Data structure and Algorithms
Tree – cấu trúc cây

Thanh-Hai Tran

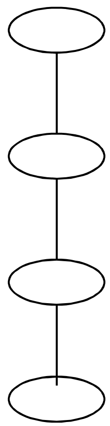
Electronics and Computer Engineering
School of Electronics and Telecommunications
Hanoi University of Science and Technology
1 Dai Co Viet - Hanoi - Vietnam

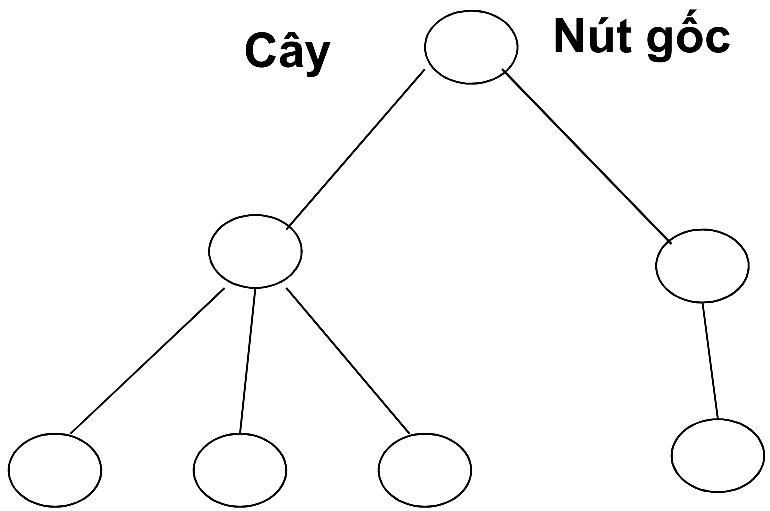
Nội dung của bài học

- **Các khái niệm**
- **Cây tổng quát**
  - Tính chất
  - Biểu diễn cây tổng quát
  - Duyệt cây tổng quát
- **Cây nhị phân**
  - Định nghĩa và tính chất
  - Biểu diễn cây nhị phân
  - Duyệt cây nhị phân
  - Cây nhị phân tìm kiếm
- **Một số ví dụ ứng dụng cây**

Định nghĩa cây

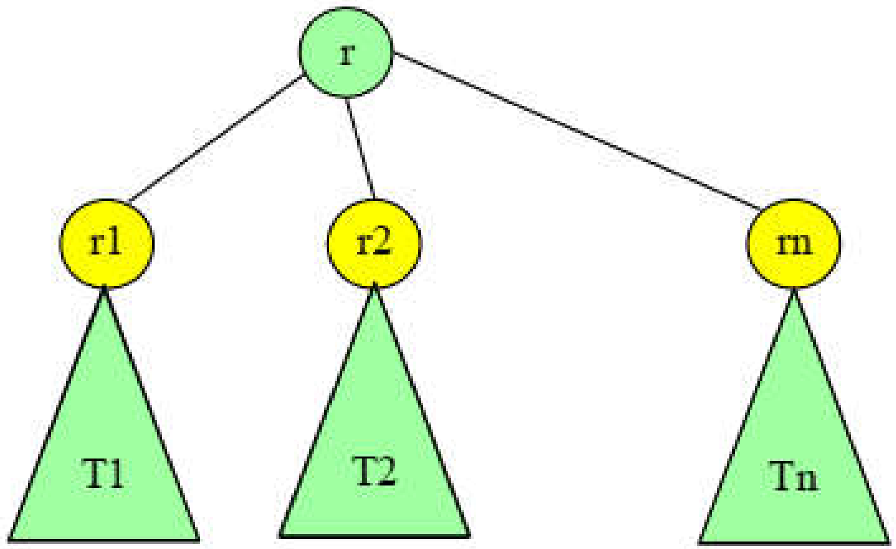
- **Cây là một cấu trúc phi tuyến**
- **Thiết lập trên một tập hữu hạn các "nút"**
  - Tồn tại một nút đặc biệt gọi là "gốc" (root)
  - Tồn tại một quan hệ phân cấp hay gọi là quan hệ cha con giữa các nút
  - Một nút (trừ nút gốc) chỉ có một cha
  - Một nút có thể có từ 0 đến n con





Định nghĩa cây

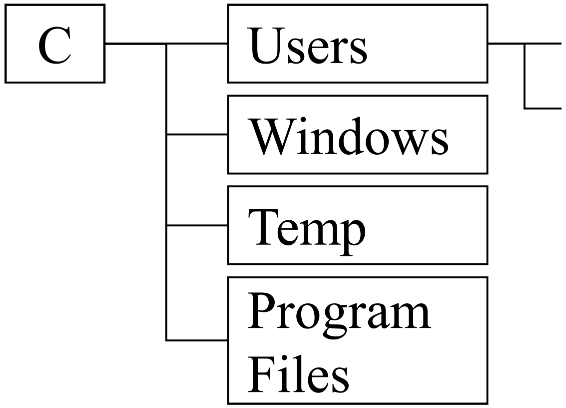
- Cây có thể định nghĩa một cách đệ quy
  - Một nút tạo thành cây (nút gốc)
  - Khi có n cây: T_1, T_2, …, T_n,
    - mỗi cây này có các nút gốc tương ứng là r_1, r_2, …, r_n
    - r là quan hệ cha con với r_1, r_2, …, r_n
    - Tồn tại cây mới T nhận r là nút gốc

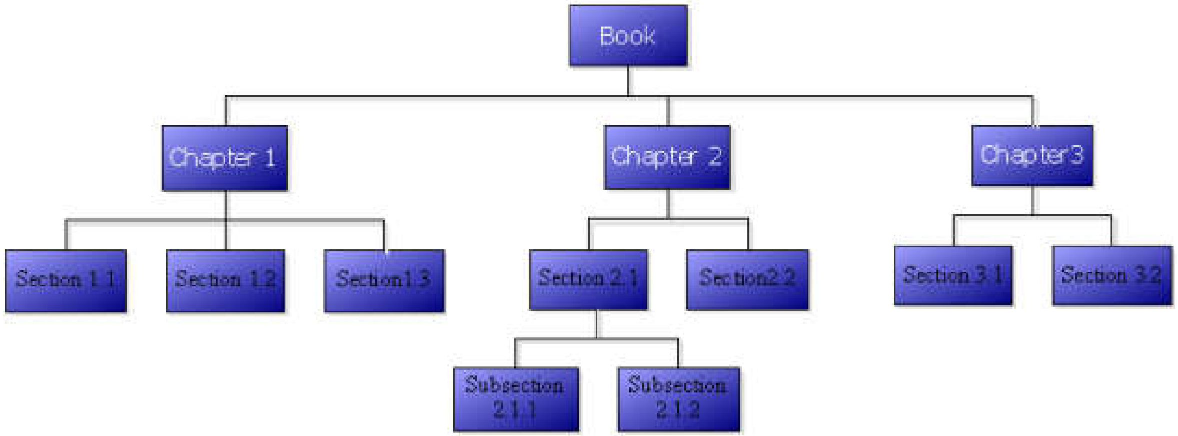


SET
2020

Ví dụ về cây

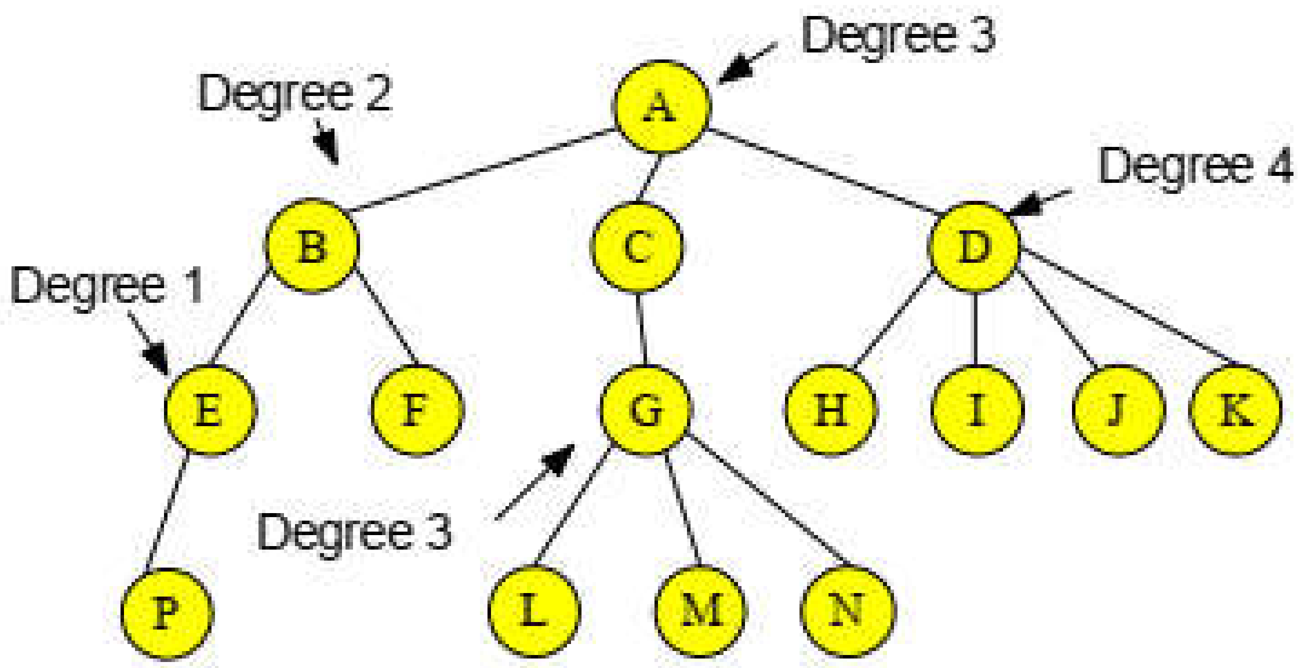
- Cấu trúc lưu trữ thư mục trong máy tính
- Cấu trúc mục lục của sách / tài liệu
- Cấu trúc các chức năng của một hệ thống thông tin





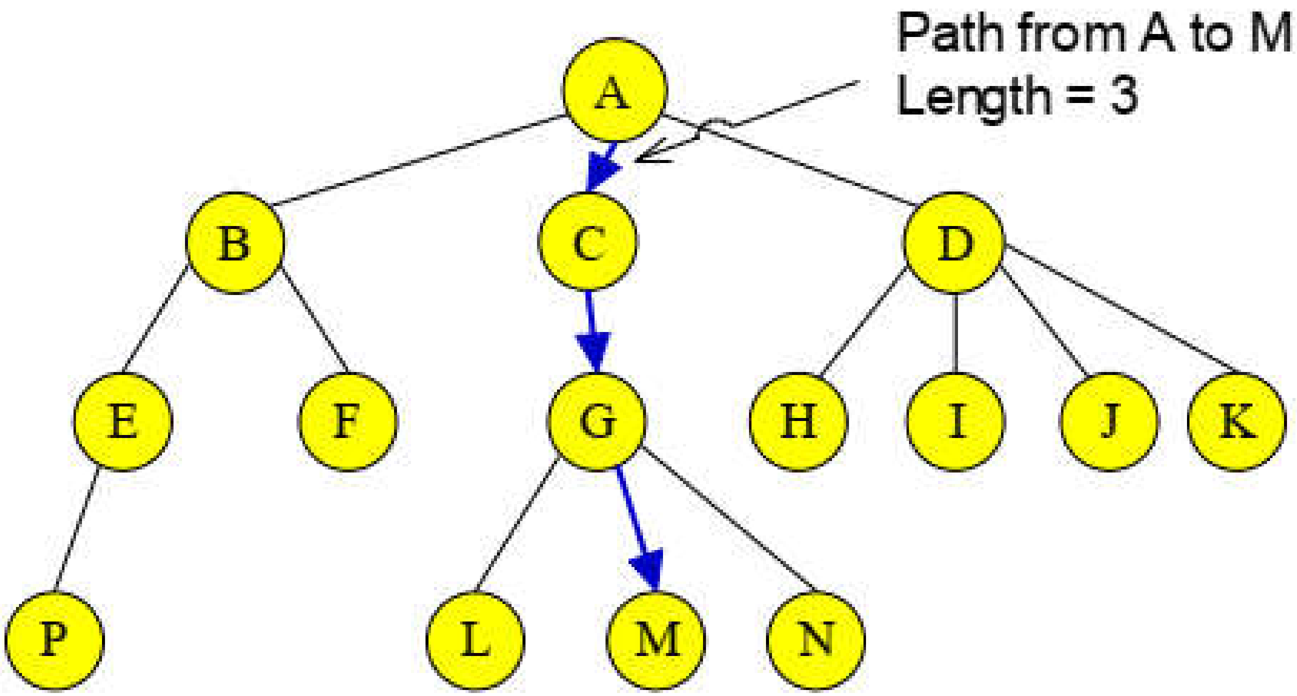
Các thuật ngữ

- Cấp (degree) của một nút: số các nút con của nút đó
- Cấp của một cây: cấp cao nhất của một nút trên cây



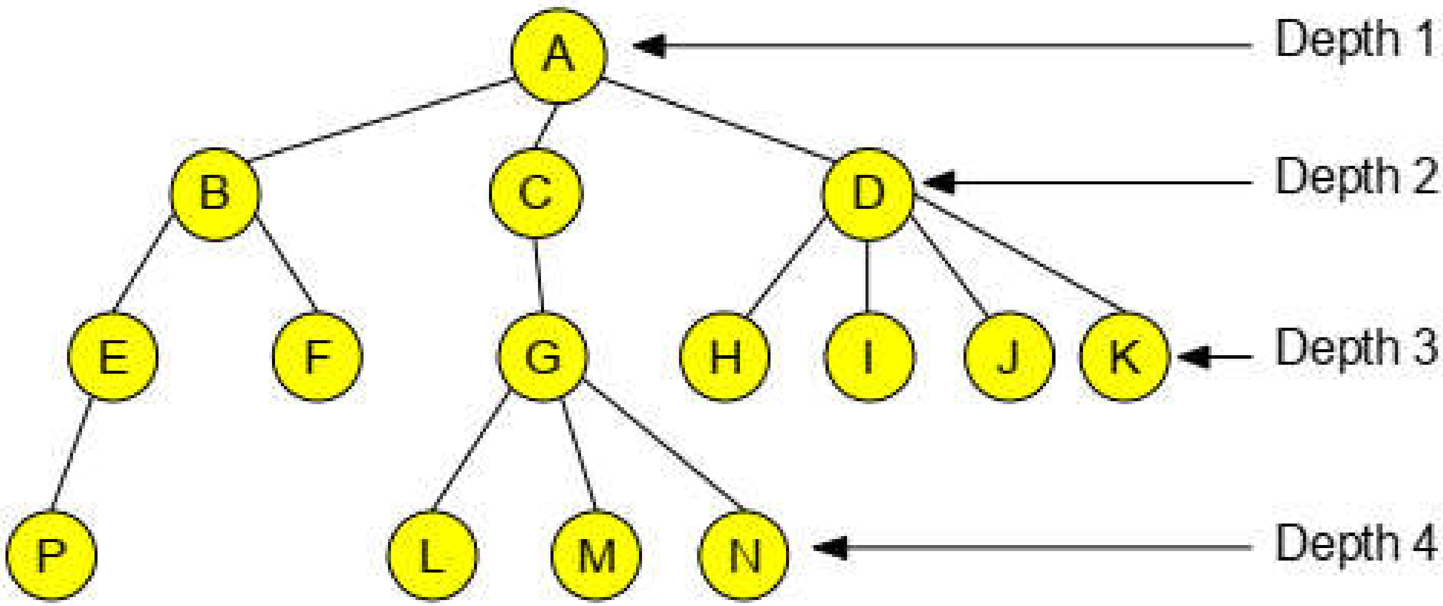
Các thuật ngữ

- **Đường đi trên một cây:**
  - dãy các nút $n_1, n_2, \ldots, n_k$
  - n_i là nút cha của n_{i+1} $(i = 1..k-1)$



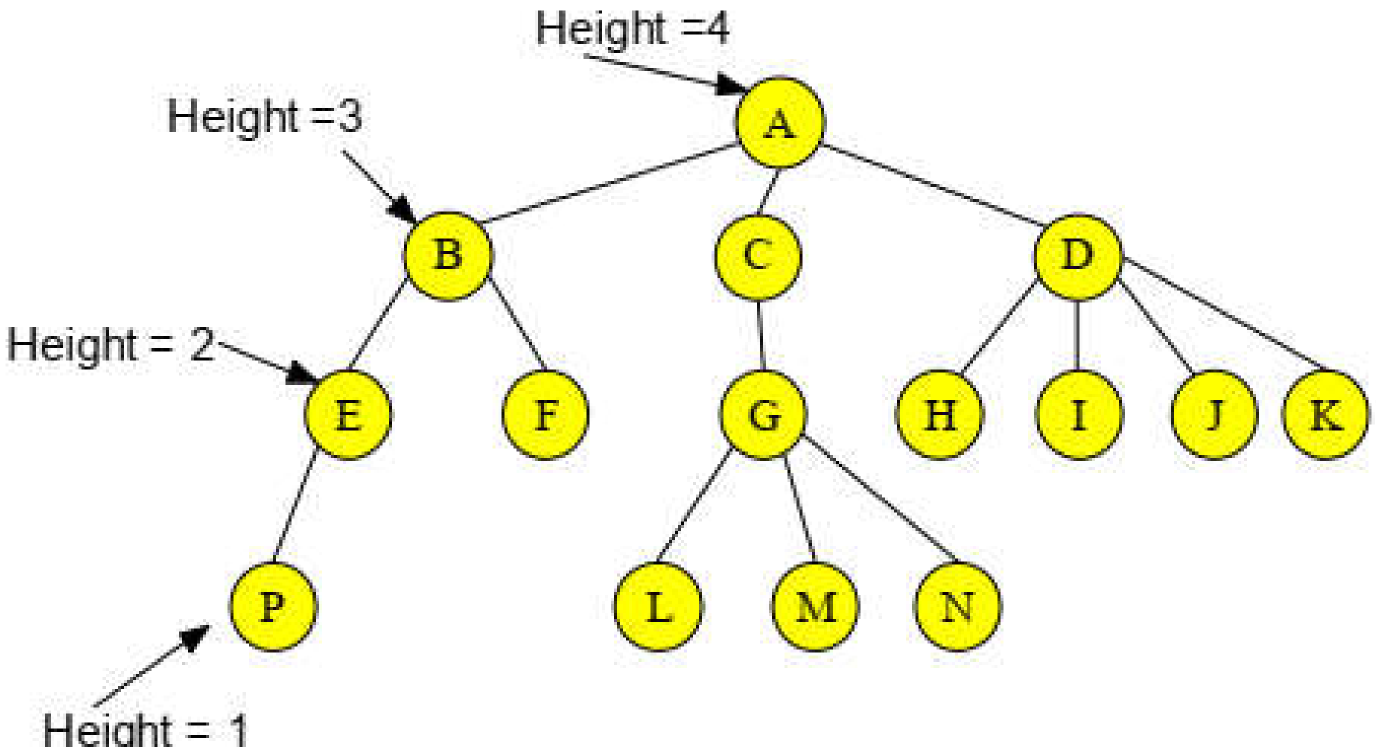
Các thuật ngữ

- **Độ sâu (depth – level) của một nút:**
  - Là độ dài đường đi từ nút gốc đến nút đó + 1
  - Ví dụ nút gốc r, nút xem xét là r_i: $d(r_i)$ = length (r, r_i) +1



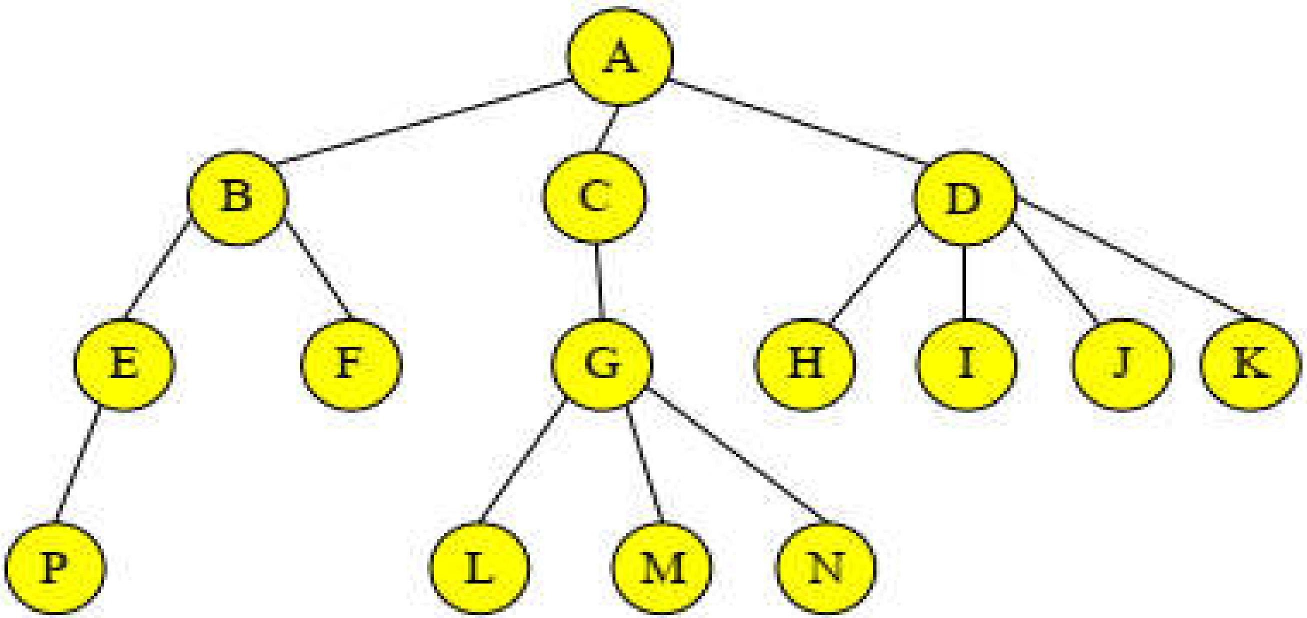
Các thuật ngữ

- **Độ cao (height) của một nút**
  - Độ dài đường đi dài nhất của nút đó đến một nút lá trong cây + 1
  - Chiều cao của cây là chiều cao của nút gốc



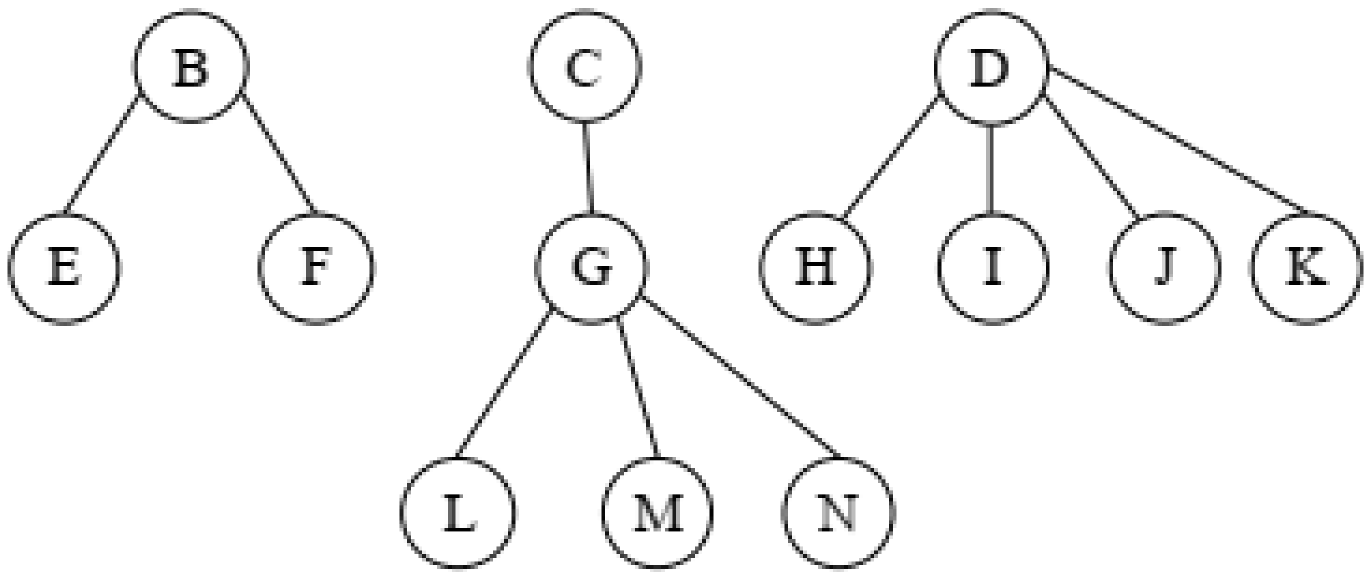
Các thuật ngữ

- **Tổ tiên (ancestor):** A,C, G là tổ tiên của M
- **Hậu duệ (descendants):** E, F, G, H, L,M …đều là hậu duệ của A
- **Anh em (siblings):** E, F là một cặp anh em ; L, N là một cặp anh em



Các thuật ngữ

- **Rừng (Forest):** một tập hợp hữu hạn các cây phân biệt , không giao nhau



Các tính chất của cây

- **Kích thước của cây:** là tổng số nhánh + 1
- **Cây có tính chất đệ quy:** một cây được tạo bởi nhiều cây con
- **Cấu trúc dữ liệu động:** Một cây có kích thước biến đổi
- **Cấu trúc cây là cấu trúc phi tuyến:** phân cấp
- **Chỉ tồn tại duy nhất một đường đi từ nút gốc đến nút khác**

Các thao tác cơ bản trên cây

- **Thao tác khởi tạo cây**: Chuẩn bị cấu trúc để lưu trữ cây
- **Bổ sung một nút mới vào cây:**
  - Xác định vị trí cần chèn
  - Xác định quan hệ nút mới và nút tại vị trí cần bổ sung
- **Lấy ra một nút:**
  - Xác định vị trí
  - Cấu trúc lại cây

Các thao tác cơ bản trên cây

- Các thao tác truy nhập cây
  - root() : trả ra nút gốc của cây
  - parent(Tree T, Node p): trả ra nút cha của nút p trong cây T
  - children(Tree T, Node p): trả ra danh sách các nút con của nút p trong cây T
  - left_most_child(Tree T, Node p) : trả ra nút con cực trái của nút p
  - right_most_child(Tree T, Node p) : trả ra nút con cực phải của nút p
  - left_sibling (Tree T, Node p) : trả ra nút anh em kề cận bên trái của nút p
  - right_sibling(Tree T, Node p) : trả ra nút anh em kề cận bên phải của nút p

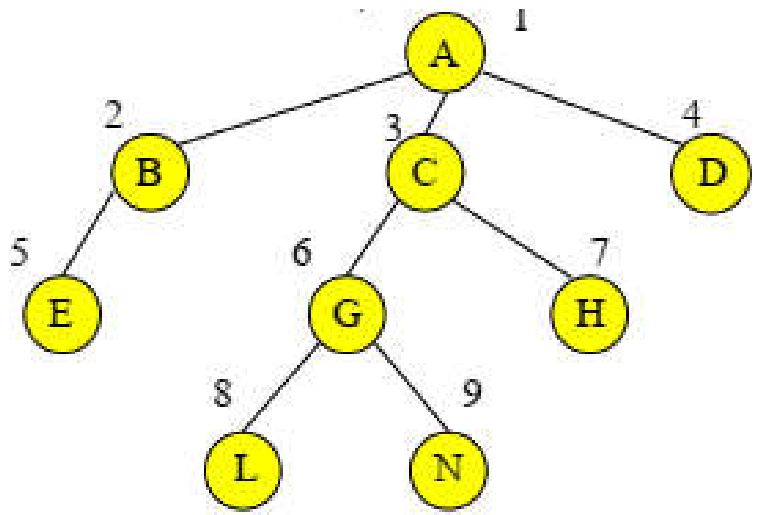
Các thao tác cơ bản trên cây

- Các thao tác khác
  - height (Tree T)
  - size(Tree T)
  - isRoot (Tree T, Node p);
  - isLeaf (Tree T, Node p);
  - isInternal (Tree T, Node p);

Biểu diễn cây tổng quát

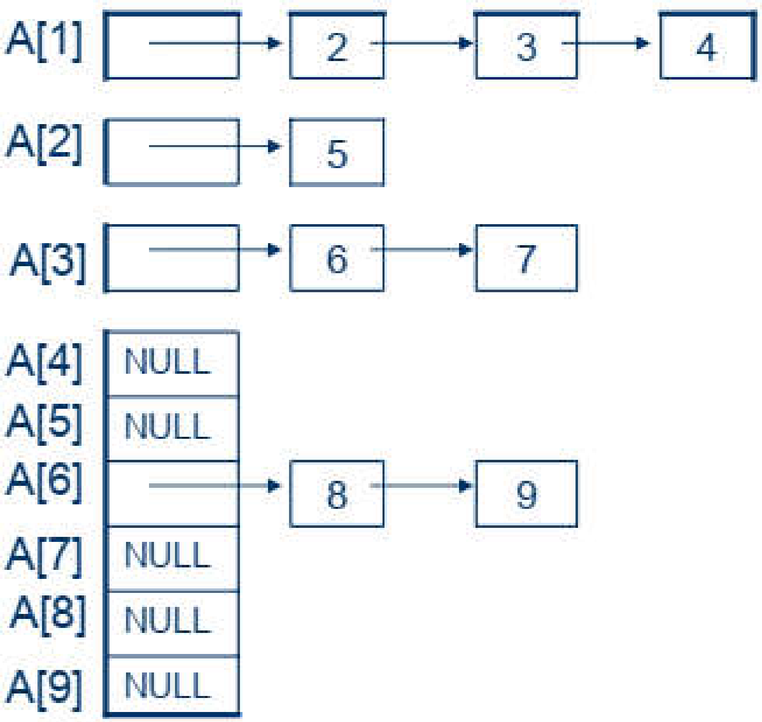
- **Dựa trên tham chiếu đến nút cha:**
  - Cây T có các nút được đánh số từ 1 đến n
  - Cây T được biểu diễn bằng một danh sách tuyến tính trong đó nút thứ i sẽ chứa một thành phần tham chiếu đến cha của nó
  - Nếu dùng mảng, A[i] = j nếu j là cha của nút i ; nếu i là gốc thì A[i] = 0;

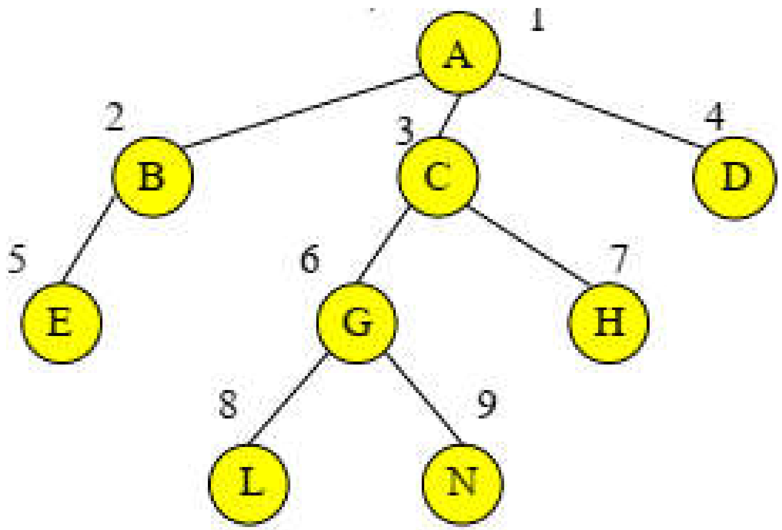
0	1	1	1	2	3	3	6	6
A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]



Biểu diễn cây tổng quát

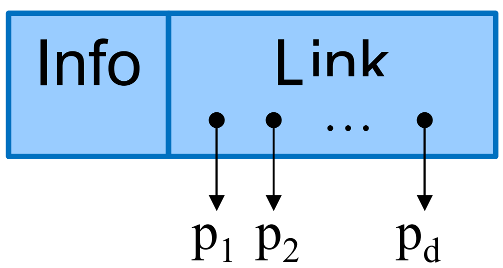
- **Dựa trên danh sách các nút con:**
  - 1 nút trong cây có một danh sách các nút con
  - Danh sách các nút con thường là danh sách móc nối
  - Trong trường hợp sử dụng danh sách móc nối, các nút đầu danh sách được lưu trong một mảng





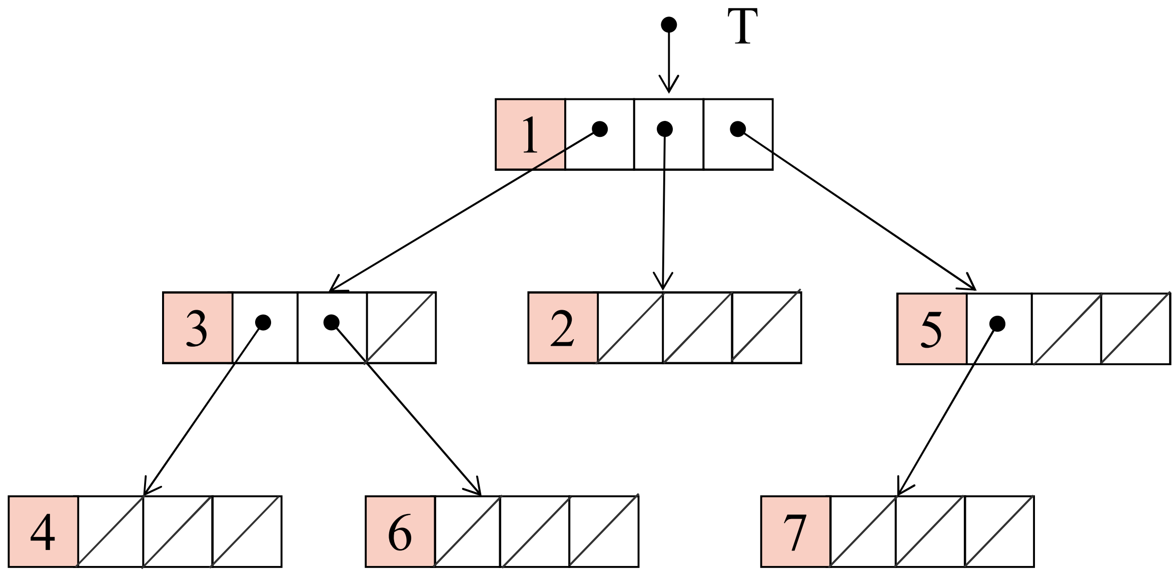
Biểu diễn cây tổng quát

- Giả sử một cây có cấp độ d
- Cấu trúc của một nút sẽ bao gồm các thông tin sau:
  - Info: chứa thông tin của nút
  - Link: chứa d con trỏ p_1, p_2,…p_d trỏ đến các con của nó (lớn nhất là d con)



Biểu diễn cây tổng quát

- Giả sử một cây có cấp độ d
- Tổ chức của cây tổng quát:
  - Một nút truy nhập vào nút gốc
  - Với mỗi nút, các con trỏ trỏ đến các con của nó



Biểu diễn cây tổng quát

- **Hạn chế của phương pháp biểu diễn dạng danh sách liên kết**
  - Nếu cây có kích thước N, cấp d thì số con trỏ NULL là N(d-1)+1
  - Khi d >=2 thì số con trỏ NULL lớn hơn kích thước của cây => lãng phí bộ nhớ
  - Đòi hỏi phải biết trước cấp của cây (d), điều này không phải lúc nào cũng thỏa mãn.

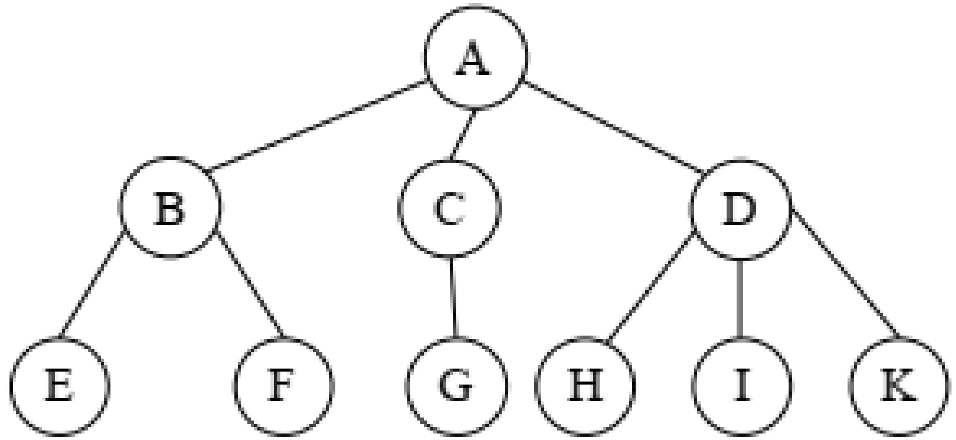
Biểu diễn cây tổng quát

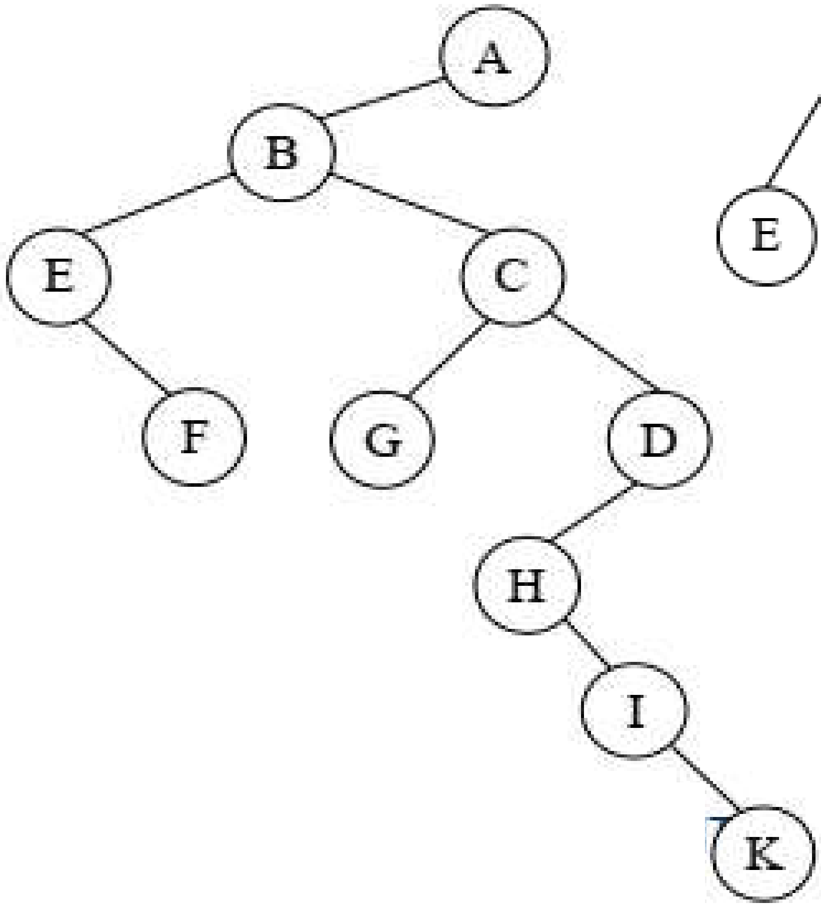
- **Thông qua một cây cấp 2:**
  - Với một nút trong cây, chỉ quan tâm tới 2 quan hệ:
    - Quan hệ 1-1 giữa nút đó và nút con cực trái của nó (con cả)
    - Quan hệ 1-1 giữa nút đó và nút em kế cận bên phải của nó
  - Dựa vào nhận định này, người ta biểu diễn được một cây tổng quát dưới dạng một cây nhị phân gọi là cây nhị phân tương đương (equivalent binary tree)
  - Quy cách của 1 nút trên cây nhị phân tương đương sẽ như sau

LCHILD	INFO	RSIBLING

Biểu diễn cây tương đương

- Cây tổng quát





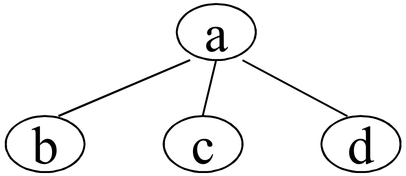
- Cây nhị phân tương đương

Cây nhị phân tương đương

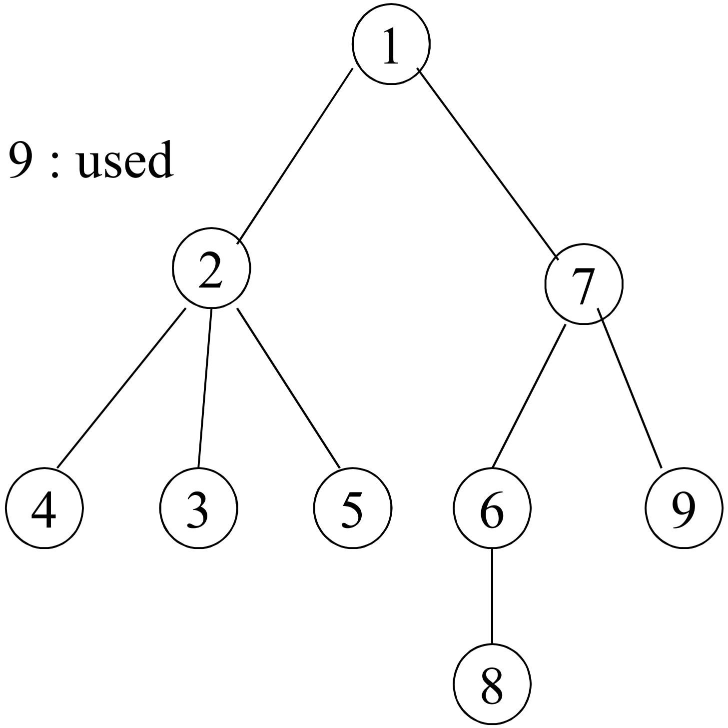
- **Luật ánh xạ giữa cây tổng quát và cây nhị phân tương đương như sau:**
  - Sắp xếp lại cây tổng quát (nếu cần)
  - Với mỗi nút:
    - Con cả của một nút trở thành nút trái của cây nhị phân
    - Em kề cận trở thành nút phải của cây nhị phân
- **Phương pháp biểu diễn này:**
  - Sử dụng bộ nhớ hiệu quả
  - Tuy nhiên việc đưa thêm các luật ánh xạ sẽ làm cho các thao tác trên cây trở nên phức tạp

Duyệt cây

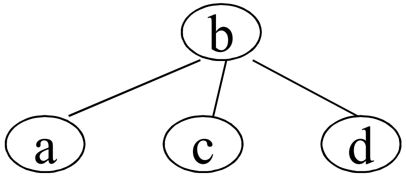
- **Preorder** (duyệt trước): 1, 2, 4, 3, 5, 7, 6, 8, 9



=> a, b, c, d

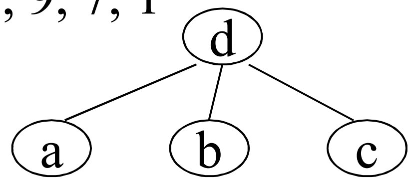


- **Inorder** (duyệt giữa): 4, 2, 3, 5, 1, 8, 6, 7, 9 : used for binary trees



=> a, b, c, d

- **Postorder** (duyệt sau): 4, 3, 5, 2, 8, 6, 9, 7, 1



=> a, b, c, d

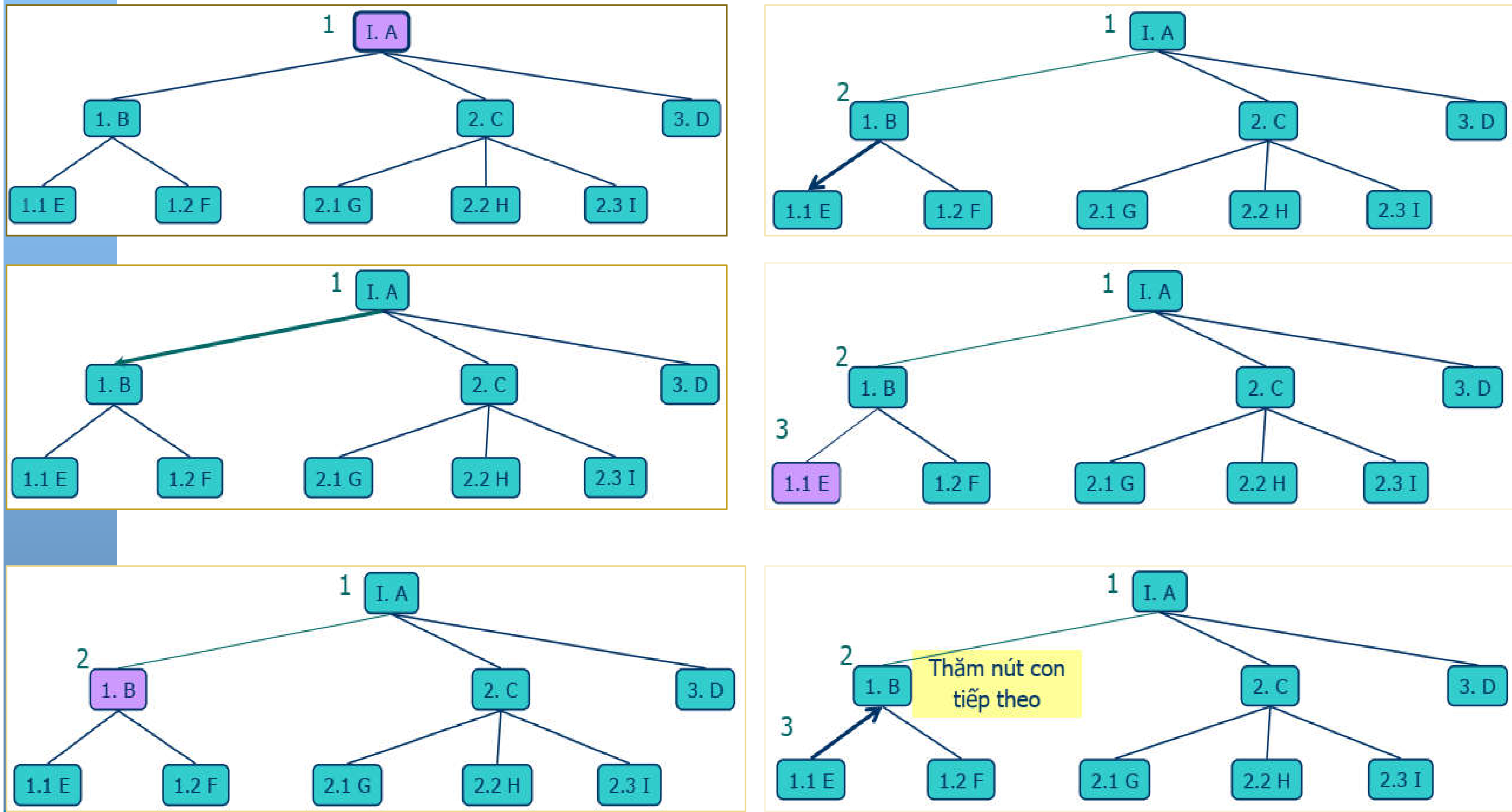
Duyệt cây theo thứ tự trước

- **Duyệt cây** là thăm các nút trên cây theo một thứ tự nhất định, mỗi nút thăm 1 lần
- **Khi duyệt theo thứ tự trước**, một nút sẽ được thăm trước các hậu duệ của nó
- **Ứng dụng:** In ra các mục lục của một tài liệu
- **Giải thuật:**

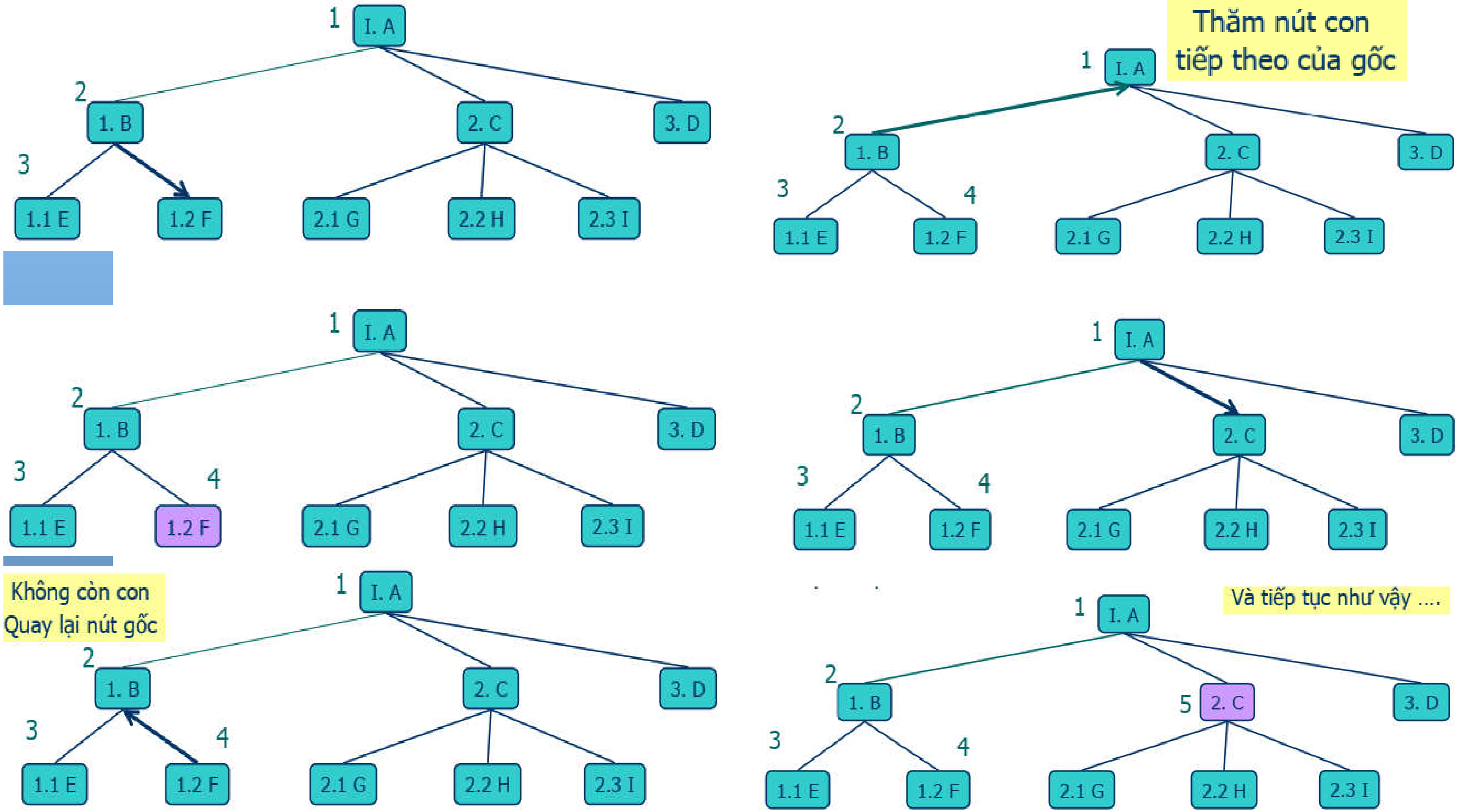
```
Algorithm preOrder(v)
    visit(v)
    for each child w of v
        preOrder(w)
```

SET 2020

Duyệt cây theo thứ tự trước



Duyệt cây theo thứ tự trước



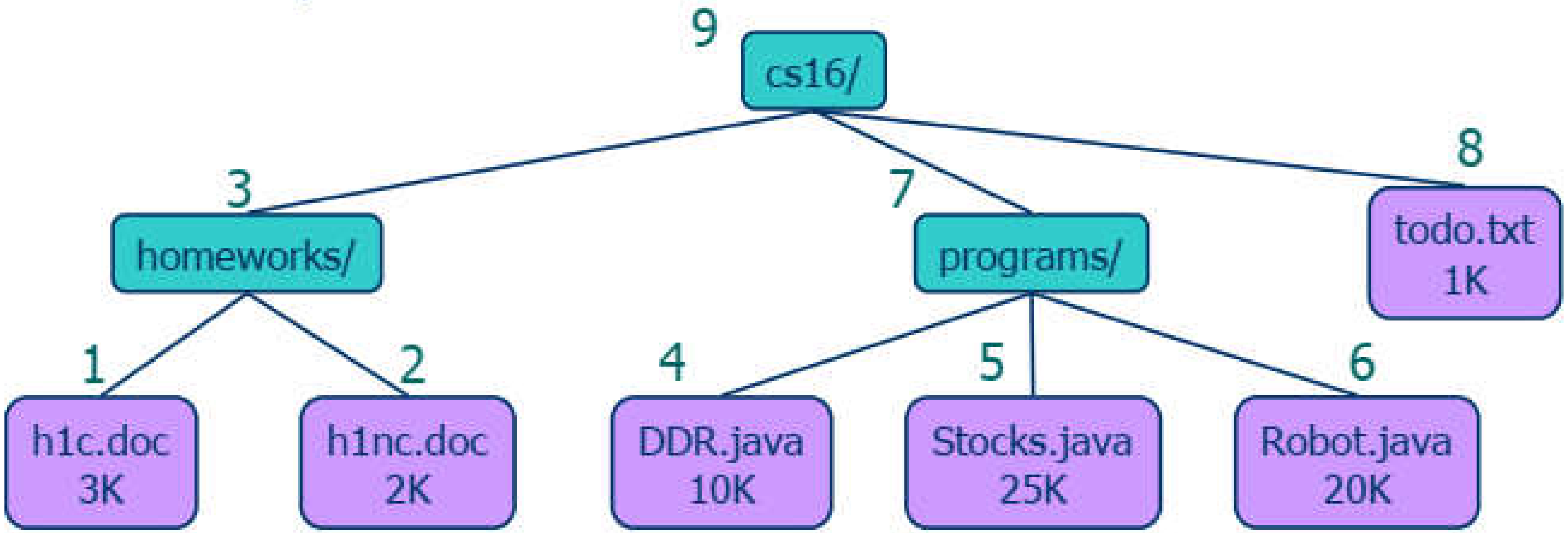
Duyệt cây theo thứ tự sau

- Duyệt theo thứ tự sau thì một nút sẽ được thăm sau các hậu duệ của nó
- Ứng dụng: Xác định kích thước của các tệp trong một thư mục và các thư mục con của nó

```
Algorithm postOrder(v)
    for each child w of v
        postOrder(w)
    visit(v)
```

```
Algorithm preOrder(v)
    visit(v)
    for each child w of v
        preOrder(w)
```

Duyệt cây theo thứ tự sau

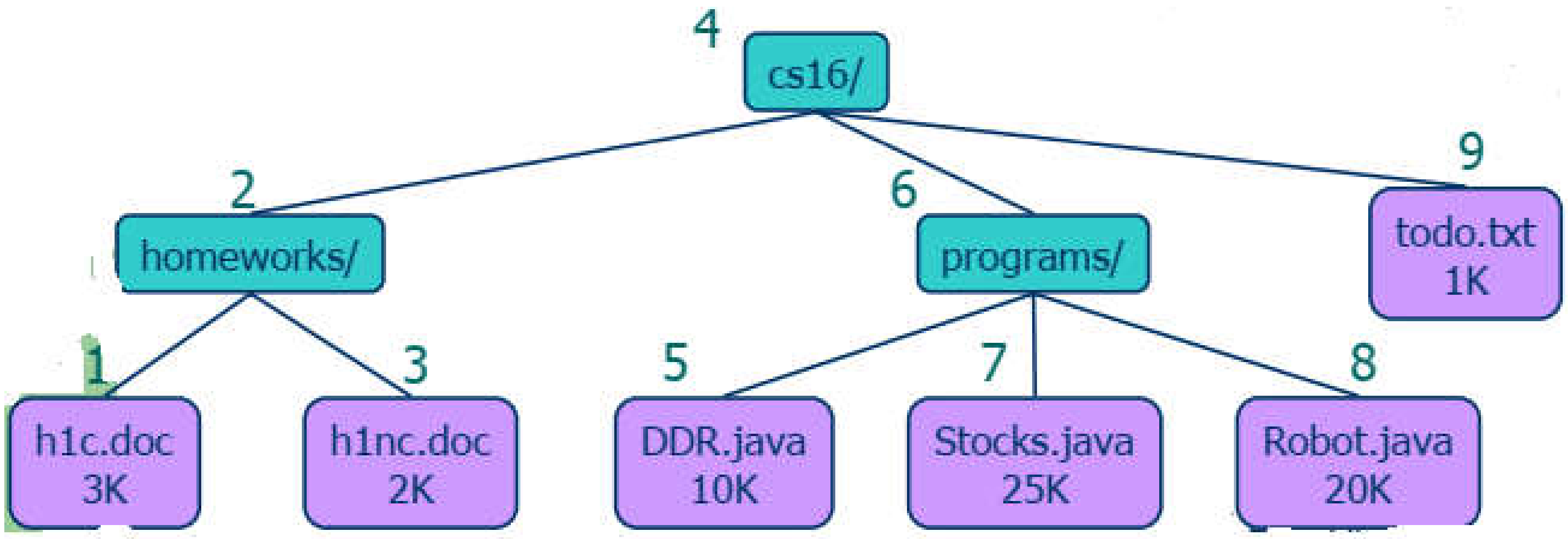


Duyệt cây theo thứ tự giữa

- Duyệt theo thứ tự giữa: một nút sẽ được thăm
  - sau các hậu duệ của nó trong cây con cực trái
  - và trước các hậu duệ trong các cây con tiếp theo

```
Algorithm inOrder(v)
    if (isLeaf(v)) then visit(v)
    else
        inOrder(left_most_child(v))
        visit(v)
        for each child w of v (w is
        not the left most child)
                inOrder(w)
```

Duyệt cây theo thứ tự giữa



Bài tập về nhà

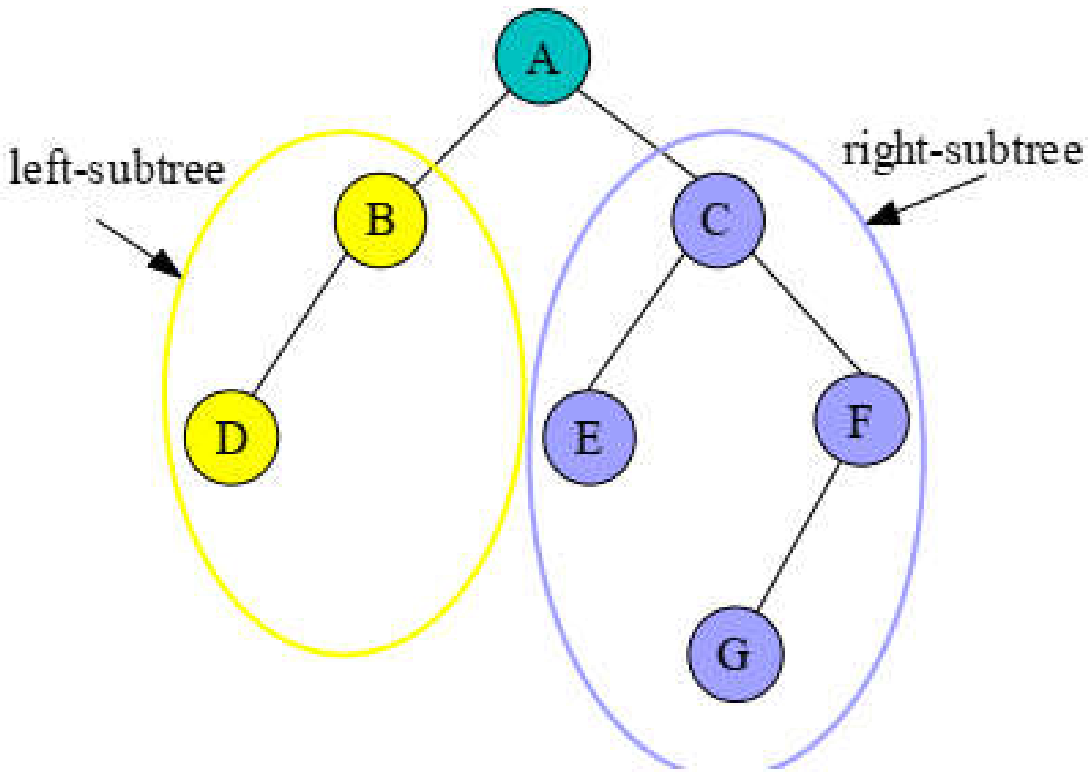
- **Khai báo cấu trúc của cây của một mục lục quyển sách bằng C/C++ theo một trong số các cách sau**
  - Mảng: quan hệ con-cha
  - Danh sách: quan hệ cha-con
  - Danh sách: quan hệ con cả - em kề cận
- **Khởi tạo cây**
- **In ra mục lục của sách theo cách duyệt cây theo thứ tự trước**

Nội dung của bài học

- Các khái niệm
- Cây tổng quát
  - Tính chất
  - Biểu diễn cây tổng quát
  - Duyệt cây tổng quát
- Cây nhị phân
  - Định nghĩa và tính chất
  - Biểu diễn cây nhị phân
  - Duyệt cây nhị phân
- Một số ví dụ ứng dụng cây

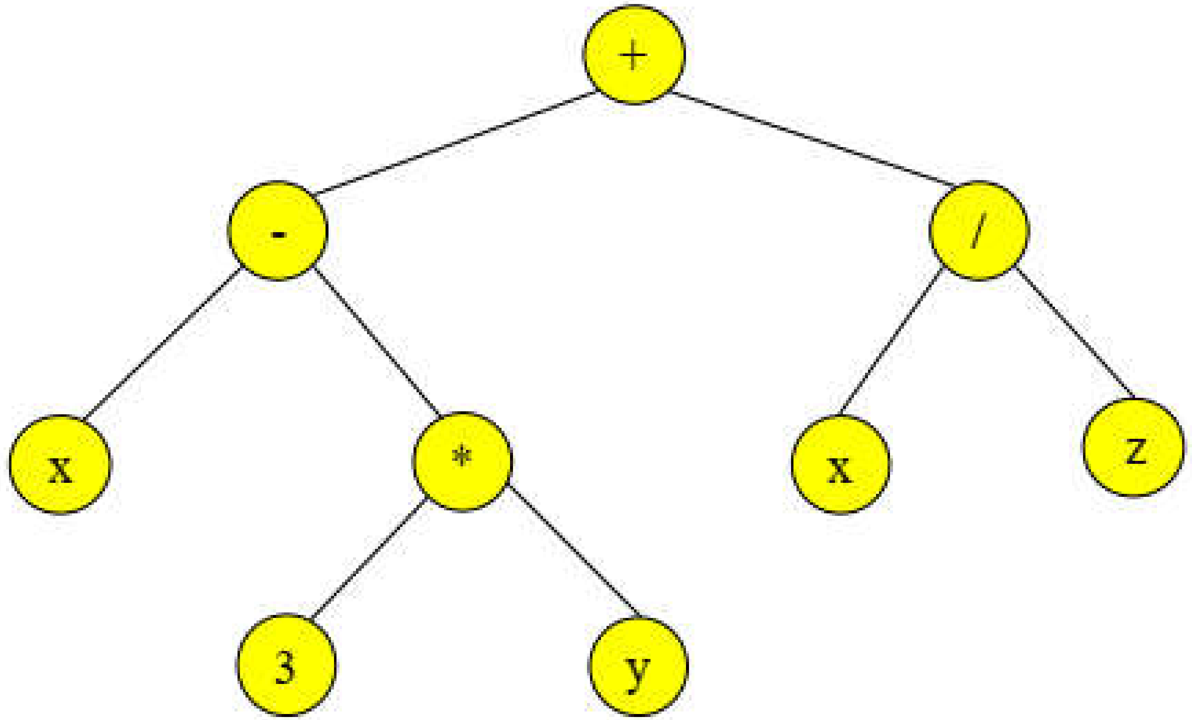
Cây nhị phân

- Là cây mà mọi nút trên cây chỉ có tối đa là 2 con.
- Cây con của một nút cũng cần phải được phân biệt rõ ràng thành cây con trái (left subtree) và cây con phải (right subtree)



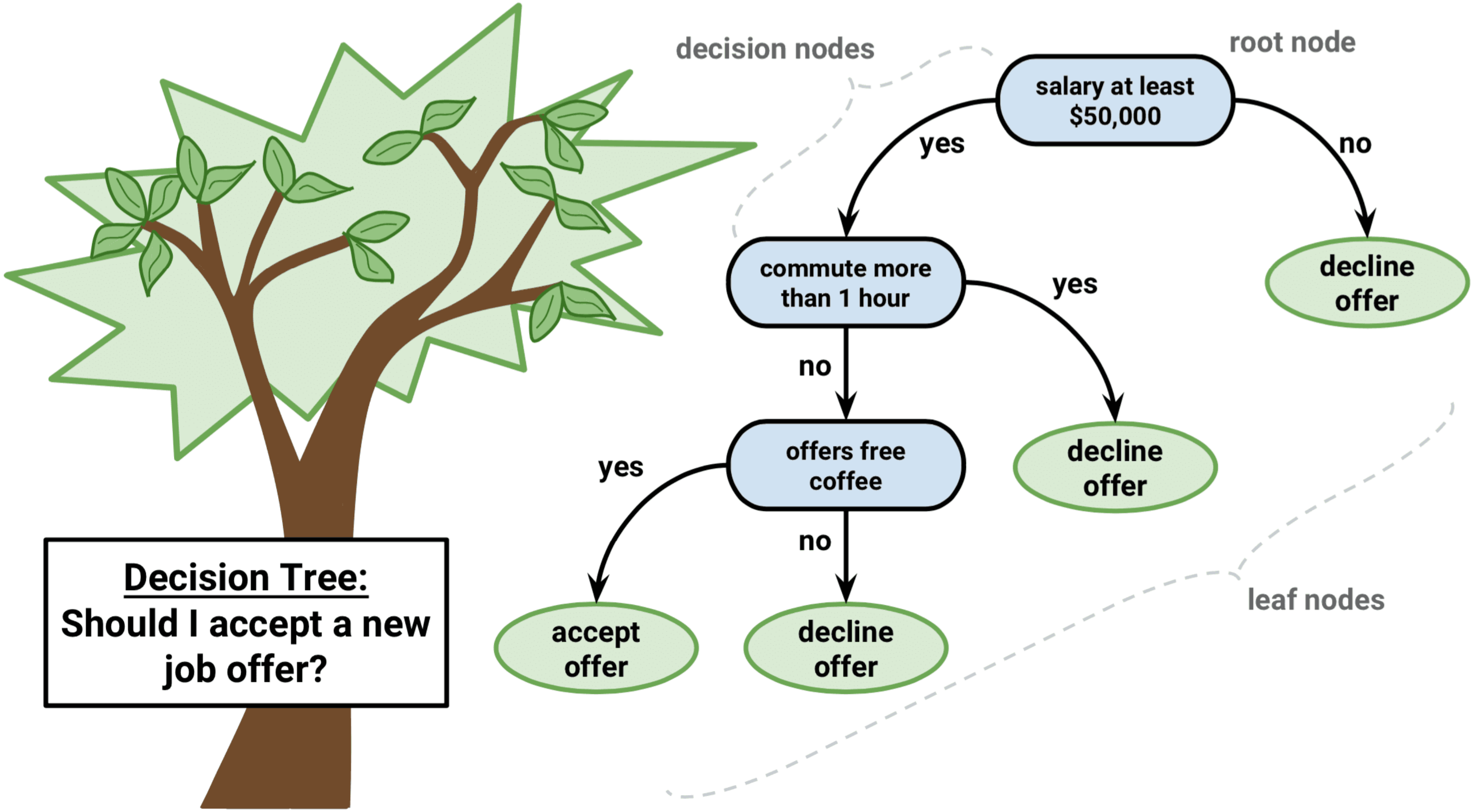
Ví dụ cây nhị phân

- Cây biểu thức số học với các phép toán 2 ngôi



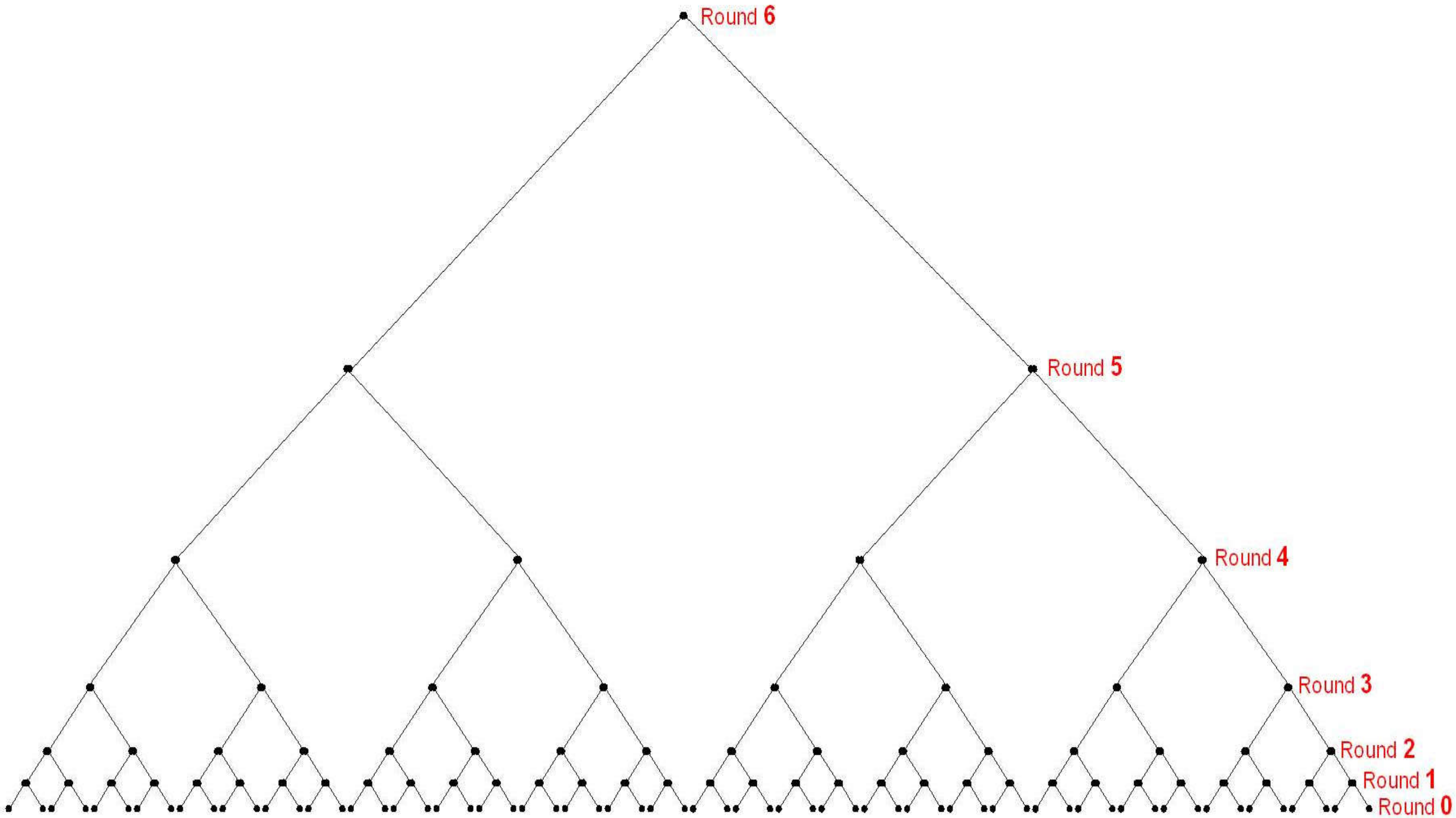
$x - 3*y + x/z$

Cây quyết định



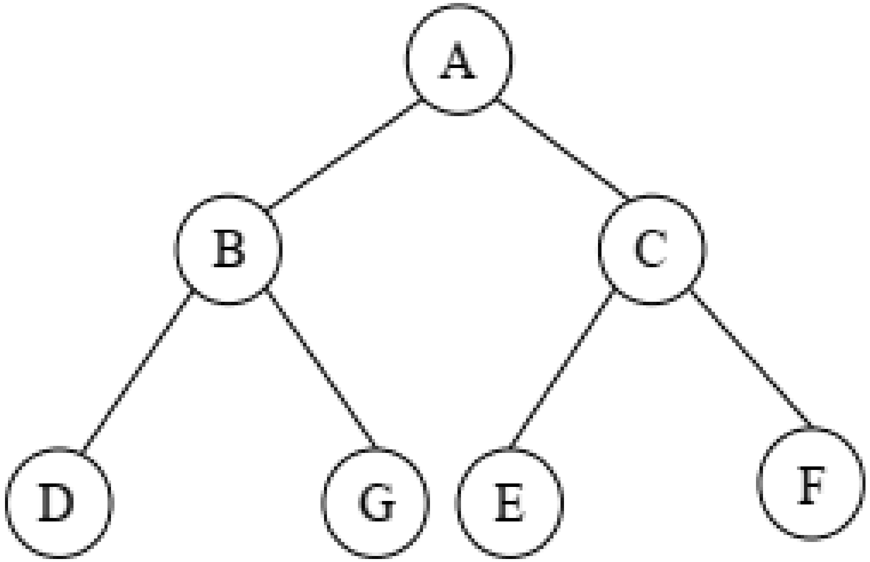
Cây nhị phân

- Vòng thi đấu thể thao theo từng cặp



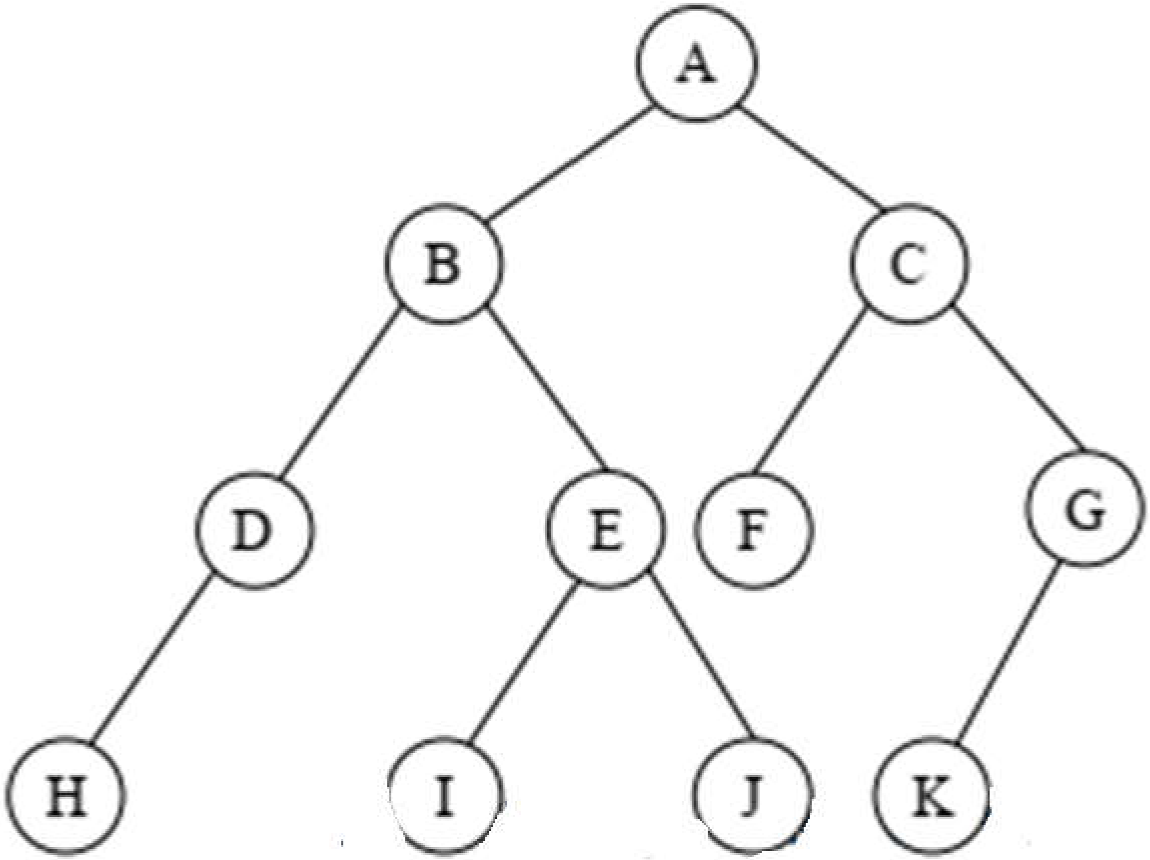
Dạng đặc biệt của cây nhị phân

- **Cây nhị phân đầy đủ:** (full binary tree) Mỗi nút trong của cây đều có đầy đủ 2 con



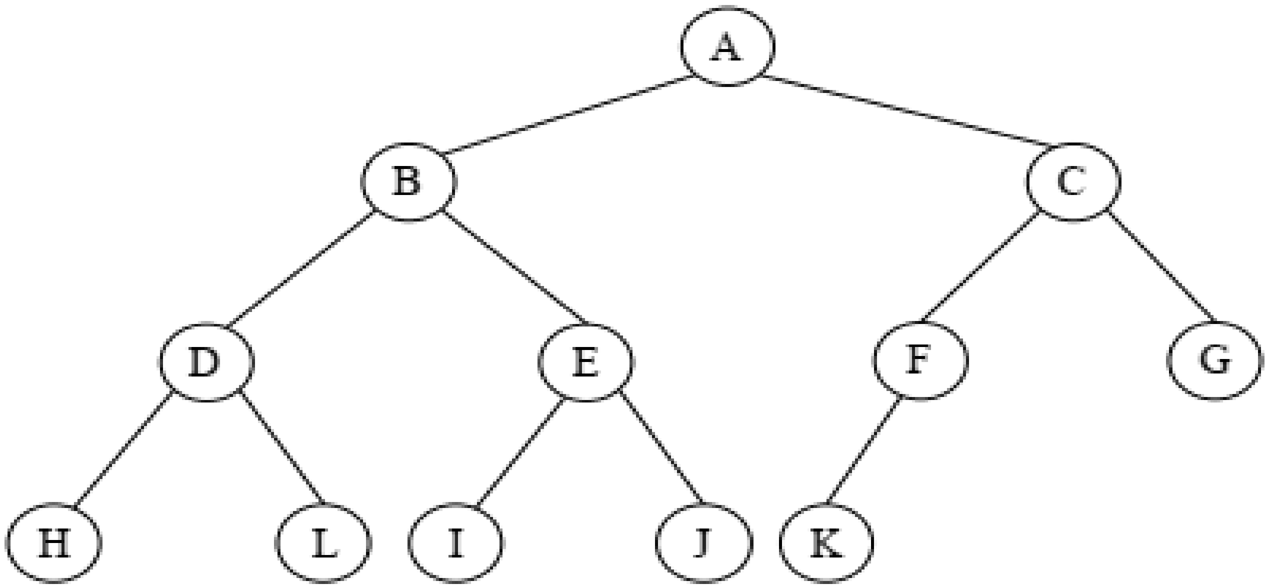
Dạng đặc biệt của cây nhị phân

- **Cây nhị phân gần đầy đủ:** ở mức cuối không có đầy đủ các nút



Dạng đặc biệt của cây nhị phân

- **Cây nhị phân hoàn chỉnh**
  - Là cây nhị phân gần đầy
  - Tất cả các nút ở mức cuối cùng đều lệch về bên trái nhất có thể



- **Cây nhị phân cân đối:** cây con trái và cây con phải lệch nhau không quá một đơn vị

Tính chất của cây nhị phân

- Số lượng tối đa của các nút ở mức i trên một cây nhị phân là 2^{i-1} (i >= 1)
- Số lượng tối đa các nút trên một cây nhị phân có chiều cao là h là $2^h - 1$ (h >= 1)
- Một cây nhị phân có n nút có chiều cao tối thiểu là

$$\lceil \log_2(n+1) \rceil$$

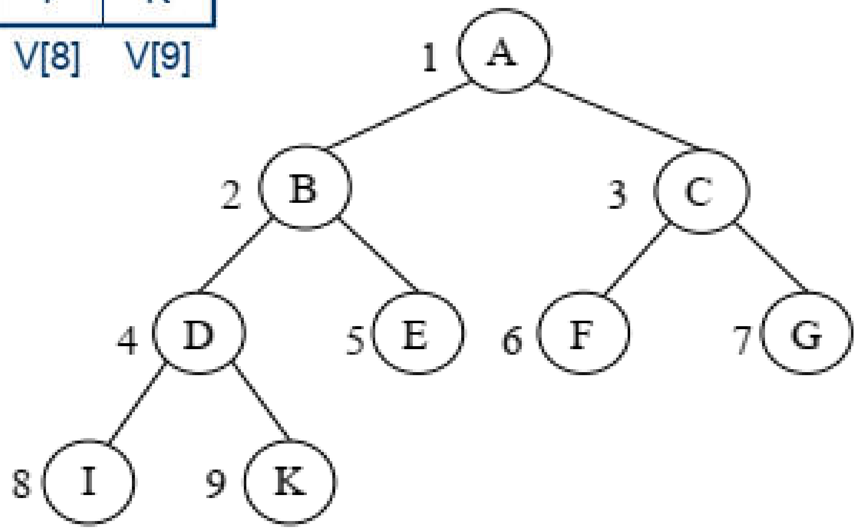
- Một cây nhị phân đầy đủ có độ sâu n thì có 2^n -1 nút
- Một cây nhị phân hoàn chỉnh có chiều cao h có số lượng nút nằm trong khoảng 2^{h-1} đến $2^h - 1$
- Trong một cây nhị phân có n_0 nút lá và n_2 nút cấp 2 thì ta có $n_0 = n_2 + 1$

SET
2020

Biểu diễn cây nhị phân

- **Biểu diễn kế tiếp sử dụng mảng:**
  - Đánh số các nút trên cây theo trình tự từ mức 1, hết mức này đến mức khác, từ trái sang phải
  - Lưu trữ trong vector lưu trữ V theo nguyên tắc phần tử V[i] sẽ lưu thông tin của nút được đánh số i

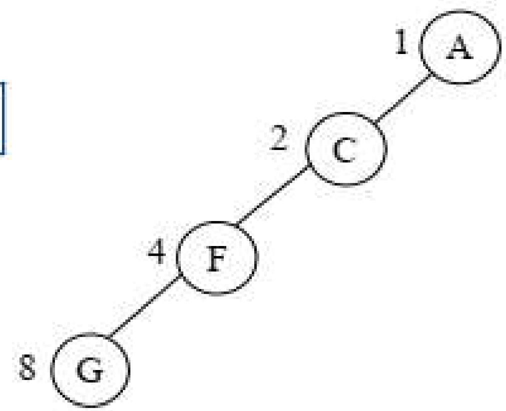
A	B	C	D	E	F	G	I	K
V[1]	V[2]	V[3]	V[4]	V[5]	V[6]	V[7]	V[8]	V[9]



Biểu diễn cây nhị phân

- Cách lưu trữ kế tiếp phù hợp để lưu trữ cây nhị phân gần đầy hoặc đầy đủ
- Với các dạng khác có thể dẫn đến lãng phí bộ nhớ

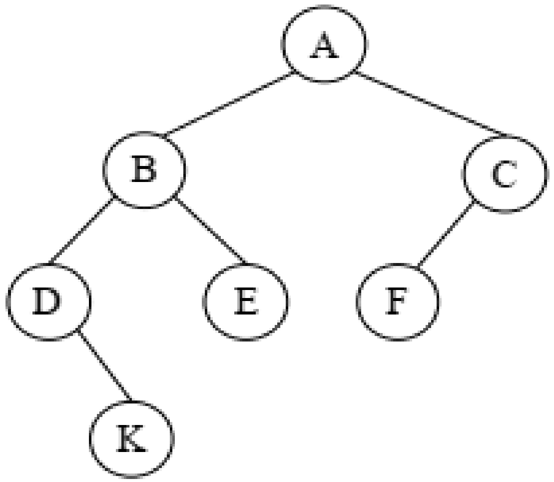
A	C		F				8
V[1]	V[2]	V[3]	V[4]	V[5]	V[6]	V[7]	V[8]

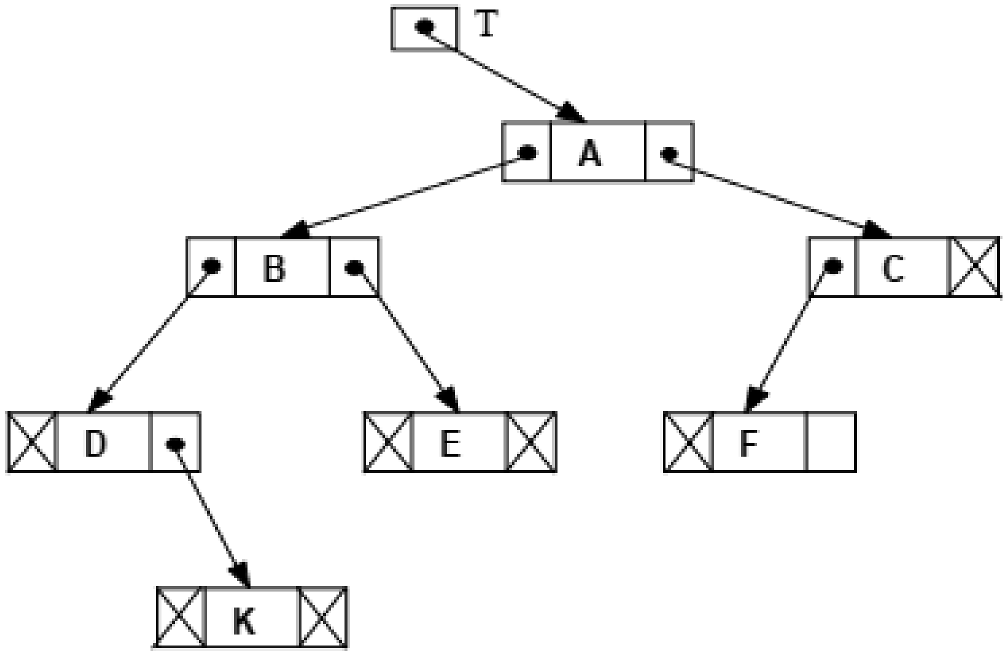


Biểu diễn cây nhị phân

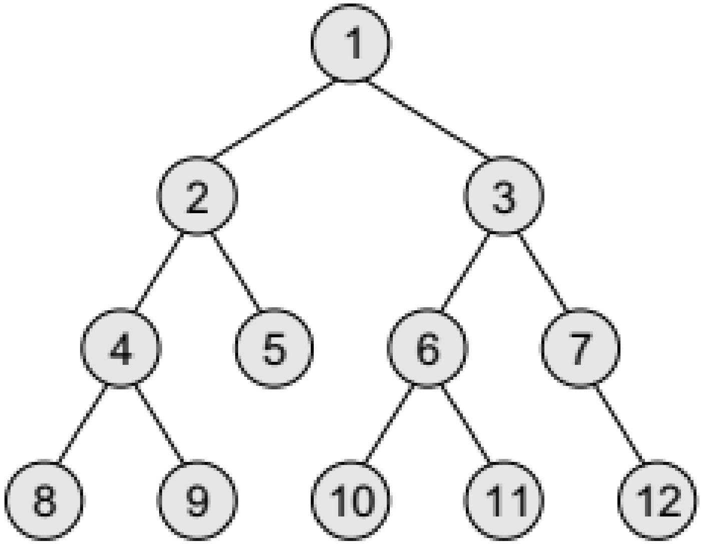
- **Biểu diễn móc nối sử dụng con trỏ:**
  - Mỗi nút trên cây được lưu trữ bởi một phần tử có quy cách như sau:
    - INFO: chứa dữ liệu của nút
    - LPTR: chứa địa chỉ của nút gốc của cây con trái
    - RPTR: chứa địa chỉ của nút gốc của cây con phải
  - Cần nắm một con trỏ T trỏ tới nút gốc của cây.
  - Nếu cây rỗng thì T = NULL

Biểu diễn cây nhị phân





Lưu trữ cấu trúc cây trong bộ nhớ



ROOT

3

	LEFT	DATA	RIGHT
1	–1	8	–1
2	–1	10	–1
3	5	1	8
4			
5	9	2	14
6			
7			
8	20	3	11
9	1	4	12
10			
11	–1	7	18
12	–1	9	–1
13			
14	–1	5	–1
15			
16	–1	11	–1
17			
18	–1	12	–1
19			
20	2	6	16

15

AVAIL

S E T

2020

Biểu diễn cây nhị phân

- *struct Tnode{*

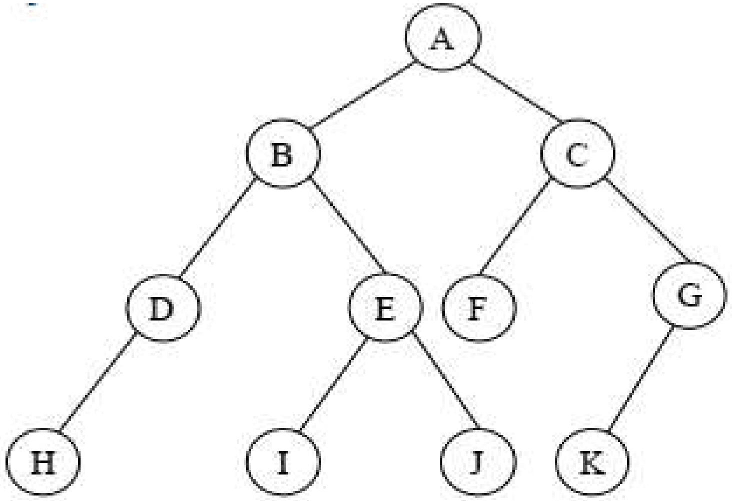
```
struct Tnode{
        int info;
        struct Tnode * lptr;
        struct Tnode * rptr;
} ;
typedef struct Tnode TREENODE;
typedef TREENODE *TREENODEPTR;
```

Duyệt cây nhị phân

- Phép duyệt cây nhị phân: Phép duyệt một cây là phép "thăm" lần lượt các nút trên cây đó sao cho mỗi nút chỉ được thăm một lần
- Tồn tại 3 phép duyệt khác nhau đối với 1 cây nhị phân
  - Duyệt cây theo thứ tự trước
  - Duyệt cây theo thứ tự giữa
  - Duyệt cây theo thứ tự sau:

Duyệt cây nhị phân

- Ví dụ: Thực hiện duyệt cây
- **Duyệt theo thứ tự trước**
  A B D H E I J C F G K
- **Duyệt theo thứ tự giữa**
  H D B I E J A F C K G
- **Duyệt theo thứ tự sau**
  H D I J E B F K G C A



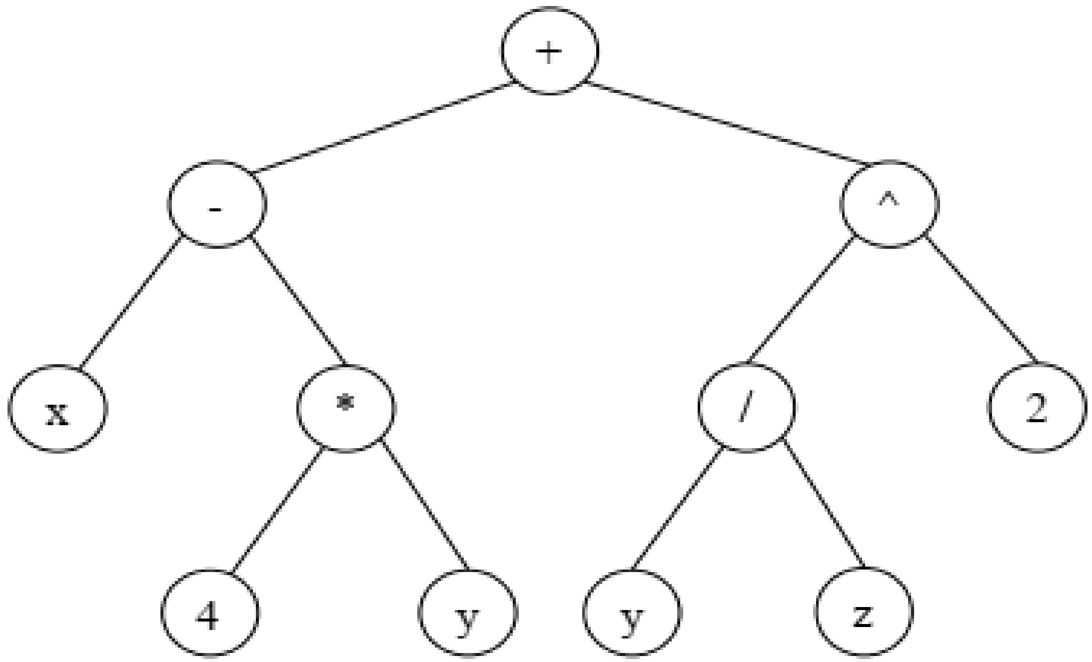
Duyệt cây nhị phân theo thứ tự trước

```
void PREORDER(TREENODEPTR tree) {
        if (tree != NULL) {
                printf("%3d", tree->info;
                PREORDER(tree->lptr);
                PREORDER(tree->rptr);
        }
}
```

SET 2020

Duyệt cây nhị phân

- Ví dụ 2: Cho cây nhị phân biểu diễn biểu thức số học sau
  - Hãy đưa ra dãy các nút được thăm khi thực hiện các phép duyệt theo thứ tự trước, giữa và sau.
  - Nhận xét về các dãy thu được

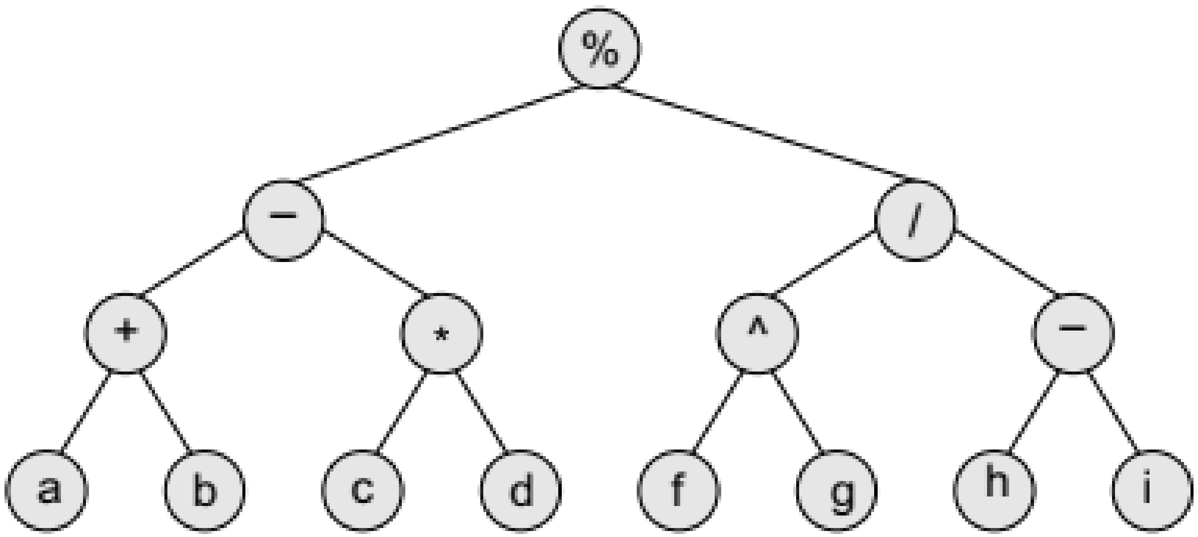


Cây biểu thức

- **Biểu diễn của biểu thức:**
  - Biểu thức = biểu thức <toán tử> biểu thức
  - Trường hợp đặc biệt: biểu thức = const
  - Toán tử (phép toán): +,-,*,/,exp, !, v.v
- **Sử dụng cây nhị phân**
  - Các nút nhánh là các toán tử
  - Các nút lá là các toán hạng
- **Duyệt cây:**
  - Thứ tự trước: biểu thức tiền tố
  - Thứ tự giữa: biểu thức trung tố
  - Thứ tự sau: biểu thức hậu tố

Cây biểu thức

- Bài toán 1: Dựng cây biểu diễn biểu thức số học:
- Cho một biểu thức số học dưới dạng hậu tố, dựng cây biểu diễn biểu thức số học đó
- Ví dụ: Cho biểu thức ((a+b)-c*d)%(f^g / (h-i)).
- Dựng được cây biểu diễn biểu thức này như sau



Ký pháp Ba Lan

- **Trong ký pháp trung tố:**
  - toán tử được đặt giữa hai toán hạng
  - Việc sử dụng các dấu ngoặc để biểu diễn thứ tự ưu tiên của các toán hạng là cần thiết
  - Ví dụ: (A+B)*C khác với A+B*C
  - Nếu không sử dụng dấu ngoặc thi phải thực hiện theo ưu tiên của các phép toán
  - Việc sử dụng ký pháp trung tố với dấu ngoặc hoặc phải theo thứ tự ưu tiên làm cho việc tính toán giá trị biểu thức trở nên cồng kềnh.
- **Ký pháp Ba Lan (Polish notation):** cho phép biểu diễn dạng hậu tố hoặc tiền tố

Biểu thức hậu tố

- **Ký pháp hậu tố:** Toán tử được đặt sau toán hạng 1 và toán hạng 2

Trung tố	Hậu tố
A+B	AB+
E/F	EF/
(A+B)*C	AB+C*
A+B*C	ABC*+

- **Ký pháp tiền tố**: Toán tử được đặt trước toán hạng 1 và 2

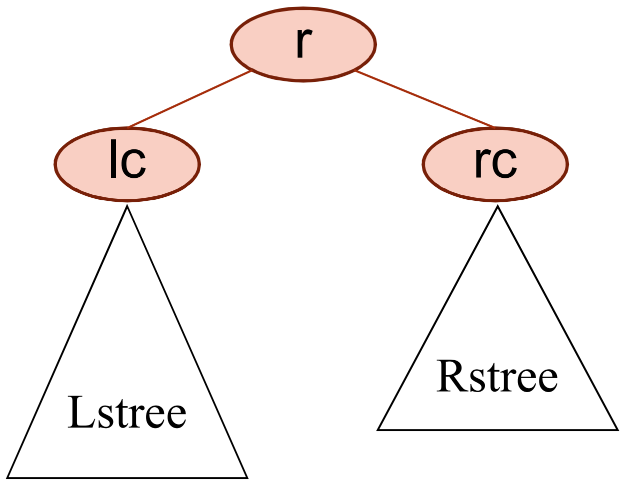
Trung tố	Hậu tố
E/F	/EF
A+B*C	+A*BC
(A+B)/(C-D)+E	+/+AB-CDE

Cây nhị phân tìm kiếm

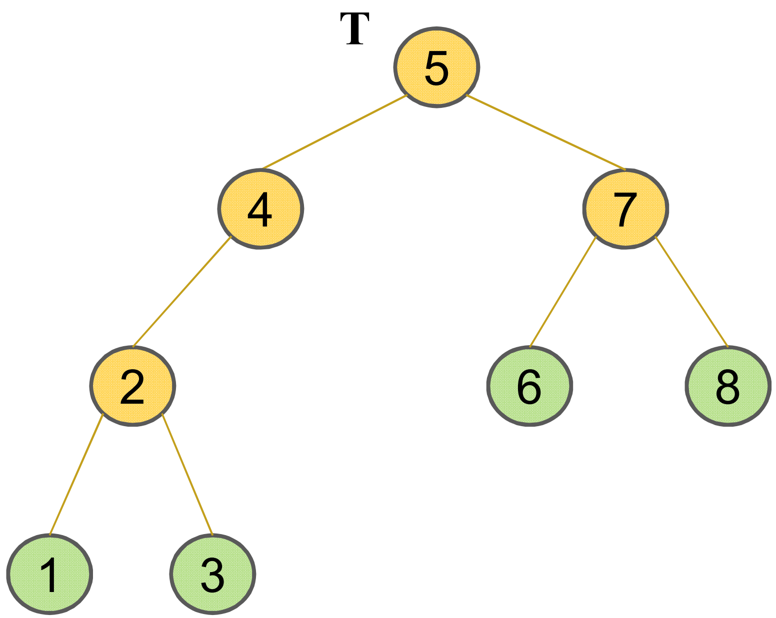
- **Vấn đề:** việc tìm kiếm một nút trong cây nhị phân theo các cách duyệt (trước, giữa, sau) khá chậm do phải duyệt qua tất cả các nút của cây.
- **Giải pháp:** xây dựng một cấu trúc cây đặc biệt phục vụ cho việc tìm kiếm nhanh (cây nhị phân tìm kiếm – Binary Search Trees)

Cây nhị phân tìm kiếm

- Giả sử có một cây nhị phân T
- Mỗi nút có một trường đặc biệt gọi là khóa (key), Gọi khóa của một node p là key(p)
- Gọi r là nút gốc của T, Hai con trái và phải của r là lc và rc
- Hai cây con tương ứng với hai nút lc và rc là Lstree và Rstree
- T là cây nhị phân tìm kiếm (BST) nếu thỏa mãn:
  - key(lc) < key(r);
  - key(r) < key(rc);
  - Lstree và Rstree là cây nhị phân tìm kiếm



Cây nhị phân tìm kiếm



Duyệt giữa: 1 2 3 4 5 6 7 8

Cây nhị phân tìm kiếm

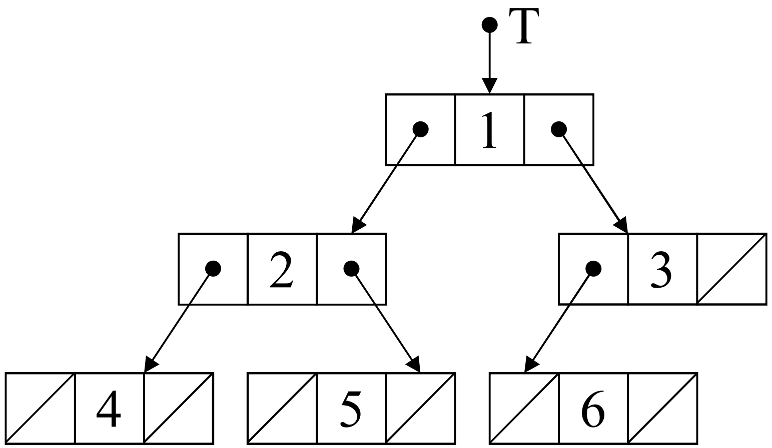
```
struct  Node  {
   keytype  key;
   Node *LP, *RP;
};

typedef  Node*  PNode;
typedef  PNode BinaryTree;
typedef  BinaryTree  BSearchTree;
```

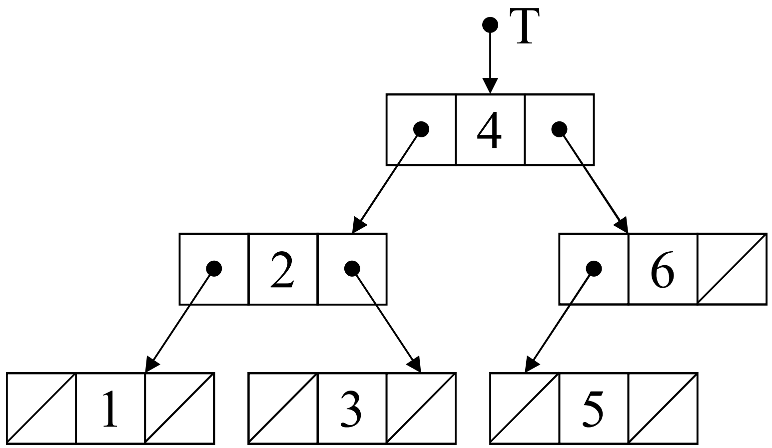
S E T
2020

Cây nhị phân tìm kiếm

- Cách tổ chức giống cây nhị phân thường



Binary tree



Binary search tree

Cây nhị phân tìm kiếm

- **Các thao tác cơ bản:**
  - Tìm kiếm: tìm một nút có giá trị x trong BST, trả về con trỏ đến nút tìm thấy, nếu không trả về NULL
  - Lời giải:
    - Base case: if (T=NULL or key(T) = x) return T;
    - Recursive case:

      otherwise if (x<key(T)) search for a node in left sub-tree of T

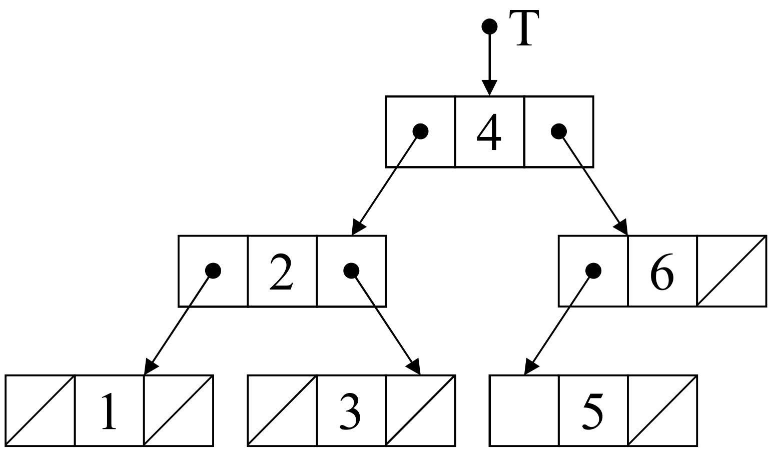
      else search for a node in right sub-tree of T
  - Code

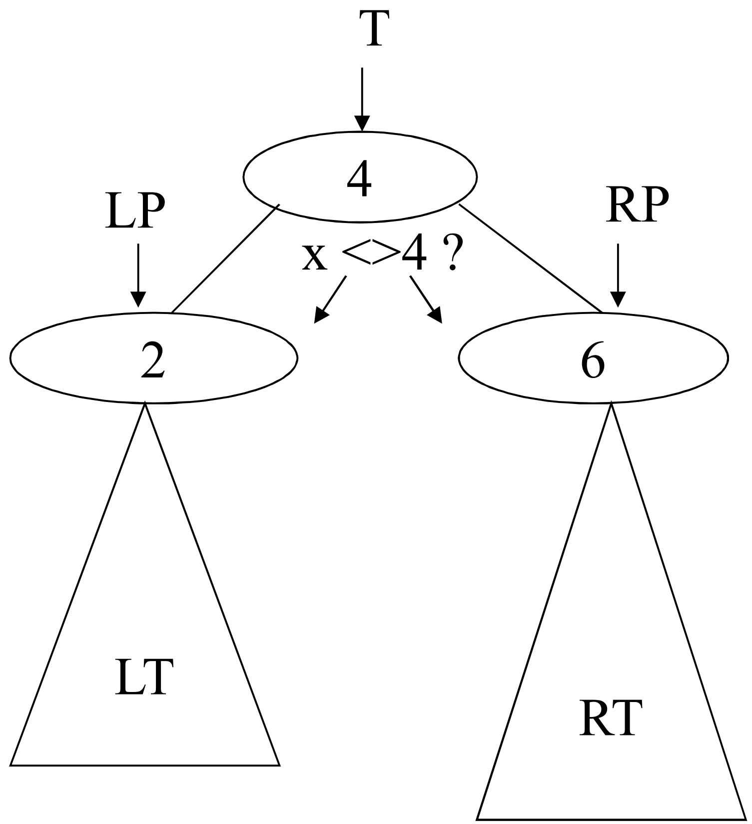
```
PNode Search (BSearchTree T, keytype  x){
   if  (T==NULL)  return  NULL;
   if  (x == T->Key) return  T;
   else
      if  (x < T->Key)  return  Search(T->LP, x);
      else  return  Search(T->RP, x);
}
```

$O(Search) = log_2(N)$, N is size of T

Cây nhị phân tìm kiếm

- Các thao tác cơ bản:
  - Chèn: chèn vào cây T một nút mới có giá trị x





Cây nhị phân tìm kiếm

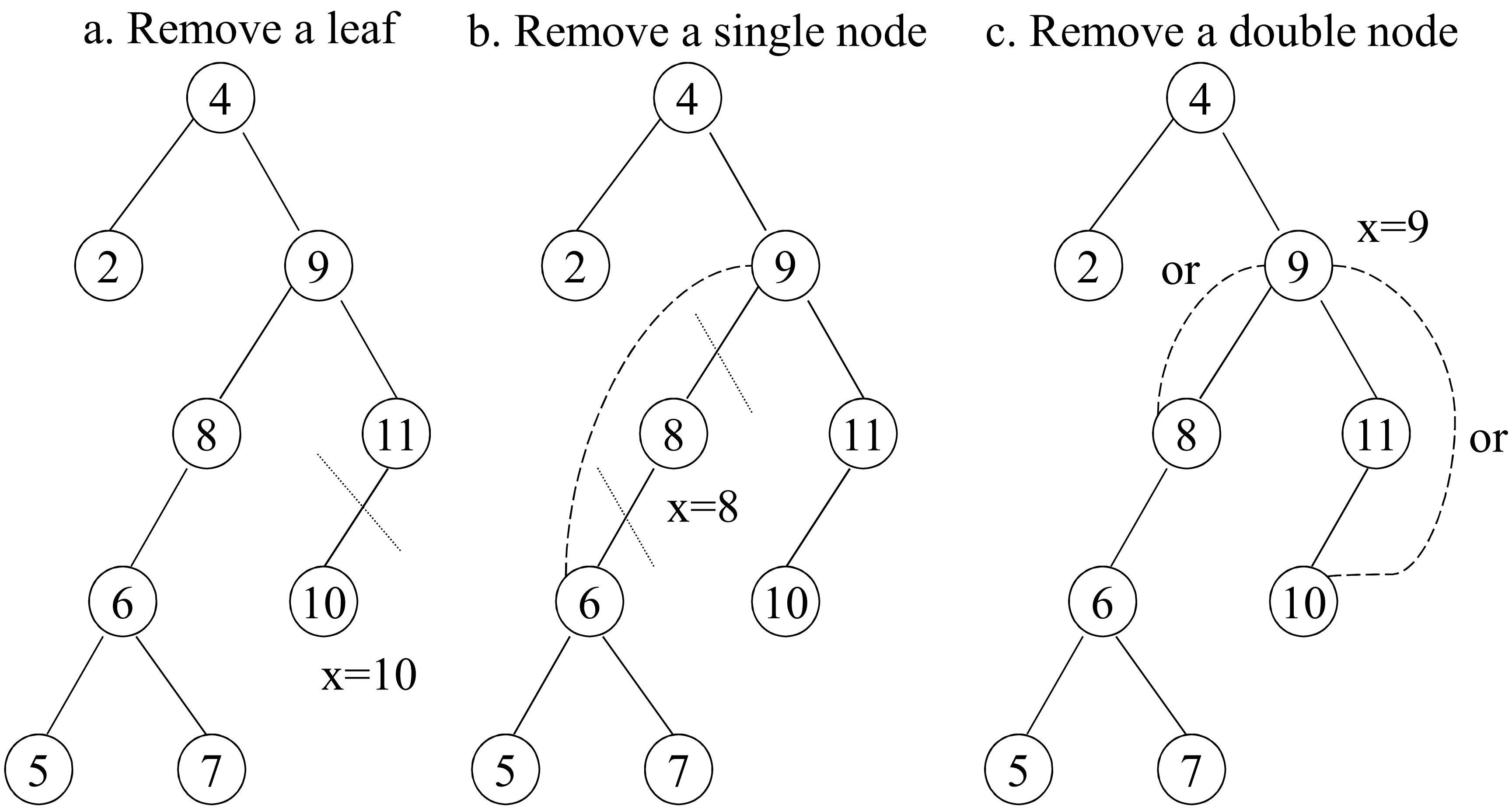
- **Các thao tác cơ bản:**
  - Chèn: chèn vào cây T một nút mới có giá trị x

```
void InsertT(BSearchTree & Root, keytype x){
   PNode Q;
   if (Root==NULL) {          //tree is empty
        Q = new Node;
        Q->Key = x;
        Q->LP =        Q->RP = NULL;
        Root = Q;
   }
   else {
        if (x < Root->Key) InsertT(Root->LP, x);
        else if (x > Root->Key) InsertT(Root->RP, x);
   }
}
```

Cây nhị phân tìm kiếm

- **Các thao tác cơ bản:**
  - Bỏ một nút có khóa x của cây BST
  - Giải thuật:
    - Tìm nút có khóa (with `key = x`)
    - Loại bỏ nút và xếp lại cây nếu cần. 3 trường hợp xảy ra:
      - Nếu nút loại bỏ là nút lá : không cần sắp lại cây
      - Nếu nút đó có 1 con: con sẽ thay thế nút
      - Nếu nút có 2 con: `LTree`, `RTree` => thay nút đó bởi Max(LTree) (maximal node in LTree) or Min(RTree) (minimal node in RTree)
    - Notes:
      - `Max(LTree)` là nút bên phải nhất của cây `LTree`.
      - `Min(RTree)` là nút bên trái nhất của cây `RTree`.

Cây nhị phân tìm kiếm



Cây nhị phân tìm kiếm

```
void  DeleteT(BSearchTree & Root, keytype  x){
    if  (Root != NULL) {
        if  (x < Root->Key)  DeleteT(Root->LP, x);
        else  if  (x > Root->key)  DeleteT(Root->RP, x);
        else  DelNode (Root);          //remove the root
    }
}
```

```
void  DelNode(PNode & P) { //remove node P & re-arrange if
    necessary
    PNode Q, R;
    if (P->LP == NULL) {       //the node has only right child
        Q = P;
        P = P->RP;
    }
    else
        if (P->RP = NULL)      //the node has only left child
        {                      //continue in next page
```

Cây nhị phân tìm kiếm

```
            Q = P;
            P = P->LP;
        }
        else {                //Remove a double node
            Q = P->LP;
            if  (Q->RP == NULL) {
                P->Key = Q->Key;
                P->LP = Q->LP;
            }
            else {
                do {          //R used to store parent of Q
                    R = Q;
                    Q = Q->RP;
                } while  (Q->RP != NULL);
                P->Key = Q->Key;
                R->RP = Q->LP;
            }
        }
    delete  Q;
}
```

S E T

2020