Chương 3: CÂY

Nguyễn Văn Linh
Khoa Công nghệ Thông tin & Truyền thông
nvlinh@ctu.edu.vn

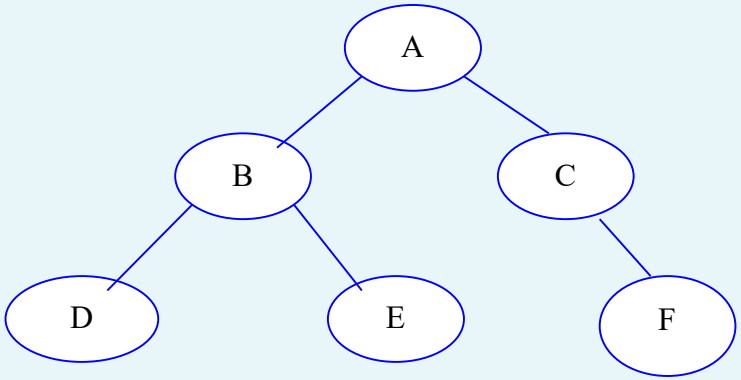
CANTHO UNIVERSITY

NỘI DUNG

- CÁC THUẬT NGỮ CƠ BẢN
- CÁC PHÉP TOÁN
- CÀI ĐẶT CÂY
- CÂY NHỊ PHÂN
- CÂY TÌM KIẾM NHỊ PHÂN

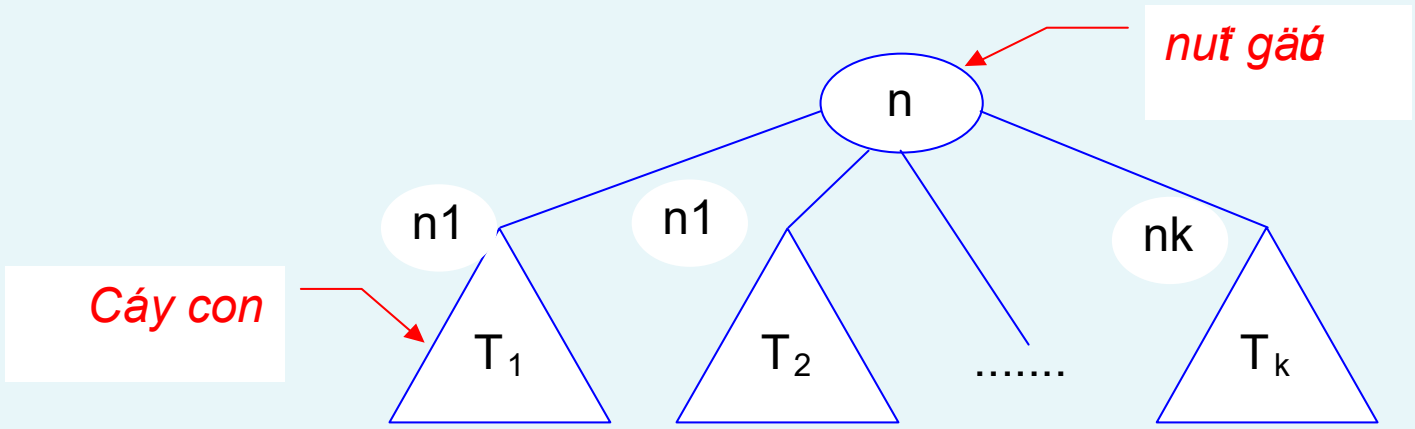
CÁC THUẬT NGỮ CƠ BẢN (1)

- Định nghĩa
  - Cây (tree): một tập hợp hữu hạn các phần tử gọi là các nút (nodes) và tập hợp hữu hạn các cạnh nối các cặp nút lại với nhau mà không tạo thành chu trình.
  - Nút gốc và quan hệ cha - con



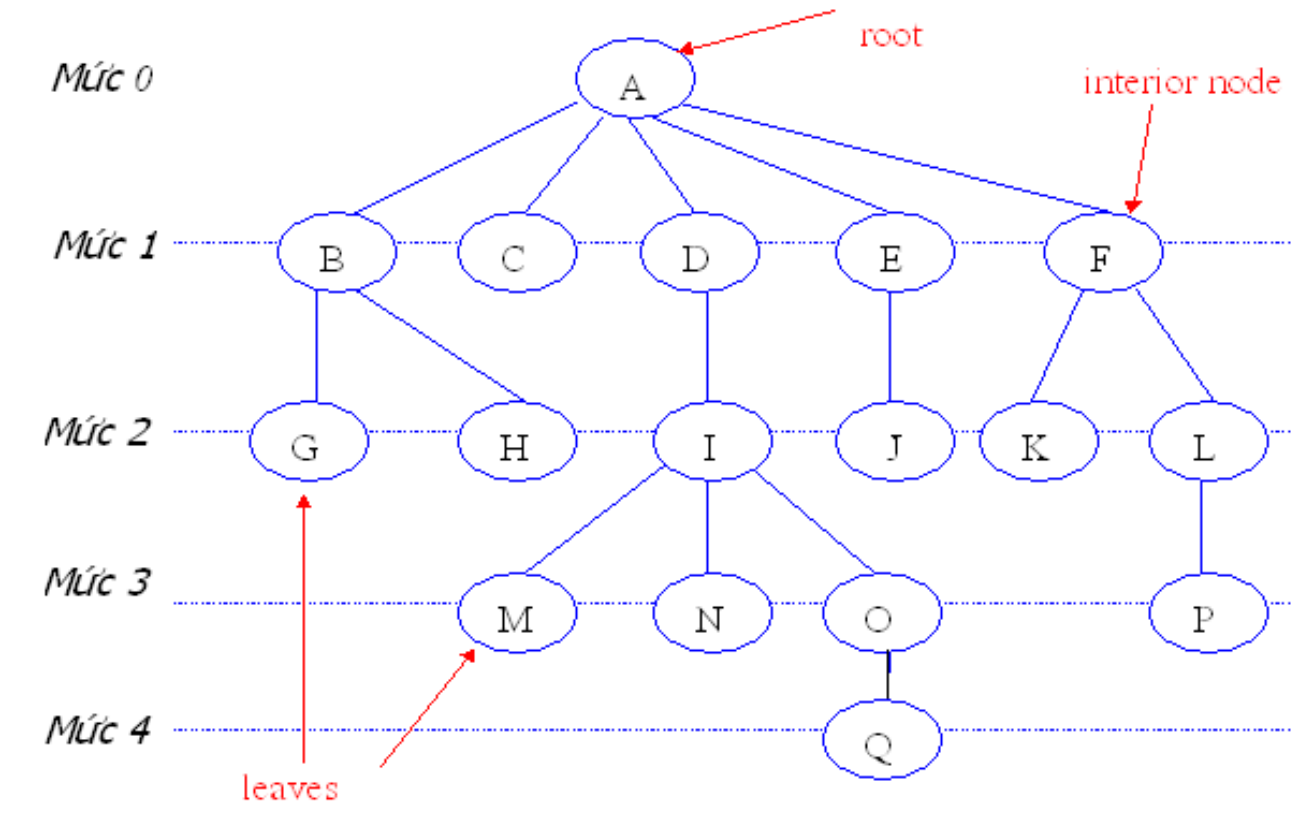
CÁC THUẬT NGỮ CƠ BẢN (2)

- Ta có thể định nghĩa cây 1 cách đệ qui:
  - Một nút đơn độc là 1 cây, nút này cũng là nút gốc của cây.
  - Nút n là nút đơn độc và k cây riêng lẻ T1, T2, ...Tk có các nút gốc lần lượt là n1, n2,...nk. Khi đó ta có được 1 cây mới bằng cách cho n là cha của các nút n1, n2, … nk.



CÁC THUẬT NGỮ CƠ BẢN (3)

- Ví dụ



CÁC THUẬT NGỮ CƠ BẢN (4)

- Nút cha con: nút A là cha của nút B khi nút A ở mức i và nút B ở mức i+1, đồng thời giữa A và B có cạnh nối.
  - VD: Ở cây trên, nút B là cha của G và H. Nút I là con của D.
- Bậc của nút là số cây con của nút đó, bậc nút lá =0.
  - VD: A có bậc 5, C có bậc 0, O có bậc 1
- Bậc của cây là bậc lớn nhất của các nút trên cây.
  - VD: cây trên có bậc 5.
- Cây n-phân là cây có bậc n.
  - VD: Bậc của cây là 5 hay cây ngũ phân

CÁC THUẬT NGỮ CƠ BẢN (5)

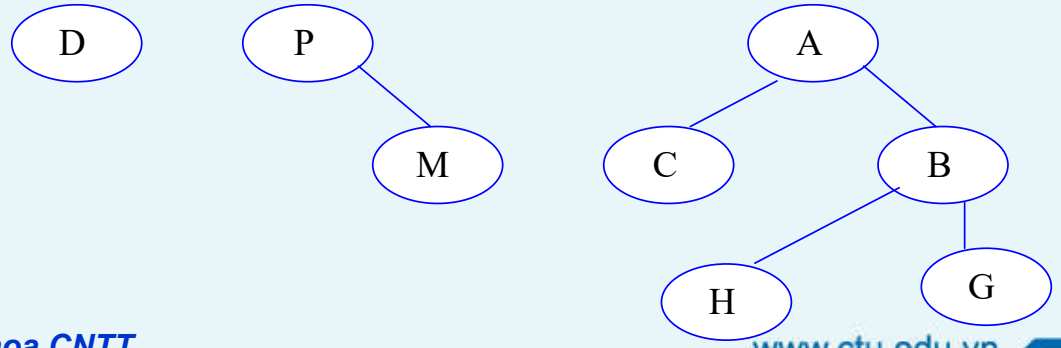
- Nút gốc (root) là nút không có cha.
  - VD: nút gốc A
- Nút lá (leaf) là nút không có con.
  - VD: các nút C, G, H, J, K, M, N, P, Q.
- Nút trung gian (interior node): nút có bậc khác 0 và không phải là nút gốc
  - VD: các nút B, D, E, F, I, L, O
- Nút tiền bối (descendant) & nút hậu duệ (ancestor): Nếu có đường đi từ nút a đến nút b thì nút a là tiền bối của b, còn b là hậu duệ của a.
  - VD: D là tiền bối của Q, còn Q là hậu duệ của D
- Cây con của 1 cây là 1 nút cùng với tất cả các hậu duệ của nó.

CÁC THUẬT NGỮ CƠ BẢN (6)

- Đường đi là một chuỗi các nút n1, n2, ..., nk trên cây sao cho ni là nút cha của nút ni+1 (i=1..k-1)
  - VD: có đường đi A, D, I, O, Q
- Độ dài đường đi bằng số nút trên đường đi trừ 1
  - VD: độ dài đường đi A,D,I,O,Q = 5-1=4
- Chiều cao của 1 nút là độ dài đường đi từ nút đó đến nút lá xa nhất.
  - VD: nút B có chiều cao 1, nút D có chiều cao 3
- Chiều cao của cây là chiều cao của nút gốc
  - VD: chiều cao của cây là 4

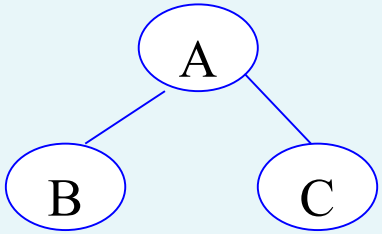
CÁC THUẬT NGỮ CƠ BẢN (7)

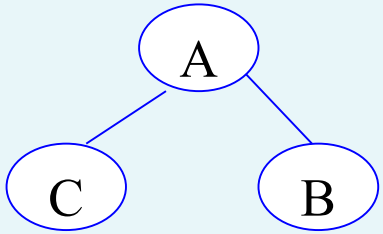
- Độ sâu của 1 nút là độ dài đường đi từ nút gốc đến nút đó, hay còn gọi là mức (level) của nút đó.
  - VD: I có độ sâu 2, E có độ sâu 1
    M, N, O, P có cùng mức 3
- Nhãn của một nút không phải là tên mà là giá trị được lưu trữ tại nút đó.
- Rừng là một tập hợp nhiều cây.



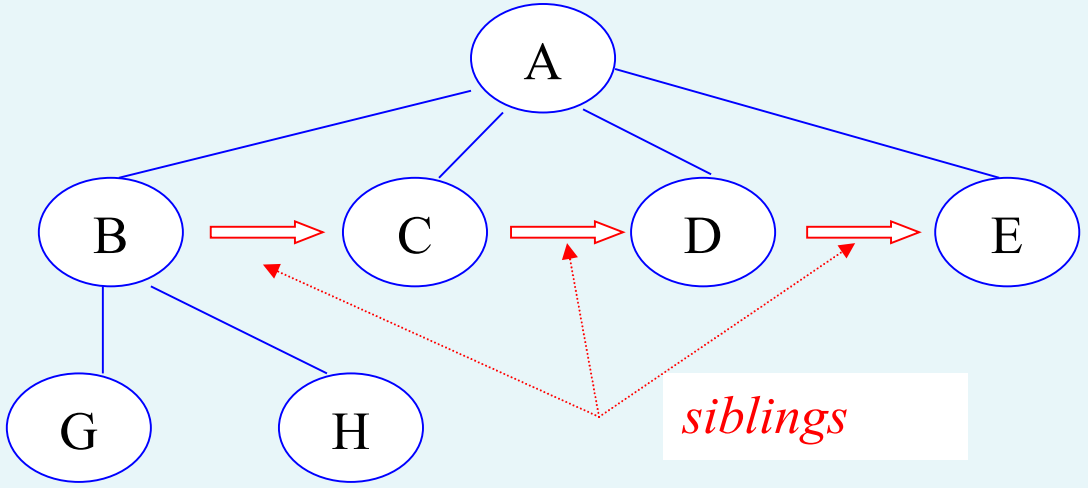
CÁC THUẬT NGỮ CƠ BẢN (8)

- Cây có thứ tự
  - Nếu ta phân biệt thứ tự các nút trong cùng 1 cây thì ta gọi cây đó có thứ tự. Ngược lại, gọi là cây không có thứ tự.
  - Trong cây có thứ tự, thứ tự qui ước từ trái sang phải.





CÁC THUẬT NGỮ CƠ BẢN (9)



- Các nút con cùng một nút cha gọi là các nút anh em ruột (siblings)
- Mở rộng: nếu ni và nk là hai nút anh em ruột và nút ni ở bên trái nút nk thì các hậu duệ của nút ni là bên trái mọi hậu duệ của nút nk

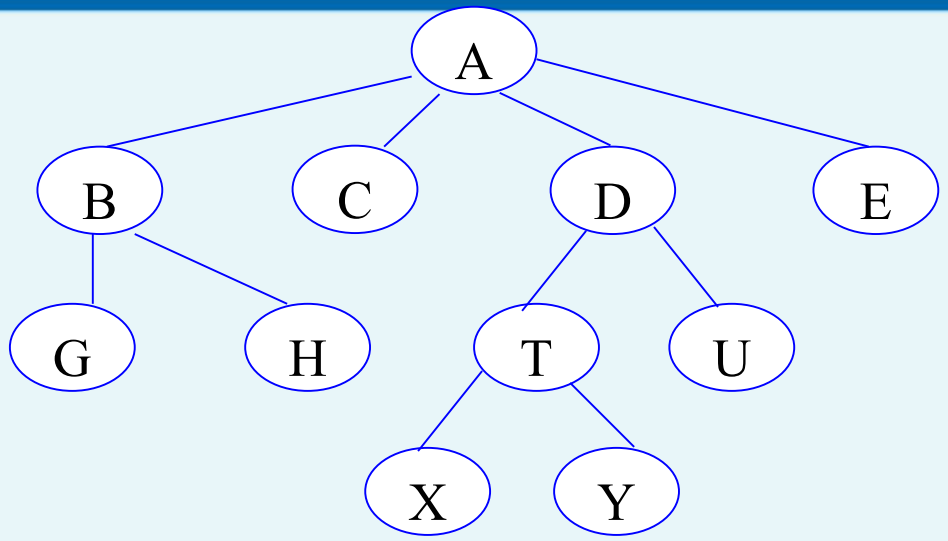
CÁC THUẬT NGỮ CƠ BẢN (10)

- Duyệt cây:
  - Quy tắc: đi qua lần lượt tất cả các nút của cây, mỗi nút đúng một lần
  - Danh sách duyệt cây: là danh sách liệt kê các nút theo thứ tự đi qua
  - Có 3 phương pháp duyệt tổng quát:
    - tiền tự (preorder)
    - trung tự (inorder)
    - hậu tự (posorder)

CÁC THUẬT NGỮ CƠ BẢN (11)

- Định nghĩa đệ qui các phép duyệt
  - Cây rỗng hoặc cây chỉ có một nút: cả 3 biểu thức duyệt là rỗng hay chỉ có một nút tương ứng
  - Ngược lại, giả sử cây T có nút gốc là n và các cây con là T1, T2 ,...,Tn thì:
    - Biểu thức duyệt tiền tự của cây T là nút n, kế tiếp là biểu thức duyệt tiền tự của các cây T1, T2 ,...,Tn theo thứ tự đó
    - Biểu thức duyệt trung tự của cây T là biểu thức duyệt trung tự của cây T1, kế tiếp là nút n rồi đến biểu thức duyệt trung tự của các cây T2 ,...,Tn theo thứ tự đó
    - Biểu thức duyệt hậu tự của cây T là biểu thức duyệt hậu tự của các cây T1, T2 ,...,Tn theo thứ tự đó rồi đến nút n

CÁC THUẬT NGỮ CƠ BẢN (13)



- tiền tự: **A** B G H C D T X Y U E
- trung tự: G B H **A** C X T Y D U E
- hậu tự: G H B C X Y T U D E **A**

CÁC THUẬT NGỮ CƠ BẢN (14)

Thuật duyệt đệ qui duyệt tiền tự

```
void Preorder(Node n, Tree T){
    liệt kê nút n;
    for (mỗi cây con c của nút n theo thứ tự từ trái sang phải)
        Preorder(c, T);
} //Preorder
```

CÁC THUẬT NGỮ CƠ BẢN (15)
Thuật duyệt đệ qui duyệt trung tự

```
void Inorder(Node n, Tree T){
   if (n là nút lá) liệt kê nút n
    else {
            Inorder(con trái nhất của n, T)
            Liệt kê nút n;
            for(mỗi cây con c của nút n, trừ cây con trái nhất, từ trái
   sang phải)
              Inorder(c, T);
          }
} //Inorder
```

CÁC THUẬT NGỮ CƠ BẢN (16)
Thuật duyệt đệ qui duyệt hậu tự

```
void Posorder (Node n, Tree T){
   if (n là nút lá)  Liệt kê nút n
   else {
       for (mỗi nút con c của nút n từ trái sang phải)
          Posorder(c, T);
       liệt kê nút n;
       }
    }; //Posorder
```

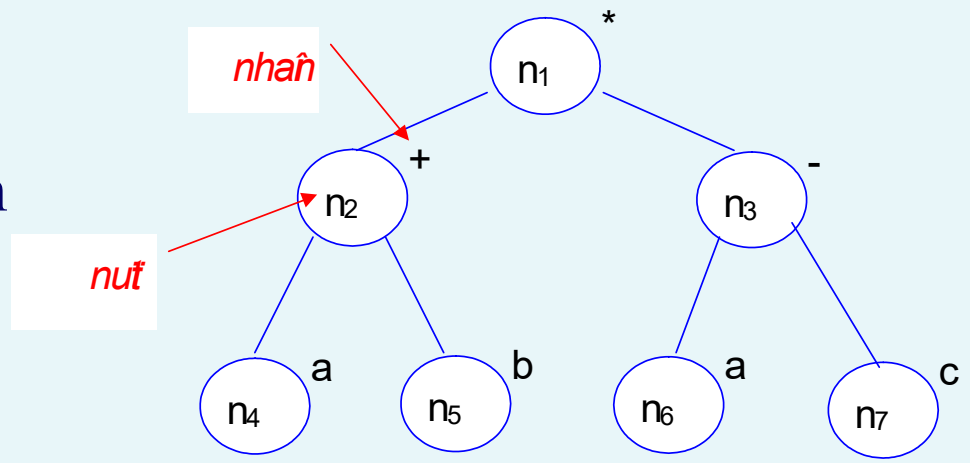
CÁC THUẬT NGỮ CƠ BẢN (16)

- Cây có nhãn và cây biểu thức

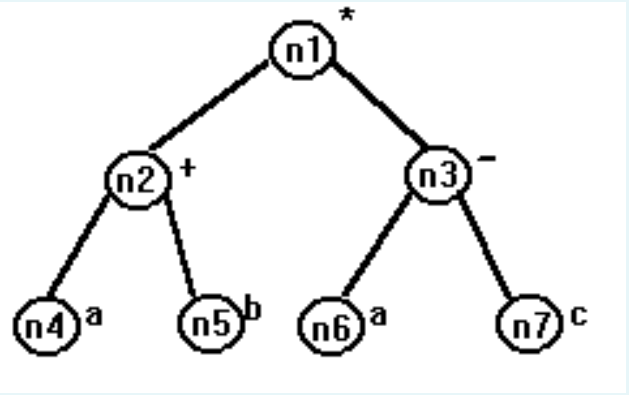
  (labeled trees and expression trees)

  – Lưu trữ kết hợp một nhãn (label) hoặc một giá trị (value) với một nút trên cây

  - Nhãn: giá trị được lưu trữ tại nút đó, còn gọi là khóa của nút



Cây biểu thức (a+b)*(a-c)



- Biểu thức tiền tố: * + a b – a c
- Biểu thức trung tố: a + b * a – c
- Biểu thức hậu tố: a b + a c - *

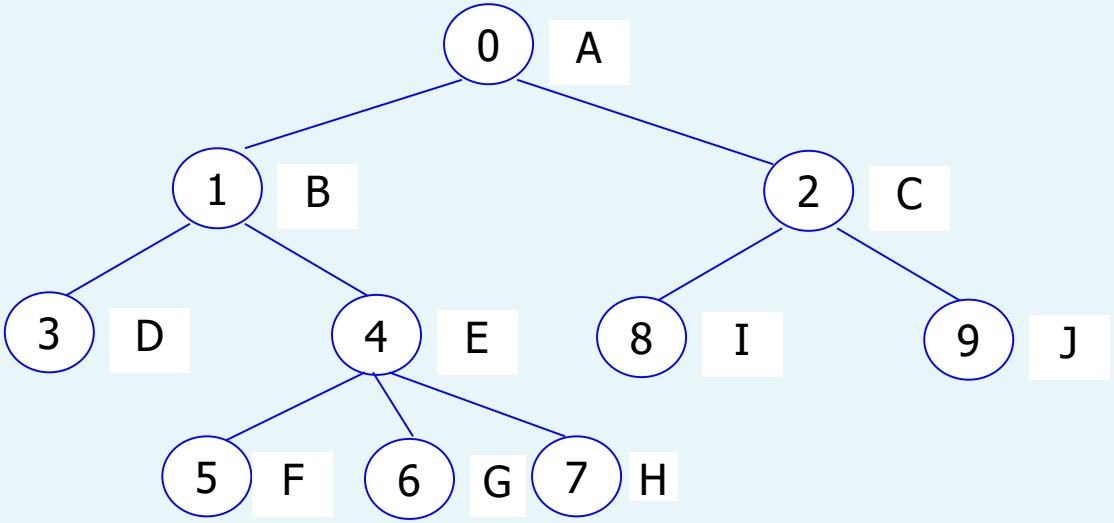
CÁC PHÉP TOÁN CƠ BẢN TRÊN CÂY

Hàm	Diễn giải
Make_Null_Tree(T)	Tạo cây T rỗng
Empty_Tree(T)	Kiểm tra xem cây T có rỗng không
Parent(n,T)	Trả về nút cha của nút n trên cây T, nếu n là nút gốc thì hàm cho giá trị NIL.
Left_Most_Child(n,T)	Trả về nút con trái nhất của nút n trên cây T, nếu n là lá thì hàm cho giá trị NIL.
Right_Sibling(n,T)	Trả về nút em ruột phải nút n trên cây T, nếu n không có em ruột phải thì hàm cho giá trị NIL.
Label_Node(n,T)	Trả về nhãn tại nút n của cây T.
Root(T)	Trả về nút gốc của cây T. Nếu Cây T rỗng thì hàm trả về NIL.

CÀI ĐẶT CÂY BẰNG MẢNG (1)

- Đánh số theo thứ tự tăng dần bắt đầu tại nút gốc.
- Nút cha được đánh số trước các nút con.
- Các nút con cùng một nút cha được đánh số lần lượt từ trái sang phải

CÀI ĐẶT CÂY BẰNG MẢNG (2)



A	B	C	D	E	F	G	H	I	J	…	
-1	0	0	1	1	4	4	4	2	2	…	
0	1	2	3	4	5	6	7	8	9	…	

Nhãn của các nút Data

Cha của các nút Parent

Chỉ số của mảng

CÀI ĐẶT CÂY BẰNG MẢNG (3)

- Khai báo

```
#define Max_Length ... //chỉ số tối đa của mảng
#define NIL -1
typedef  ... Data_Type;
typedef int Node;
typedef struct {
   Data_Type Data[Max_Length];  //Lưu trữ nhãn (dữ liệu) của nút trong cây
   Node Parent[Max_Length];     //Lưu trữ cha của các nút trong cây
   int Max_Node; //Số nút thực sự trong cây
} Tree;
```

Khởi tạo cây rỗng

```
void Make_Null_Tree (Tree &T){
    T.Max_Node=0;
}
```

Kiểm tra cây rỗng

```
int Empty_Tree(Tree T) {
    return T.Max_Node == 0;
}
```

Xác định nút cha của nút trên cây

```
Node Parent(Node n, Tree T){
    if (n==0) return NIL;
    return T.Parent[n];
}
```

Xác định nhãn của nút trên cây

```
Data_Type Label_Node(Node n, Tree T){
   return T.Data[n];
}
```

Hàm xác định nút gốc trong cây

```
Node Root(Tree T){
    return 0;
}
```

Xác định con trái nhất của một nút

Hàm Left_Most_Child(n,T)

Xét nút i, bắt đầu từ n+1 đến nút cuối cùng

Nếu có một i sao cho Parent(i,T) == n thì i là con trái nhất của n

Ngược lại thì n không có con trái nhất.

Xác định con trái nhất của một nút

```
Node Left_Most_Child(Node n, Tree T) {
    Node i=n+1;//Vị trí nút đầu tiên hy vọng là con của nút n
    int found =0;
    while ((i<=T.Max_Node-1) && !found)
        if (T.Parent[i]==n) found=1;
        else i=i+1;
  if (found) return i;
  else return NIL;
 }
```

Xác định em ruột phải của một nút

Hàm **Right_Sibling**(n,T)

Em ruột phải của n, chỉ có thể là n+1

Nếu n+1 là một node và có cùng cha với n thì trả về n+1

Ngược lại trả về NIL

Hàm xác định em ruột phải của một nút

```
Node Right_Sibling(Node n,Tree T){
        if ((n+1<T.Max_Node)
           && (T.Parent[n]==T.Parent[n+1]))
            return n+1;
       else return NIL;
}
```

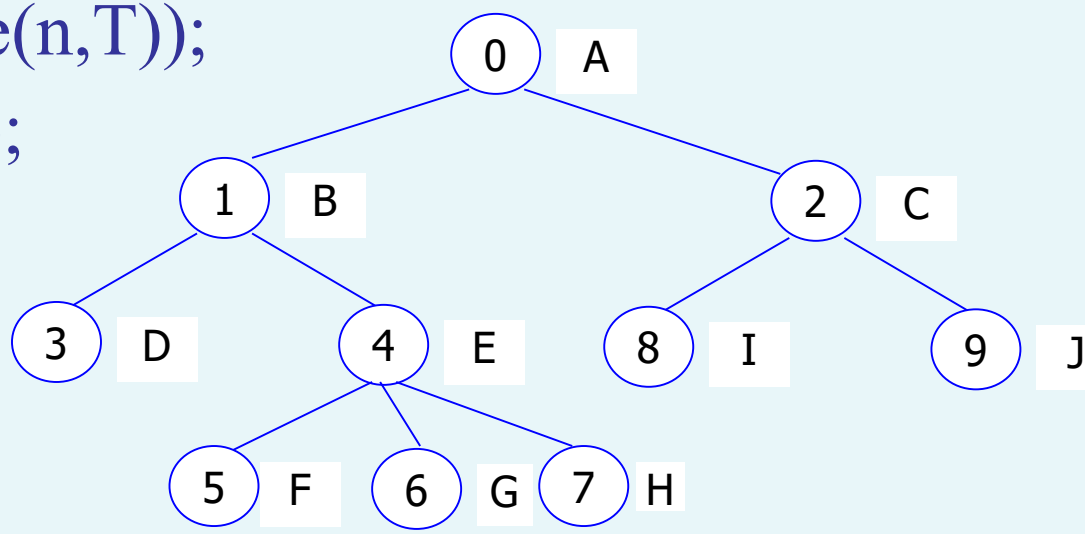
DUYỆT TIỀN TỰ

```
void Preorder(Node n, Tree T)
{
        liệt kê nút n;
        for (mỗi cây con c của nút n
      theo thứ tự từ trái sang phải)
            Preorder(c, T);
    } //Preorder
```

- Hàm Preorder(n, T)
- In ra giá trị của nút n
- Đặt c là con trái nhất của n
- Trong khi c # NIL
  - Preorder(c,T)
  - Đặt c bằng em ruột phải của c

DUYỆT TIỀN TỰ

```
void Preorder(Node n,Tree T) {
   Node c;
   printf("%c ",Label_Node(n,T));
   c=Left_Most_Child(n,T);
   while (c!=NIL)   {
      Preorder(c,T);
      c=Right_Sibling(c,T);
   }
}
```



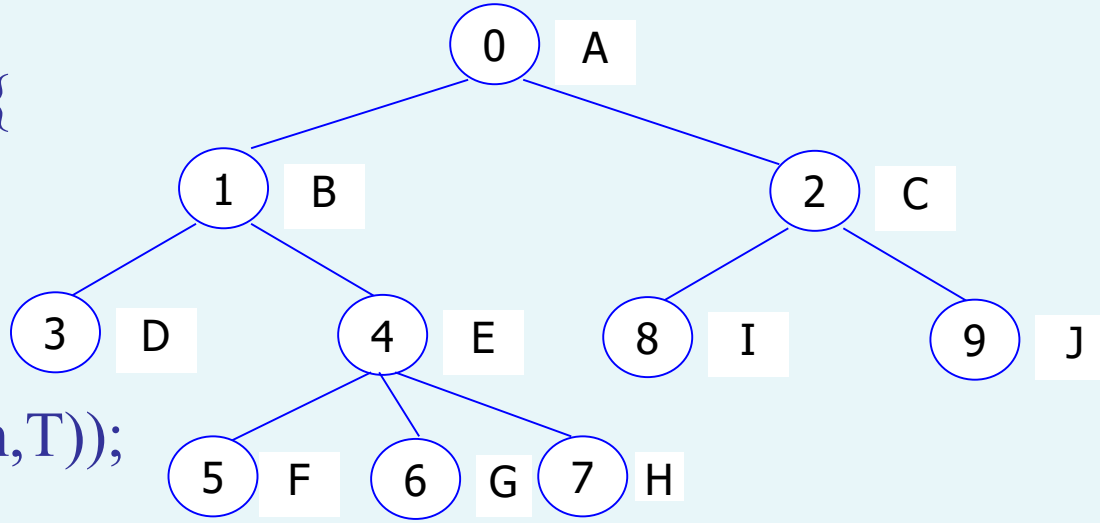
DUYỆT TRUNG TỰ

```
void Inorder(Node n, Tree T){
    if (n là nút lá) liệt kê nút n
     else {
            Inorder(con trái nhất của n,
    T)
            Liệt kê nút n;
            for(mỗi cây con c của nút n,
    trừ cây con trái nhất, từ trái sang
    phải)
                Inorder(c, T);
          }
} //Inorder
```

- Hàm Inorder(Node n, Tree T){
- Đặt c là con trái nhất của n
- Nếu c#NIL Inorder(c,T)
- In giá trị của nút n
- Đặt c là em ruột phải của c
- Trong khi c#NIL
  - Inorder(c,T)
  - Đặt c là em ruột phải của c

DUYỆT TRUNG TỰ

```
void Inorder(Node n,Tree T) {
    Node c;
    c=Left_Most_Child(n,T);
    if (c!=NIL) Inorder(c,T);
    printf("%c ",Label_Node(n,T));
    c=Right_Sibling(c,T);
    while (c!=NIL) {
        Inorder(c,T);
        c=Right_Sibling(c,T);
    }
}
```



DUYỆT HẬU TỰ

```
void Posorder (Node n, Tree
   T){
   if (n là nút lá)  Liệt kê nút n
   else {
       for (mỗi nút con c của nút
   n từ trái sang phải)
           Posorder(c, T);
       liệt kê nút n;
       }
   }; //Posorder
```

- Hàm Posorder(n,T)
- Đặt c là con trái nhất của n
- Trong khi c#NIL
  - Posorder(c,T)
  - Đặt c bằng em ruột phải của c
- In giá trị của nút n

DUYỆT HẬU TỰ

```
void Posorder(Node n,Tree T){
    Node c;
    c=Left_Most_Child(n,T);
    while (c!=NIL) {
        Posorder(c,T);
        c=Right_Sibling(c,T);
    }
    printf("%c ",Label_Node(n,T));
}
```

BÀI TẬP (1)

- Viết chương trình nhập dữ liệu vào cho cây từ bàn phím như:
  - Tổng số nút trên cây
  - Ứng với từng nút thì phải nhập nhãn của nút, cha của nút
- Hiển thị danh sách duyệt cây theo các phương pháp duyệt tiền tự, trung tự, hậu tự

BÀI TẬP (2)

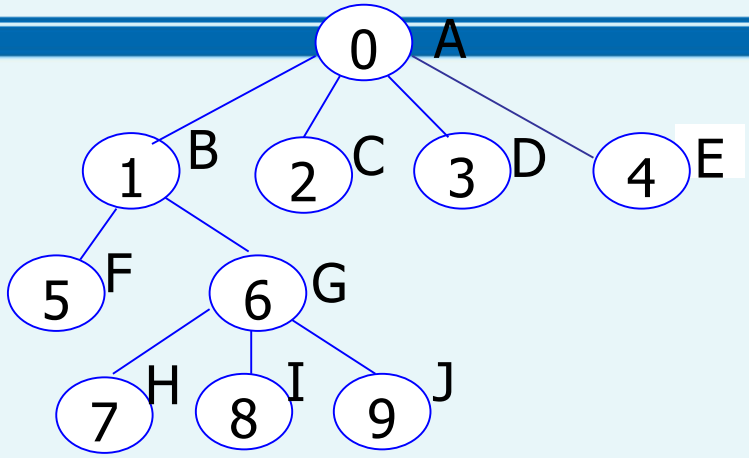
```
void Read_Tree(Tree &T){
  int i;
  printf("Cay co bao nhieu nut? ");    scanf("%d",&T.Max_Node);
 if (T.Max_Node > 0){
    fflush(stdin);
    printf("Nhap nhan (gia tri) cho nut goc (nut 0):");    scanf("%c",&T.Data[0]);
    T.Parent[0]=NI L;
    printf("Nhap cac nut con lai\n");
    for(i=1;i<=T.Max_Node-1;i++){
       printf("Cha cua nut %d la: ",i);  scanf("%d",&T.Parent[i]);
       printf("  Nhan (gia tri) cua nut %d la: ",i);
       fflush(stdin);
      scanf("%c",&T.Data[i]);
   }\
 }
}
```

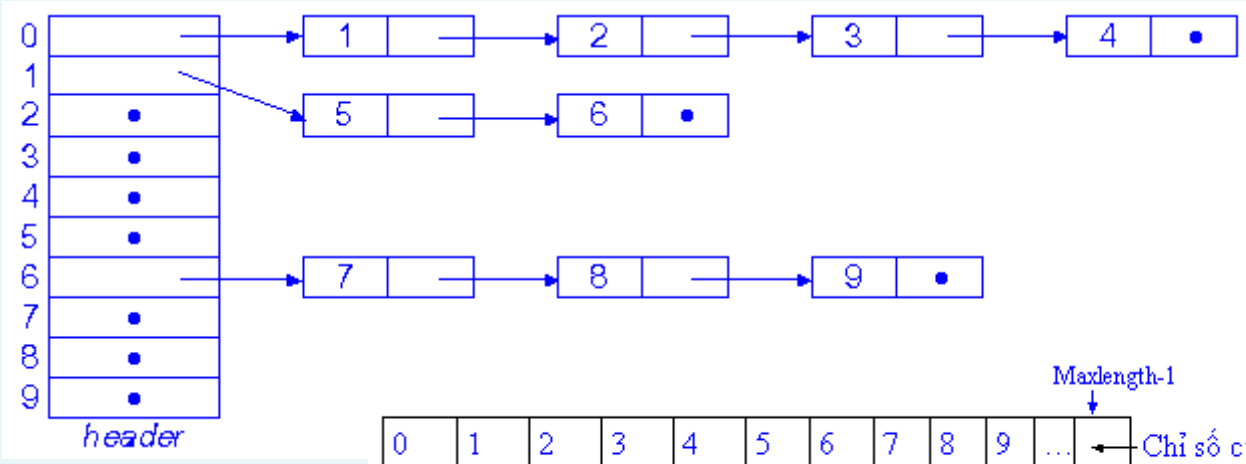
BÀI TẬP (3)

```
void main(){
    Tree T;
     Make_Null_Tree(T);
     printf("Nhap du lieu cho cay tong quat\n");
     Read_Tree(T);
     printf("Danh sach duyet tien tu cua cay la\n");
     Pre_Order(Root(T),T);
     printf("\nDanh sach duyet trung tu la\n");
     In_Order(Root(T),T);
     printf("\nDanh sach duyet hau tu cua cay la\n");
     Post_Order(Root(T),T);
     getch();
}
```

CÀI ĐẶT CÂY BẰNG DS CÁC NÚT CON (1)

- Minh họa





CÀI ĐẶT CÂY BẰNG DS CÁC NÚT CON (2)

- Mỗi nút có một danh sách các nút con
- Thường sử dụng cấu trúc danh sách liên kết để cài đặt các nút con do số lượng các nút con này biến động

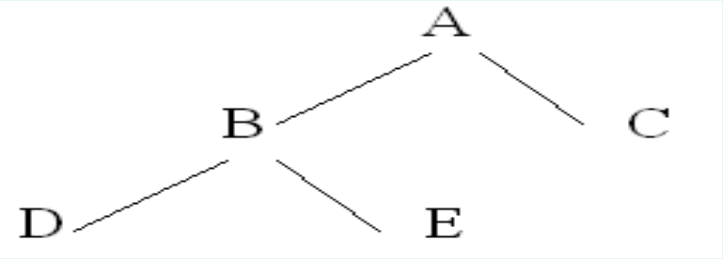
- Khai báo:

```
typedef int Node;
typedef ….. . Label_Type
typedef ….. .  List;
typedef struct
  { List header[Max_Length];
    Label_Type  Labels[Max_Length];
    Node  Root;
   }Tree;
```

CÀI ĐẶT CÂY THEO PHƯƠNG PHÁP CON TRÁI NHẤT VÀ ANH EM RUỘT PHẢI

CANTHO UNIVERSITY

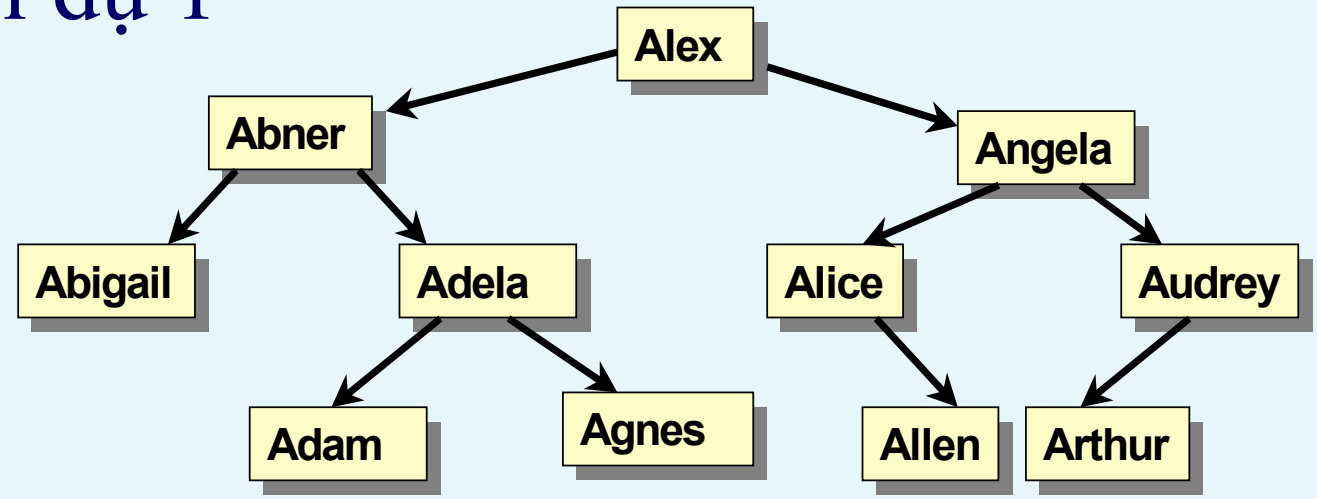
- Ví dụ



	Chỉ số	labels	leftmost_child	right_Sibling	parent
	1	D	null	4	3
Available	2			8	
	3	B	1	7	5
	4	E	null	null	3
Root	5	A	3	null	null
	6			null	
	7	C	null	null	3
	8			6	

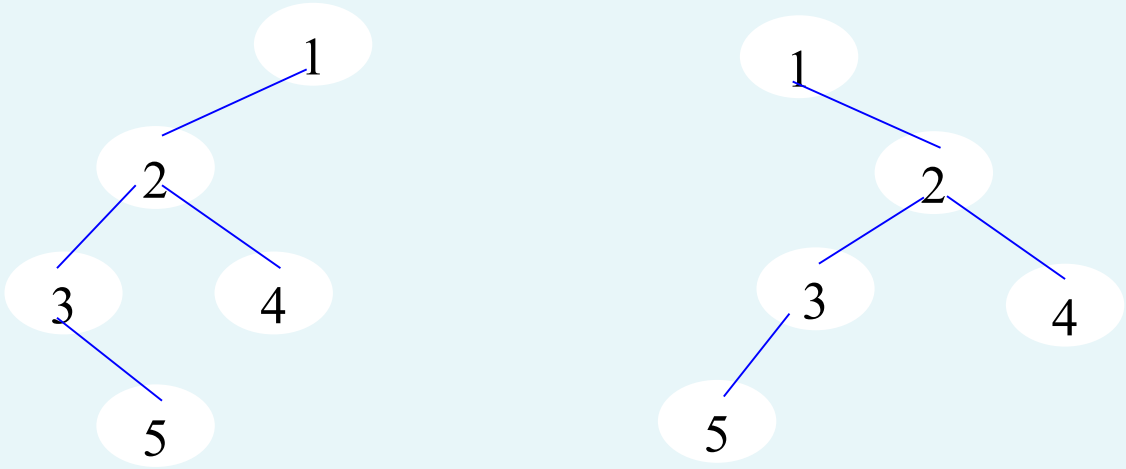
CÂY NHỊ PHÂN (1)

- Định nghĩa
  - Là cây rỗng hoặc mỗi nút có tối đa hai nút con
  - Hai nút con có thứ tự phân biệt rõ ràng
    - Con trái (left child): nằm bên trái nút cha
    - Con phải (right child): nằm bên phải nút cha
- Ví dụ 1



CÂY NHỊ PHÂN (2)

- Ví dụ 2

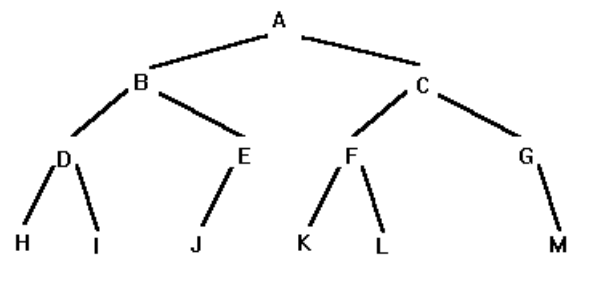


=>Là 2 cây nhị phân khác nhau

DUYỆT CÂY NHỊ PHÂN

- Các biểu thức duyệt: (N:Node, R:Right, L:Left)
  - Tiền tự (NLR): thăm nút gốc, duyệt tiền tự con trái, duyệt tiền tự con phải.
  - Trung tự (LNR): duyệt trung tự con trái, thăm nút gốc, duyệt trung tự con phải.
  - Hậu tự (LRN): duyệt hậu tự con trái, duyệt hậu tự con phải, thăm nút gốc.
- Danh sách duyệt trung tự của cây nhị phân có thể khác với DS duyệt trung tự theo cây tổng quát (do trong cây nhị phân con phải nằm bên phải)

Ví dụ về sự khác nhau của DS duyệt trung tự



	Các danh sách duyệt cây nhị phân	DS duyệt cây tổng quát
Tiền tự:	ABDHIEJCFKLGM	
Trung tự:	HDIBJEAKFLC**GM**	HDIBJEAKFLCMG
Hậu tự:	HIDJEBKLFMGCA	

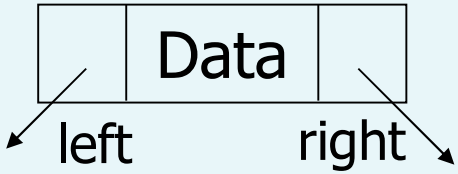
Ví dụ về sự khác nhau của DS duyệt trung tự

	Các danh sách duyệt cây nhị phân
Tiền tự:	ABDHIEJCFKLGM
Trung tự:	HDIBJEAKFLC**GM**
Hậu tự:	HIDJEBKLFMGCA

CÀI ĐẶT CÂY NHỊ PHÂN (1)

- Khai báo

```
typedef … Data_Type;
typedef struct Node
{Data_Type Data;
        Node* Left, Right;
};
typedef Node*  B_Tree;
```



Tạo cây rỗng
Đặt cây T = NULL

```
void Make_Null_Tree(B_Tree &T){
        T=NULL;
}
```

Kiểm tra cây rỗng

```
int Empty_Tree(B_Tree T){
    return T==NULL;
}
```

Xác định con trái

```
B_Tree Left_Child(B_Tree n){
        return n->Left;
}
```

Xác định con phải

```
B_Tree Right_Child(B_Tree n){
    return n->Right;
}
```

Kiểm tra xem một nút có phải là lá không

```
int Is_Leaf(B_Tree n){
  return (Left_Child(n)==NULL) &&  (Right_Child(n) ==
      NULL);
}
int Is_Leaf(B_Tree n){
  return (n->Left==NULL) && (n->Right == NULL);
}
int Is_Leaf(B_Tree n){
  return (Empty_Tree(n->Left)) && (Empty_Tree(n->Right));
}
```

Duyệt tiền tự

```
void Pre_Order(B_Tree T) {
    printf("%c ",T->Data);
    if (Left_Child(T)!=NULL)
        Pre_Order(Left_Child(T));
    if (Right_Child(T)!=NULL)
        Pre_Order(Right_Child(T));
}
```

Duyệt tiền tự

```
void Pre_Order(B_Tree T) {
    if (T!=NULL) {
        printf("%c ",T->Data);
        Pre_Order(Left_Child(T));
        Pre_Order(Right_Child(T));
    }
}
```

Duyệt trung tự

```
void In_Order(B_Tree T){
    if (Left_Child(T)!=NULL)
            In_Order(Left_Child(T));
    printf("%c ",T->Data);
    if(Right_Child(T)!=NULL)
            In_Order(Right_Child(T));
}
```

Duyệt trung tự

```
void In_Order(B_Tree T){
    if (T!=NULL) {
            In_Order(Left_Child(T));
            printf("%c ",T->Data);
           In_Order(Right_Child(T));
    }
}
```

Duyệt hậu tự

```
void Pos_Order(B_Tree T){
    if(Left_Child(T)!=NULL)
            Pos_Order(Left_Child(T));
    if(Right_Child(T)!=NULL)
            Pos_Order(Right_Child(T));
    printf("%c ",T->Data);
}
```

Duyệt hậu tự

```
void Pos_Order(B_Tree T){
    if(T!=NULL){
            Pos_Order(Left_Child(T));
            Pos_Order(Right_Child(T));
           printf("%c ",T->Data);
    }
}
```

Xác định số nút trong cây

```
int Nb_Nodes(B_Tree T){
    if(Empty_Tree(T)) return 0;
   return 1 + Nb_Nodes(Left_Child(T)) +
              Nb_Nodes(Right_Child(T));
 }
```

Tạo cây mới từ hai cây có sẵn

- B_Tree **Create2**(v, L, R): Trả về cây NP
- Cấp phát một nút N
- Đặt Data của N bằng v
- Đặt Left của N bằng L
- Đặt Right của N bằng R
- Trả về N

Tạo cây mới từ hai cây có sẵn

```
B_Tree Create2(Data_Type v, B_Tree L, B_Tree R)
{       B_Tree N;
        N=(Node*)malloc(sizeof(Node));
        N->Data=v;
        N->Left=L;
        N->Right=R;
        return N;
}
```

Bài tập: Tìm một giá trị trên cây nhị phân

- Hàm Find_B_Tree(Data_Type X, B_Tree T)
- Input: Giá trị X, Cây NP T
- Output: trả về 1 nếu tồn tại một node trên cây T có giá trị bằng X, ngược lại trả về 0

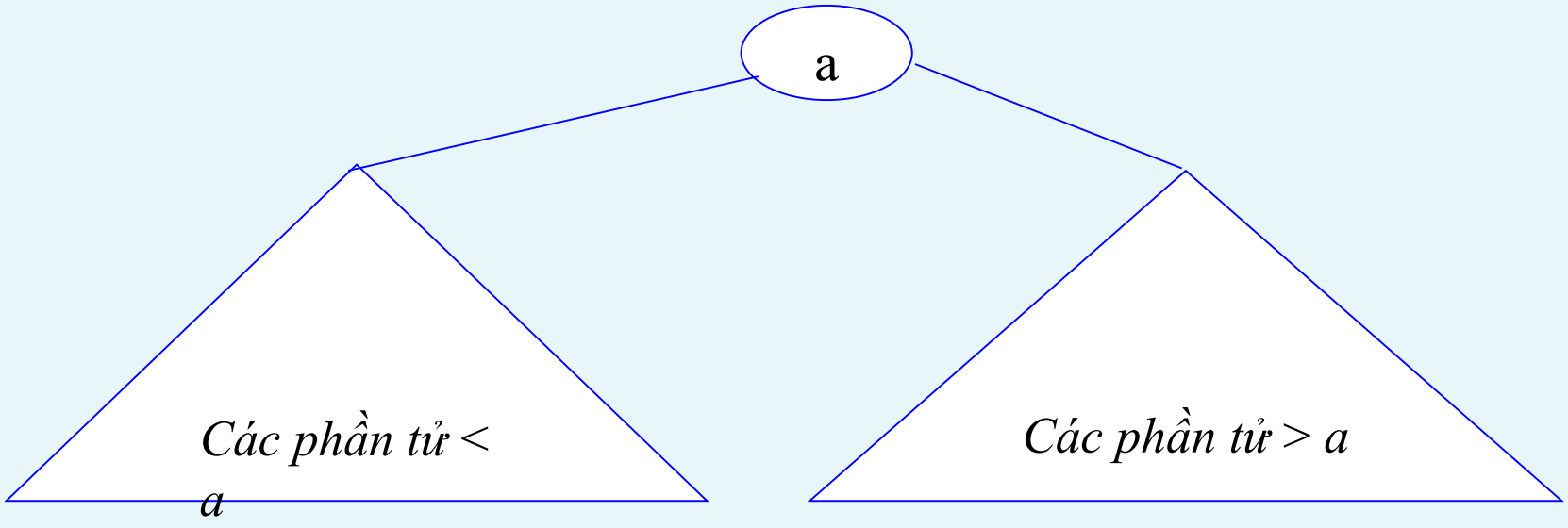
CÂY TÌM KIẾM NHỊ PHÂN (Binary Search Tree - BST)

CANTHO UNIVERSITY

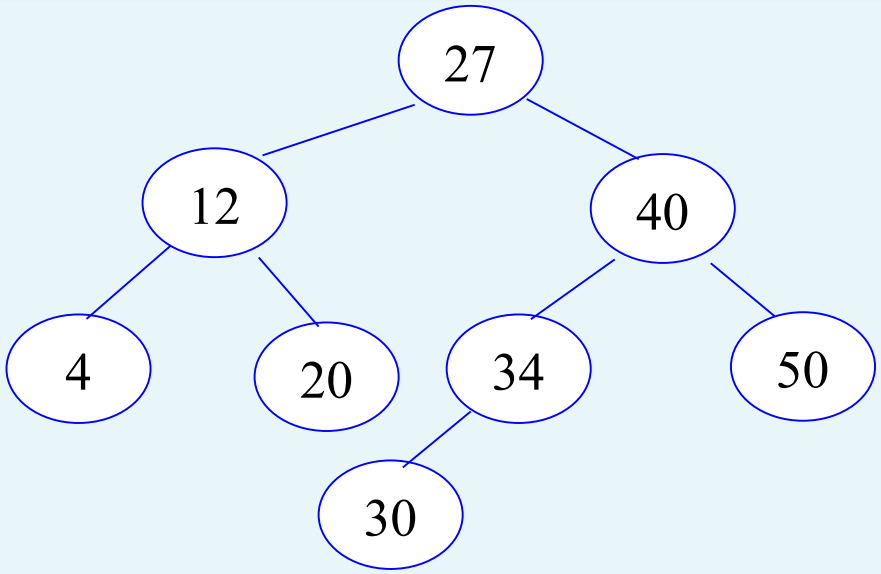
- **Định nghĩa**

  Cây BST là cây nhị phân mà nhãn tại mỗi nút lớn hơn nhãn của tất cả các nút thuộc cây con bên trái và nhỏ hơn nhãn của tất cả các nút thuộc cây con bên phải.

- **Ưu điểm:** ***Cho phép tìm kiếm nhanh***



CÂY TÌM KIẾM NHỊ PHÂN



- Nhận xét
  - Trên cây BST không có 2 nút trùng khóa.
  - Cây con của 1 BST là 1 BST.
  - Biểu thức duyệt trung tự là dãy nhãn có giá trị tăng: 4, 12, 20, 27, 30, 34, 40, 50

CÀI ĐẶT CÂY BST

- Khai báo

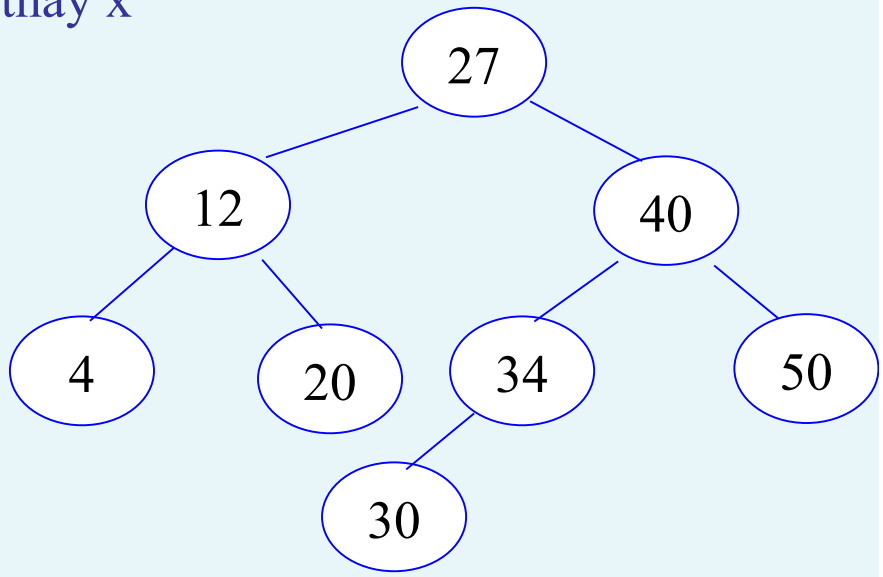
```
typedef ... Key_Type;
typedef struct Node
   {Key_Type Key;
    Node* Left, Right;
   }
typedef Node* BST;
```

CÀI ĐẶT CÂY BST

- Tìm kiếm một nút có khoá X:
- Input: Khóa X, BST Root (Nút Gốc)
- Output: Node có Key = X / NULL
- Thuật toán:
  - Nếu nút gốc bằng NULL thì khóa X không có trên cây.
  - Nếu X bằng khóa nút gốc thì giải thuật dừng vì đã tìm gặp X trên cây.
  - Nếu X nhỏ hơn khoá nút gốc: tìm X trên cây con bên trái
  - Nếu X lớn hơn khoá nút gốc: tìm X trên cây con bên phải

CÀI ĐẶT CÂY BST

```
BST Search(Key_Type x, BST Root){
    if (Root == NULL) return Root;//không tìm thấy x
    if (Root->Key == x) // tìm thấy khoá x
            return Root;
    if (Root->Key < x)
            //tìm tiếp trên cây bên phải
            return Search(x, Root->Right);
    //tìm tiếp trên cây bên trái
    return Search(x, Root->Left);
}
```



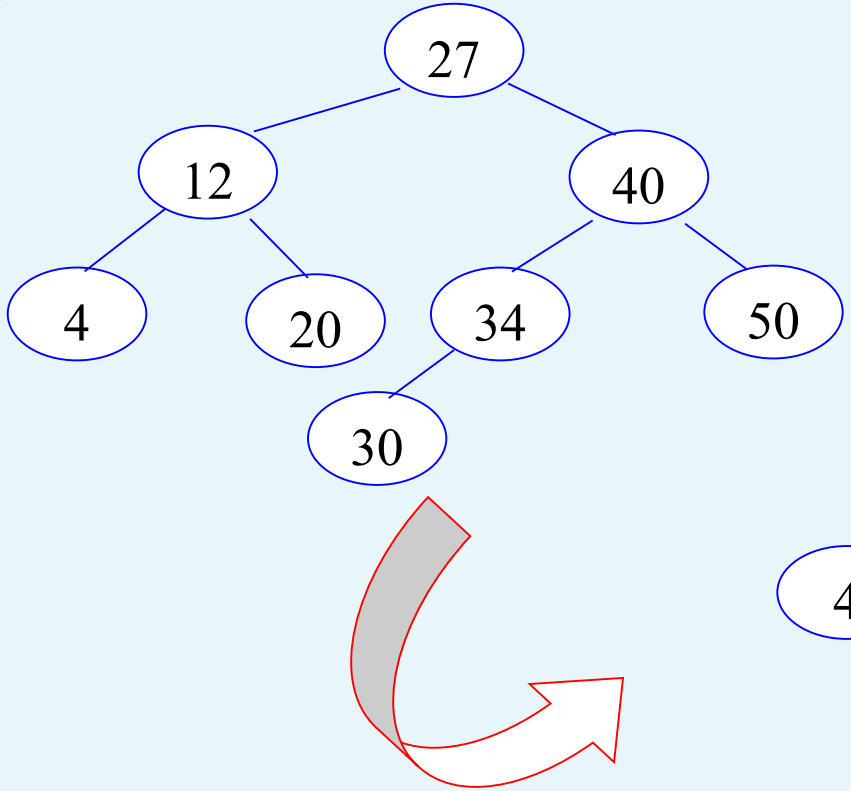
CÀI ĐẶT CÂY BST

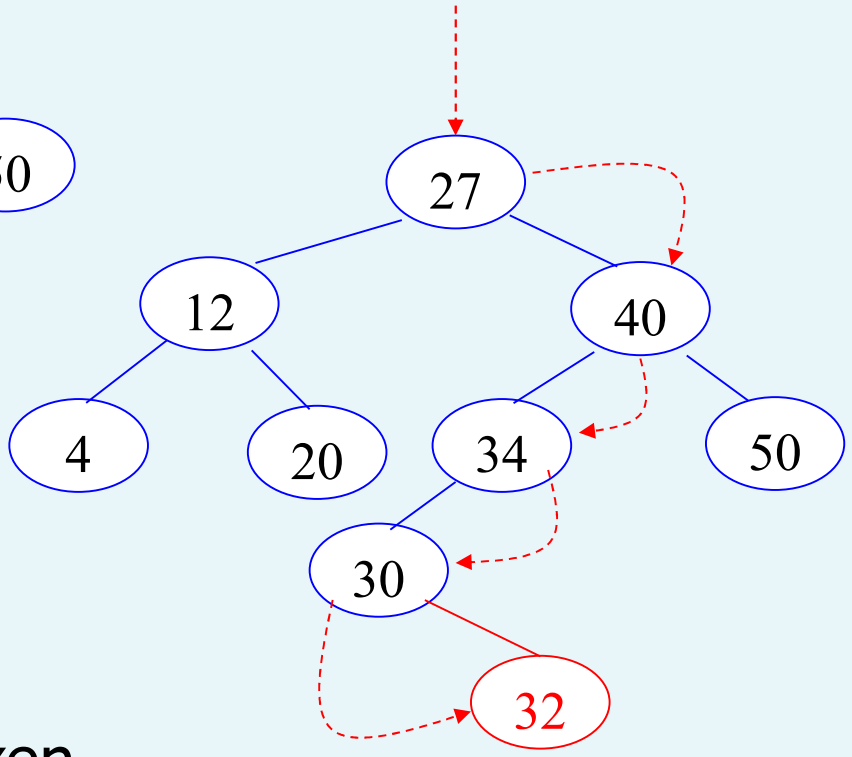
- Thêm một nút có khoá x vào cây: Muốn thêm 1 nút có khóa X vào cây BST, trước tiên ta phải tìm kiếm xem đã có X trên cây chưa.

  Nếu có thì giải thuật kết thúc, nếu chưa thì ta mới thêm vào. Việc thêm vào không làm phá vỡ tính chất cây BST.
  - Giải thuật thêm vào như sau: bắt đầu từ nút gốc ta tiến hành các bước sau:
  - Nếu nút gốc bằng NULL thì khóa X chưa có trên cây, do đó ta thêm 1 nút mới.
  - Nếu X bằng khóa nút gốc thì giải thuật dừng vì X đã có trên cây.
  - Nếu X nhỏ hơn khoá của nút gốc: thêm X vào cây con bên trái
  - Nếu X lớn hơn khoá của nút gốc: thêm X vào cây con bên phải

CÀI ĐẶT CÂY BST

- Ví dụ: Xen nút có khóa 32





Các thao tác xen

CÀI ĐẶT CÂY BST

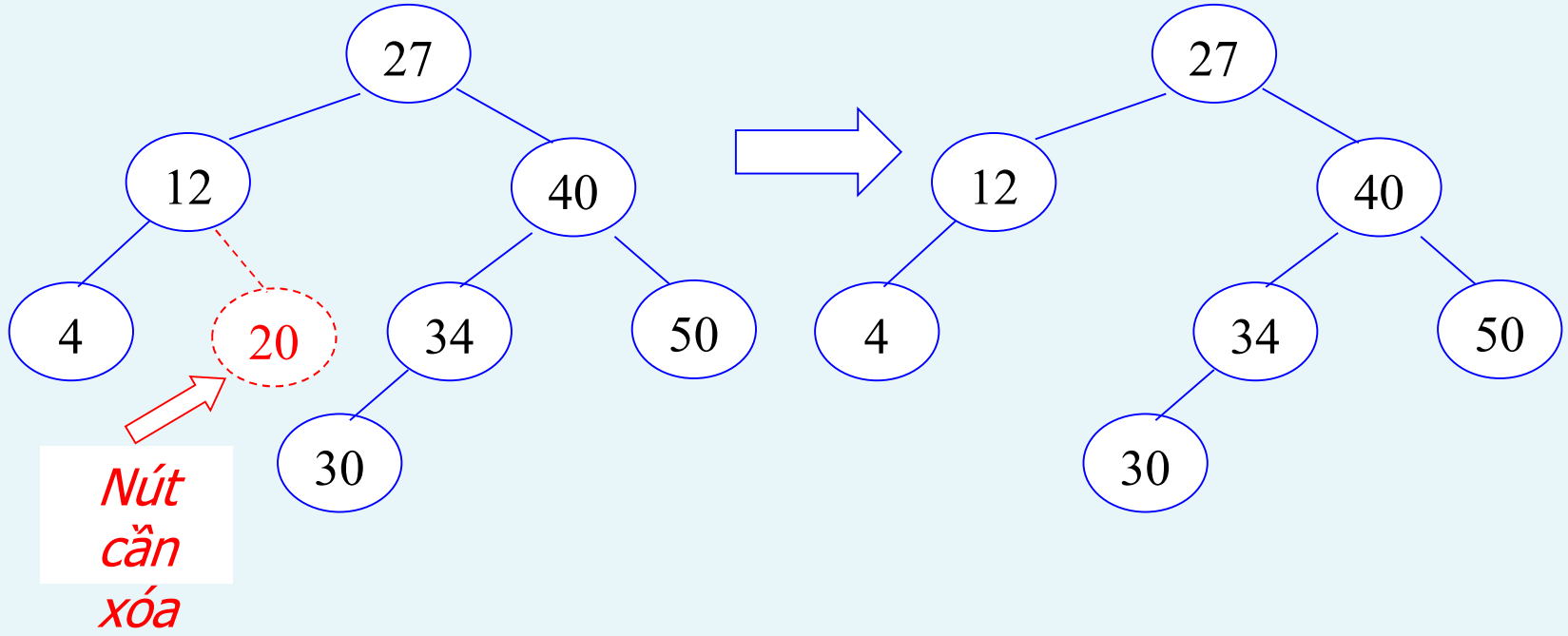
```
void Insert_Node(Key_Type x,BST &Root ){
 if (Root == NULL) //thêm nút mới chứa khoá x
{    Root=(Node*)malloc(sizeof(Node));
     Root->Key = x;
     Root->Left = NULL;
     Root->Right = NULL;
     }
 else if (x < Root->Key)
              Insert_Node(x,Root->Left);
       else if (x>Root->Key)
                    Insert_Node(x,Root->Right);
  }
```

CÀI ĐẶT CÂY BST

- Xóa một nút khóa X khỏi cây
  - Muốn xóa 1 nút có khóa X trên cây BST. Trước tiên ta phải tìm xem có X trên cây không.
  - Nếu không thì giải thuật kết thúc
  - Nếu gặp nút N chứa khóa X, có 3 trường hợp xảy ra

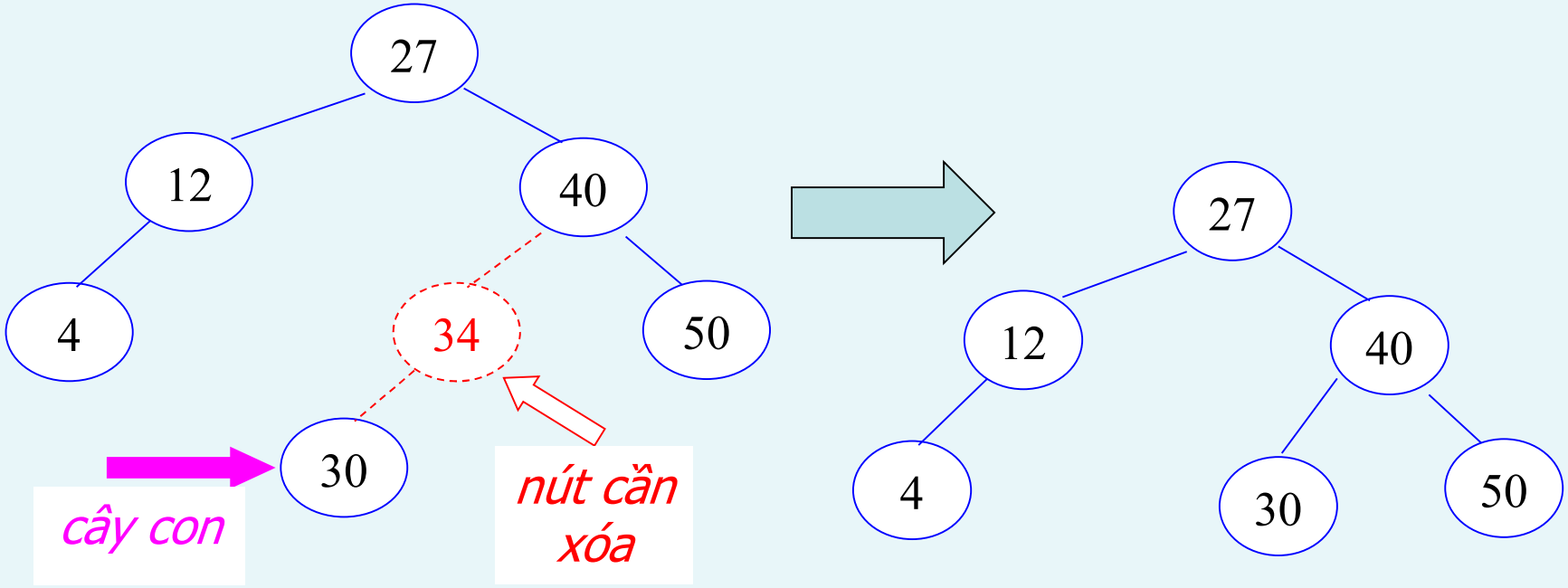
CÀI ĐẶT CÂY BST

- Trường hợp 1:
  - N là nút lá: thay nút này bởi NULL
  - Ví dụ: Xóa nút nhãn 20



CÀI ĐẶT CÂY BST

- Trường hợp 2
  - N có một cây con: thay nút này bởi cây con của nó
  - Ví dụ: xóa nút có nhãn 34

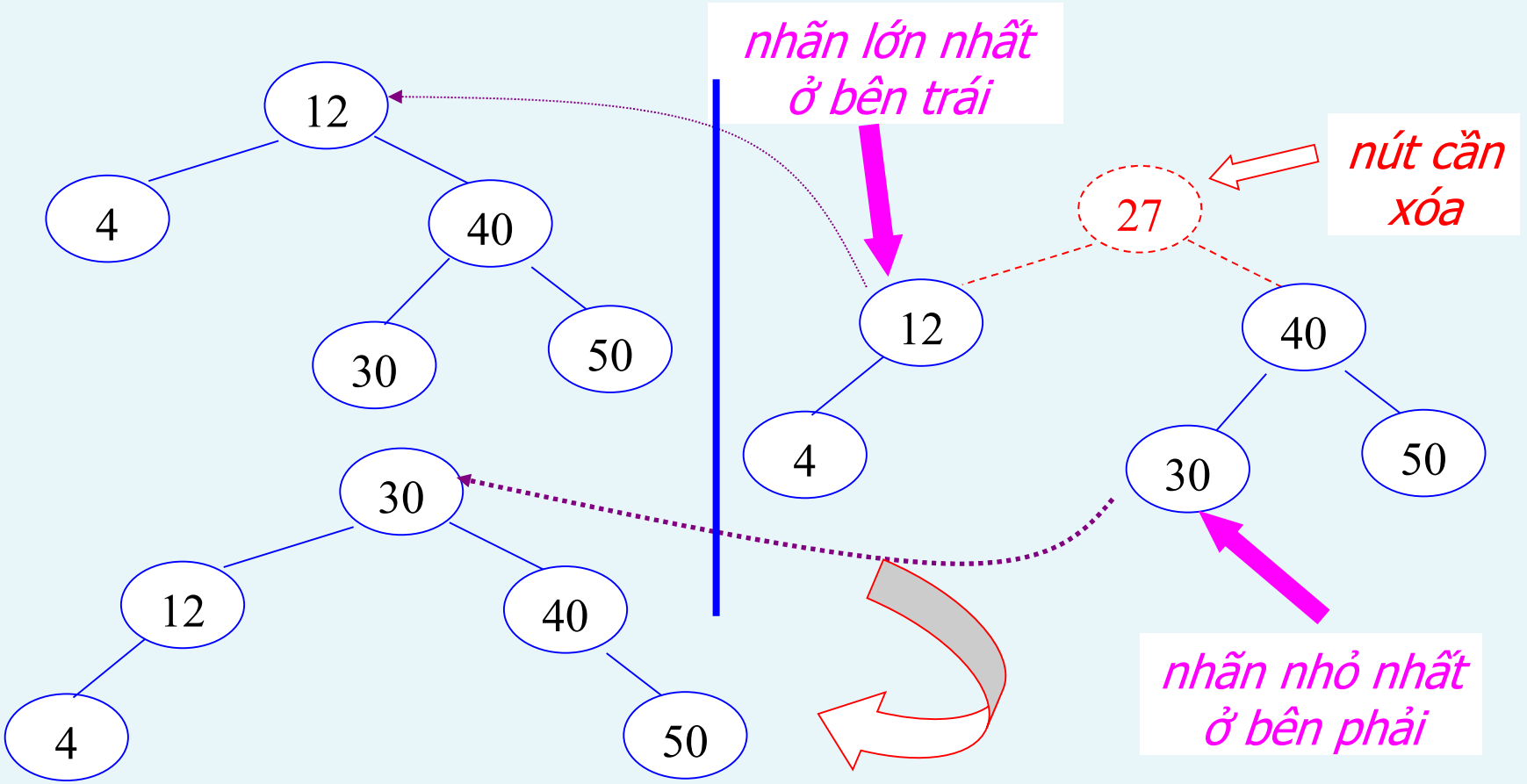


CÀI ĐẶT CÂY BST

- Trường hợp 3
  - N có hai cây con: thay giá trị của nút này bởi
    - nhãn lớn nhất của cây con bên trái và xóa nút có nhãn lớn nhất này hoặc
    - nhãn nhỏ nhất của cây con bên phải và xóa nút có nhãn nhỏ nhất này

CÀI ĐẶT CÂY BST

- Ví dụ: Xoá nút có nhãn 27



CÀI ĐẶT CÂY BST

```
void Delete_Node(Key_Type X, BST &Root) {
  if (Root!=NULL)
      if(X < Root->Key) Delete_Node(X, Root->Left);
      else if(X > Root->Key) Delete_Node(X, Root->Right);
          else if (Is_Leaf(Root))
                  Root=NULL; // Trường hợp 1
              else if(Root->Left == NULL)       Root = Root->Right;
                  else if(Root->Right==NULL) Root = Root->Left;
                                  // Trường hợp 2
                          else Root->Key=Delete_Min(Root->Right);
                          // Trường hợp 3: thay nhãn bằng nhãn nhỏ nhất của
    cây con bên phải và xóa nút có nhãn nhỏ nhất này
}
```

CÀI ĐẶT CÂY BST

```
Key_Type Delete_Min (BST &Root ){
    Key_Type k;
    if (Root->Left == NULL){
        k=Root->Key;
        Root = Root->Right;
        return k;
    }
     else
        return Delete_Min(Root->Left);
}
```

BÀI TẬP

- Viết hàm Delete_Node trong đó, ở trường hợp thứ 3, thay giá trị của nút cần xoá bằng giá trị lớn nhất của cây con trái và xóa nút có giá trị lớn nhất này

CÀI ĐẶT CÂY BST

```
void Delete_Node(Key_Type X, BST &Root) {
  if (Root!=NULL)
      if(X < Root->Key) Delete_Node(X, Root->Left)
      else if(X > Root->Key) Delete_Node(X, Root->Right)
          else if(Root->Left==NULL)&&(Root->Right==NULL)
                  Root=NULL; // Trường hợp 1
              else if(Root->Left == NULL)       Root = Root->Right
                  else if(Root->Right==NULL) Root = Root->Left
                                  // Trường hợp 2
                          else Root->Key=Delete_Max(Root->Left);
                          // Trường hợp 3: thay nhãn bằng nhãn lớn nhất của cây con bên
    trái và xóa nút có nhãn lớn nhất này
}
```

CÀI ĐẶT CÂY BST

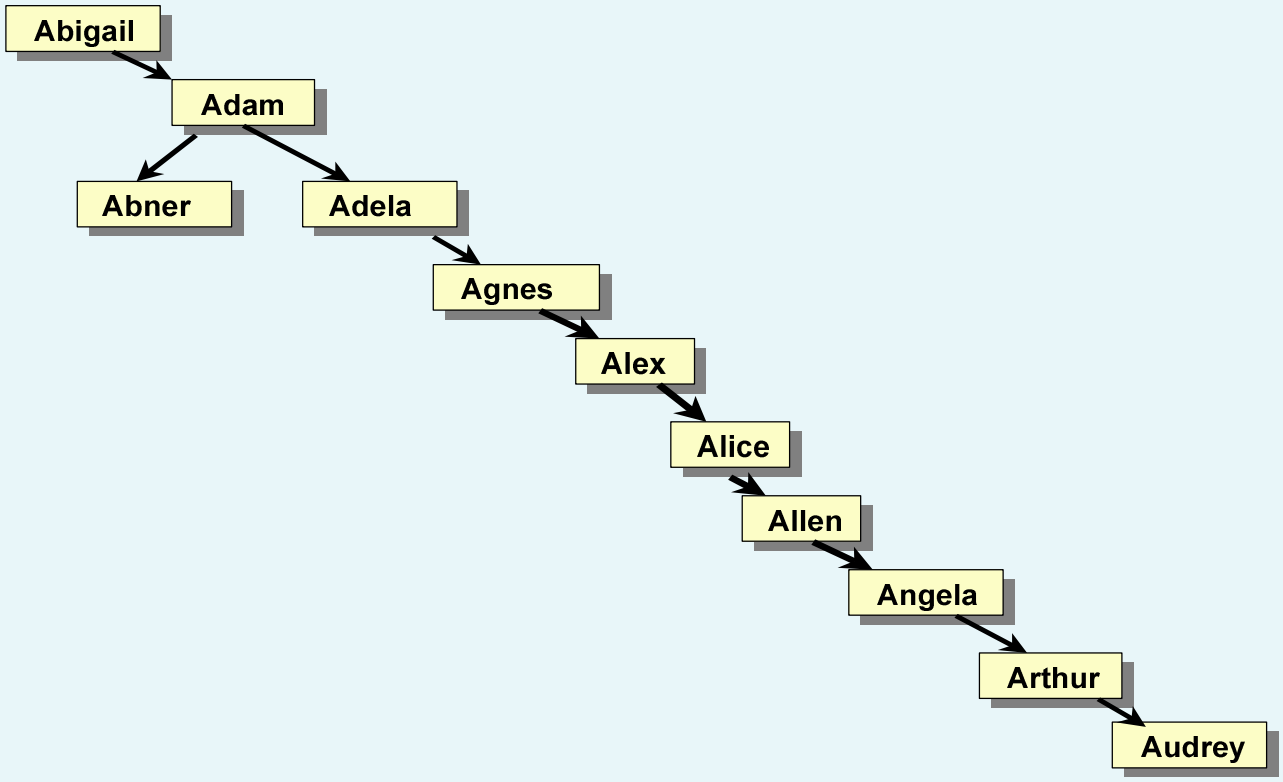
```
Key_Type Delete_Max (BST &Root ){
    Key_Type k;
    if (Root->Right == NULL){
        k=Root->Key;
        Root = Root->Left;
        return k;
        }
     else Delete_Max(Root->Right);
    }
```

KIẾN THỨC BỔ SUNG (1)

- Thời gian tìm kiếm một giá trị trên một cây TKNP có N nút là:
  - O(logN) nếu cây "cân bằng" (balanced)
  - O(N) nếu cây "không cân bằng" (unbalanced)

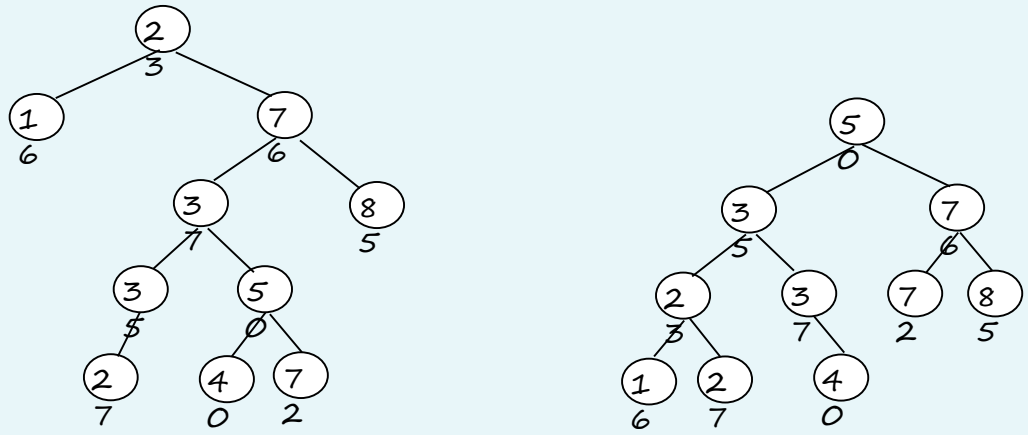
KIẾN THỨC BỔ SUNG (2)

- Ví dụ về một cây TKNP phân "không cân bằng"



CÂY CÂN BẰNG AVL

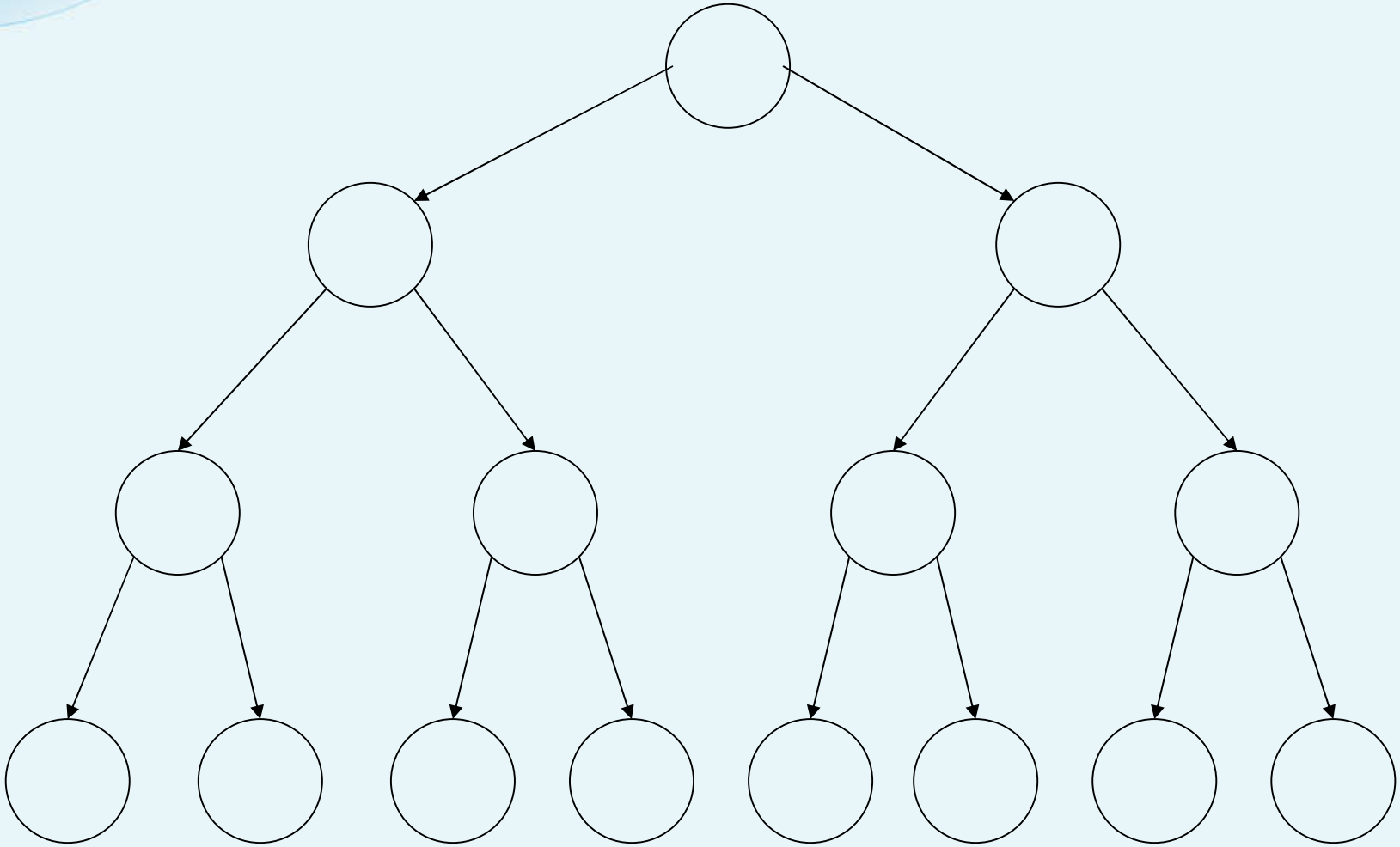
- Cây cân bằng (AVL) là một **cây tìm kiếm nhị phân** mà tại mỗi nút chiều cao của hai cây con sai khác nhau không quá một.



CÂY NHỊ PHÂN ĐẦY ĐỦ (1)
(full binary tree)

- Một cây nhị phân là "**cây nhị phân đầy đủ**" nếu và chỉ nếu
  - Mỗi nút không phải lá có chính xác 2 nút con
  - Tất cả các nút lá có chiều cao bằng nhau

CÂY NHỊ PHÂN ĐẦY ĐỦ (2)



CÂY NHỊ PHÂN ĐẦY ĐỦ (3)

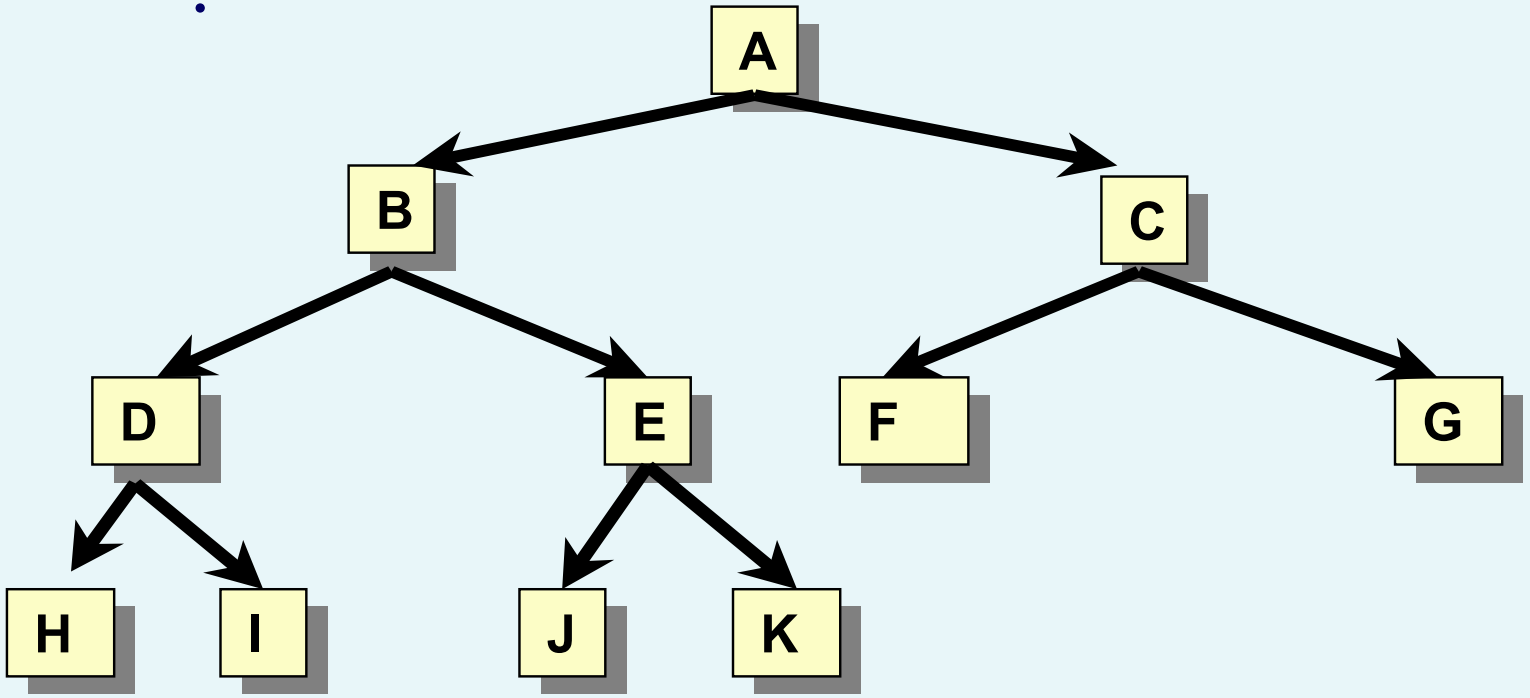
- **Câu hỏi về cây nhị phân đầy đủ:**
  - **Một cây nhị phân đầy đủ chiều cao h sẽ có bao nhiêu nút lá?**
  - **Một cây nhị phân đầy đủ chiều cao h sẽ có tất cả bao nhiêu nút?**

CÂY NHỊ PHÂN HOÀN CHỈNH (1) (complete binary tree)

- Một cây nhị phân hoàn chỉnh (về chiều cao) thỏa mãn các điều kiện sau:
  - Mức 0 đến h-1 là trình bày một cây nhị phân đầy đủ chiều cao h-1
  - Một hoặc nhiều nút ở mức h-1 có thể có 0, hoặc 1 nút con
  - Nếu j, k là các nút ở mức h-1, khi đó j có nhiều nút con hơn k nếu và chỉ nếu j ở bên trái của k

CÂY NHỊ PHÂN HOÀN CHỈNH (2)

- Ví dụ



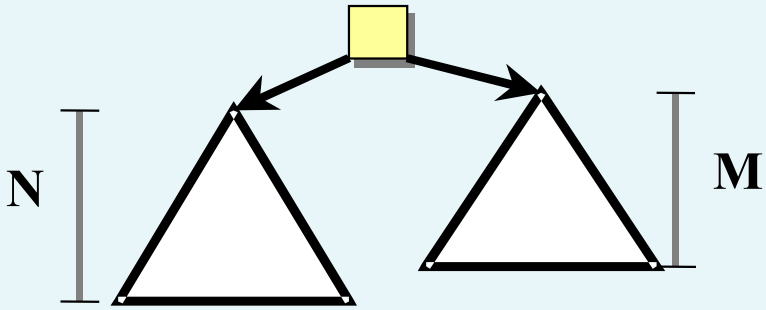
CÂY NHỊ PHÂN HOÀN CHỈNH (3)

- Được cho một tập hợp N nút, một cây nhị phân hoàn chỉnh của những nút này cung cấp số nút lá nhiều nhất - với chiều cao trung bình của mỗi nút là nhỏ nhất
- Cây hoàn chỉnh n nút phải chứa ít nhất một nút có chiều cao là $\lfloor \log n \rfloor$

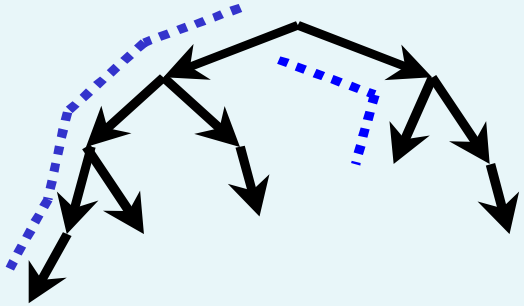
CÂY NHỊ PHÂN CÂN BẰNG VỀ CHIỀU CAO (Height-balanced Binary Tree)

- Một cây nhị phân **cân bằng về chiều cao** là một cây nhị phân như sau:
  - Chiều cao của cây con trái và phải của bất kỳ nút nào khác nhau không quá một đơn vị
  - Chú ý: mỗi cây nhị phân hoàn chỉnh là một cây cân bằng về chiều cao

CÂY CÂN BẰNG VỀ CHIỀU CAO – VÍ DỤ



N - M<=1



Cân bằng về chiều cao là một thuộc tính cục bộ

ƯU ĐIỂM CỦA CÂY CÂN BẰNG

- Cây nhị phân cân bằng về chiều cao là cây "cân bằng"
- Thời gian tìm kiếm một nút trên cây N nút là O(logN)

Thank you