HUST

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

ET2100

ONE LOVE. ONE FUTURE.

Chương IV: Cấu trúc CÂY

Giảng viên: TS. Trần Thị Thanh Hải
Khoa Kỹ thuật Truyền thông – Trường Điện Điện Tử

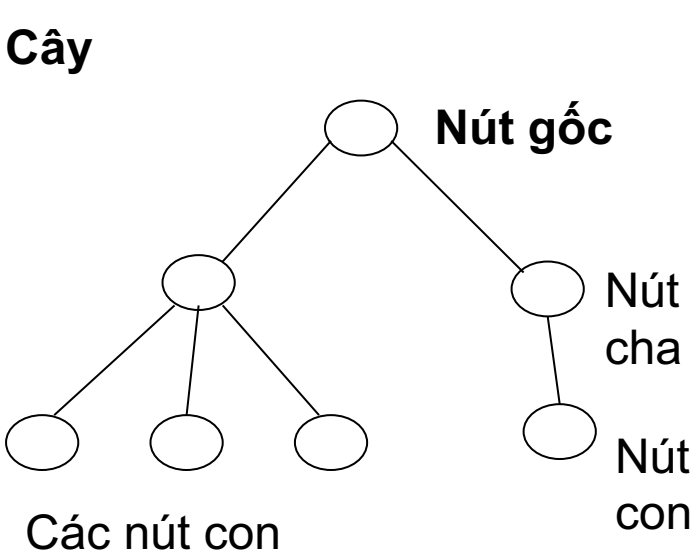
Các nội dung chính

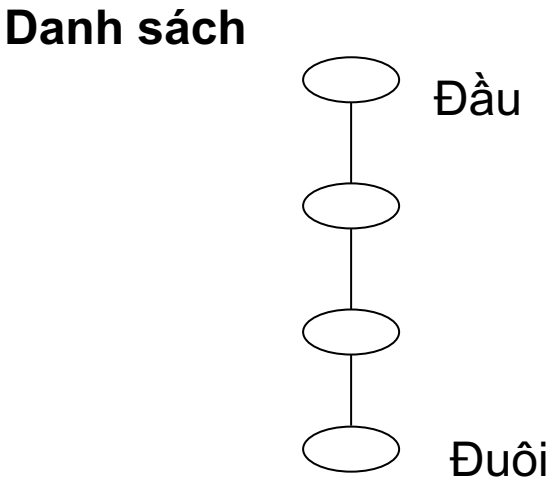
- 1. Giới thiệu
  - Mô tả cấu trúc cây (tree)
  - Các khái niệm cơ bản trong cây
  - Các tính chất cơ bản
  - Các thao tác cơ bản
- 2. Cây nhị phân (binary tree)
  - Khái niệm, tính chất của cây nhị phân
  - Các thao tác cơ bản
  - Duyệt cây nhị phân
  - Cài đặt cây nhị phân
  - Cây nhị phân tìm kiếm
- 3. Cây tổng quát
  - Biểu diễn cây tổng quát
  - Cài đặt cây tổng quát
- 4. Ứng dụng của cấu trúc cây
- 5. Cây cân bằng

1. Giới thiệu

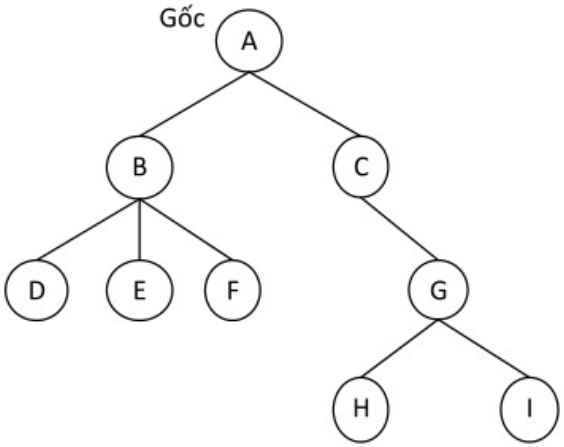
Định nghĩa cây

- Cây là một cấu trúc phi tuyến
- Thiết lập trên một tập hữu hạn các phần tử / "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 (đường nối gọi là nhánh)
  - Một nút (trừ nút gốc) chỉ có một cha
  - Một nút có thể có từ *0* đến *n* con



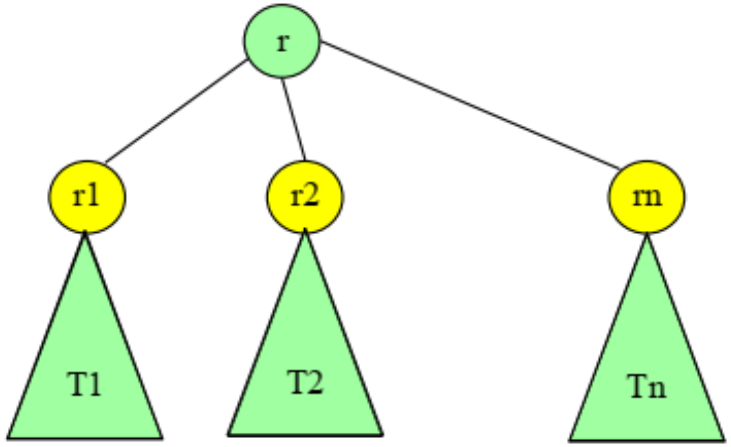


Định nghĩa cây



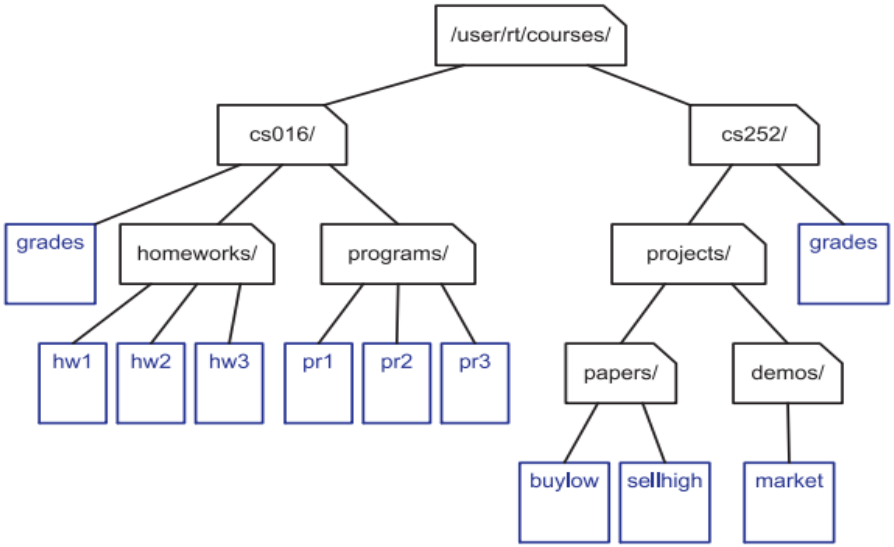
Nút gốc: A
Các nhánh: (A, B), (A, C), ..., (G, I)

- Cây rỗng: cây không một nút nào
- Một nút tạo thành cây (nút gốc)
- Cây có thể định nghĩa một cách đệ quy



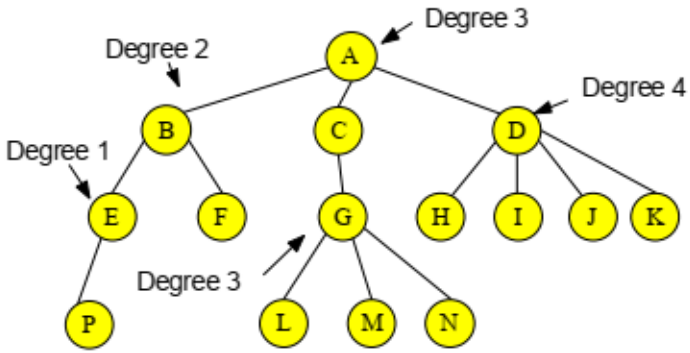
Ví dụ về cây

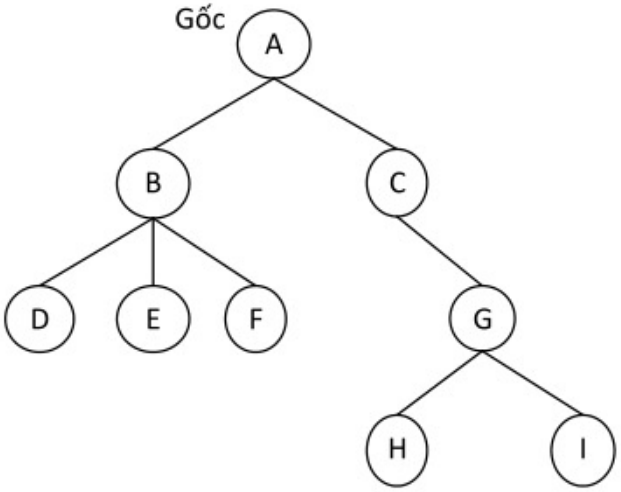
- Cấu trúc lưu trữ thư mục trong máy tính
- Cấu trúc mục lục của sách / tài liệu
- Cấu trúc các chức năng của một hệ thống thông tin



Các khái niệm cơ bản

- Gọi cây là T:
  - **Kích thước** cây (size) **S(T)**: là số nút có trên cây
  - **Cấp** của một nút A (degree) ***d(A)***: là số con mà nút này có
  - **Cấp** của cây ***d(T)***:
    - là cấp của nút có số con nhiều nhất trên cây





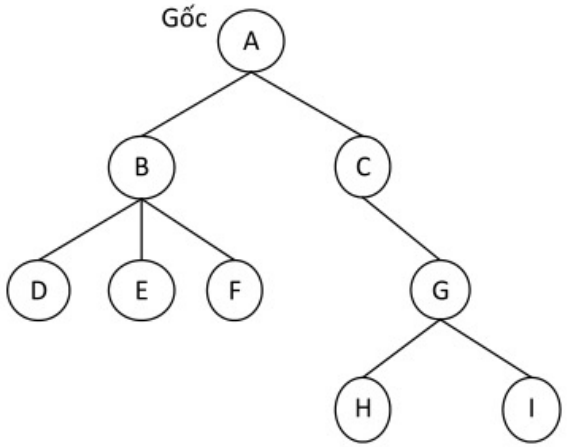
S(T)	S(T) = 9
d(A), d(G)	d(A) = d(G) = 2;
d(B)	d(B) = 3;
d(H), d(I)	d(H) = d(I) =0.
d(T), d(B)	d(T) = d(B) = 3;

Các khái niệm cơ bản

- Gọi cây là T:
  - Nút nhánh (branch node)
    - là nút có ít nhất một con
    - tên gọi khác: nút trong, nút không tận cùng
  - Nút lá (leaf node, terminal node)
    - là nút không có con nào
    - tên gọi khác: nút ngoài, nút tận cùng
  - **Mức** của một nút A (level) **L(A)**:
    - Nếu A là nút gốc thì L(A) = 1 (hoặc 0)
    - Nếu A là cha của B thì L(B) = L(A) +1
  - Anh chị em của một nút (sibling):
    - Các nút cùng mức có cùng nút cha

Các khái niệm cơ bản

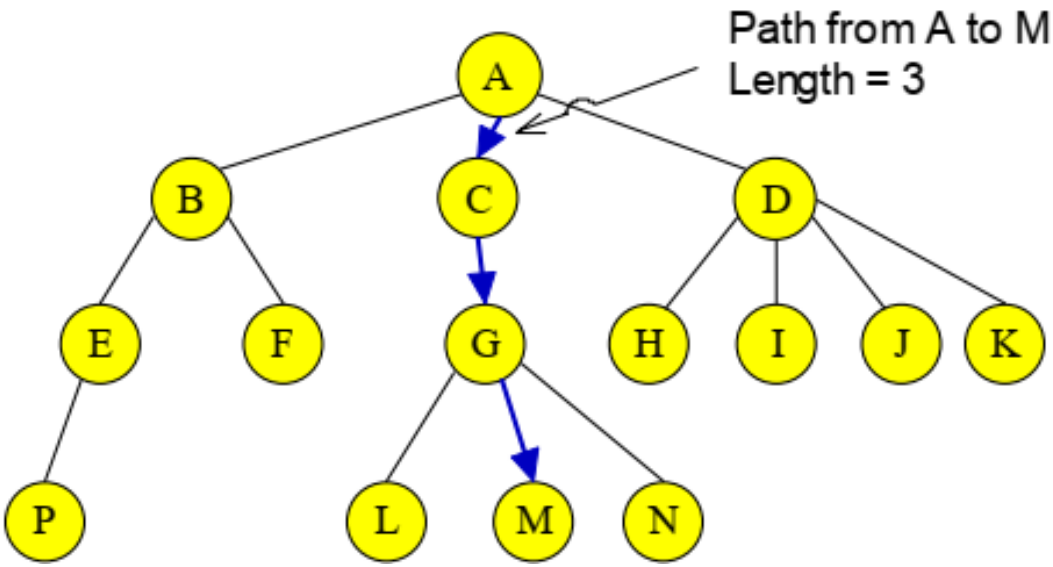
- Đường đi giữa nút gốc và một nút khác (path): ***p(A)***
  - là dãy các nhánh nối từ gốc đến nút đó
  - Hoặc dãy các nút $n_1, n_2, ..., n_k$ với n_i là nút cha của n_{i+1} (i = 1..k-1)



p(G) = {(A,C), (C,G)}
p(G) = { A, C, G }

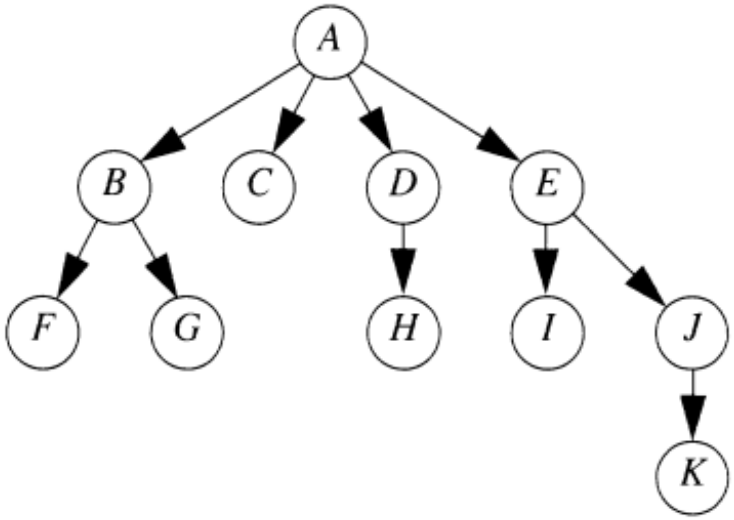
Các khái niệm cơ bản

- Độ dài đường đi (path length): ***PL(...)***
  - là tổng độ dài các nhánh tạo thành đường đi
  - Thực tế: quy ước độ dài của tất cả các nhánh trong một cây đều bằng 1 (đơn vị)



Các khái niệm cơ bản

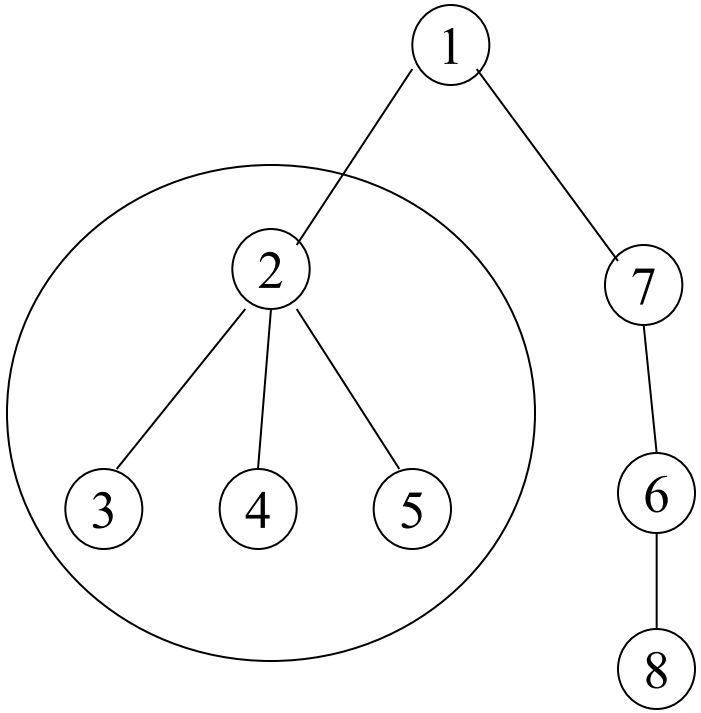
- Độ sâu của một nút (depth)
  - khoảng cách giữa nút và gốc của nó. Độ sâu nút gốc = 0
  - độ dài đường đi từ nút gốc tới nút
- Chiều cao của một nút (height)
  - Độ dài đường đi từ nút tới nút lá sâu nhất.
- Chiều cao của cây: ***h(T)***
  - = Chiều cao nút gốc



Node	Height	Depth
A	3	0
B	1	1
C	0	1
D	1	1
E	2	1
F	0	2
G	0	2
H	0	2
I	0	2
J	1	2
K	0	3

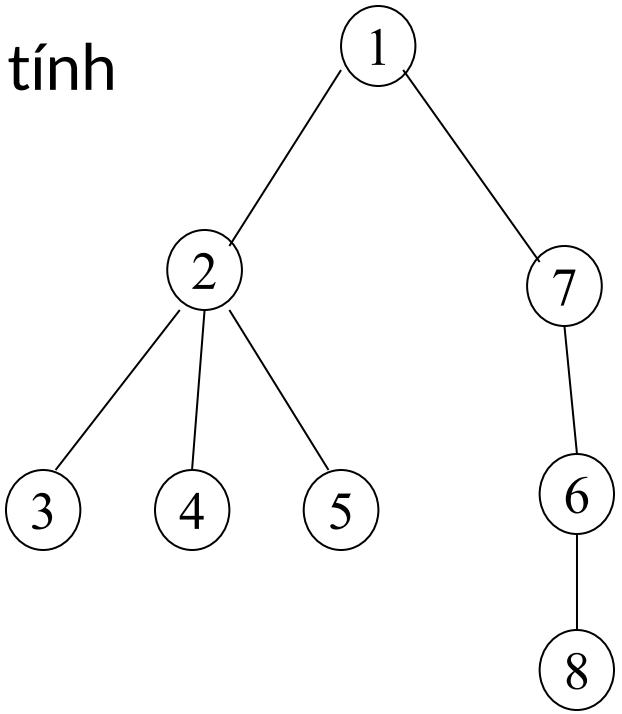
Các khái niệm cơ bản

- Cây con
- Cây được sắp (có thứ tự - ordered tree)
  - Có thứ tự (ordered tree): có trật tự tuyến tính giữa các nút con
  - Không có thứ tự (unordered tree): không có trật tự tuyến tính
- Rừng (forest)
  - là cấu trúc bao gồm nhiều cây
  - có thể có rừng rỗng



Các tính chất cơ bản

- Số nút của cây bằng số nhánh cộng 1
- Cây có tính chất đệ quy: cây được hình thành từ việc tổ chức các cây con theo cấu trúc phân cấp.
- Cây là một cấu trúc dữ liệu động, kích thước của nó (số nút) có thể thay đổi.
- Cấu trúc cây không còn cấu trúc tuyến tính nữa mà là cấu trúc phân cấp.
- Chỉ tồn tại duy nhất một đường đi từ gốc đến một nút khác.

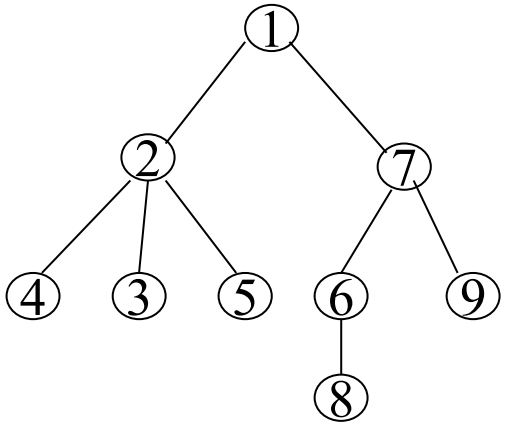


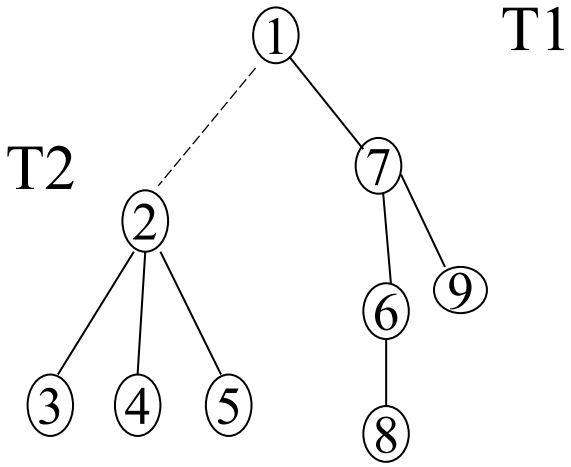
Các thao tác cơ bản trên cây

- Khởi tạo: chuẩn bị CTLT để lưu trữ cấu trúc cây
- Bổ sung một nút mới vào cây
  - Xác định vị trí cần bổ sung
  - Xác định quan hệ nút mới và nút tại vị trí bổ sung
- Lấy ra một nút
  - Xác định vị trí nút lấy ra
  - Cấu trúc lại cây sau loại bỏ
- Tìm cha mỗi nút
  - Parent(x)
- Tìm con bên trái ngoài cùng (con trưởng) của mỗi nút
  - EldestChild(x)
- Tìm anh/em liền kề mỗi nút
  - NextSibling(x)

Các thao tác cơ bản trên cây

- Duyệt cây: truy nhập vào tất cả các nút của cây, mỗi nút đúng một lần
  - Preorder: 1, 2, 4, 3, 5, 7, 6, 8, 9
- Tìm kiếm: một hay nhiều nút thoả một điều kiện nào đó.
- Ghép cây
  - MergeTree(T1,T2,x): x=1
  - x: nút ghép thuộc cây T1, gốc của T2 sẽ là con của x, có thể ghép trái, ghép phải
- Cắt cây
  - CutTree(T1,x,T2): x=2



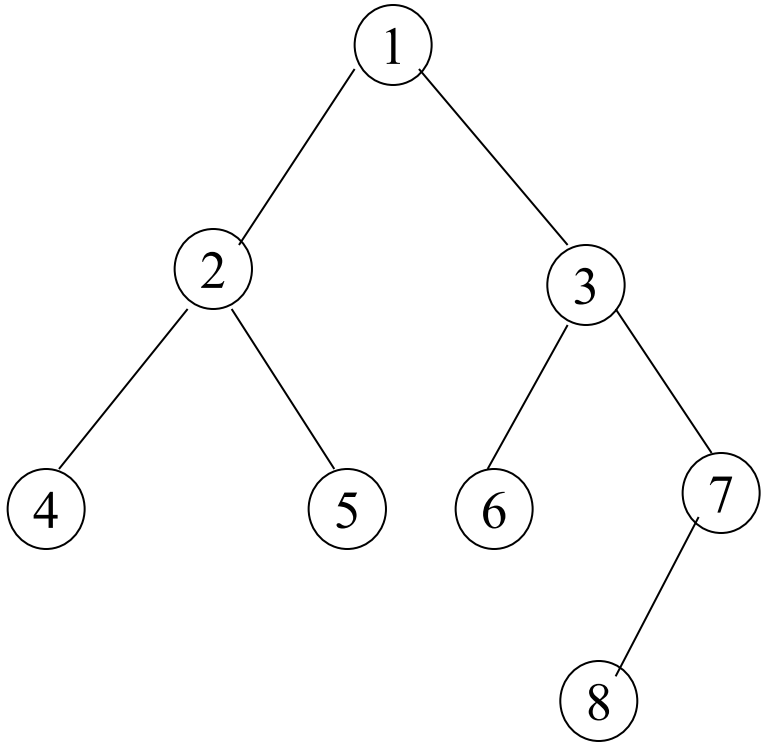


2. Cây nhị phân

Khái niệm cơ bản

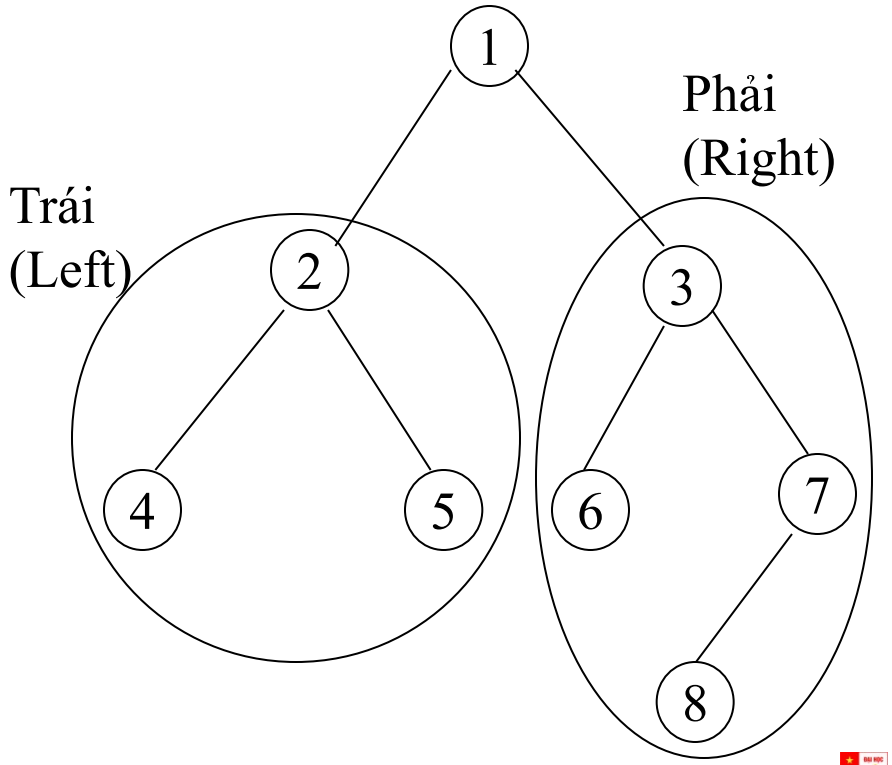
- Mô tả cây nhị phân (binary tree)
  - Cây có thứ tự và là cây cấp hai, tức là mỗi nút có ***tối đa*** 2 con.
  - Hai con của một nút được phân biệt thứ tự
  - Quy ước: nút trước gọi là nút con trái
    nút sau gọi là nút con phải

→ Cần phân biệt rõ ràng giữa con trái và con phải, nhất là khi nút chỉ có một con



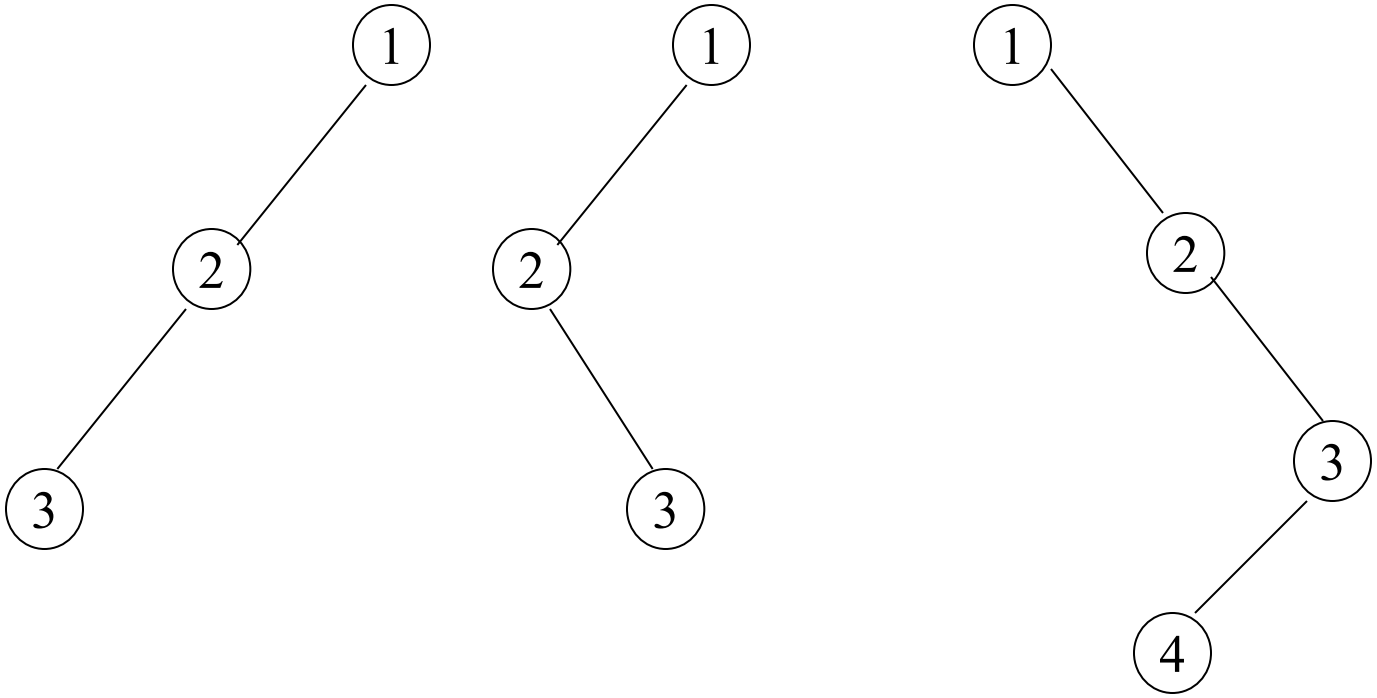
Khái niệm cơ bản

- Loại nút:
  - Nút có đủ hai con được gọi là nút kép: 2, 3
  - Nút chỉ có một con gọi là nút đơn: 7
  - Nút không có con được gọi là nút lá: 4, 5, 6, 8
- Cây con có gốc là nút con trái/phải được gọi là cây con trái/phải



Một số loại cây nhị phân đặc biệt

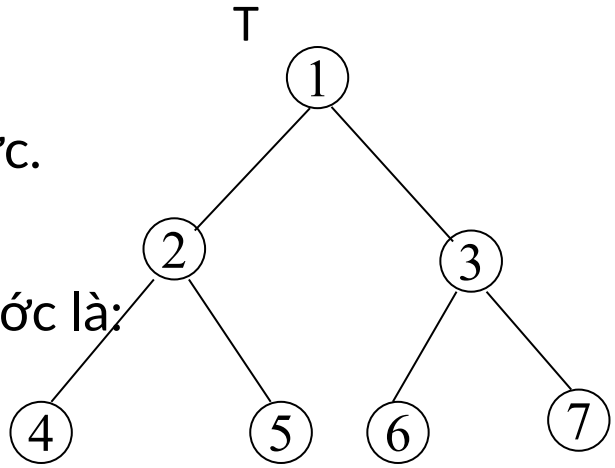
- Cây suy biến (degenerate binary tree)
  - Là cây nhị phân mà mỗi nút chỉ có tối đa một con
    - → suy biến về cấu trúc danh sách.
    - danh sách là dạng suy biến, trường hợp đặc biệt của cấu trúc cây, và nó vẫn giữa được tính chất đệ quy của cây



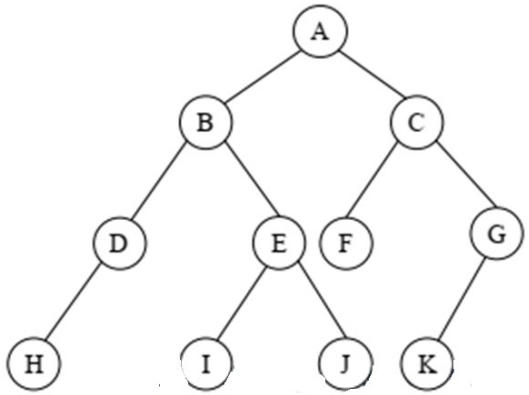
Một số loại cây nhị phân đặc biệt

- Cây đầy đủ (full binary tree)
  - Là cây nhị phân có số nút tối đa ở tất cả các mức.
    - Ở mức *i* sẽ có đúng 2^{i-1} nút.
    - Nếu cây nhị phân có chiều cao *h* thì kích thước là:

      $S(T) = 2^0+2^1+...+2^h = 2^{h+1}-1$;

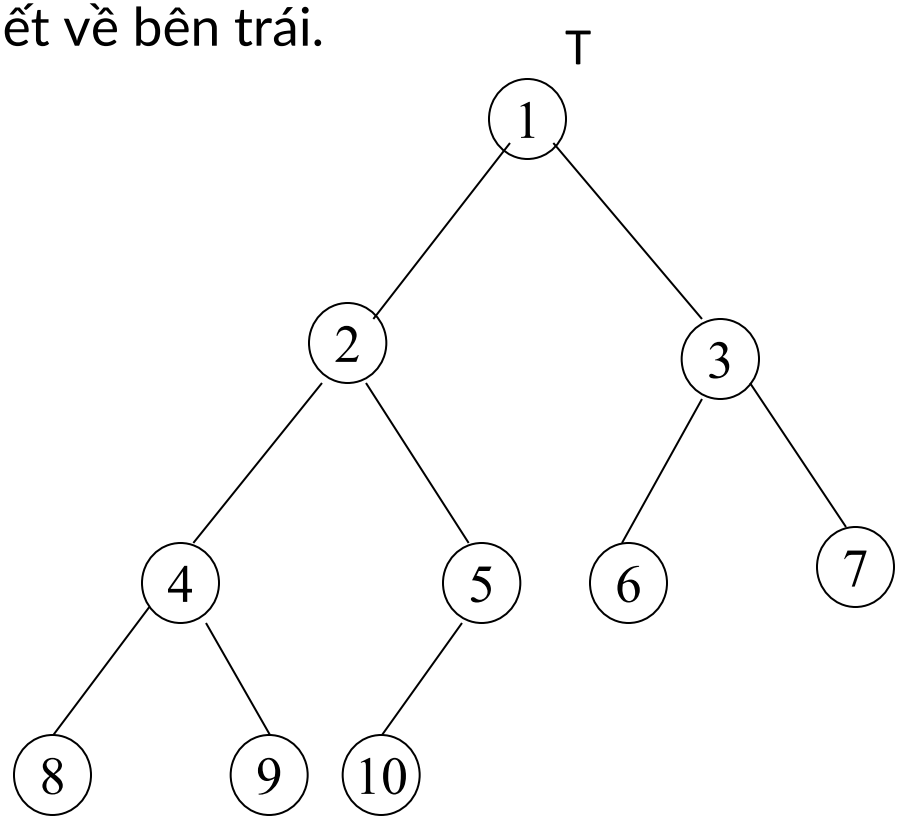


  - Cây nhị phân gần đầy đủ:
    ở mức cuối không có đầy đủ các nút



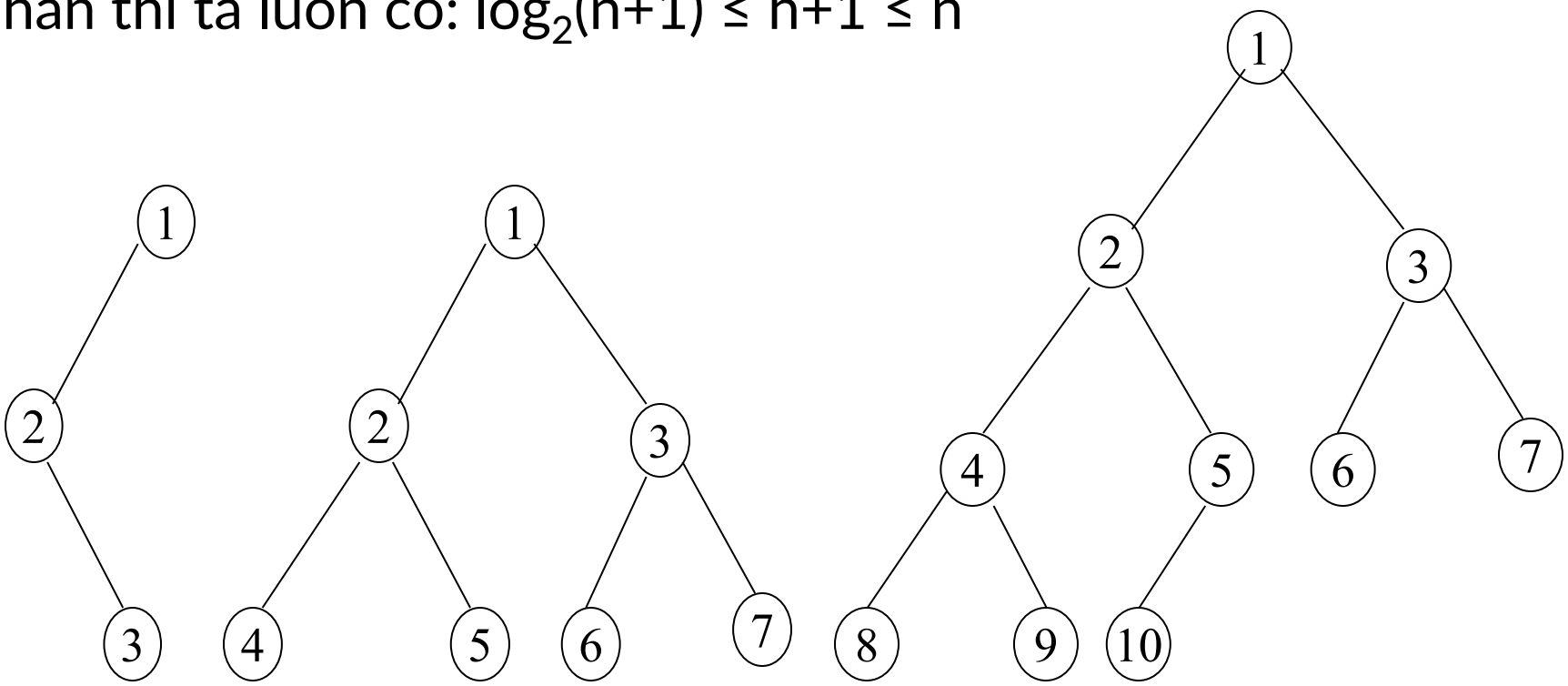
Một số loại cây nhị phân đặc biệt

- Cây hoàn chỉnh (complete binary tree)
  - Gọi *h* là chiều cao của cây T.
  - T là cây hoàn chỉnh nếu:
    - T là cây đầy đủ đến độ sâu *h-1*
    - Các nút có chiều cao *h* thì dồn hết về bên trái.



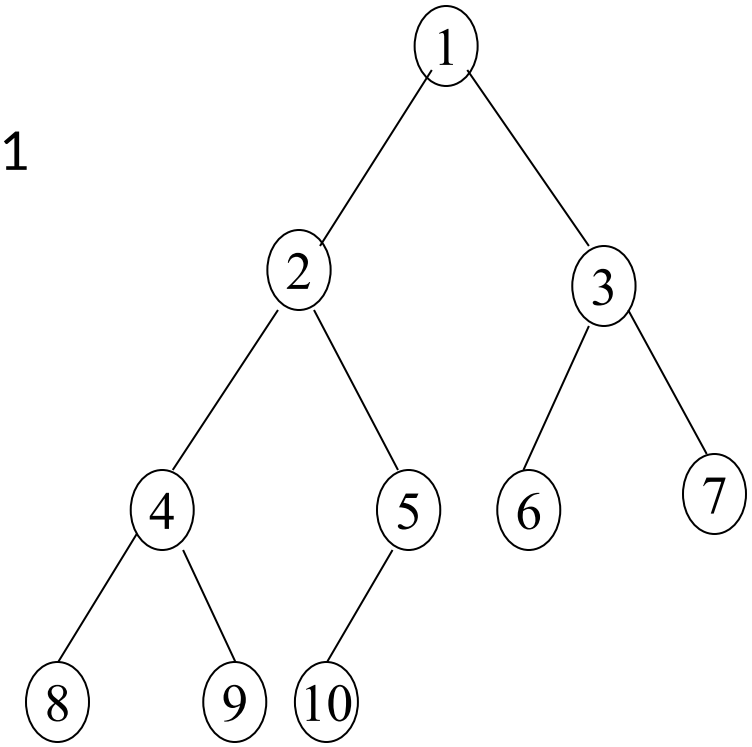
Một số tính chất của cây nhị phân

- TC1: Tính đệ quy: nếu ta chặt một nhánh bất kì của cây nhị phân thì ta sẽ thu được hai cây con cũng đều là cây nhị phân.
- TC2: Gọi *h* và *n* lần lượt là chiều cao và kích thước của cây nhị phân thì ta luôn có: $\log_2(n+1) \le h+1 \le n$



Một số tính chất của cây nhị phân

- TC3: Đối với cây hoàn chỉnh và cây đầy đủ: nếu ta đánh số các nút lần lượt từ 1..N như hình vẽ (N là kích thước của cây):
- Tại nút có số thứ tự i:
  - nếu 2i>N: nút i là nút lá
  - nếu 2i=N: nút i là nút đơn
  - nếu 2i<N: nút i là nút kép và có hai con trái, phải tương ứng là 2i và 2i+1
- Tại nút có số thứ tự j:
  - nếu j=1: nút j là nút gốc
  - nếu j>1: nút j/2 (nếu j chẵn) hoặc (j-1)/2 (nếu j lẻ) là nút cha của nút j.



Các thao tác cơ bản

- Khởi tạo
- Bổ sung một nút mới vào cây
- Lấy ra một nút
- Tìm cha mỗi nút: Parent(x)
- Tìm con bên trái của mỗi nút : LeftChild(x)
- Tìm con bên phải của mỗi nút : RightChild(x)
- Ghép cây
- Cắt cây
- **Duyệt cây (!)**

Phép duyệt cây

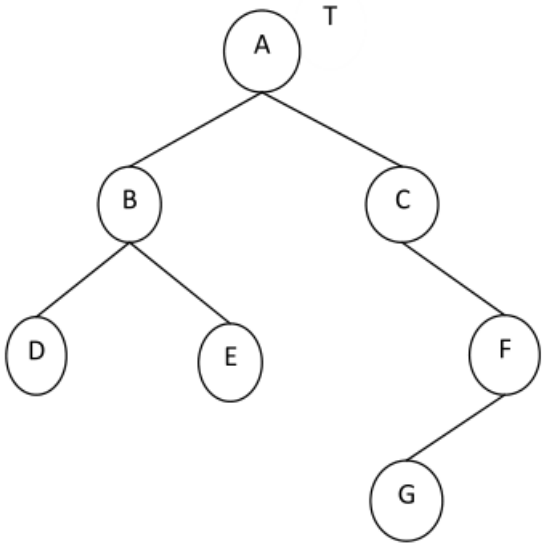
- Bao gồm 2 thao tác:
  - Phép thăm một nút (visit): là thao tác truy nhập vào một nút của cây để xử lý nút đó.
  - Phép duyệt (traversal): là phép thăm một cách hệ thống tất cả các nút của cây, mỗi nút đúng một lần.

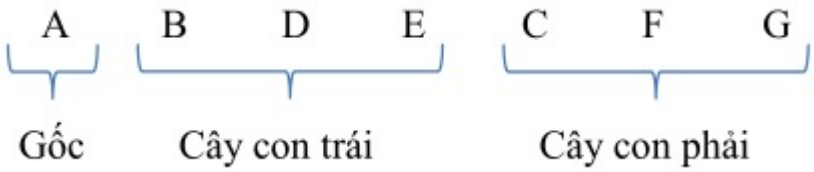
Giải thuật duyệt cây

- Hướng thứ nhất: dựa vào tính chất đệ quy của cây
  - Duyệt theo thứ tự trước (preorder traversal) còn gọi là duyệt cây theo chiều sâu
  - Duyệt theo thứ tự giữa (inorder traversal)
  - Duyệt theo thứ tự sau (postorder traversal)
- Hướng thứ hai: Chuyển cấu trúc cây về CT danh sách
  - Đưa bài toán duyệt cây về bài toán duyệt danh sách đơn giản hơn
  - Sử dụng một danh sách để lưu các nút của cây, sau đó sẽ duyệt danh sách
  - Tuỳ theo loại danh sách (hàng đợi, ngăn xếp,...) và thứ tự duyệt trong danh sách mà chúng ta có những cách duyệt cây khác nhau
  - Duyệt theo từng mức (tầng) - Duyệt theo chiều rộng

Giải thuật duyệt cây

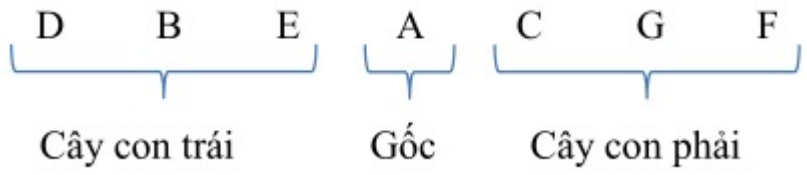
- Duyệt theo thứ tự trước (preorder traversal)
  - Trường hợp đệ quy: Nếu T khác rỗng, nó được duyệt theo các bước sau:
    - Thăm gốc: cây chỉ có một nút thì phép duyệt chính là phép thăm
    - Duyệt cây con trái theo thứ tự trước
    - Duyệt cây con phải theo thứ tự trước
  - Trường hợp điểm dừng: khi cây T rỗng

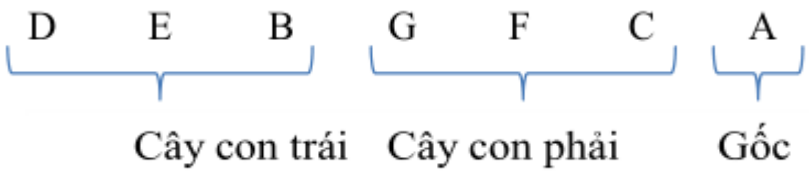


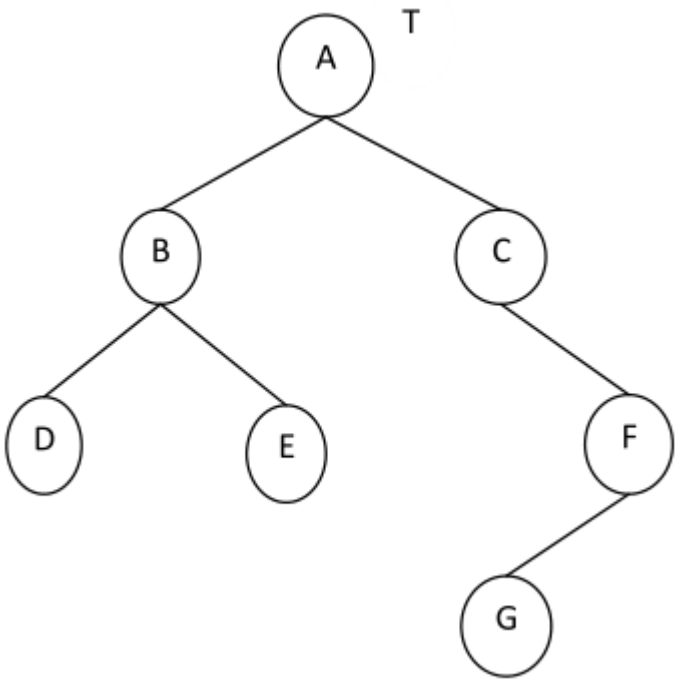


Giải thuật duyệt cây

- Duyệt theo thứ tự giữa (inorder traversal)
- Duyệt theo thứ tự sau (postorder traversal)





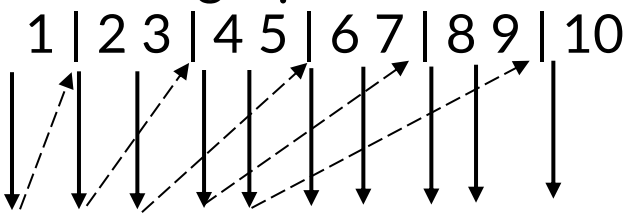


Giải thuật duyệt cây

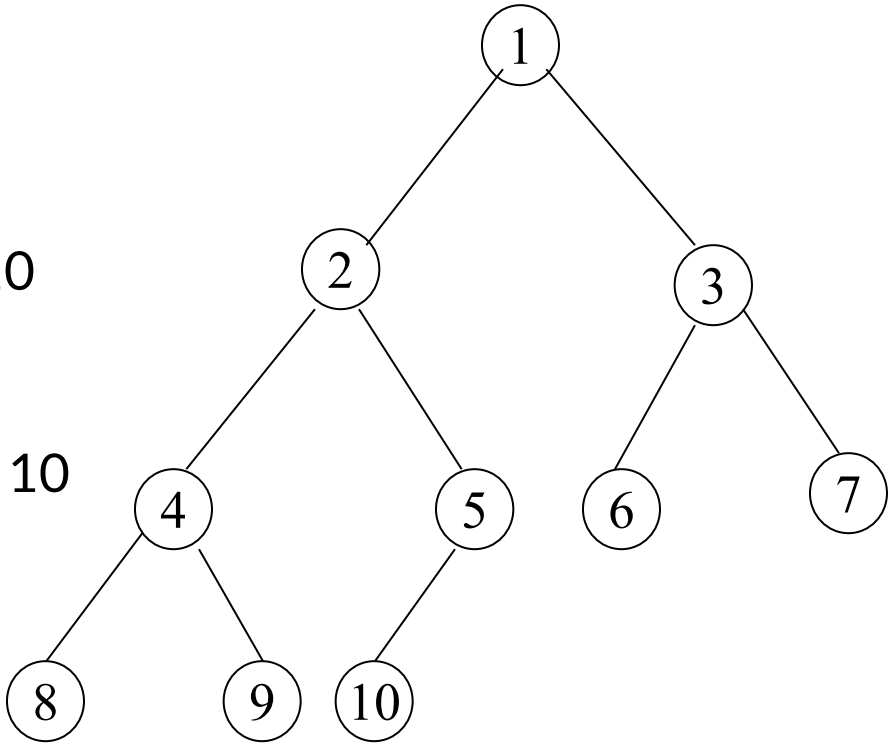
- Giải thuật duyệt cây theo từng mức (theo chiều rộng): giải thuật ***không*** đệ quy
  - Đưa bài toán duyệt cây về duyệt danh sách.
  - Để thứ tự duyệt là theo từng mức (từng tầng) và sử dụng một cấu trúc danh sách kiểu hàng đợi Q.

VD:

- Thứ tự đưa vào hàng đợi:

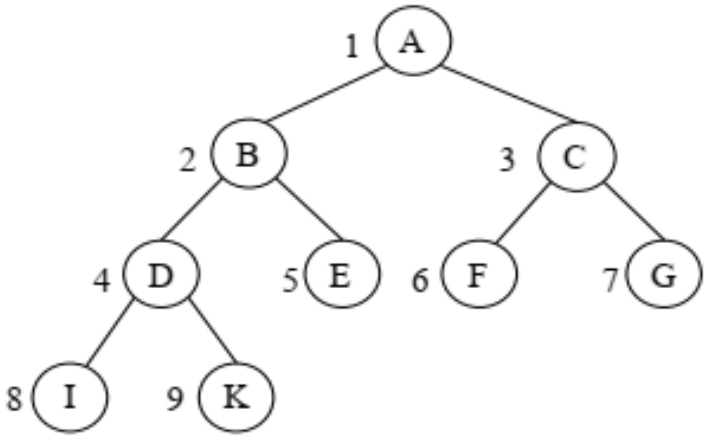


- Thứ tự thăm: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10



Cài đặt cây nhị phân

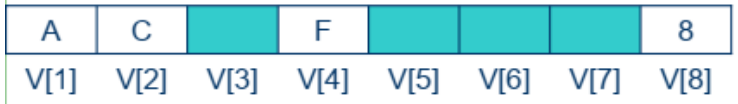
- Sử dụng CTLT tuần tự:
  - Đánh số các nút trên cây theo trình tự từ mức 1, hết mức này đến mức khác, từ trái sang phải
  - Lưu trữ trong vector lưu trữ V theo nguyên tắc phần tử V[i] sẽ lưu thông tin của nút được đánh số i

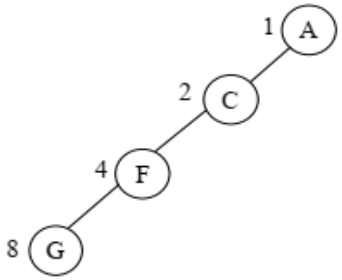


A	B	C	D	E	F	G	I	K
V[1]	V[2]	V[3]	V[4]	V[5]	V[6]	V[7]	V[8]	V[9]

Cài đặt cây nhị phân

- Sử dụng CTLT tuần tự:
  - Thường thích hợp với các cây NP đặc biệt: cây nhị phân gần đầy hoặc đầy đủ
    - Với cá





  - Đối với cây thông thường, phải bổ sung thêm các nút để nó trở thành đặc biệt (thành cây đầy đủ hoặc hoàn chỉnh). Do đó, làm cho việc cài đặt phức tạp và tốn bộ nhớ, nên ít khi được sử dụng.
  - Thường chỉ áp dụng khi hệ thống không cung cấp CTLT móc nối.

Cài đặt cây nhị phân

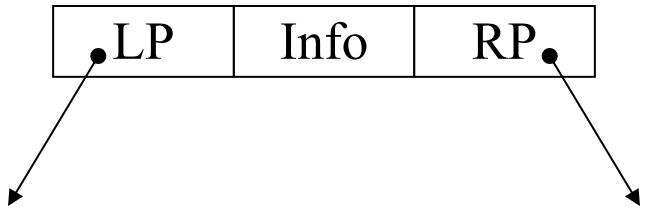
- Sử dụng CTLT móc nối:
  - Thường được dùng do tính linh hoạt của CTLT móc nối, rất thích hợp để cài đặt các CTDL phi tuyến.
- Nguyên tắc cài đặt sử dụng cấu trúc lưu trữ móc nối
  - Đối với cây nhị phân, cần lưu trữ hai thành phần:
  - Các phần tử (nút) của cấu trúc cây:
    - Dùng các nút của CTLT để lưu trữ các phần tử.
  - Các nhánh (quan hệ cha-con) giữa các cặp nút:
    - Sử dụng con trỏ để biểu diễn các nhánh.
    - Do mỗi nút có nhiều nhất hai con nên trong mỗi nút của CTLT sẽ có hai con trỏ để trỏ đến tối đa hai con này.

Cài đặt cây nhị phân

- Dùng cấu trúc móc nối
  - Cấu tạo mỗi nút:
    - Trường Info: chứa thông tin về một phần tử.
    - Trường LP, RP: hai con trỏ sẽ trỏ vào hai nút con trái và con phải.
    - Nếu không có con thì giá trị con trỏ tương ứng sẽ rỗng (NULL)

```
struct Node {
    type Info;
    Node *LP, *RP;
};

typedef Node* PNode;
```



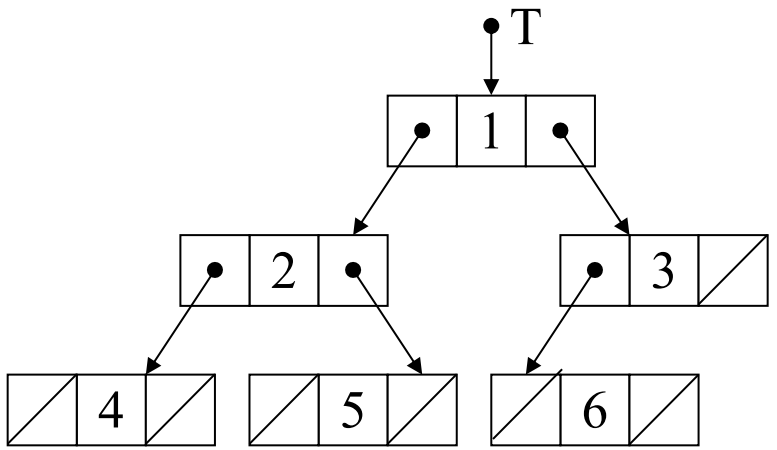
Cài đặt cây nhị phân

- Dùng cấu trúc móc nối
  - Cấu trúc cây:
    - Ít nhất một con trỏ T trỏ vào nút gốc: đại diện cho cây, điểm truy nhập vào các nút khác trong cây.
    - Khi kích thước cây là N thì sẽ có N+1 con trỏ rỗng.
    - chỉ có một chiều truy nhập từ nút cha xuống nút con mà không có chiều ngược lại

Cách 1: typedef PNode **BinaryTree**;

Cách 2: struct **BinaryTree** {

```
    PNode  T;    //con trỏ trỏ vào nút gốc
    int  n;      //kích thước cây
}
```

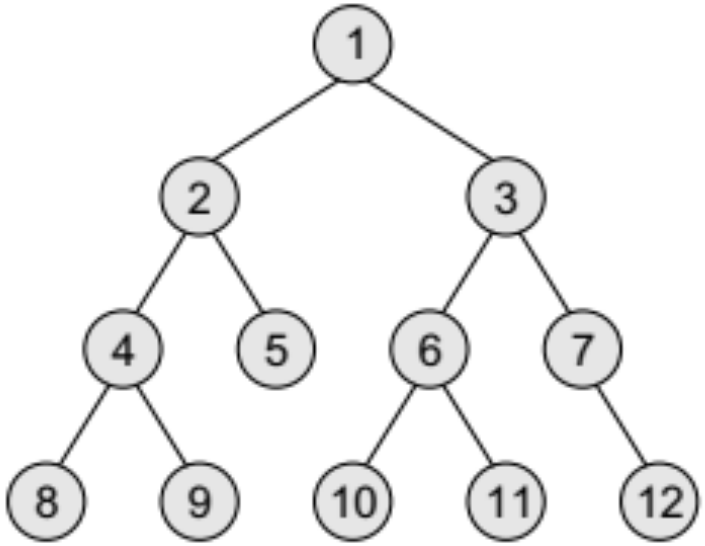


Lưu trữ cấu trúc cây trong bộ nhớ

ROOT
3

15
AVAIL

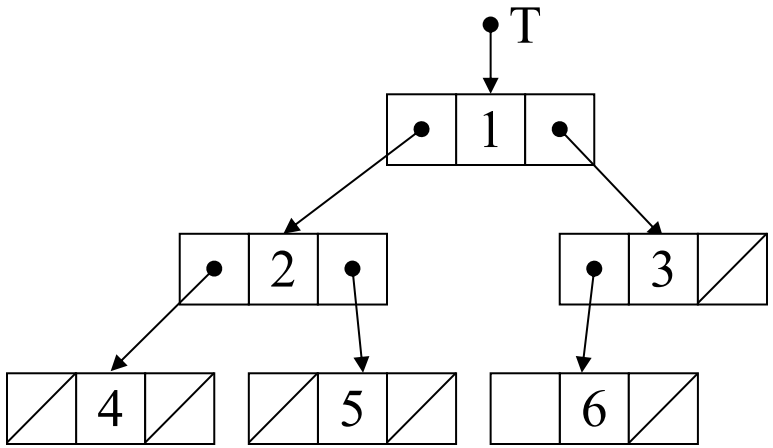
	LEFT	DATA	RIGHT
1	–1	8	–1
2	–1	10	–1
3	5	1	8
4			
5	9	2	14
6			
7			
8	20	3	11
9	1	4	12
10			
11	–1	7	18
12	–1	9	–1
13			
14	–1	5	–1
15			
16	–1	11	–1
17			
18	–1	12	–1
19			
20	2	6	16



Cài đặt cây nhị phân

- Dùng cấu trúc móc nối - Cài đặt một số thao tác cơ bản
  - Thao tác khởi tạo: tạo ra một cây rỗng.

```
void InitBT (BinaryTree T) {
    T = NULL;
}
```

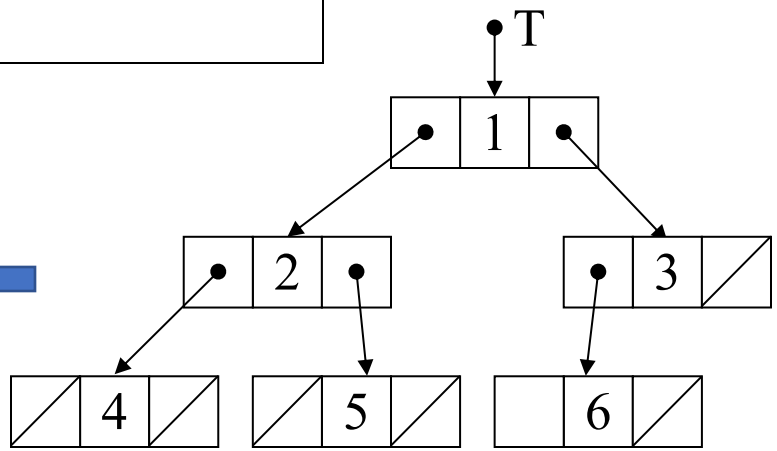


Cài đặt cây nhị phân

- Dùng cấu trúc móc nối - Cài đặt một số thao tác cơ bản
  - Duyệt cây theo thứ tự trước (preorder traversal)

```
//Duyệt cây theo thứ tự trước
void PreOrderTraversal (BinaryTree T) {
    if (T==NULL) return;
    Visit(T);
    PreOrderTraversal(T->LP);
    PreOrderTraversal(T->RP);
}
```

1, 2, 4, 5, 3, 6

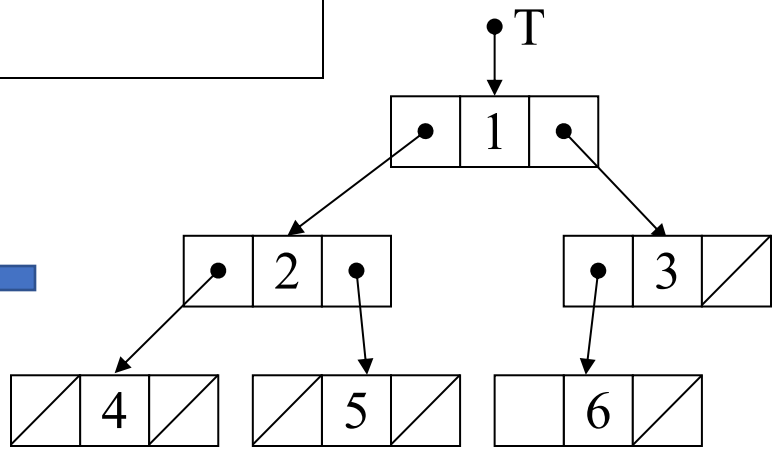


Cài đặt cây nhị phân

- Dùng cấu trúc móc nối - Cài đặt một số thao tác cơ bản
  - Duyệt cây theo thứ tự giữa (inorder traversal)

```
//Duyệt cây theo thứ tự giữa
void InOrderTraversal (BinaryTree T) {
    if (T==NULL) return;
    InOrderTraversal(T->LP);
    Visit(T);
    InOrderTraversal(T->RP);
}
```

4, 2, 5, 1, 6, 3

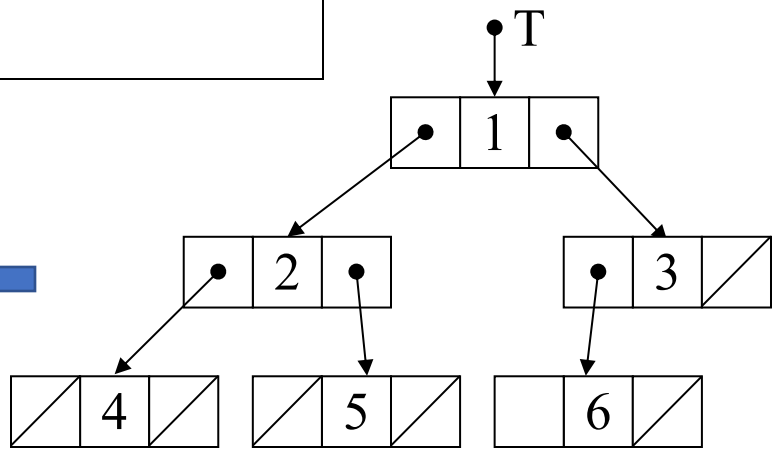


Cài đặt cây nhị phân

- Dùng cấu trúc móc nối - Cài đặt một số thao tác cơ bản
  - Duyệt cây theo thứ tự sau (postorder travelsal)

```
//Duyệt cây theo thứ tự sau
void   PostOrderTraversal (BinaryTree  T) {
        if  (T==NULL) return;
        PostOrderTraversal(T->LP);
        PostOrderTraversal(T->RP);
        Visit(T);
}
```

4, 5, 2, 6, 3, 1



Giải thuật duyệt cây

- Giải thuật duyệt cây theo từng mức (theo chiều rộng): giải thuật ***không*** đệ quy:
  - Đưa bài toán duyệt cây về duyệt danh sách.
  - Để thứ tự duyệt là theo từng mức (từng tầng) và sử dụng một cấu trúc danh sách kiểu hàng đợi Q.

```
Initialize (Q)                          // Khởi tạo: Q=∅.
InsertQ (T, Q);                         //Bổ sung nút gốc vào Q
While ( Q <> ∅ ) do
      P = DeleteQ (Q);                  //Lấy một nút ra khỏi Q để chuẩn bị thăm
      Visit (P);                        //Thăm P
      if <P có con trái LP> then InsertQ (LP, Q);
      if <P có con phải RP> then InsertQ (RP, Q);
End While ;
```

Cài đặt cây nhị phân

- Dùng cấu trúc móc nối - Cài đặt một số thao tác cơ bản
  - Tìm kiếm một nút trên cây có giá trị bằng K cho trước

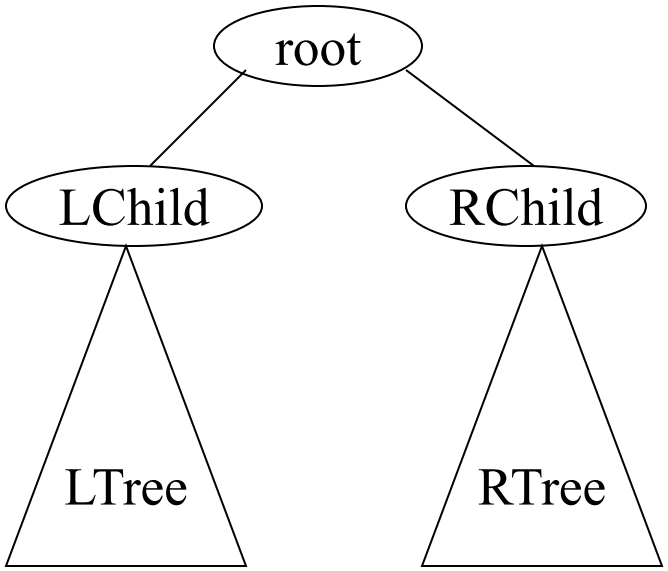
```
PNode Find (BinaryTree T, type K){
    if (T==NULL) return NULL;
    if (T->Info==K) return T;
    else {
       PNode found = Find(T->LP, K);
       if (found != NULL) return found;
       else return Find(T->RP, K);
    }
}
```

Xu hướng phải duyệt hết các nút trong cây, tốc độ tìm kiếm chậm.
Độ phức tạp là ~ O(N), với N là kích thước của cây

Cây nhị phân tìm kiếm

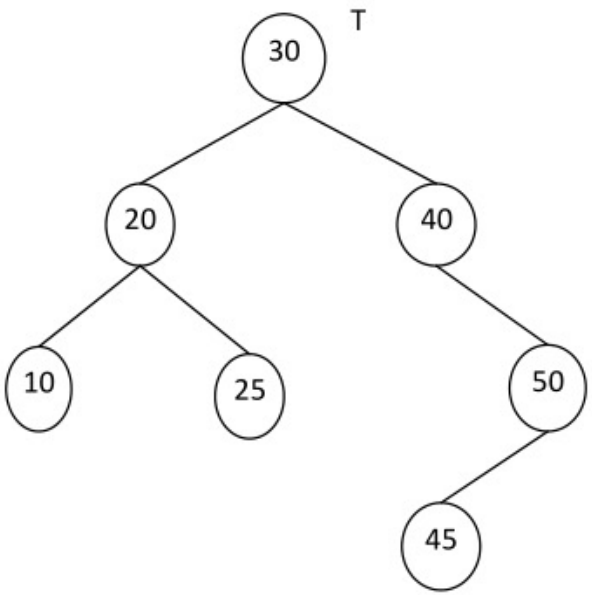
- Cây chuyên dụng, ứng dụng vào lưu trữ và tìm kiếm thông tin hiệu quả
  - Tại mọi nút A được coi là gốc,
    giá trị mọi nút thuộc cây con trái của A < giá trị tại nút A;
    giá trị mọi nút thuộc cây con phải của A > giá trị tại nút A.

BSearchTree = BinaryTree;
and **Key**(LChild) < **Key**(root);
and **Key**(root) < **Key**(RChild);
and LTree = **BSearchTree;**
and RTree = **BSearchTree;**



Cây nhị phân tìm kiếm

- Nếu duyệt cây theo thứ tự giữa thì được các nút được sắp xếp theo thứ tự tăng dần
- Giải thuật tìm kiếm nhị phân
  - so sánh K với nút gốc
  - nếu K < nút gốc, tìm K trong cây con trái của nút gốc.
  - nếu K > nút gốc, tìm K trong cây con phải của nút gốc.
  - Nếu cây cân bằng → $O(\log_2 N)$



Cây nhị phân tìm kiếm

- Khai báo cấu trúc nút

```
struct Node {
    keytype Key;      //Thường là kiểu có thứ tự, có thể so sánh được
    Node * LP, *RP;
};

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

Cây nhị phân tìm kiếm

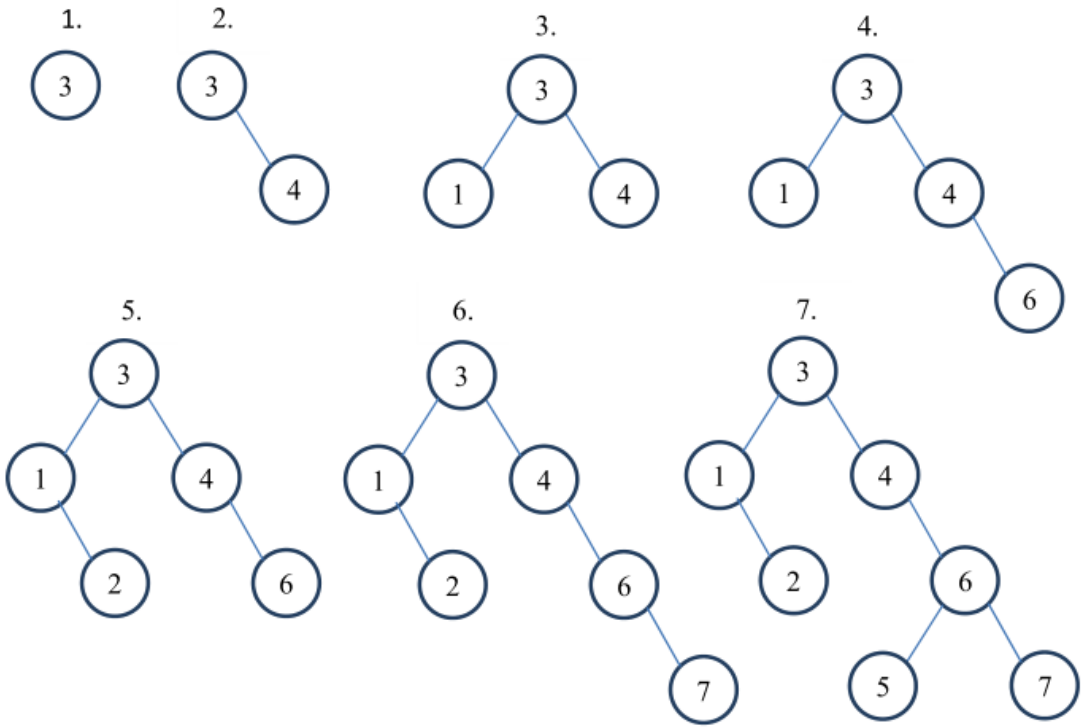
- Cài đặt các thao tác cơ bản:
  - Tìm kiếm một nút có giá trị *x* cho trước: Hàm sẽ trả về con trỏ trỏ vào nút tìm được nếu có, trái lại trả về con trỏ NULL

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

Cây nhị phân tìm kiếm

- Cài đặt các thao tác cơ bản:
  - Bổ sung một nút
    - cần đảm bảo sau khi bổ sung, cây T vẫn là cây nhị phân tìm kiếm.

(3, 4, 1, 6, 2, 7, 5)



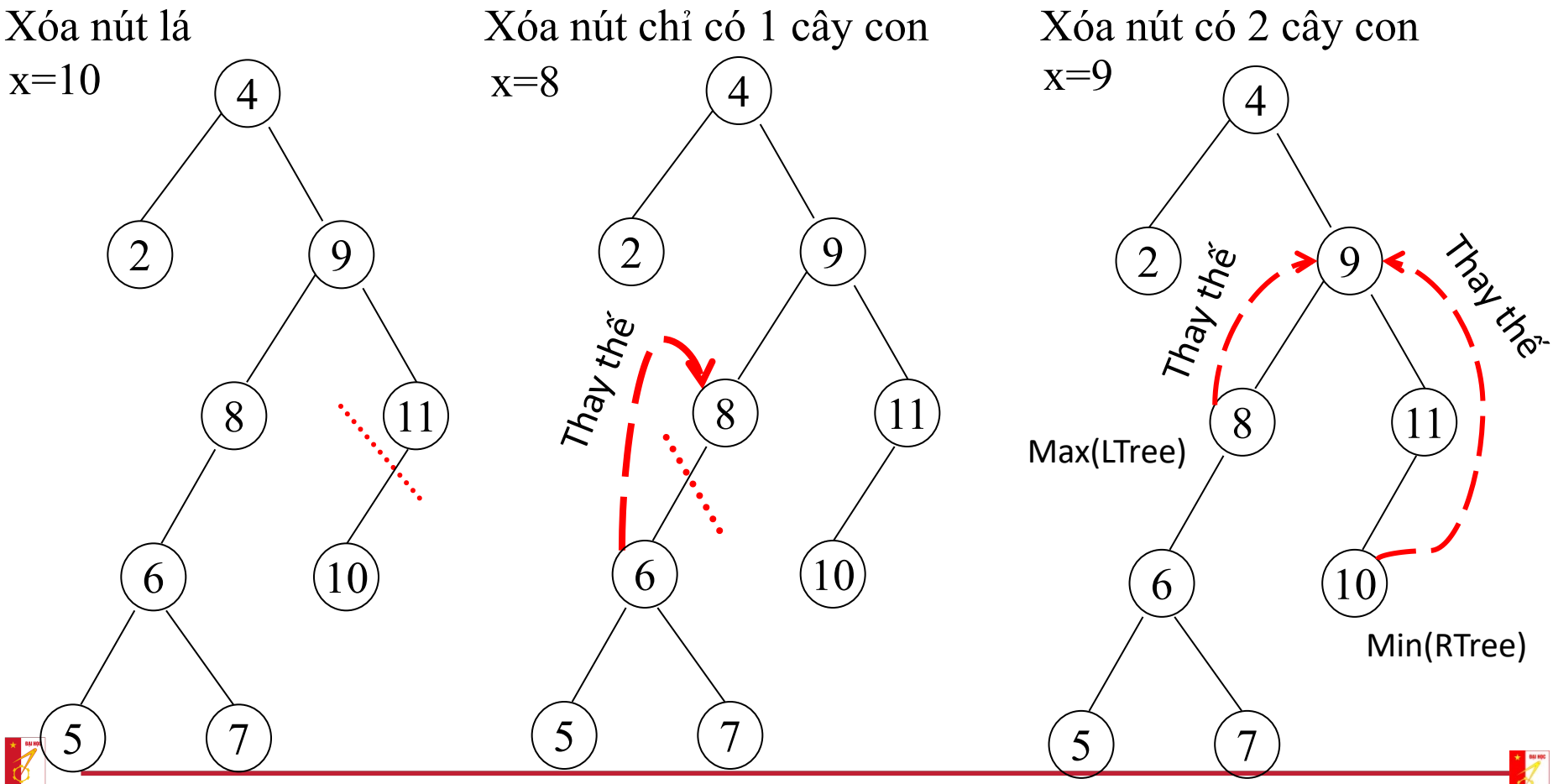
Cây nhị phân tìm kiếm

- Cài đặt các thao tác cơ bản:
  - Bổ sung một nút
    - cần đảm bảo sau khi bổ sung, cây T vẫn là cây nhị phân tìm kiếm.

```
void InsertT (BSearchTree & Root, keytype x){
    PNode Q;
    if (Root==NULL) {
        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);
    }
}
```

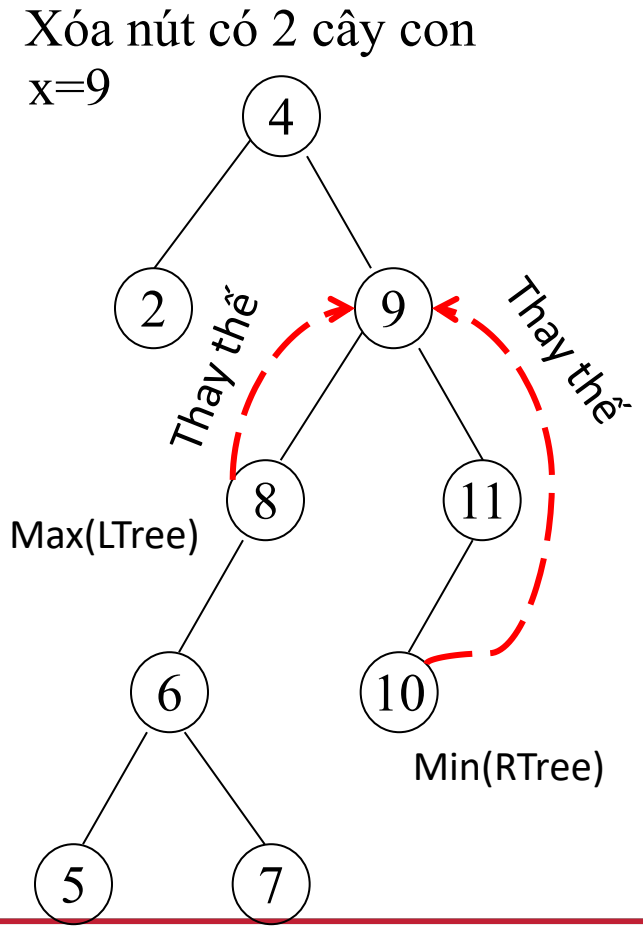
Cây nhị phân tìm kiếm

- Cài đặt các thao tác cơ bản:
  - Lấy ra (loại bỏ) một nút: cần sắp xếp lại để đảm bảo tính chất cây tìm kiếm NP



Cây nhị phân tìm kiếm

- Cài đặt các thao tác cơ bản:
  - Lấy ra (loại bỏ) một nút: cần sắp xếp lại để đảm bảo tính chất cây tìm kiếm NP



- Max(LTree) phần tử bên phải ngoài cùng của cây con Ltree:
  - *luôn đi theo bên phải của cây LTree đến khi không được nữa Q->RP=NULL thì Q là nút ngoài cùng bên phải*
- Min(RTree) phần tử bên trái ngoài cùng của cây con Rtree:
  - *luôn đi theo bên trái của cây RTree đến khi không được nữa Q->LP=NULL thì Q là nút ngoài cùng bên trái*

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);
      }
}           //Tìm đến nút cần loại bỏ, gọi hàm Xóa nút
```

```
void  DelNode (PNode & P) {  //Xóa giá trị ở nút P và sắp lại cây
      PNode Q, R;
      if (P->LP == NULL) {              // nút P chỉ có cây con phải
            Q = P;                      // lưu nút sẽ xóa là Q
            P = P->RP;
      } else if (P->RP = NULL)  {       // nút P chỉ có cây con trái
            Q = P;
            P = P->LP;
      } else
```

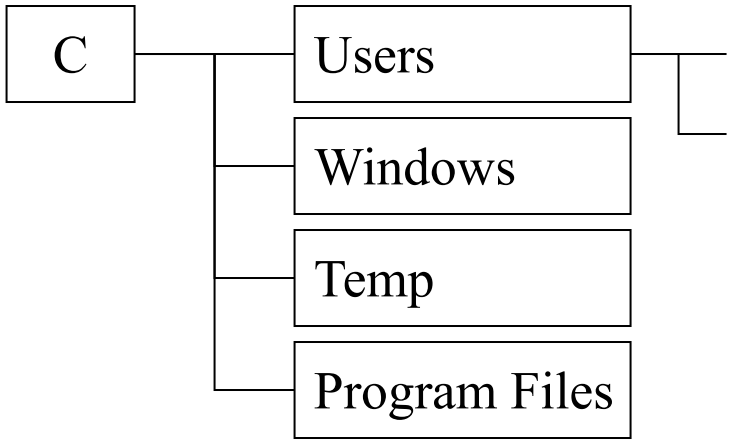
```
        // else                                    //Xóa nút có 2 cây con
        {
            Q = P->LP;
            if (Q->RP == NULL) {                   //Q không có cây con phải,
                P->Key = Q->Key;                   // Q là Max(Ltree)
                P->LP = Q->LP;
            } else {
                do {
                  R = Q;                           // R lưu parent của Q
                  Q = Q->RP;
                } while (Q->RP != NULL);
                P->Key = Q->Key;                   //Lấy giá trị ở Q đưa lên P
                R->RP = Q->LP;                     //Chuyển con của Q lên vị trí Q
            }
        }
        delete Q;                                  //Xoá Q
}
```

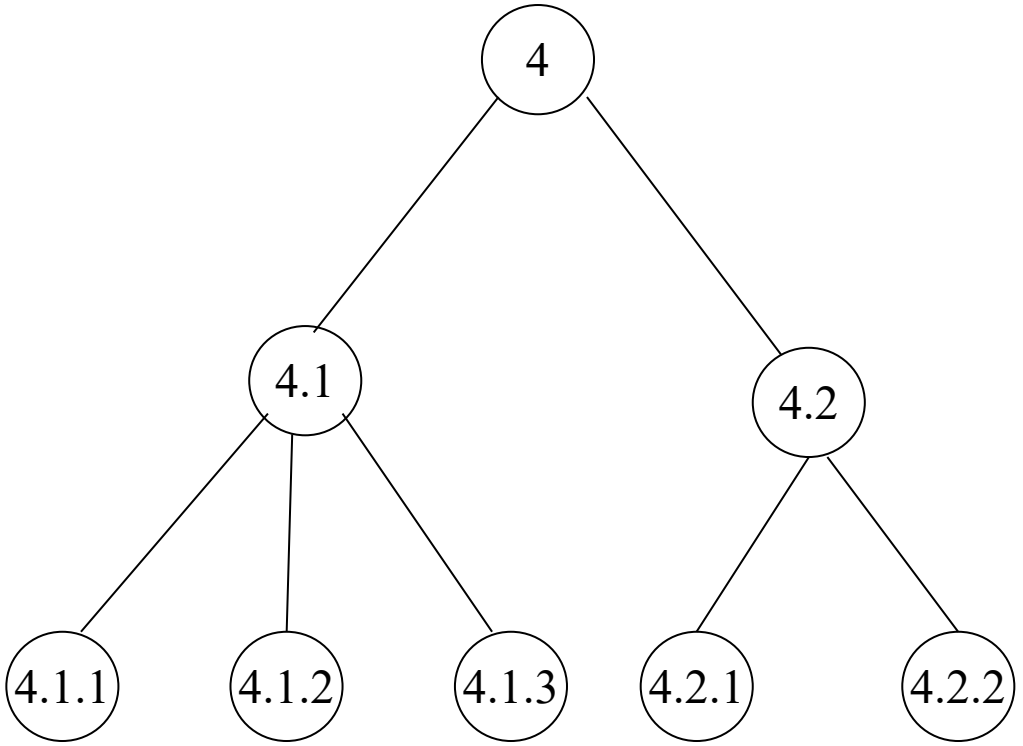
[Tìm Max(LTree): Luôn đi theo bên phải của Q, khi nào không đi đươc nữa thì đó là nút ngoài cùng bên phải]

3. Cây tổng quát

3. Cây tổng quát

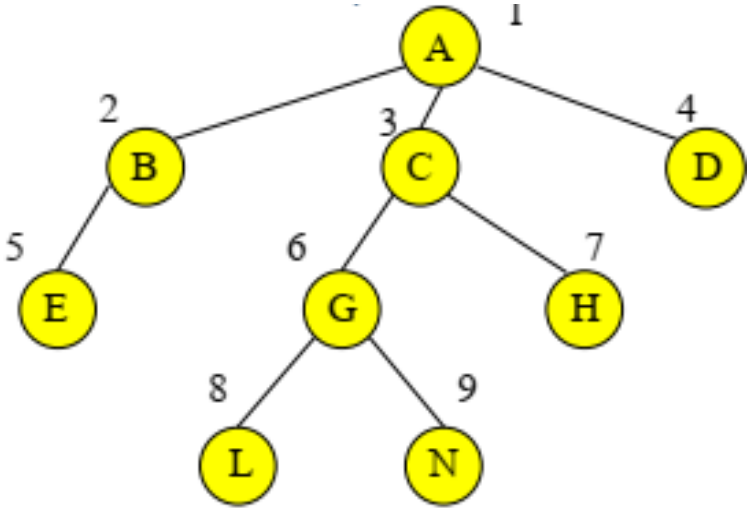
- Giới thiệu
  - Cây tổng quát là cây mà mỗi nút có thể có nhiều hơn hai con.
  - Việc cài đặt càng phức tạp, nhất là những cây mà số con tối đa thường không cố định





Biểu diễn cây tổng quát

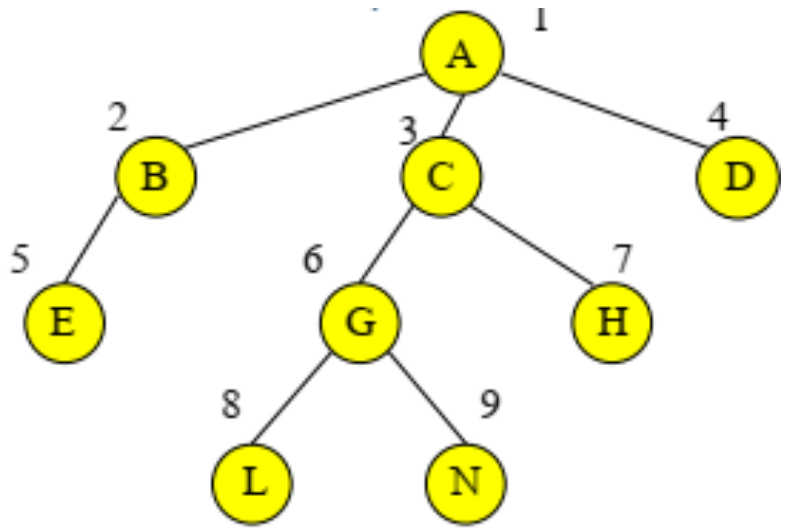
- Dựa trên tham chiếu đến nút cha:
  - Cây T có các nút được đánh số từ 1 đến n
  - Cây T được biểu diễn bằng một danh sách tuyến tính trong đó nút thứ i sẽ chứa một thành phần tham chiếu đến cha của nó
  - Nếu dùng mảng, A[i] = j nếu j là cha của nút i ; nếu i là gốc thì A[i] = 0;

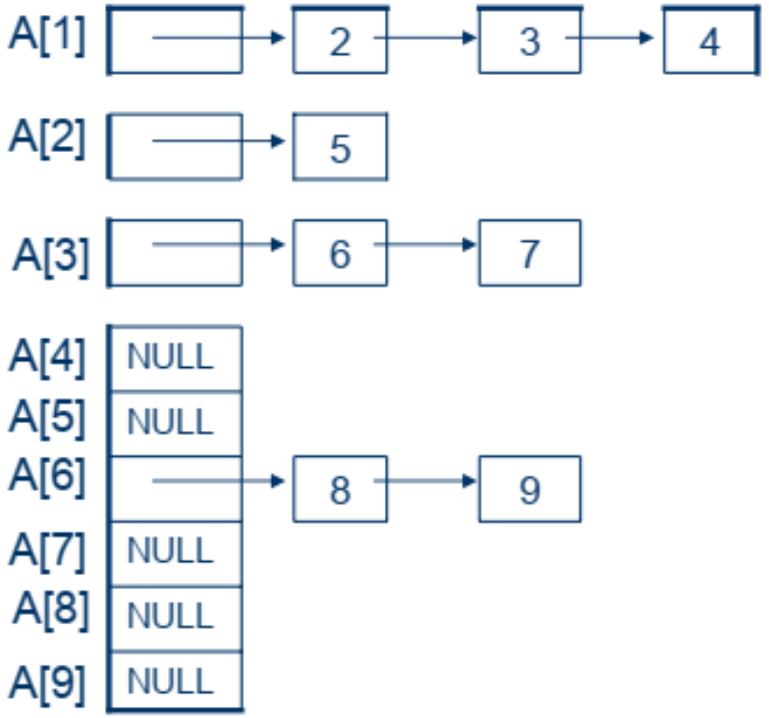


0	1	1	1	2	3	3	6	6
A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

Biểu diễn cây tổng quát

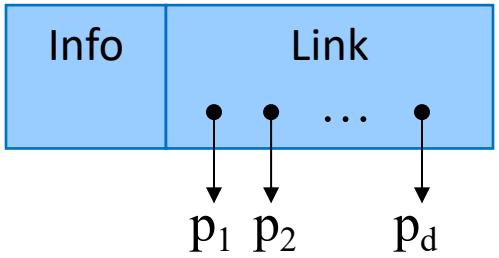
- Dựa trên danh sách các nút con:
  - 1 nút trong cây có một danh sách các nút con
  - Danh sách các nút con thường là danh sách móc nối
  - Trong trường hợp sử dụng danh sách móc nối, các nút đầu danh sách được lưu trong một mảng





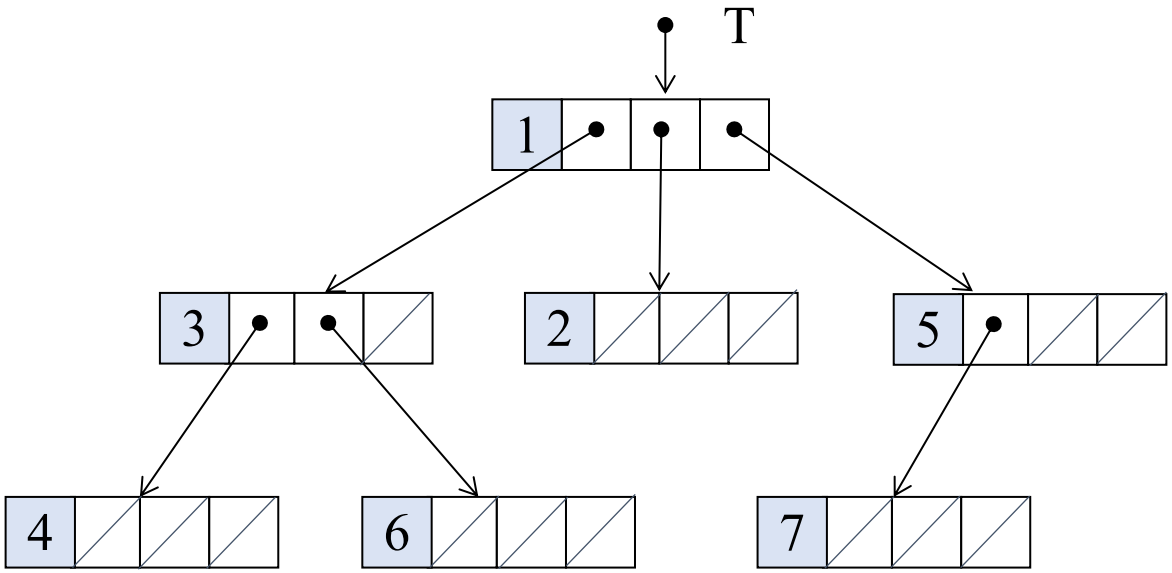
Biểu diễn cây tổng quát

- Dựa trên danh sách liên kết
  - 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 trỏ)



Biểu diễn cây tổng quát

- Dựa trên danh sách liên kết
  - 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.

Cài đặt cây tổng quát

- Cài đặt gián tiếp qua cây nhị phân

  Ý tưởng:

  - Tận dụng các kết quả cài đặt từ cây NP, đồng thời khắc phục các hạn chế của cách cài đặt trực tiếp.
  - Chuyển cây tổng quát về cây NP.
    - Xác định quy tắc chuyển đổi hai chiều từ cây tổng quát sang cây NP và ngược lại.
    - Coi cây tổng quát là cây có thứ tự. Nếu cây tổng quát không có thứ tự thì ta đưa thêm vào một thứ tự trong cây đó.
    - 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 đó, biểu diễn được một cây tổng quát dưới dạng một cây nhị phân gọi là cây nhị phân tương đương (equivalent binary tree)

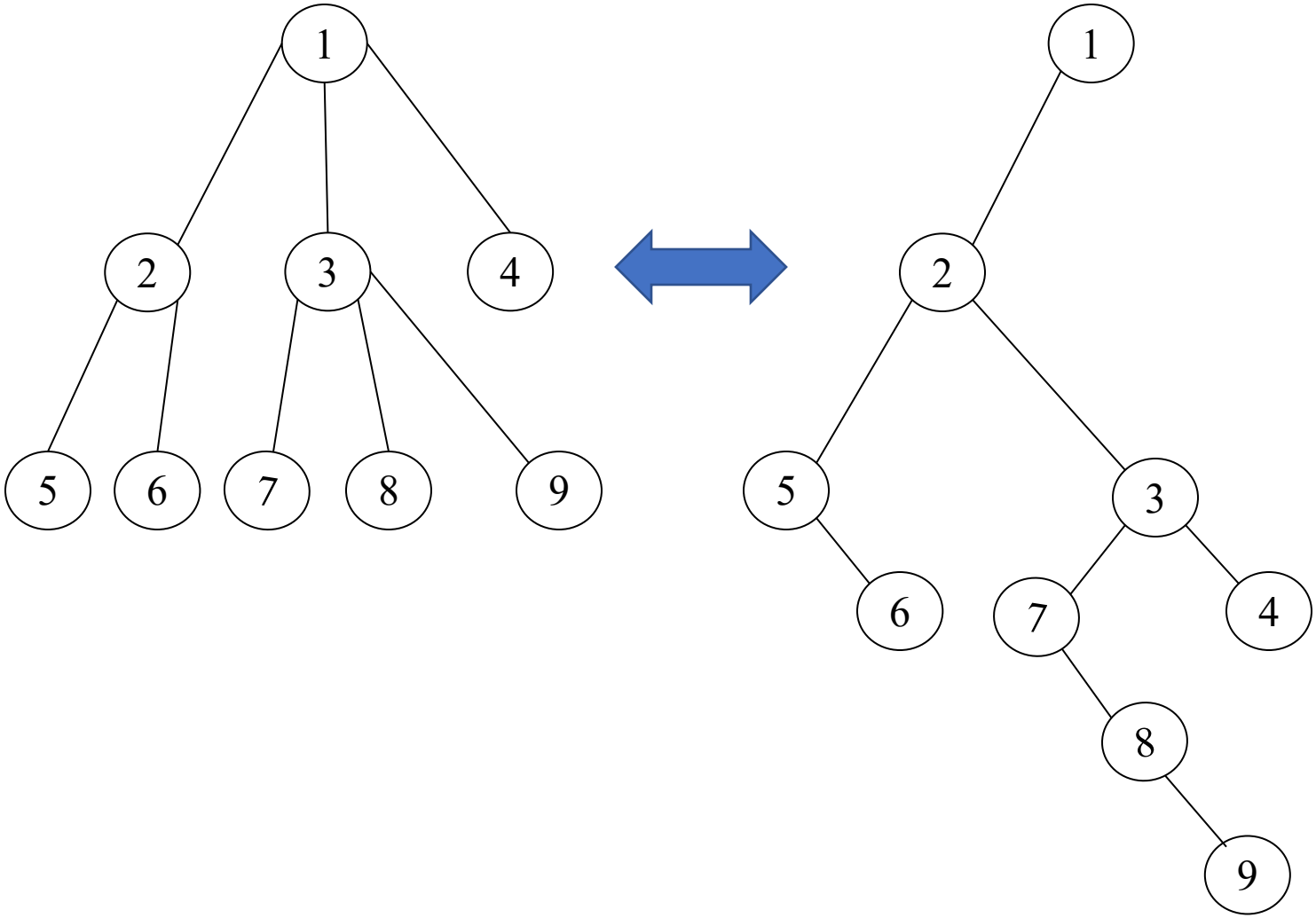
Cài đặt cây tổng quát

- Cài đặt gián tiếp qua cây nhị phân
  - Quy tắc:
    - Sắp xếp lại cây tổng quát (nếu cần)
    - Chuyển nút con trái nhất (con cả) của một nút thành nút con trái của nó
    - Chuyển nút em kế cận của một nút thành nút con phải của nút đó.
    - Quy cách của 1 nút trên cây nhị phân tương đương sẽ như sau

  - Nhận xét:
    - 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

Cài đặt cây tổng quát

- Ví dụ



Cài đặt cây tổng quát

- Cài đặt gián tiếp qua cây nhị phân

  Nhận xét:

  - Cho phép cài đặt một cây bất kì, có thể có cấp rất lớn và hiệu quả về bộ nhớ.
  - Cần thêm quá trình chuyển đổi ngữ nghĩa thuận và nghịch giữa cây tổng quát và cây nhị phân.
    - Ví dụ bổ sung một nút là con cả của một nút trong cây tổng quát sẽ thành bổ sung một nút thành nút con trái trong cây NP => giảm tốc độ thực hiện các thao tác cơ bản như bổ sung, tìm kiếm trên cây.

4. Ứng dụng của cấu trúc cây

4. Ứng dụng của cấu trúc cây

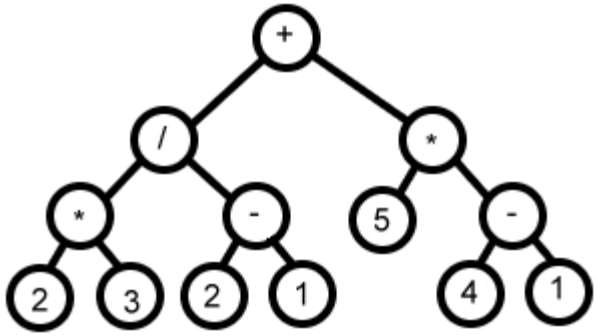
- Phân loại
- Quyết định
- Lưu trữ, sắp xếp, tìm kiếm dữ liệu
- Phương pháp tính toán

4. Ứng dụng của cấu trúc cây

- Một số ứng dụng cây NP
  - Cây nhị phân tìm kiếm: ứng dụng vào lưu trữ và tìm kiếm thông tin một các hiệu quả nhất (tốn ít bộ nhớ và thời gian chạy nhanh nhất)
  - Cây biểu thức: giúp thực hiện hàng loạt các thao tác tính toán cơ bản cũng như phức tạp (do người dùng định nghĩa)
  - Cây cân bằng
- Một số ứng dụng khác
  - Cây biểu diễn các tập hợp
  - Cây hỗ trợ ra quyết định: decision tree

Cây biểu thức

- Cây biểu thức
  - Biểu diễn biểu thức
    - Biểu thức = Biểu thức <Phép toán> Biểu thức
    - Đặc biệt: Biểu thức = const
    - Phép toán: +, / , *, -, exp, !, ...
  - Dùng cây nhị phân
    - Gốc: phép toán
    - Lá: toán hạng
  - Nhận xét:
    - Duyệt cây theo Preorder => biểu thức prefix
    - Duyệt cây theo Postorder => biểu thức postfix
    - Duyệt cây theo Inorder => biểu thức infix

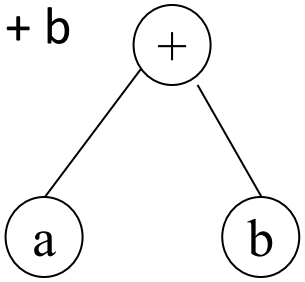


Expression tree for 2*3/(2-1)+5*(4-1)

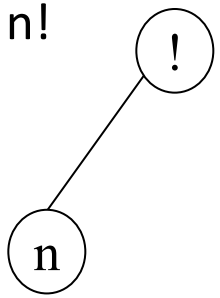
Cây biểu thức

- Cây biểu thức (Ví dụ minh họa)
  - Ví dụ: a+b, n!, exp(x), a*(d+b)

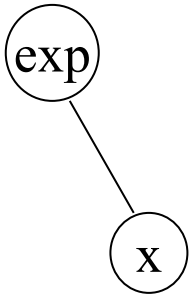
a + b



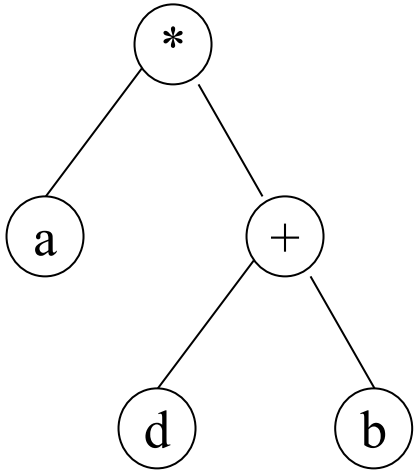
n!



e^x

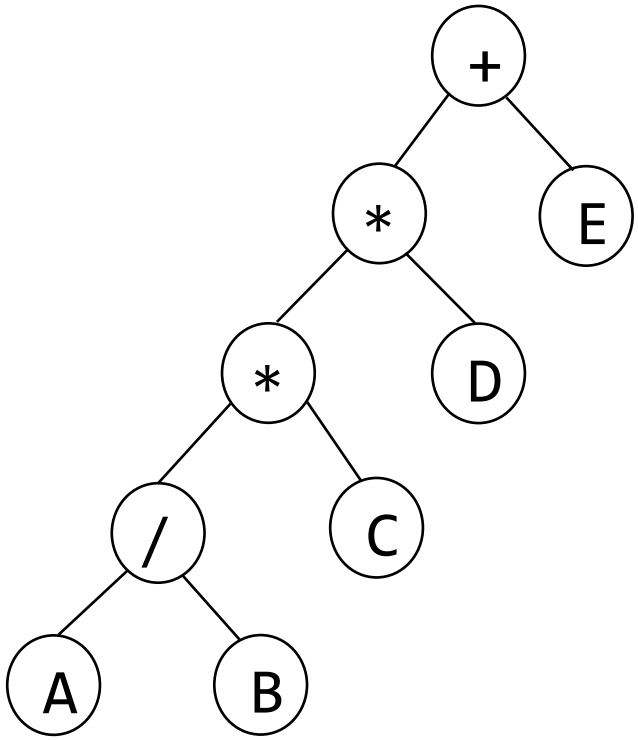


a * (d + b)



Cây biểu thức

- Cây biểu thức (Ví dụ minh họa)



inorder traversal

A / B * C * D + E

infix expression

preorder traversal

+ * * / A B C D E

prefix expression

postorder traversal

A B / C * D * E +

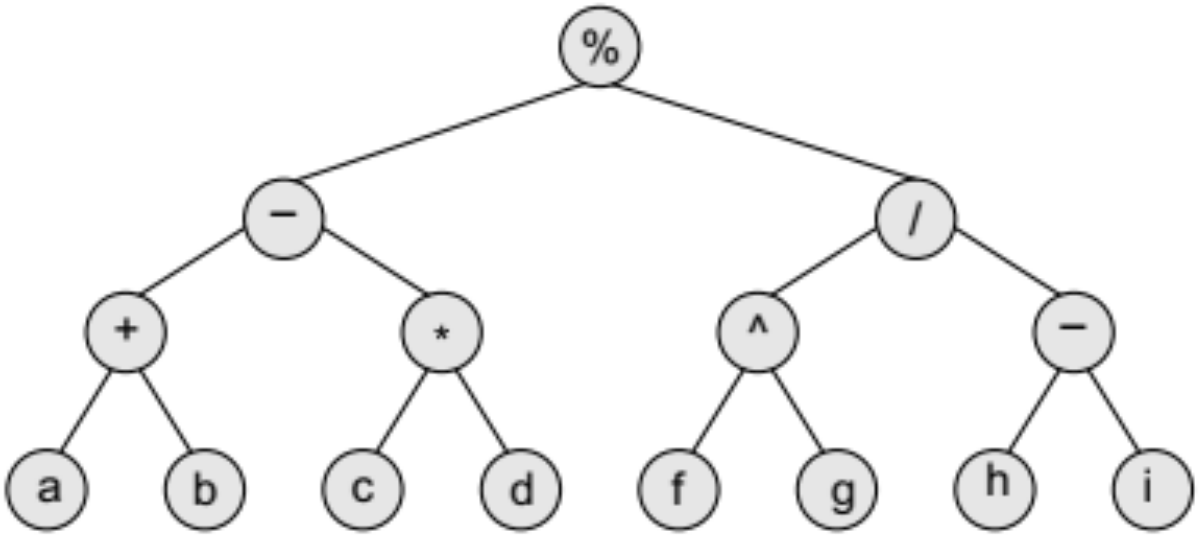
postfix expression

level order traversal

+ * E * D / C A B

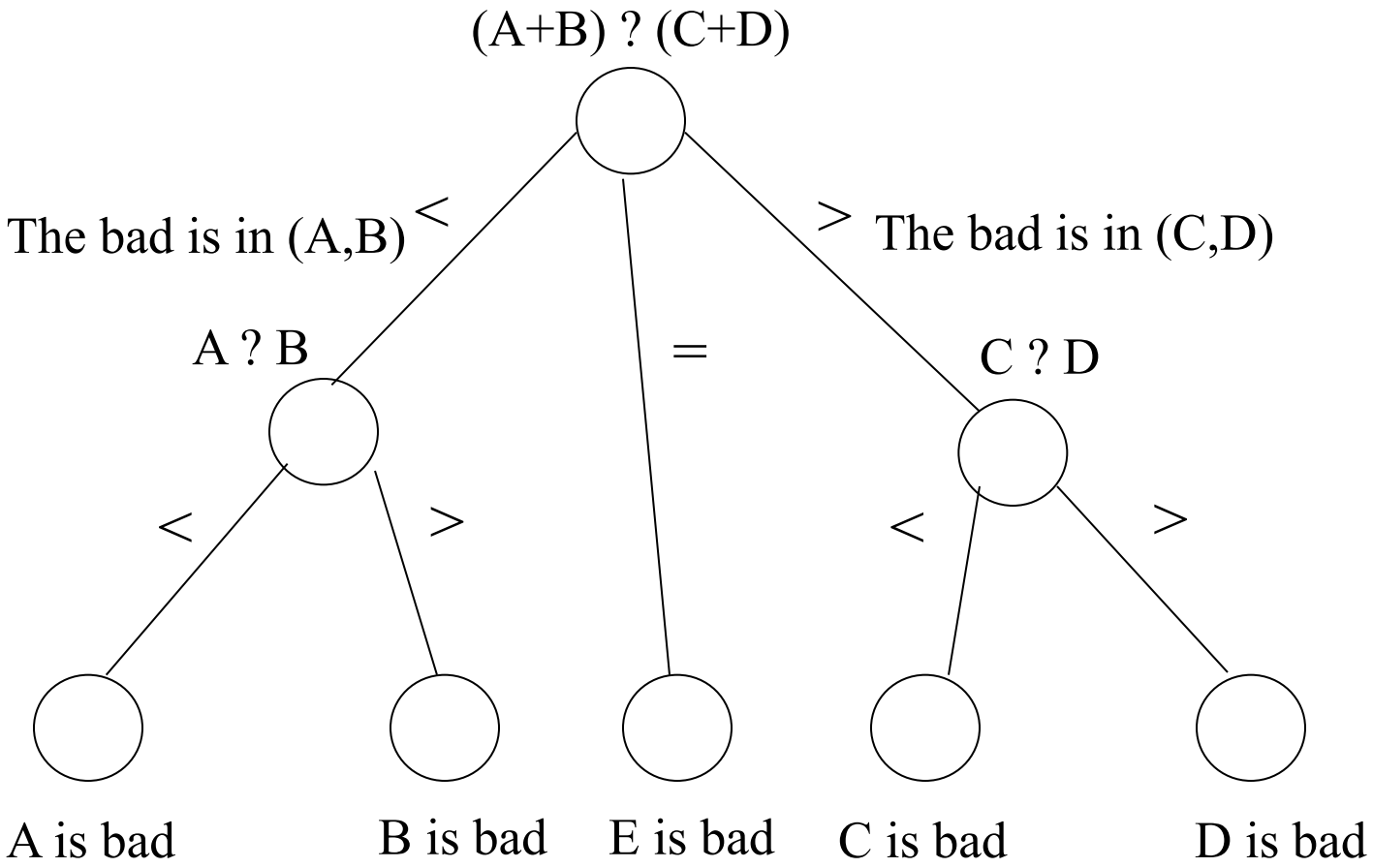
Cây biểu thức

- Cây biểu thức (Ví dụ minh họa)
  - Dựng cây biểu diễn biểu thức số học:
    - Cho một biểu thức số học, dựng cây biểu diễn biểu thức số học đó
    - Ví dụ: Cho biểu thức ((a+b)-c*d)%(f^g / (h-i)).



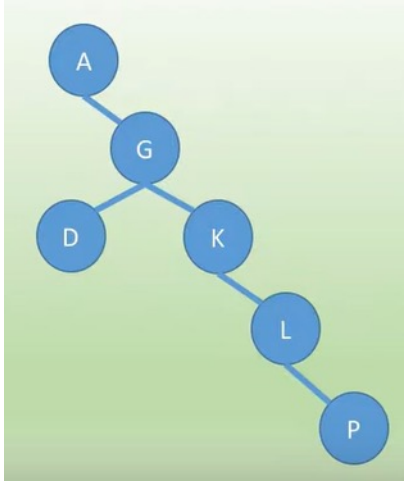
Cây quyết định

- Cây hỗ trợ ra quyết định (Ví dụ minh họa)
  - VD: Có 5 đồng tiền vàng. Trong đó có duy nhất 1 đồng bị nhẹ hơn. Tìm đồng tiền đó?

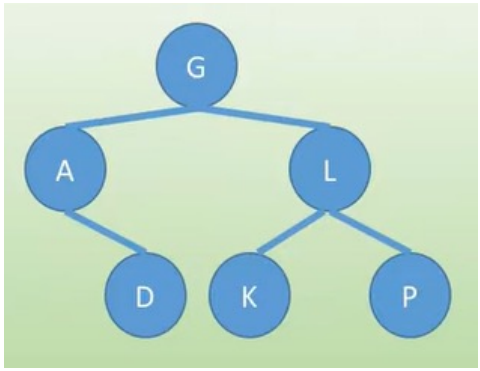


Cây cân bằng

- Giải thuật tìm kiếm đối với cây nhị phân tìm kiếm độ phức tạp phụ thuộc vào chiều cao của cây.
- Khái niệm cây cân bằng: là cây nhị phân tìm kiếm có mức độ phân bố chiều cao *cân đối* giữa từng cặp cây con trái/con phải tương ứng
  - Hạn chế mức độ suy biến của cây nhị phân tìm kiếm, nguyên nhân làm giảm tốc độ các thao tác trên loại cây này, nhất là thao tác tìm kiếm.
  - Đảm bảo độ phức tạp của tìm kiếm nhị phân luôn là $O(\log_2 N)$



vs.

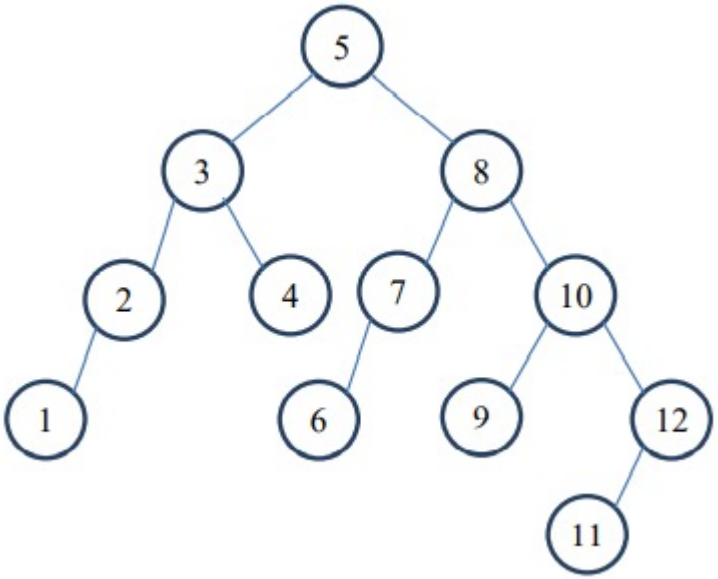


Cây cân bằng

- Cây cân bằng về chiều cao (cây cân bằng)
  - Độ chênh lệch về chiều cao giữa hai cây con trái và phải tương ứng bất kỳ trong nó không lớn hơn 1

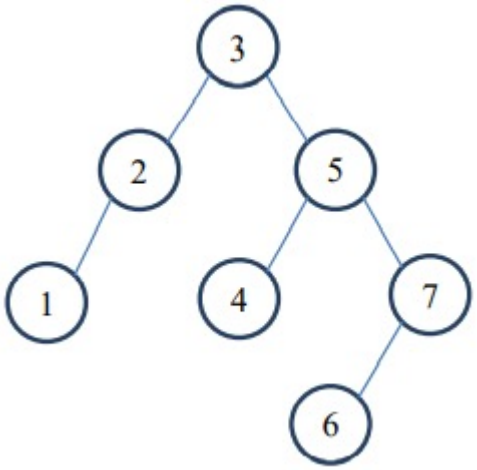
$$\forall \text{nút } A \in T, \text{ đều có } |h(A_L) - h(A_R)| \leq 1.$$

  - AL và AR là hai cây con tương ứng của nút A
  - $h(A_R)-h(A_L)$: yếu tố cân bằng (balanced factor) của nút A



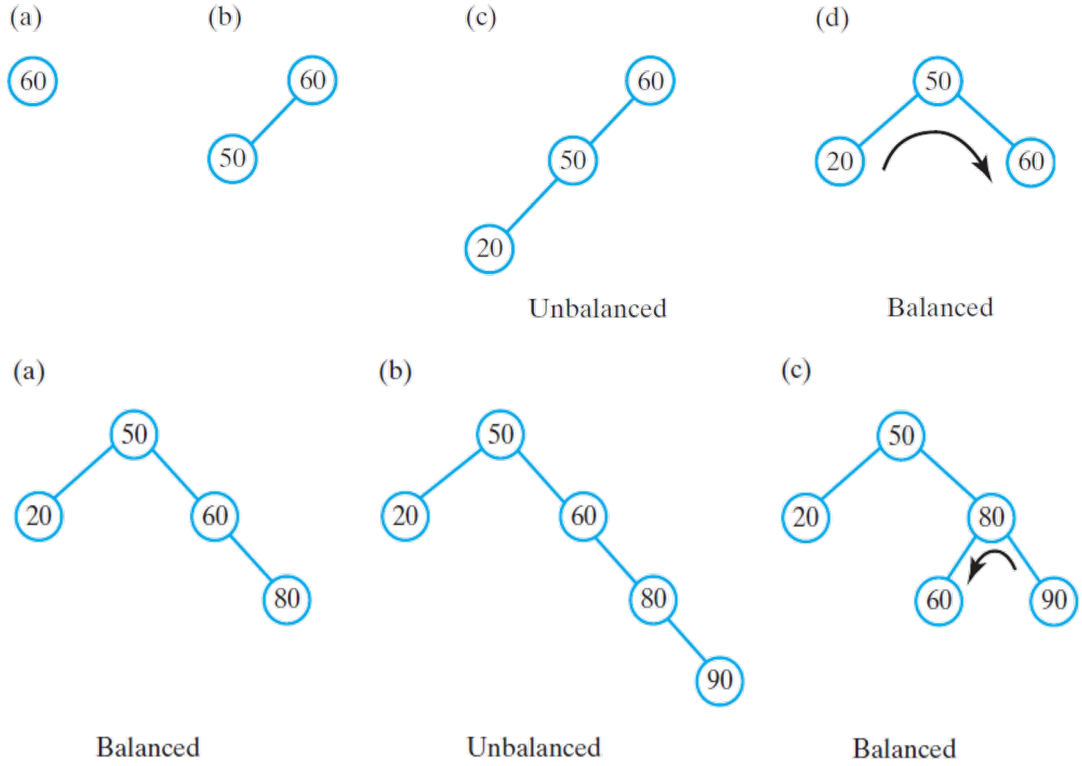
Cây cân bằng

- Cây cân bằng hoàn chỉnh
  - là một cây cân bằng mà tất cả các nút lá của nó chỉ nằm trên một hoặc hai mức



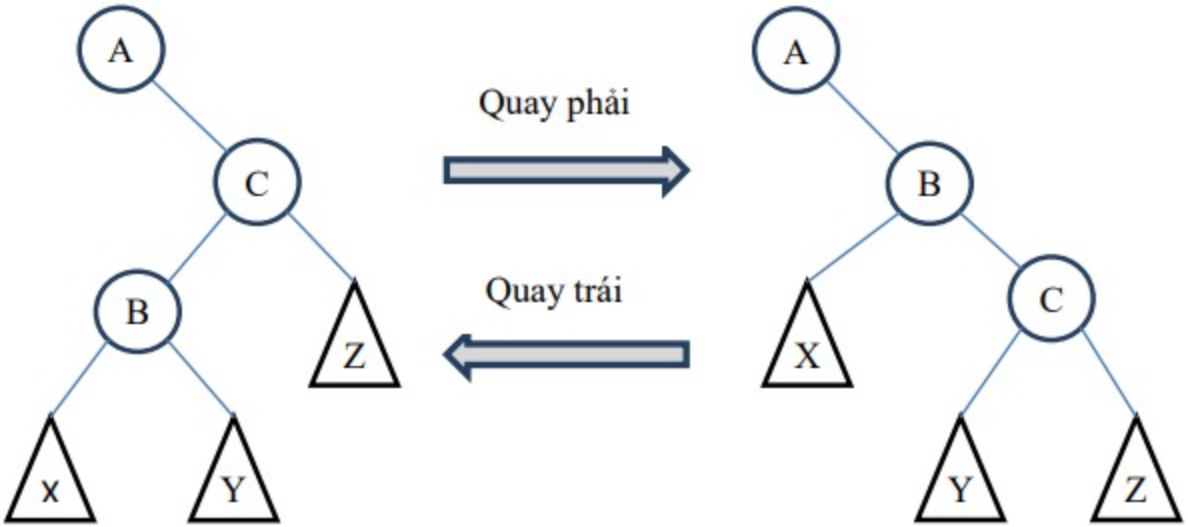
Cây cân bằng

- Phép quay: thao tác trên cây cân bằng
  - Hai thao tác đối xứng: phép quay phải và phép quay trái
  - Phép quay phải/trái tại một nút giúp giảm chiều cao cây con trái/phải và tăng chiều cao cây con phải/trái của nút đó cùng một đơn vị (yếu tố cân bằng của nút đó giảm đi hai đơn vị)



Cây cân bằng

- Phép quay



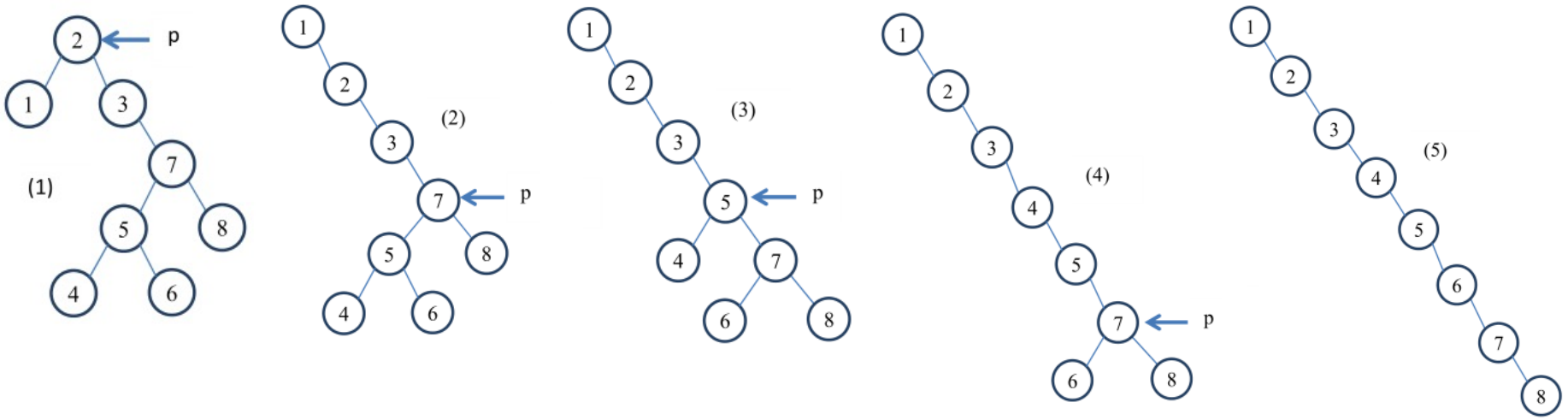
Cây cân bằng

- Một số giải thuật cân bằng cây
  - Giải thuật DSW (Day, Stout, Warren)
    - Biến đổi trực tiếp từ cây nhị phân tìm kiếm sang cây cân bằng (thực ra là sang cây cân bằng hoàn chỉnh)
  - Giải thuật AVL (Adelson, Velskii, Landis)
    - Nhóm các giải thuật nhằm duy trì lại sự cân bằng của cây mỗi khi có một thao tác: bổ sung nút, loại bỏ nút, tách/ghép cây, v.v.

Cây cân bằng

- Giải thuật DSW
  - Bước 1: Chuyển từ cây nhị phân tìm kiếm sang cây suy biến có dạng danh sách

```
P = gốc của T;
while (P ≠ NULL) {
    while (P_L ≠ NULL) quay phải tại P;
    P = P_R;
}
```

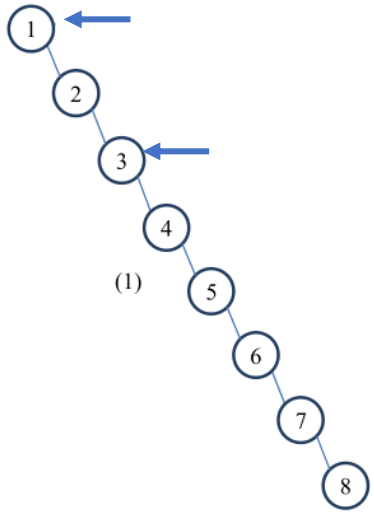


- Giải thuật DSW
  - Bước 2: Chuyển từ cây suy biến sang cây cân bằng hoàn chỉnh
  - *n* là kích thước của cây suy biến cần chuyển

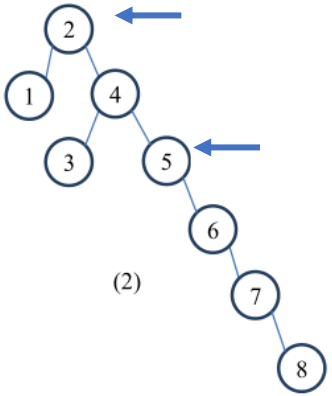
- M = $\lfloor (N-1)/2 \rfloor - 1$;
- Thực hiện M phép quay trái bắt đầu từ nút gốc rồi xuống dần nút con phải (nhưng cách một nút mới quay một lần).
- Chừng nào M>=1:
  + Thực hiện M phép quay trái bắt đầu từ nút gốc trở xuống (cũng cách một nút mới quay một lần)
  + M = M/2;

Có nhiều biến thể của Giải thuật !

M=2

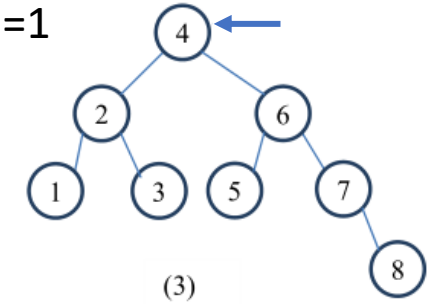


(1)

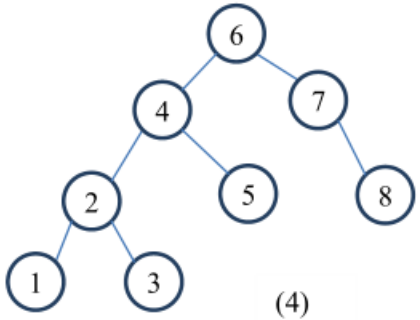


(2)

M=1

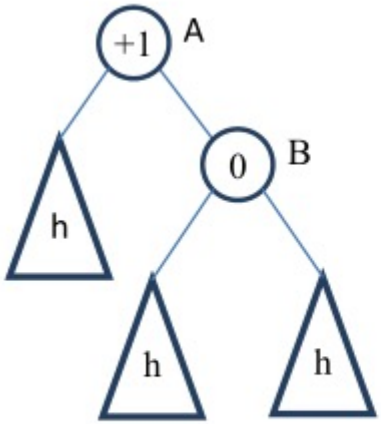


(3)



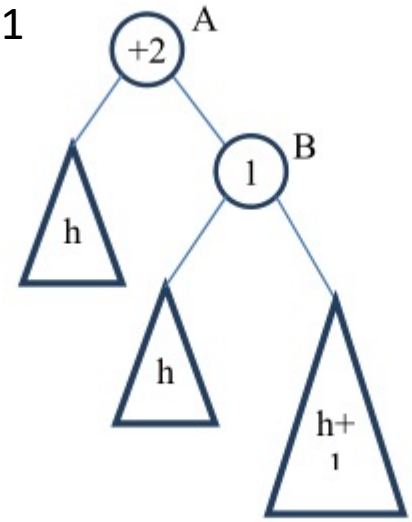
(4)

- Giải thuật con AVL khi bổ sung một nút (dựng cây, bổ sung mới)
  - Có thể làm cây mất cân bằng (khi nút được bổ sung nằm ở cây con trái hoặc con phải của nút gốc)
  - Giả sử nút được bổ sung sẽ nằm ở cây con phải
    - Giả sử chiều cao cây con trái và phải trước khi bổ sung lần lượt là h và h+1 (a1)
    - Nếu nút được bổ sung nằm ở cây con phải của cây con phải của nút gốc (a2) khi đó, để cân bằng lại cây, chỉ cần một phép quay trái từ nút gốc

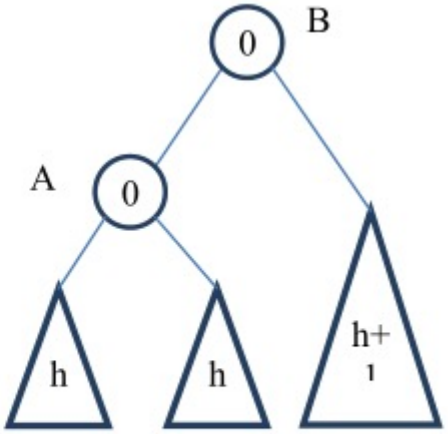


(a1)

TH1

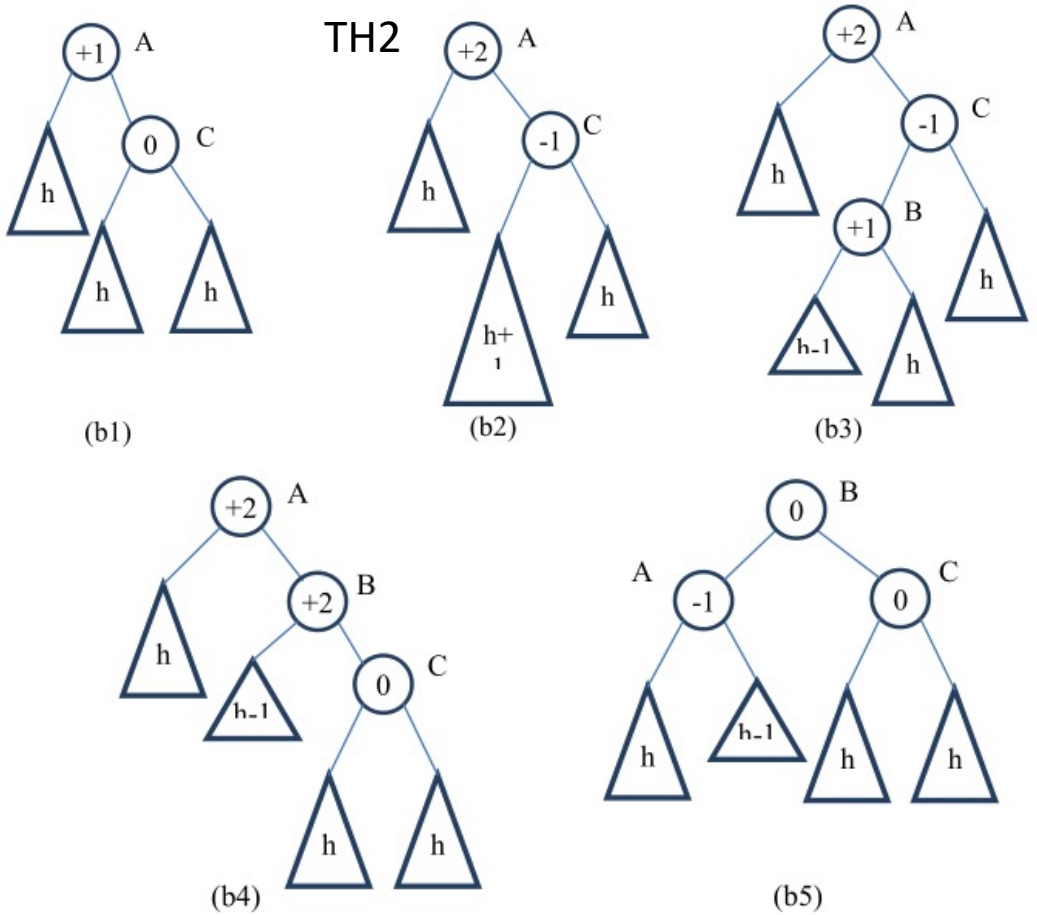


(a2)



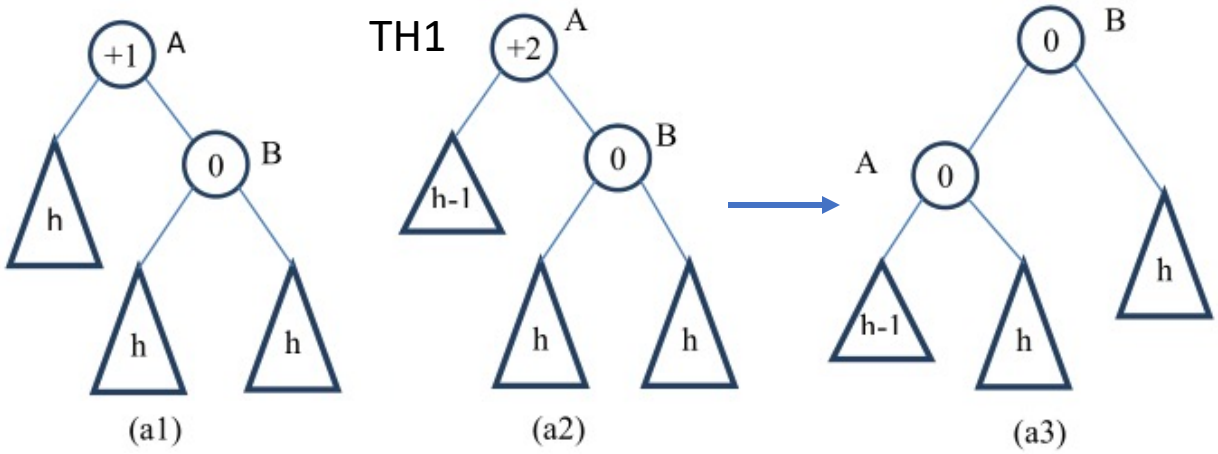
(a3)

- Giải thuật con AVL khi bổ sung một nút
  - Nếu nút được bổ sung nằm ở cây con trái (gốc C) của cây con phải của nút gốc (b123): quay phải tại C (b4) rồi sau đó quay trái tại A (b5)

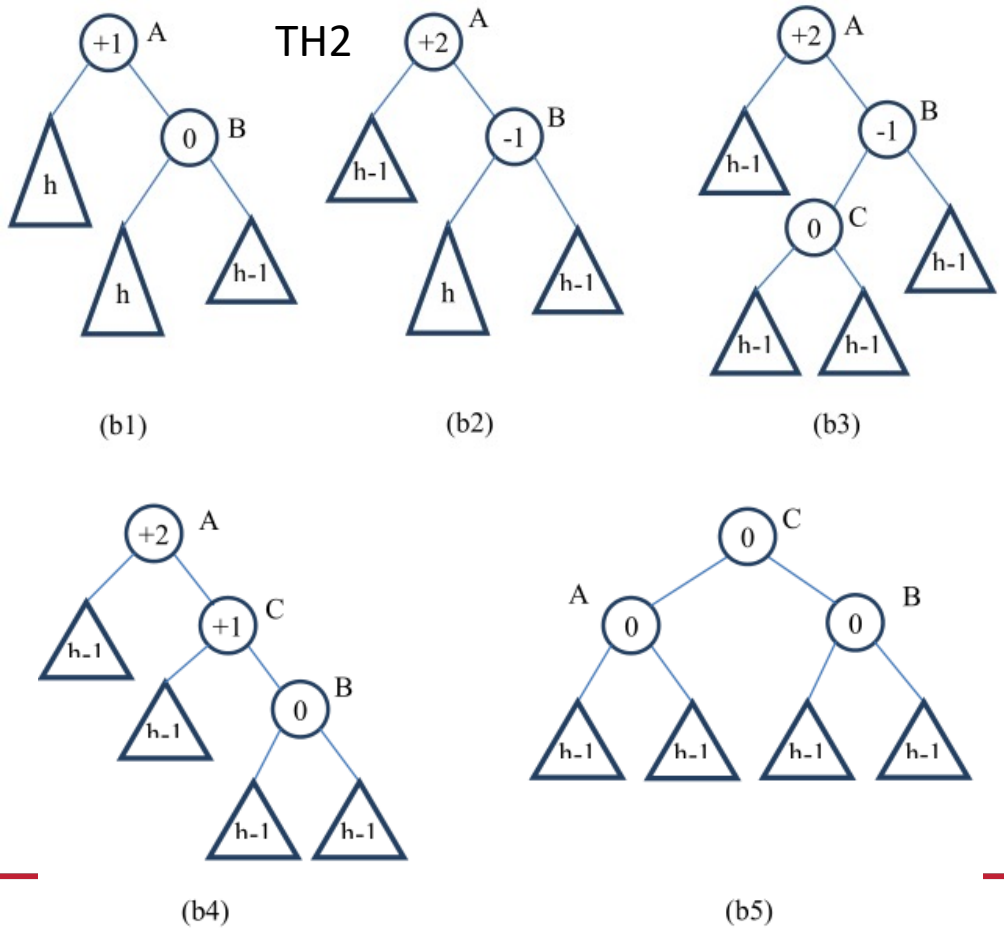


"quay kép"

- Giải thuật con AVL khi loại bỏ một nút
  - có thể làm mất cân bằng khi nút bị loại bỏ nằm ở cây con trái hoặc cây con phải của nút gốc
  - **Giả sử nút bị loại bỏ nằm ở cây con trái**, A là nút gốc và B là nút gốc của cây con phải (a1)
    - Giả sử chiều cao của cây con trái và cây con phải trước khi loại bỏ lần lượt là h và h+1
    - Nếu cây con phải của B có chiều cao h (a2), thực hiện một phép quay trái tại A (a3)



- Giải thuật con AVL khi loại bỏ một nút
  - Nếu cây con phải của B có chiều cao h-1 (b1), chiều cao cây con trái của B phải là h, sau loại bỏ chiều cao cây con trái sẽ là h-1 (b2), gọi C là nút con trái của B (b3) → thực hiện quay phải tại B, rồi quay trái tại A (b4, b5)



Bài tập

- **Bài 1**: Viết chương trình bằng C tạo cây tìm kiếm nhị phân, biết cấu trúc 1 nút được khai báo:

  struct node { int data; struct node *left; struct node *right; };

  và thực hiện tất cả các thao tác sau,

  - **void create_tree(struct node *);**
  - **struct node *insertElement(struct node *, int);**
  - **void preorderTraversal(struct node *);**
  - **void inorderTraversal(struct node *)**
  - **void postorderTraversal(struct node *);**
  - **struct node *findSmallestElement(struct node *);**
  - **struct node *findLargestElement(struct node *);**
  - **struct node *deleteElement(struct node *, int);**
  - **int totalNodes(struct node *);**
  - **int Height(struct node *);**
  - **struct node *deleteTree(struct node *);**

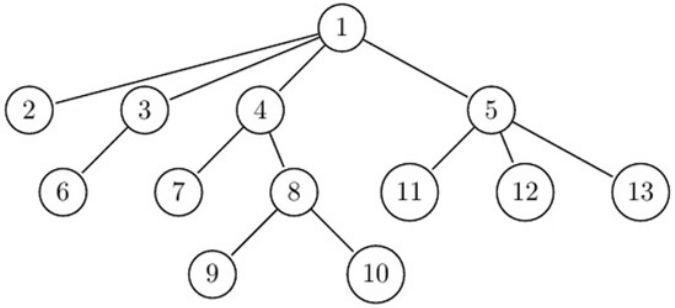
Bài tập

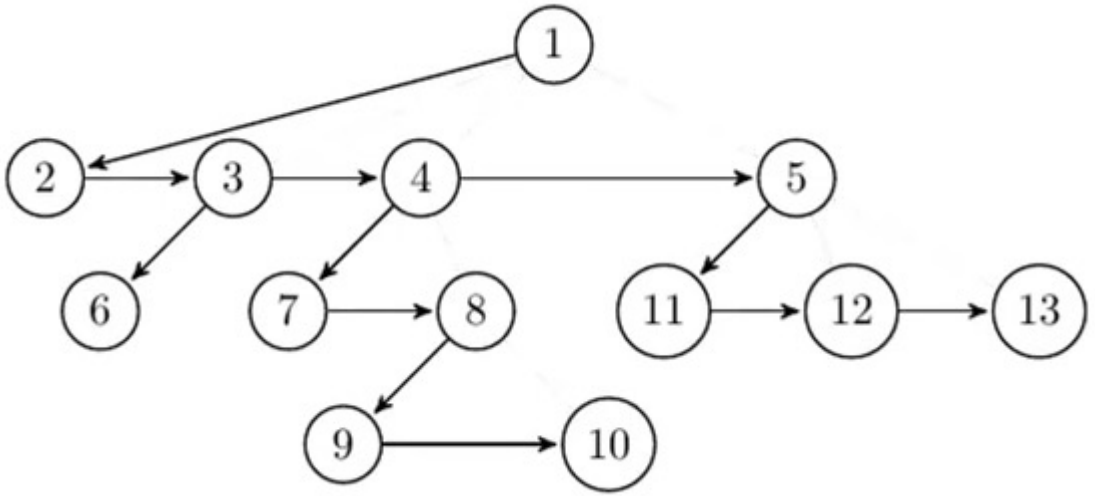
- **Bài 2**: Xây dựng một class Folder biểu diễn cây thư mục, trong đó một Folder có thể chứa nhiều File và SubFolder.
- Yêu cầu cài đặt class Folder.
- Cài đặt các hàm thực hiện các thao tác cơ bản sau:
  - Thêm một Folder/File vào cùng mức với một thư mục
  - Thêm một Folder/File vào thành con của một thư mục
  - Duyệt một thư mục, in ra các Folder/File con của thư mục đó
  - Tìm kiếm một Folder/File trong một thư mục. Hàm trả về con trỏ trỏ vào Folder/File tìm được

Bài tập

- **Bài 3:** Viết chương trình biến đổi một biểu thức toán học dạng chuỗi ký tự sang dạng cây biểu thức. Biểu thức chứa các toán tử +, -, *, /, các toán hạng dạng 1 ký tự chữ hoặc số, các dấu đóng mở ngoặc. Duyệt lại cây biểu thức đã xây dựng theo thứ tự trước, giữa, sau và in ra kết quả duyệt.

- **Bài 4:** Viết chương trình thực hiện thao tác chèn trong cây AVL.


1
2
3
4
5
6
7
8
11
12
13
9
10


1
2
3
4
5
6
7
8
11
12
13
9
10

Day	Weather	Temperature	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Cloudy	Hot	High	Weak	Yes
3	Sunny	Mild	Normal	Strong	Yes
4	Cloudy	Mild	High	Strong	Yes
5	Rainy	Mild	High	Strong	No
6	Rainy	Cool	Normal	Strong	No
7	Rainy	Mild	High	Weak	Yes
8	Sunny	Hot	High		
9	Cloudy	Hot	Norma		
10	Rainy	Mild	High