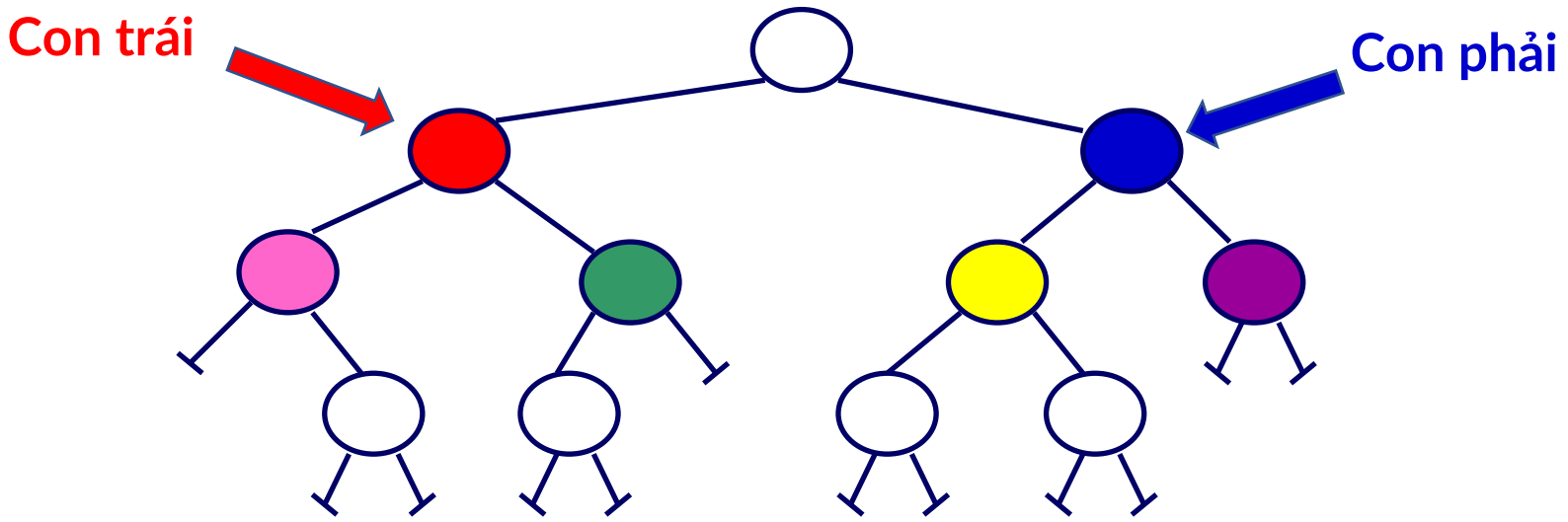
NỘI DUNG TIẾP THEO

1. Tìm kiếm tuần tự

2. Cây nhị phân

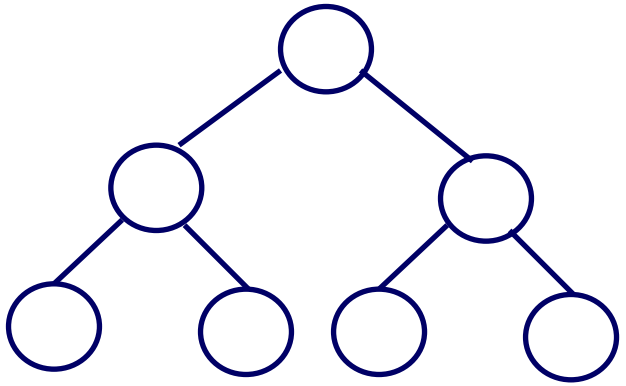
2. CÂY NHỊ PHÂN

- **2.1.** Cây nhị phân
  - **Cây nhị phân (Binary Tree):** Cây mà mỗi nút có nhiều nhất là 2 con
    - ***Con trái*** và ***con phải***
    - Mỗi nút hoặc là:
      - **Không có con**
      - **Chỉ có con trái**
      - **Chỉ có con phải**
      - Con trái và con phải

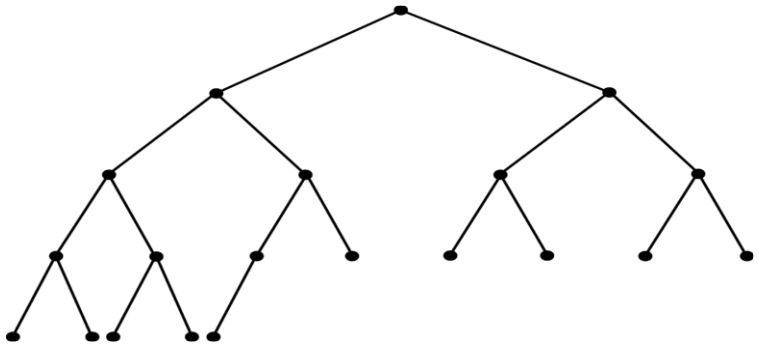


2. CÂY NHỊ PHÂN

- **2.2.** Cây nhị phân đầy đủ
  - **Cây nhị phân đầy đủ (Full Binary Trees):** Cây nhị phân thoả mãn
    - Mỗi nút lá đều có cùng độ sâu
    - Các nút trong có đúng 2 con



- **2.3.** Cây nhị phân hoàn chỉnh
  - **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.
  - Cây nhị phân hoàn chỉnh độ sâu n có số lượng nút nằm trong khoảng từ 2^{n-1} đến $2^n - 1$

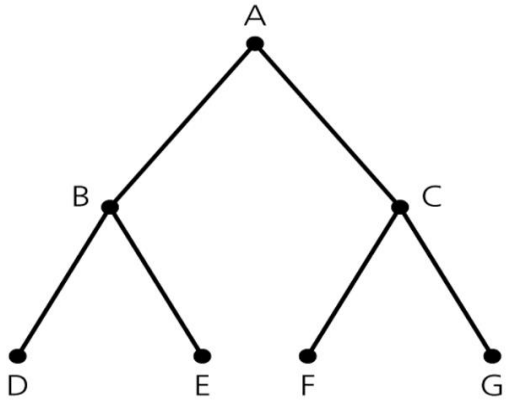


Cây nhị phân hoàn chỉnh

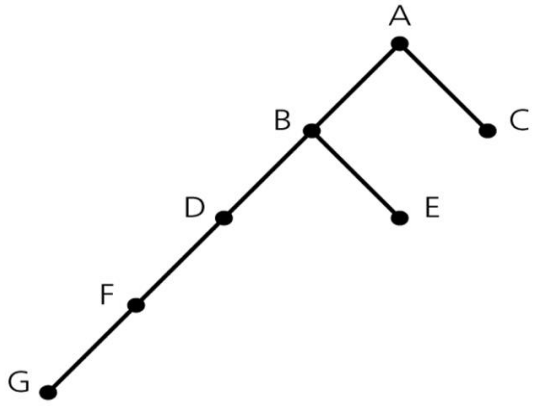
2. CÂY NHỊ PHÂN

- **2.4.** Cây nhị phân cân đối
  - **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

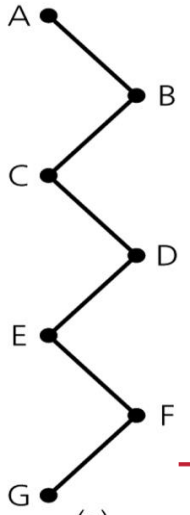
Ví dụ:



(a)



(b)



(c)

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

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Chương 6- Cây

Bài 2. Cấu trúc dữ liệu biểu diễn cây, duyệt cây

ONE LOVE. ONE FUTURE.

MỤC TIÊU

Sau bài học này, người học có thể:

1. Hiểu được khái niệm duyệt cây theo thứ tự trước, sau và giữa

2. Cài đặt được cấu trúc dữ liệu biểu diễn cây

NỘI DUNG TIẾP THEO

1. Cấu trúc dữ liệu biểu diễn cây

1.1. Biểu diễn cây bằng mảng

1.2. Biểu diễn cây bằng danh sách các con

1.3. Biểu diễn cây bằng con trái và em kế cận phải

2. Duyệt cây

1. CẤU TRÚC DỮ LIỆU BIỂU DIỄN CÂY

- 1.1. Biểu diễn cây bằng mảng
  - Giả sử *T* là cây với các nút đặt tên là 1, 2, . . . , *n*.
  - Biểu diễn *T* :
    - danh sách tuyến tính *A* trong đó mỗi phần tử *A*[*i*] chứa con trỏ đến cha của nút *i*.
    - 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.
    - thao tác ***parent*** trả về cha của một nút
  - Cách biểu diễn này dựa trên cơ sở là mỗi nút của cây (ngoại trừ gốc) đều có duy nhất một cha.

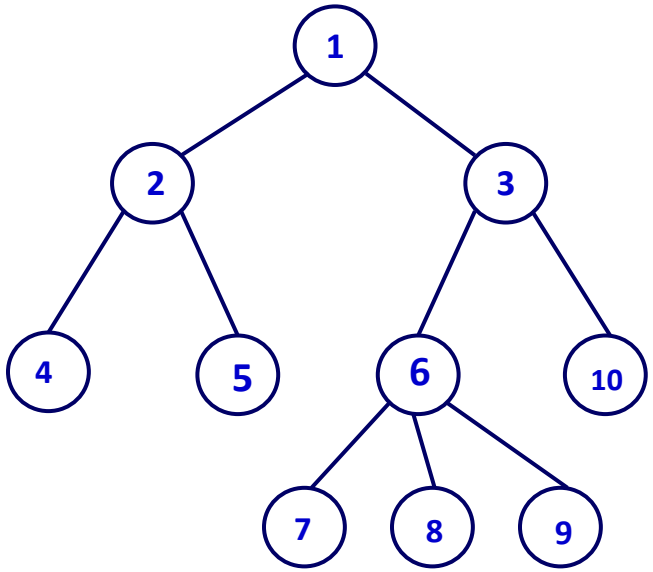
1. CẤU TRÚC DỮ LIỆU BIỂU DIỄN CÂY

- 1.1. Biểu diễn cây bằng mảng
  - Với cách biểu diễn này cha của 1 nút có thể xác định trong thời gian hằng số.
  - Đường đi từ 1 nút đến tổ tiên (kể cả đến gốc) :

    n ← **parent**(n) ← **parent**(**parent**(n)) ← …
  - Có thể dùng thêm 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
    - nhãn.

1. CẤU TRÚC DỮ LIỆU BIỂU DIỄN CÂY

- 1.1. Biểu diễn cây bằng mảng
  - **Ví dụ**



A	0	1	1	2	2	3	6	6	6	3

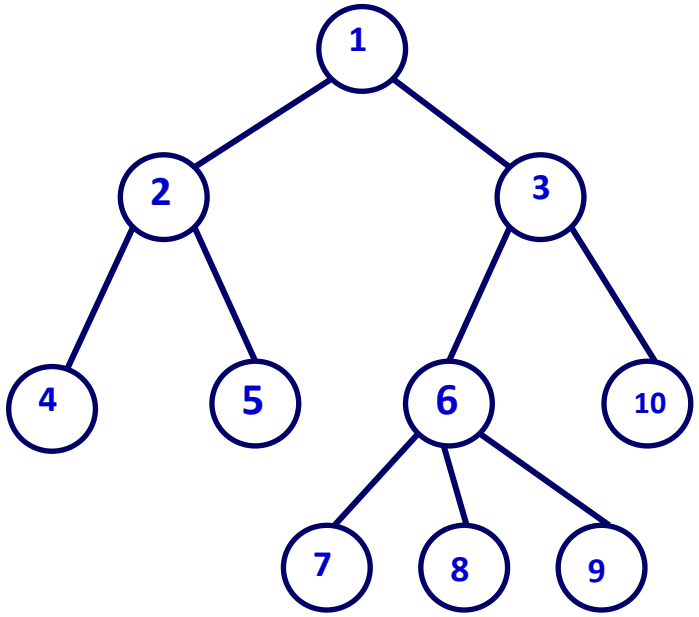
- ***Hạn chế:*** Cách dùng con trỏ cha không thích hợp cho các thao tác với con.
- Cho nút *n*, mất nhiều thời gian để xác định các con của *n*, chiều cao của *n*.
- Không cho thứ tự của các nút con → **leftmost_child** và **right_sibling** là không xác định → cách biểu diễn này chỉ dùng trong một số trường hợp nhất định.

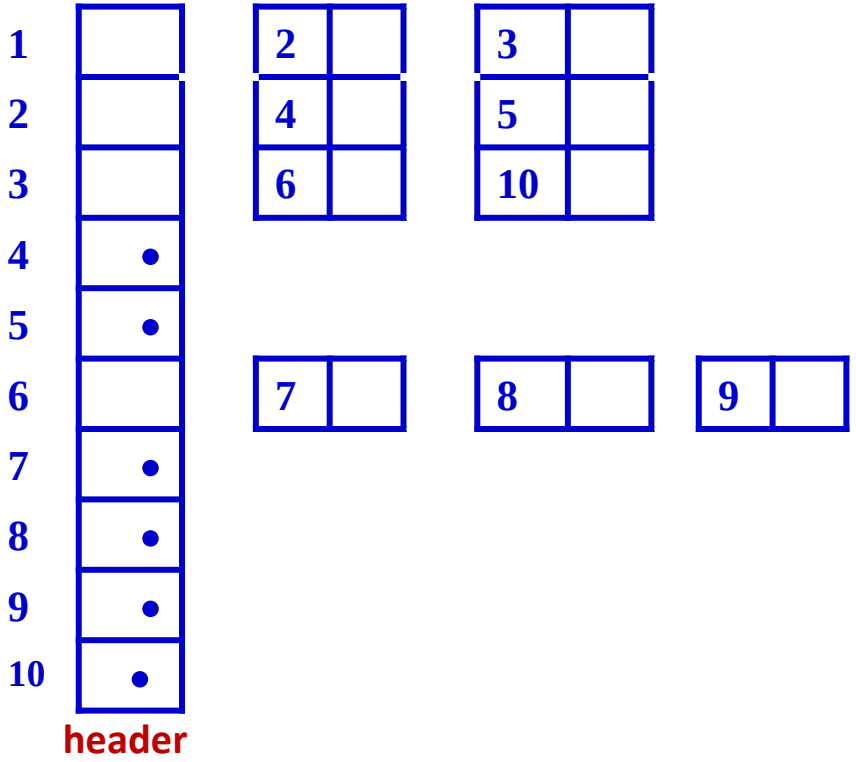
1. CẤU TRÚC DỮ LIỆU BIỂU DIỄN CÂY

- 1.2. Biểu diễn cây bằng danh sách các con
  - 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.
  - Số lượng con của các nút là rất khác nhau → danh sách móc nối thường là lựa chọn thích hợp nhất.

1. CẤU TRÚC DỮ LIỆU BIỂU DIỄN CÂY

- 1.2. Biểu diễn cây bằng danh sách các con





- 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*.

1. CẤU TRÚC DỮ LIỆU BIỂU DIỄN CÂY

- 1.2. Biểu diễn cây bằng danh sách các con
  - **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;
```

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

1. CẤU TRÚC DỮ LIỆU BIỂU DIỄN CÂY

- 1.2. Biểu diễn cây bằng danh sách các con
  - 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(retrive ( first(L), L));
}
```

1. CẤU TRÚC DỮ LIỆU BIỂU DIỄN CÂY

- 1.3. Biểu diễn cây bằng con trái và em kế cận phải

Nhận xét:

- Mỗi một nút của cây hoặc là
  - không có con
  - có đúng 1 nút con cực trái
  - không có em kế cận phải,
  - có đúng 1 nút em kế cận phải

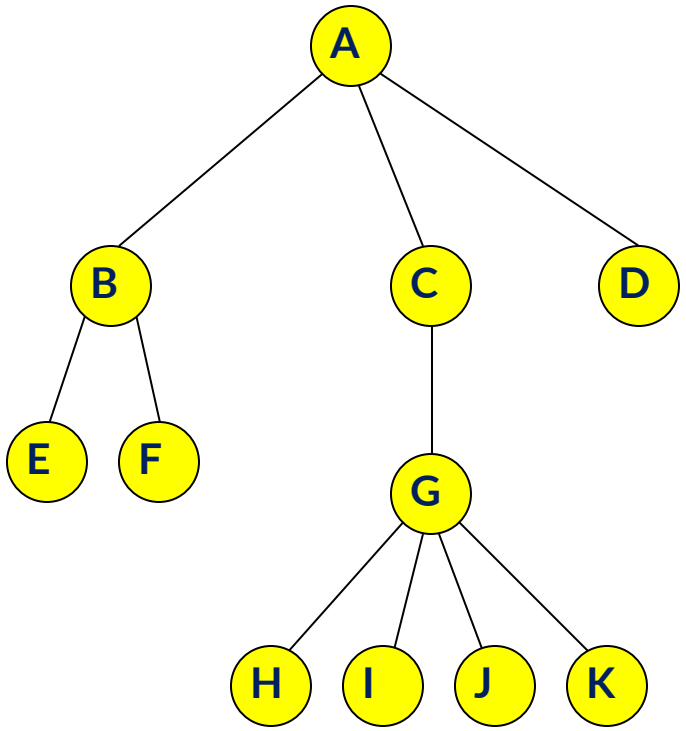
→ Để biểu diễn cây: lưu trữ thông tin về con cực trái và em kế cận phải của mỗi nút.

Sử dụng mô tả sau:

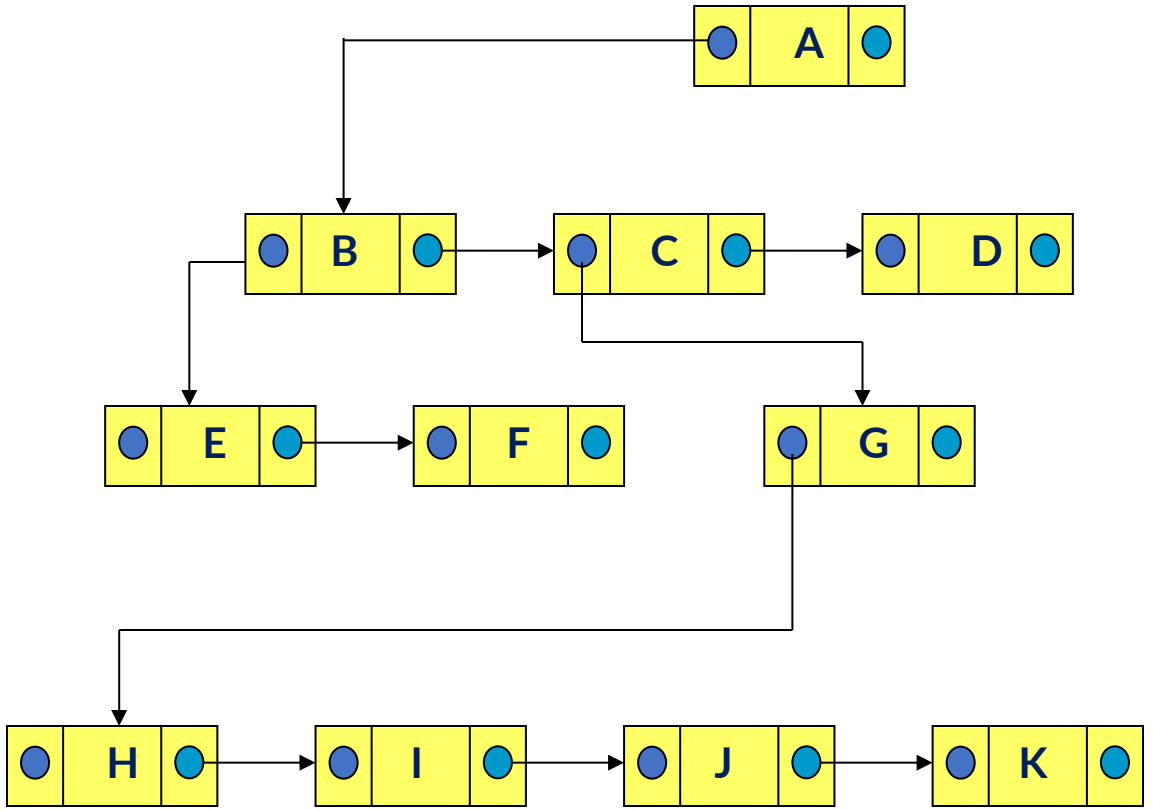
```
struct Tnode
{
   char word[20]; // Dữ liệu cất giữ ở nút
   struct Tnode *leftmost_child;
   struct Tnode *right_sibling;
};
typedef struct Tnode treeNode;
treeNode Root;
```

1. CẤU TRÚC DỮ LIỆU BIỂU DIỄN CÂY

- 1.3. Biểu diễn cây bằng con trái và em kế cận phải



Cây tổng quát



Biểu diễn cây

1. CẤU TRÚC DỮ LIỆU BIỂU DIỄN CÂY

- 1.3. Biểu diễn cây bằng con trái và em kế cận phải

Nhận xét:

- Với cách biểu diễn này, các thao tác cơ bản dễ dàng cài đặt.
- 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, bổ sung thêm một trường nữa vào bản ghi để lưu cha của nút.

NỘI DUNG TIẾP THEO

1. Cấu trúc dữ liệu biểu diễn cây

2. Duyệt cây

2.1. Duyệt theo thứ tự trước

2.2. Duyệt theo thứ tự sau

2.3. Duyệt theo thứ tự giữa

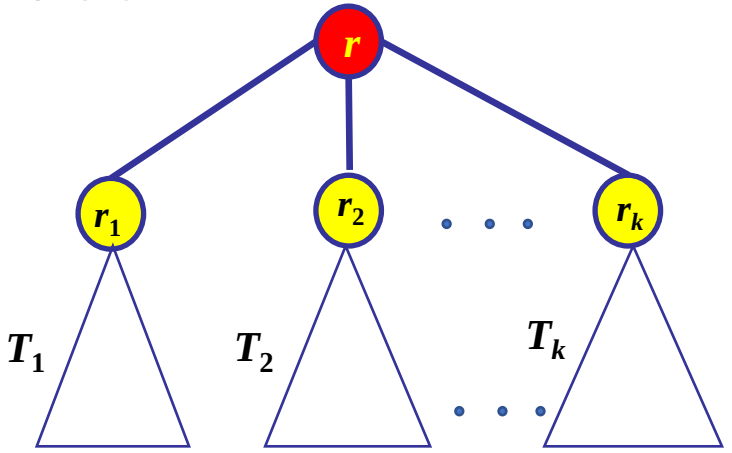
3. Cây nhị phân

2. DUYỆT CÂY

- 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,..., T_k.

2. DUYỆT CÂY

- **2.1.** Duyệt theo thứ tự trước



  - ***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.

2. DUYỆT CÂY

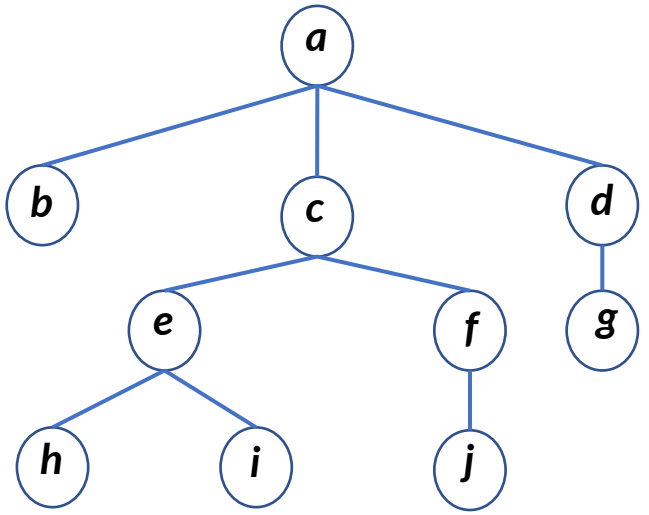
- **2.1.** Duyệt theo thứ tự trước

  Thuật toán:

```
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);
}
```

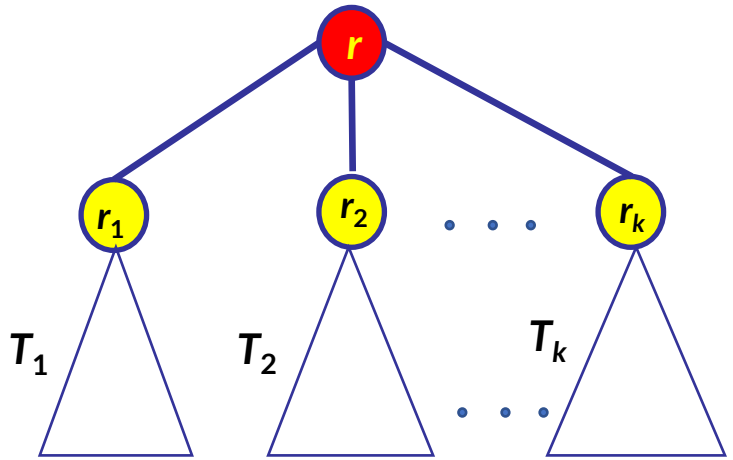
Ví dụ: Thứ tự trước của các đỉnh của cây trên hình vẽ là

a, b, c, e, h, i, f, j, d, g



2. DUYỆT CÂY

- **2.2.** Duyệt theo thứ tự sau



  - **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.

2. DUYỆT CÂY

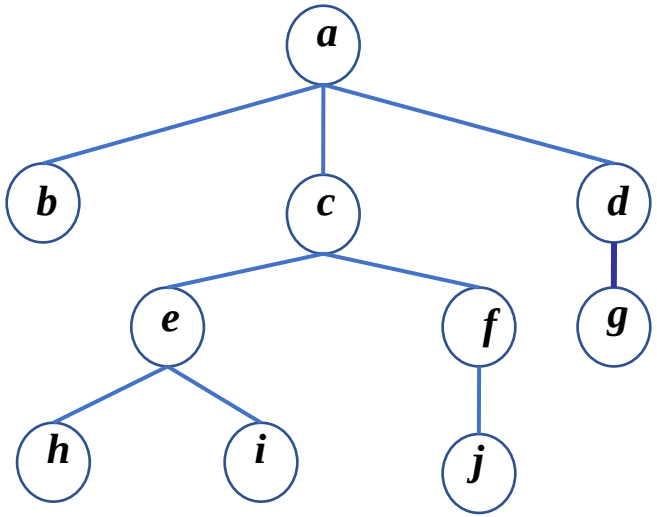
- **2.2.** Duyệt theo thứ tự sau

  Thuật toán:

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

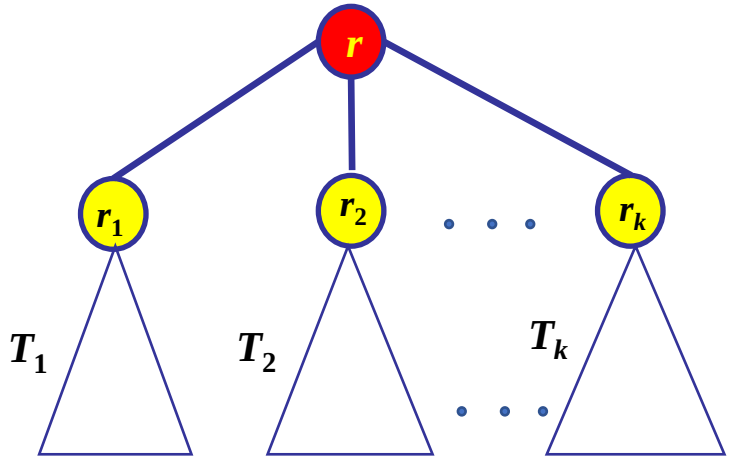
  - **Ví dụ:** Dãy các đỉnh được liệt kê theo ***thứ tự sau*** của cây trong hình vẽ là:

    b, h, i, e, j, f, c, g, d, a



2. DUYỆT CÂY

- **2.3.** Duyệt theo thứ tự giữa

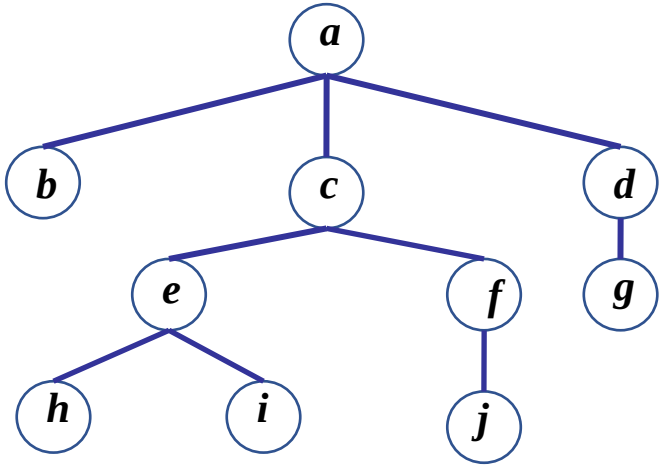


  - ***Thứ tự giữa*** của các nút của cây T là:
    - Các nút của T_1 theo thứ tự giữa,
    - Tiếp đến là nút gốc r,
    - Tiếp theo là các nút của $T_2, \ldots, T_k$, mỗi nhóm nút được xếp theo thứ tự giữa.

2. DUYỆT CÂY

- **2.3.** Duyệt theo thứ tự giữa

```
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);
   }
}
```



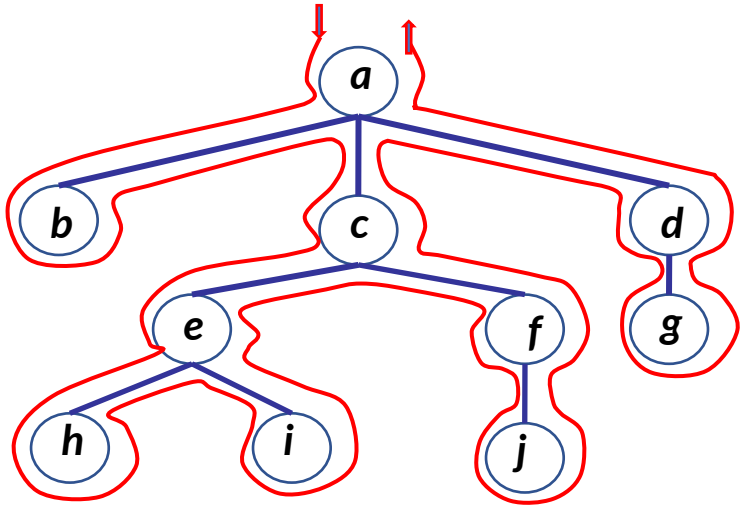
- **Ví dụ:** Dãy các đỉnh của cây trong hình vẽ được liệt kê theo thứ tự giữa:

  b, a, h, e, i, c, j, f, g, d

2. DUYỆT CÂY

- Xếp thứ tự các nút

Đi vòng quanh bên ngoài cây bắt đầu từ gốc, ngược chiều kim đồng hồ và sát theo cây nhất



- 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.

NỘI DUNG TIẾP THEO

1.Cấu trúc dữ liệu biểu diễn cây

2. Duyệt cây

3. Cây nhị phân

3.1. Biểu diễn cây nhị phân

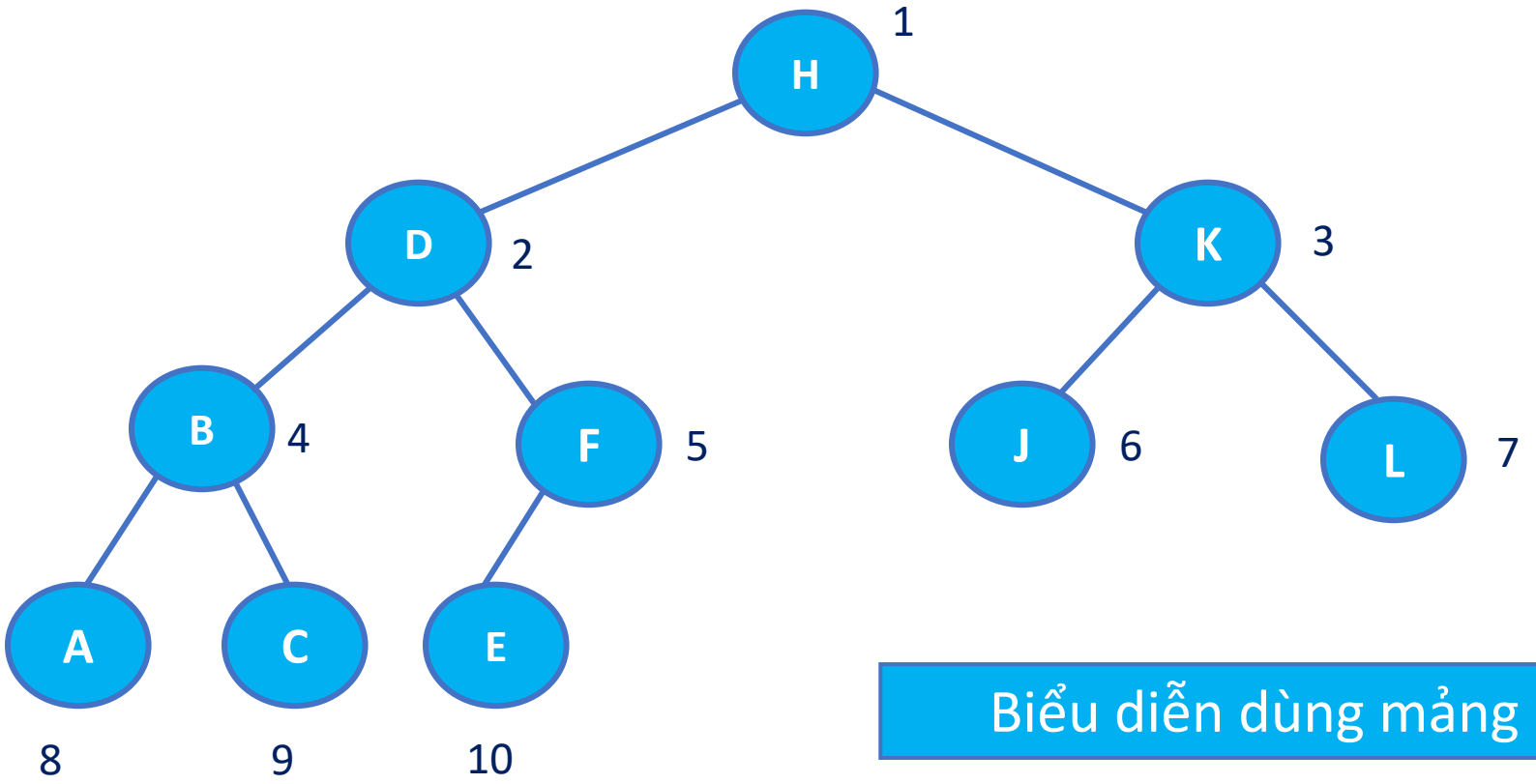
3.2. Duyệt cây nhị phân

3. CÂY NHỊ PHÂN

- **3.1.** 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,..., *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, ..., *n*.

3. CÂY NHỊ PHÂN

- **3.1.** Biểu diễn cây nhị phân
  - **Biểu diễn sử dụng mảng**

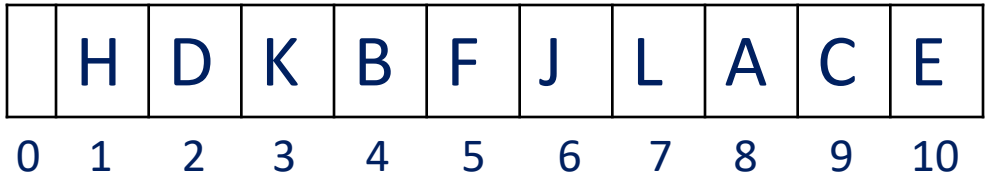


Biểu diễn dùng mảng

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

3. CÂY NHỊ PHÂN

- **3.1.** Biểu diễn cây nhị phân
  - **Biểu diễn sử dụng mảng**

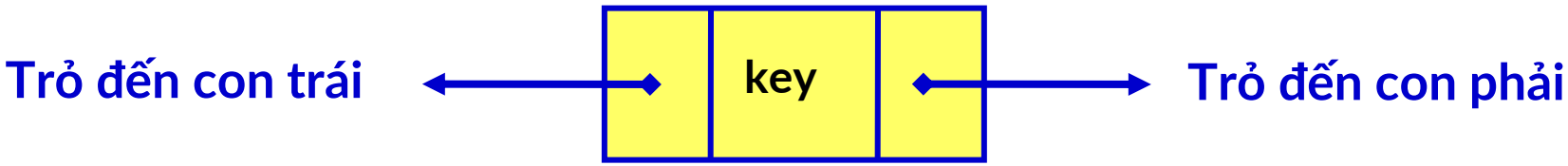


Để tìm	Sử dụng	Hạn chế
Con trái của A[i]	A[2*i]	2*i <= n
Con phải của A[i]	A[2*i + 1]	2*i + 1 <= n
Cha của A[i]	A[i/2]	i > 1
Gốc	A[1]	A khác rỗng
Thử A[i] là lá?	True	2*i > n

3. CÂY NHỊ PHÂN

- **3.1.** Biểu diễn cây nhị phân
  - **Biểu diễn sử dụng con trỏ**

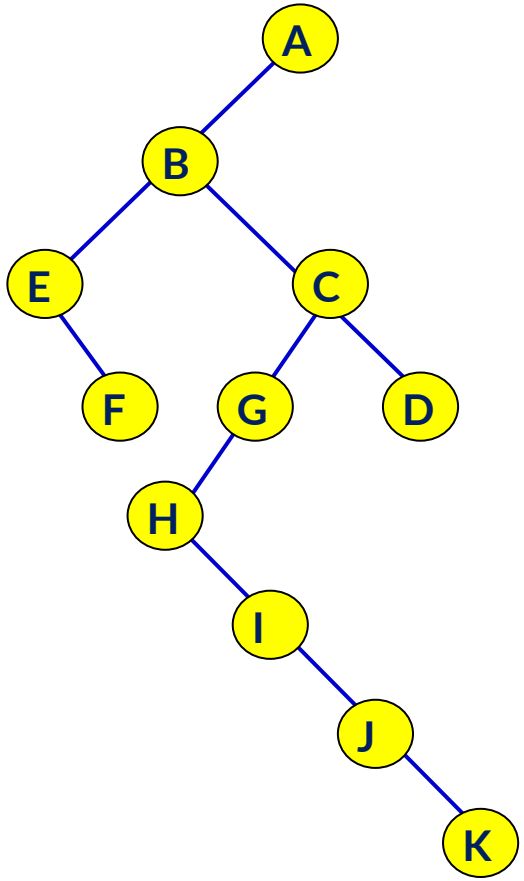
Mỗi nút của cây sẽ có con trỏ đến con trái và con trỏ đến con phải:



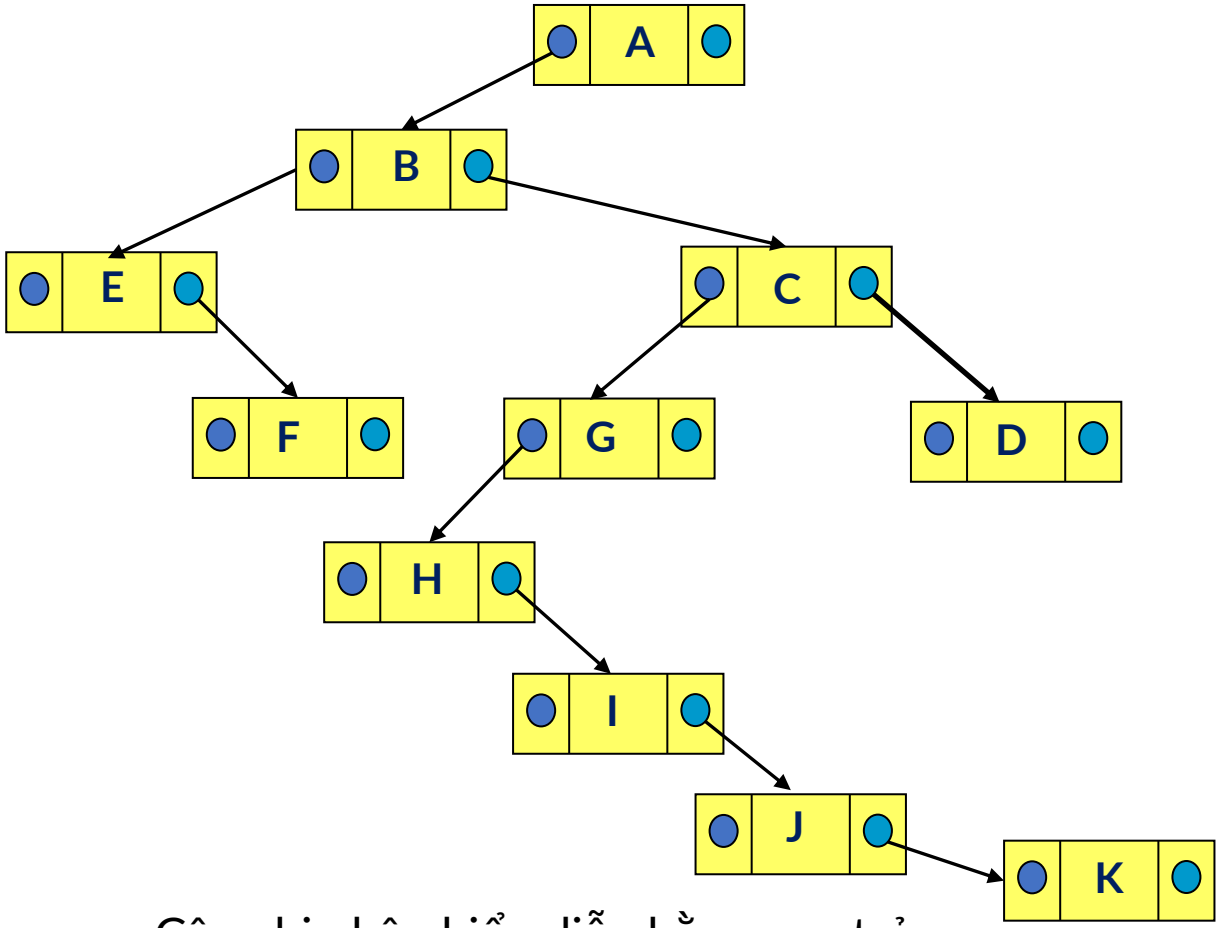

```
struct Tnode {
        DataType Item; // DataType - kiểu dữ liệu của phần tử

        struct Tnode *left;
        struct Tnode *right;
};
typedef struct Tnode treeNode;
```

3. CÂY NHỊ PHÂN

- **3.1.** Biểu diễn cây nhị phân
  - **Biểu diễn sử dụng con trỏ**



Cây nhị phân



Cây nhị phân biểu diễn bằng con trỏ

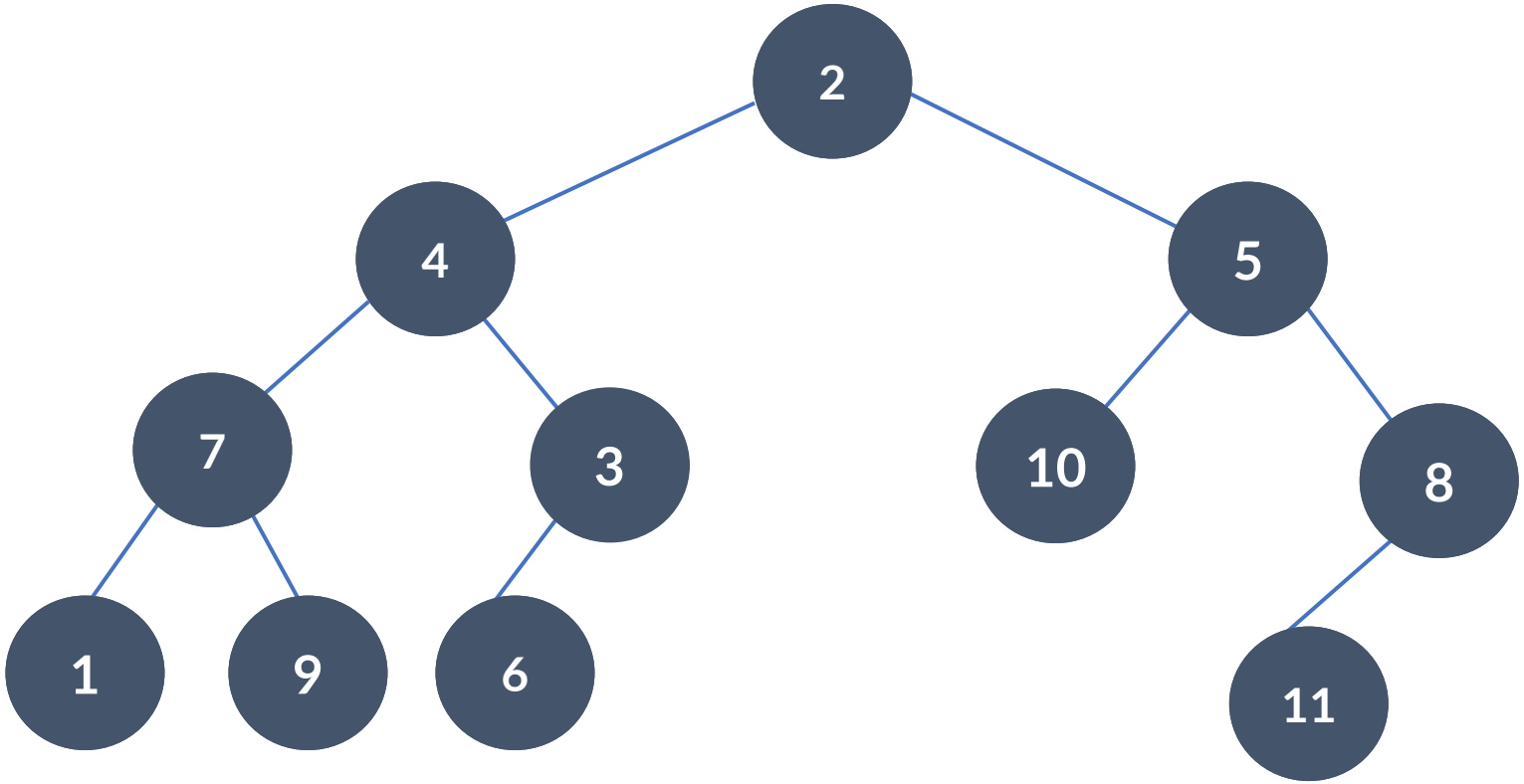
- **3.2.** Duyệt cây nhị phân
  - **Duyệt theo thứ tự trước**

- Thăm nút
- Duyệt cây con trái
- Duyệt cây con phải

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

3. CÂY NHỊ PHÂN

- **3.2.** Duyệt cây nhị phân
  - **Duyệt theo thứ tự trước**



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

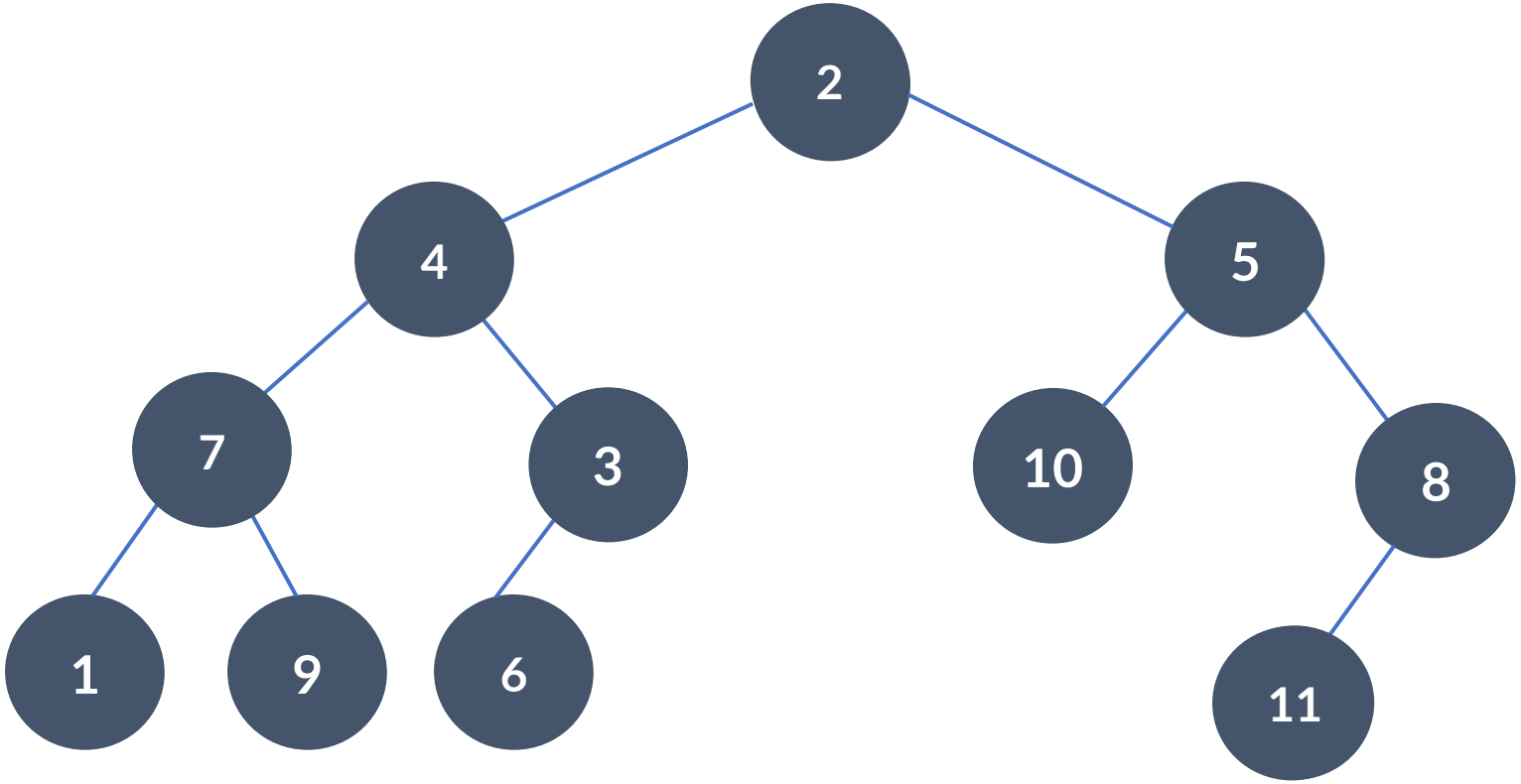
- **3.2.** Duyệt cây nhị phân
  - **Duyệt theo thứ tự giữa**

- Duyệt cây con trái
- Thăm nút
- Duyệt cây con phải

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

3. CÂY NHỊ PHÂN

- **3.2.** Duyệt cây nhị phân
  - **Duyệt theo thứ tự giữa**



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

- **3.2.** Duyệt cây nhị phân

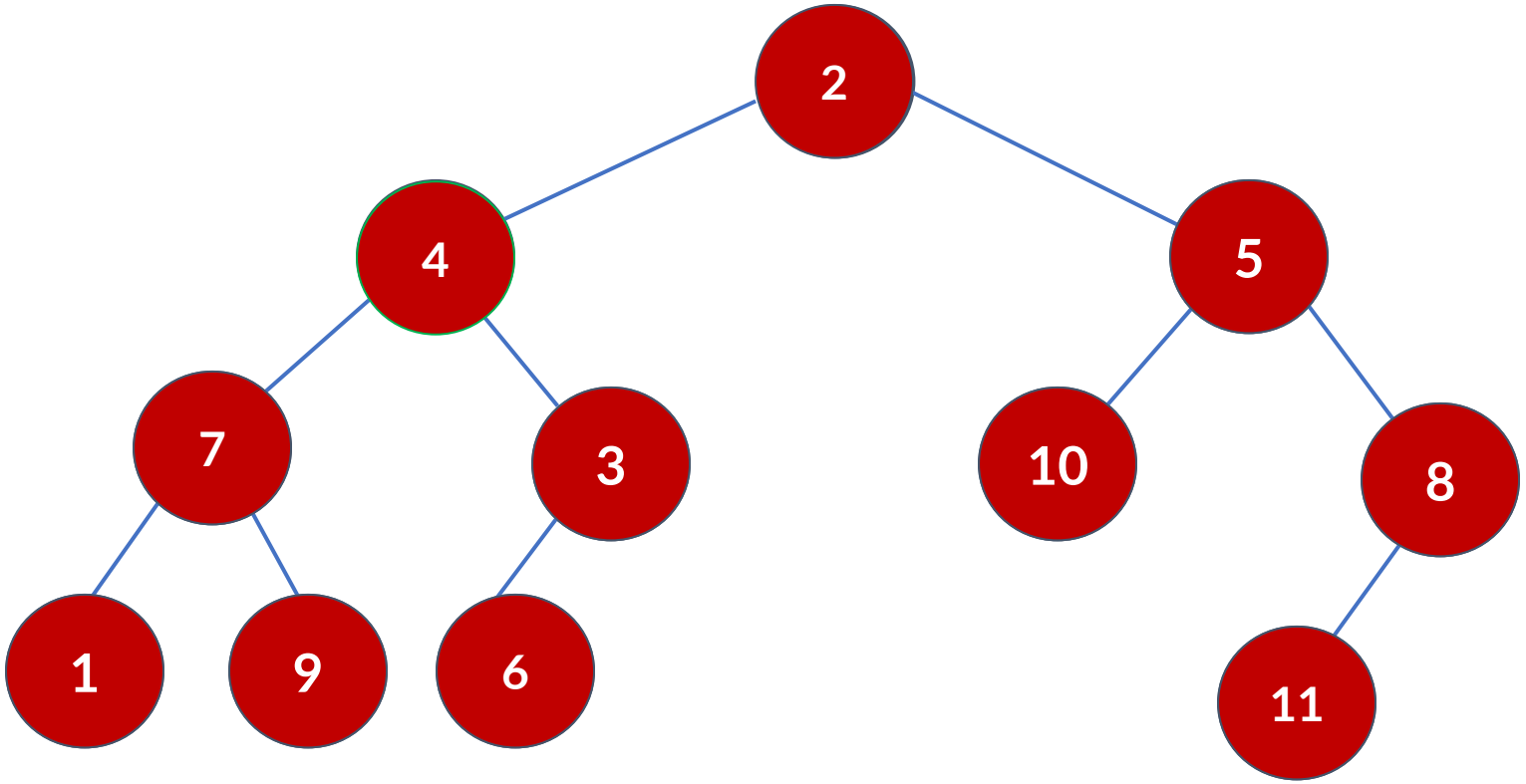
3.2.3. Duyệt theo thứ tự sau

- Duyệt cây con trái
- Thăm nút
- Duyệt cây con phải

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

3. CÂY NHỊ PHÂN

- **3.2.** Duyệt cây nhị phân
  - **Duyệt theo thứ tự sau**



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

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Chương 6- Cây

Bài 3. Tính chiều cao, độ sâu của một nút trên cây

ONE LOVE. ONE FUTURE.

MỤC TIÊU

Sau bài học này, người học có thể:

1. Hiểu được khái niệm chiều cao và độ sâu của một nút trong cấu trúc dữ liệu cây

2. Cài đặt được thuật toán tìm chiều cao và độ sâu của nút trên cây

NỘI DUNG TIẾP THEO

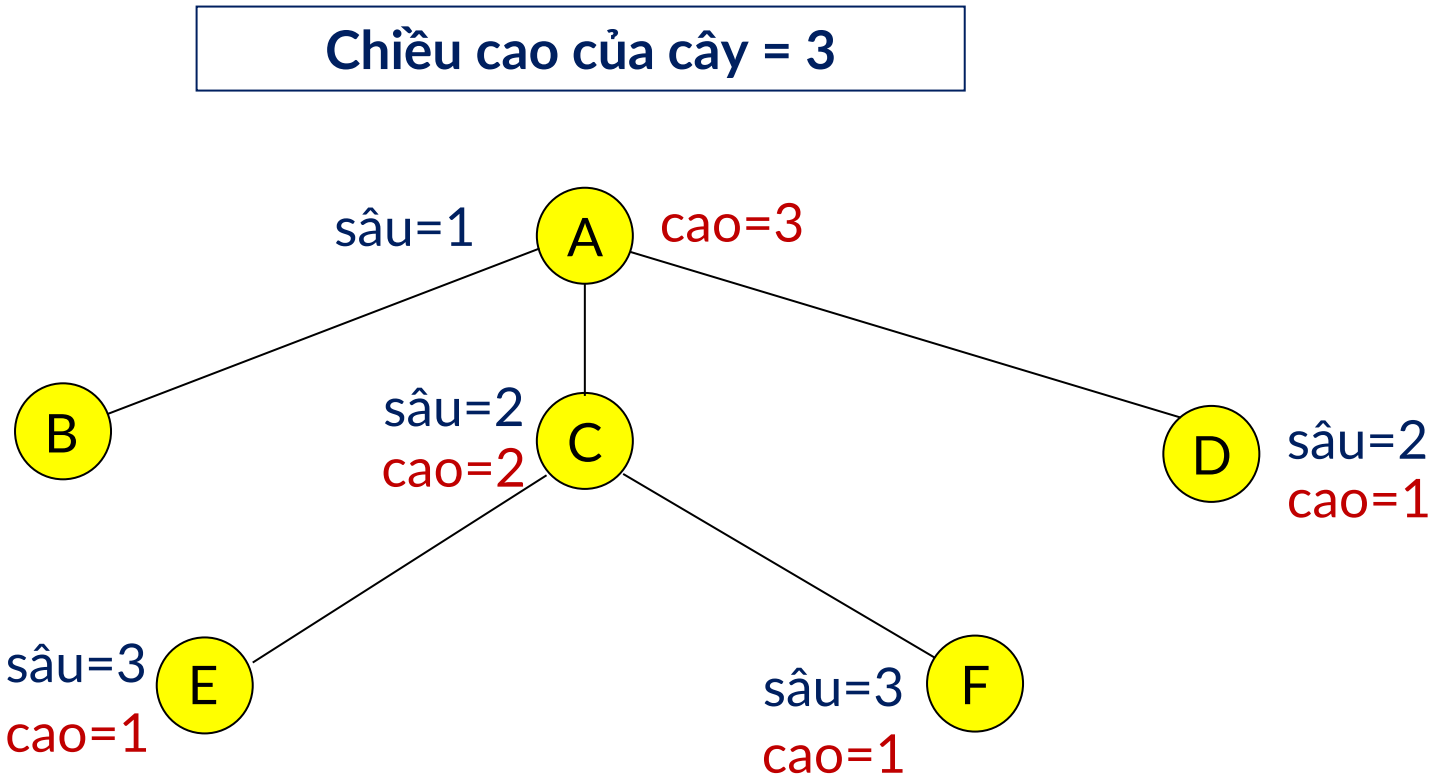
1. **Định nghĩa chiều cao và độ sâu của nút**
   1.1. Định nghĩa
   1.2. Ví dụ
2. Thuật toán tìm chiều cao và độ sâu

1. ĐỊNH NGHĨA CHIỀU CAO VÀ ĐỘ SÂU CỦA NÚT

- 1.1. Định nghĩa
  - **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.

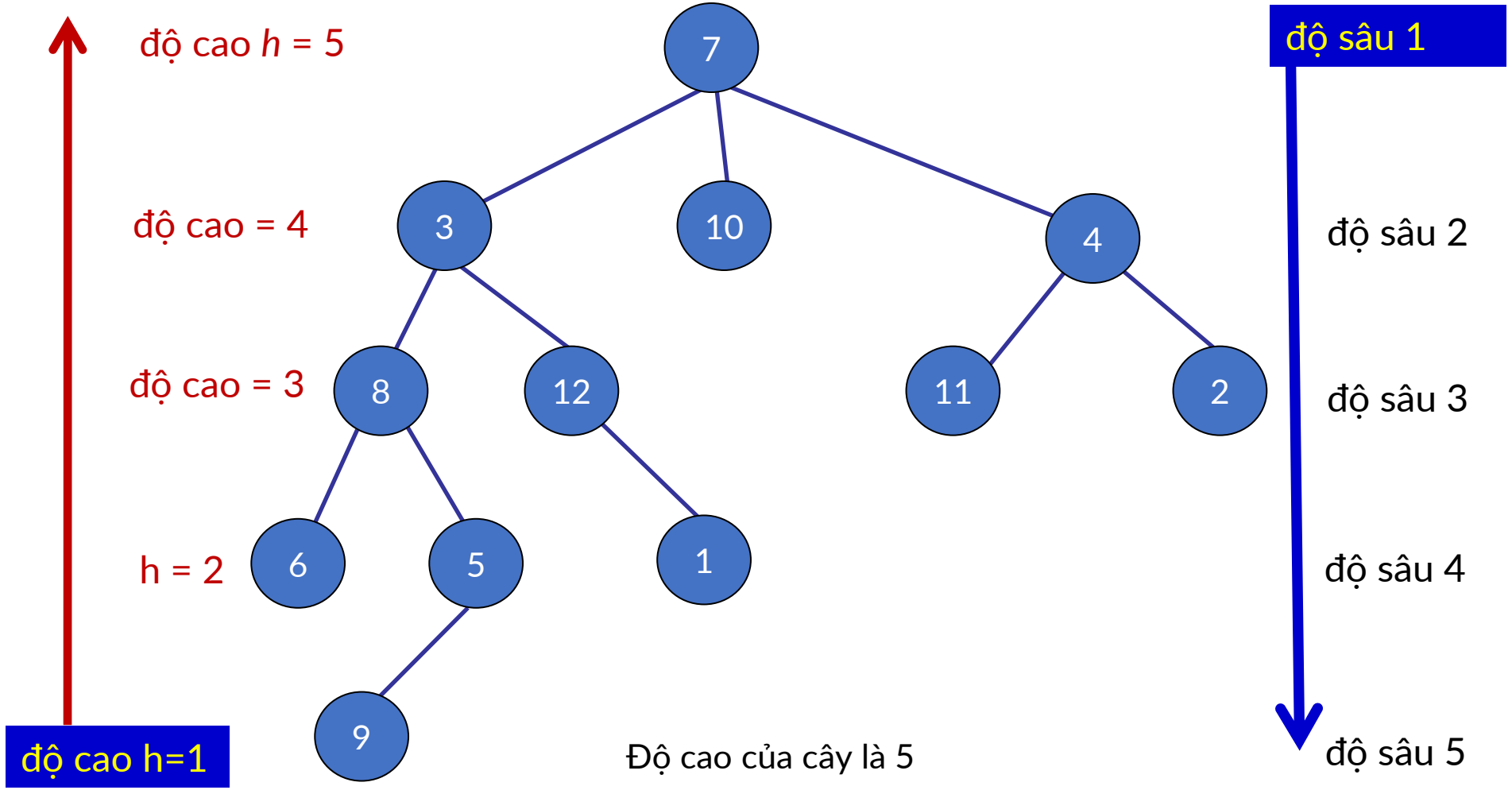
1. ĐỊNH NGHĨA CHIỀU CAO VÀ ĐỘ SÂU CỦA NÚT

- 1.2. Ví dụ



1. ĐỊNH NGHĨA CHIỀU CAO VÀ ĐỘ SÂU CỦA NÚT

- 1.2. Ví dụ



NỘI DUNG TIẾP THEO

1. Định nghĩa chiều cao và độ sâu của nút

2. Thuật toán tìm chiều cao và độ sâu

2.1 Tìm độ cao của một nút

2.2 Tìm độ sâu của một nút

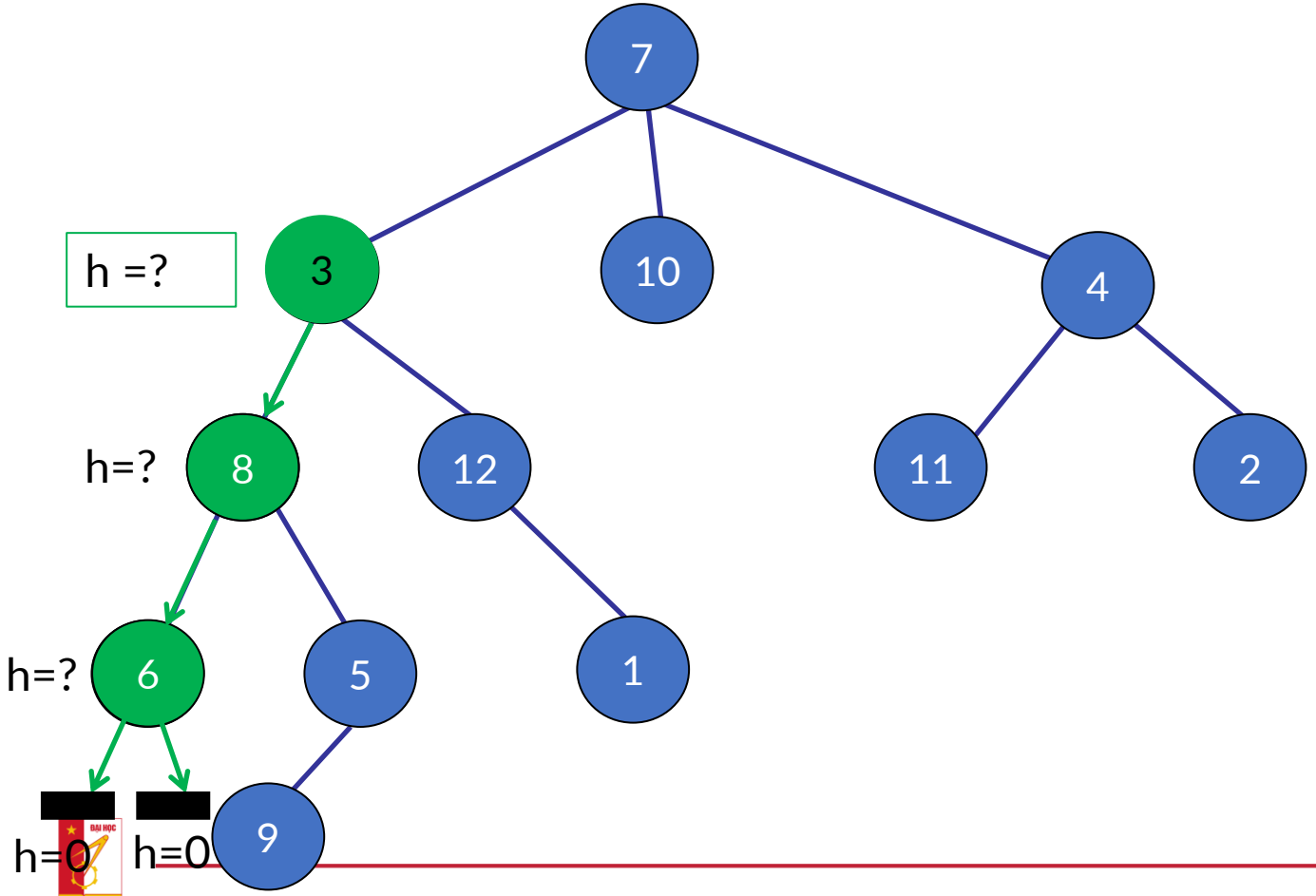
2. THUẬT TOÁN TÌM CHIỀU CAO VÀ ĐỘ SÂU

- Cấu trúc dữ liệu biểu diễn cây

```
struct Node{
    int id;
    Node* leftMostChild;
    Node* rightSibling;
};
Node* root;
```

2. THUẬT TOÁN TÌM CHIỀU CAO VÀ ĐỘ SÂU

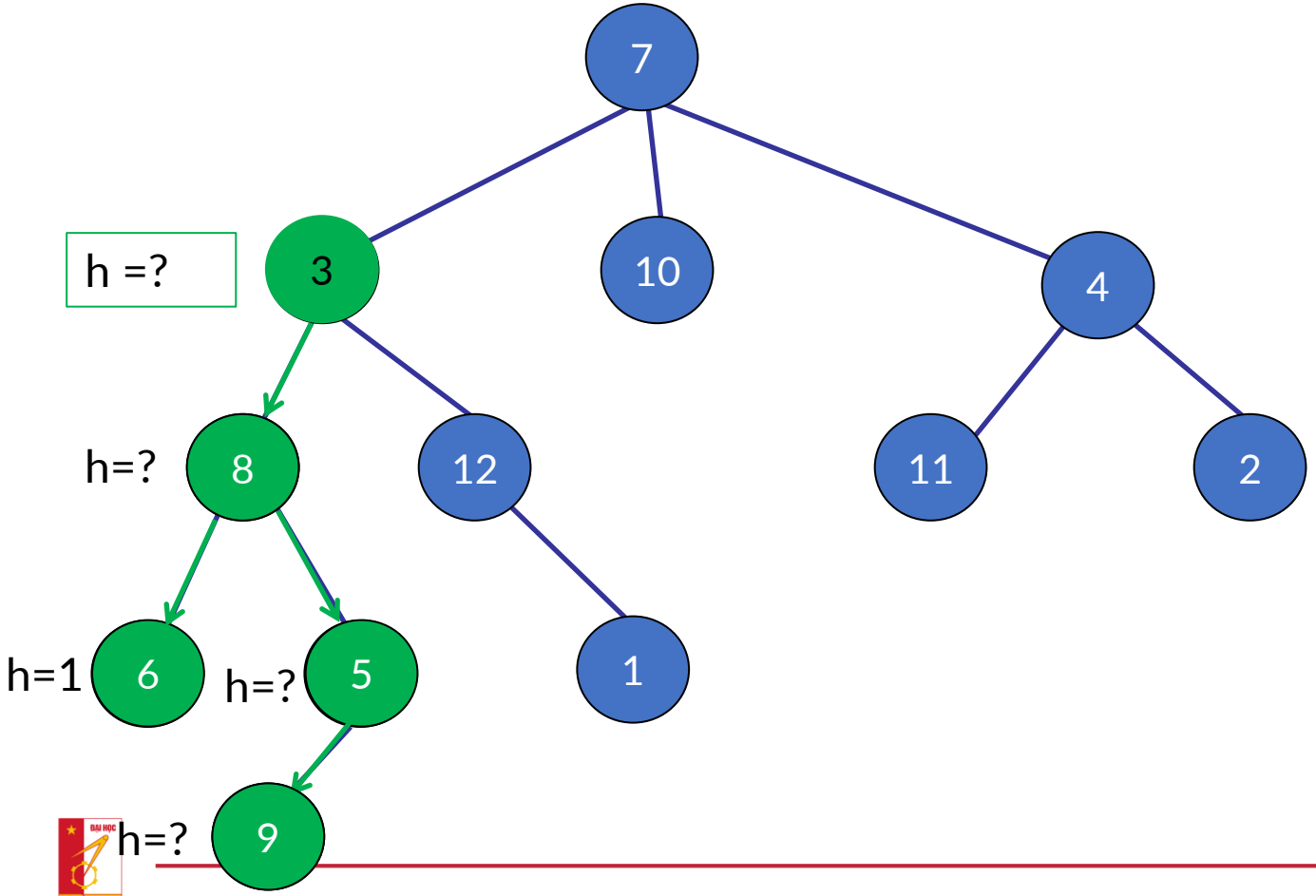
- 2.1. Tìm độ cao của một nút
  - Độ cao của một nút




```
int height(Node* p){
   if(p == NULL) return 0;
   int maxh = 0;
   Node* q = p->leftMostChild;
   while(q != NULL){
      int h = height(q);
      if(h > maxh) maxh = h;
      q = q->rightSibling;
   }
   return maxh + 1;
}
```

2. THUẬT TOÁN TÌM CHIỀU CAO VÀ ĐỘ SÂU

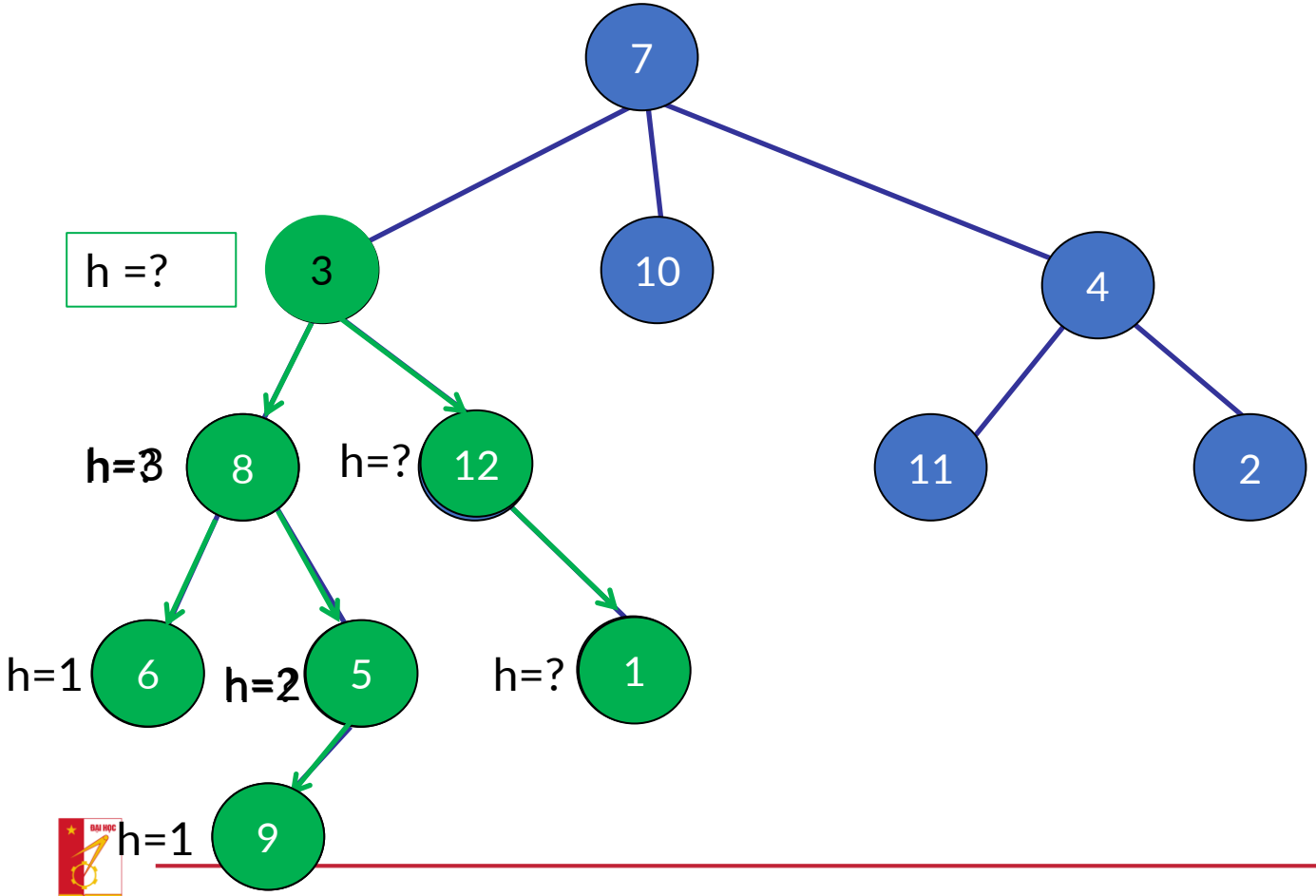
- 2.1. Tìm độ cao của một nút
  - Độ cao của một nút




```
int height(Node* p){
   if(p == NULL) return 0;
   int maxh = 0;
   Node* q = p->leftMostChild;
   while(q != NULL){
      int h = height(q);
      if(h > maxh) maxh = h;
      q = q->rightSibling;
   }
   return maxh + 1;
}
```

2. THUẬT TOÁN TÌM CHIỀU CAO VÀ ĐỘ SÂU

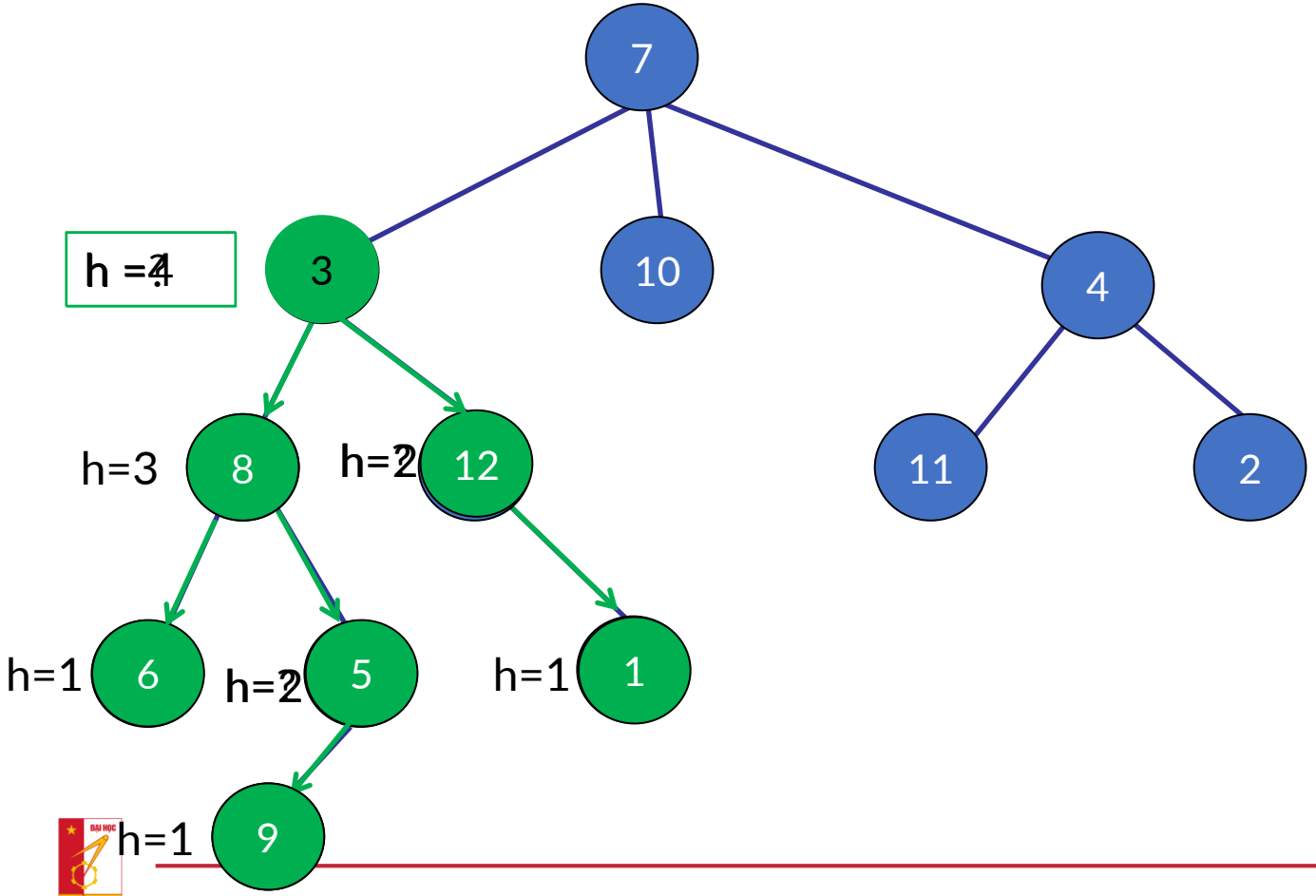
- 2.1. Tìm độ cao của một nút
  - Độ cao của một nút




```
int height(Node* p){
   if(p == NULL) return 0;
   int maxh = 0;
   Node* q = p->leftMostChild;
   while(q != NULL){
      int h = height(q);
      if(h > maxh) maxh = h;
      q = q->rightSibling;
   }
   return maxh + 1;
}
```

2. THUẬT TOÁN TÌM CHIỀU CAO VÀ ĐỘ SÂU

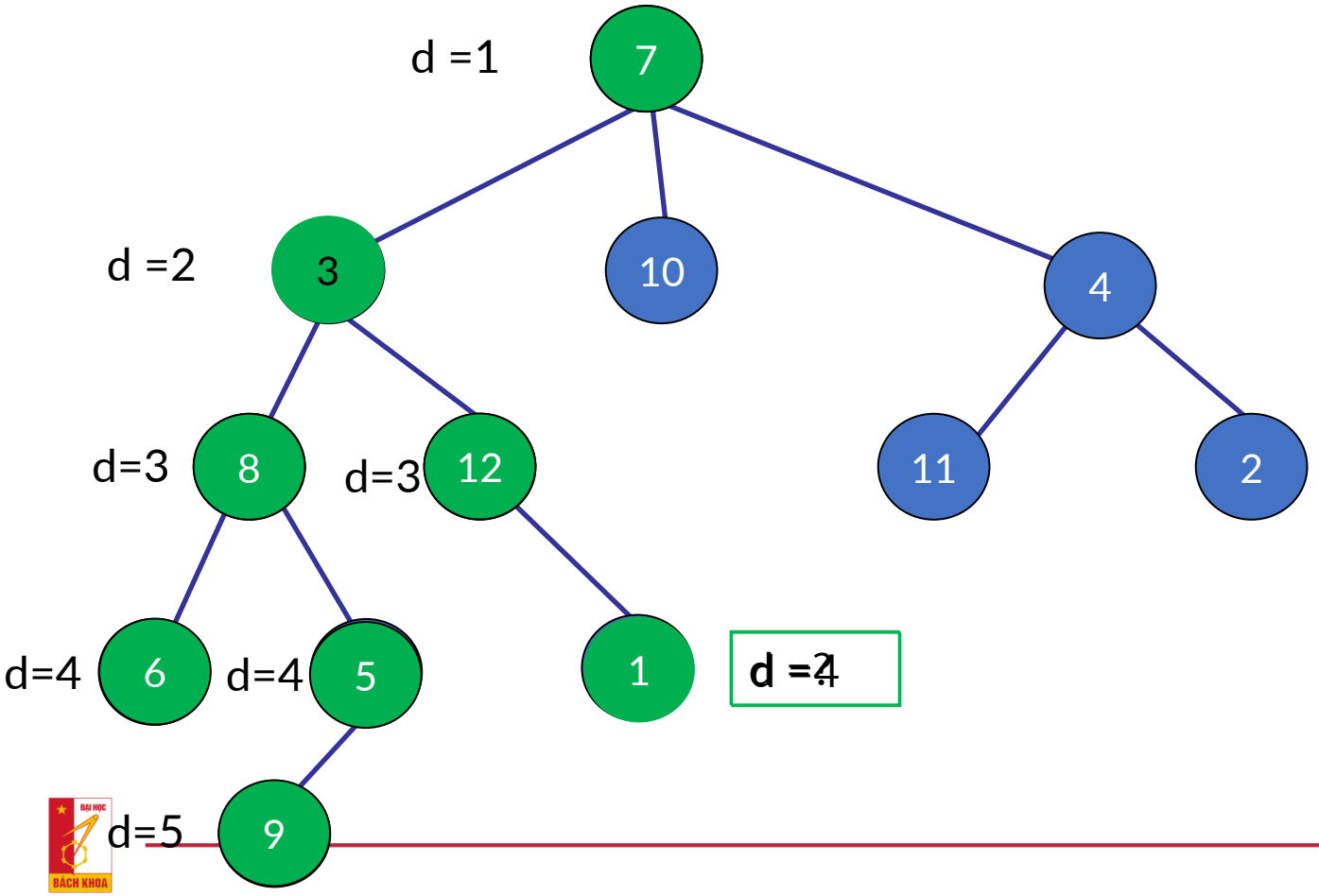
- 2.1. Tìm độ cao của một nút
  - Độ cao của một nút




```
int height(Node* p){
   if(p == NULL) return 0;
   int maxh = 0;
   Node* q = p->leftMostChild;
   while(q != NULL){
      int h = height(q);
      if(h > maxh) maxh = h;
      q = q->rightSibling;
   }
   return maxh + 1;
}
```

2. THUẬT TOÁN TÌM CHIỀU CAO VÀ ĐỘ SÂU

- 2.2. Tìm độ sâu của một nút
  - Độ sâu của một nút




```
int depth(Node* r, int v, int d){
   // d la do sau cua nut r
   if(r == NULL) return -1;
   if(r->id == v) return d;
   Node* p = r->leftMostChild;
   while(p != NULL){
      if(p->id == v) return d+1;
      int dv = depth(p,v,d+1);
      if(dv > 0) return dv;
      p = p->rightSibling;
   }
   return -1;
}
int find_depth(Node* r, int v){
   return depth(r,v,1);
}
```

HUST

hust.edu.vn fb.com/dhbkhn

THANK YOU !