CÁC KIỂU DỮ LIỆU TRỪU TƯỢNG CƠ BẢN

CẤU TRÚC DỮ LIỆU CÂY

Đỗ Thanh Nghị

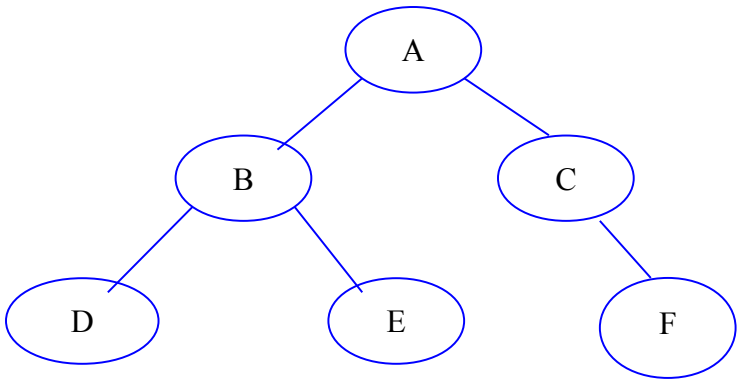
dtnghi@cit.ctu.edu.vn

NỘI DUNG

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

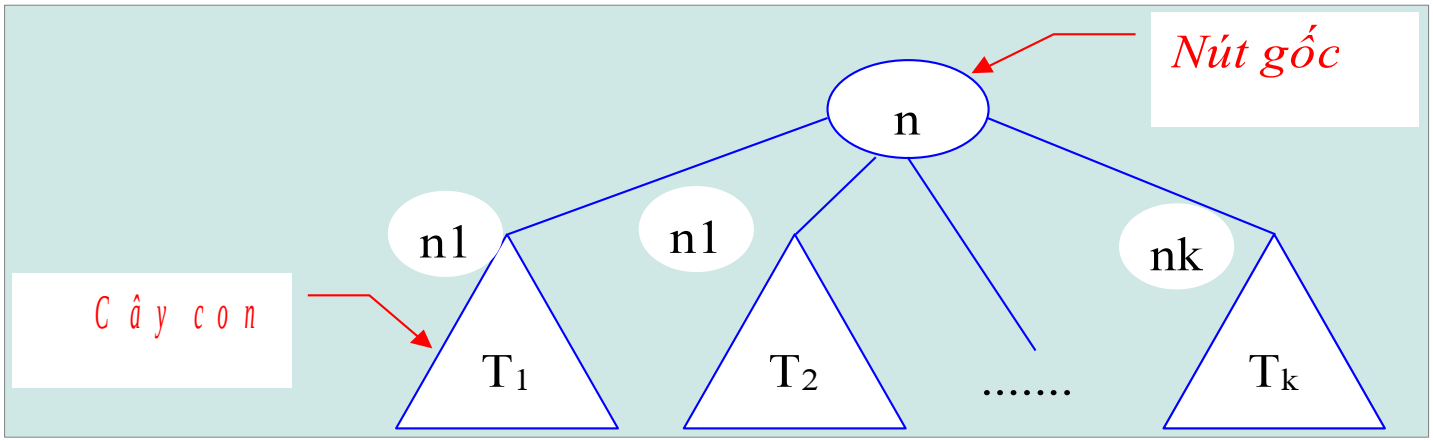
CÁC THUẬT NGỮ CƠ BẢN (1)

- Định nghĩa
  - Cây (tree)
  - Nút (nodes)
  - Ví dụ:



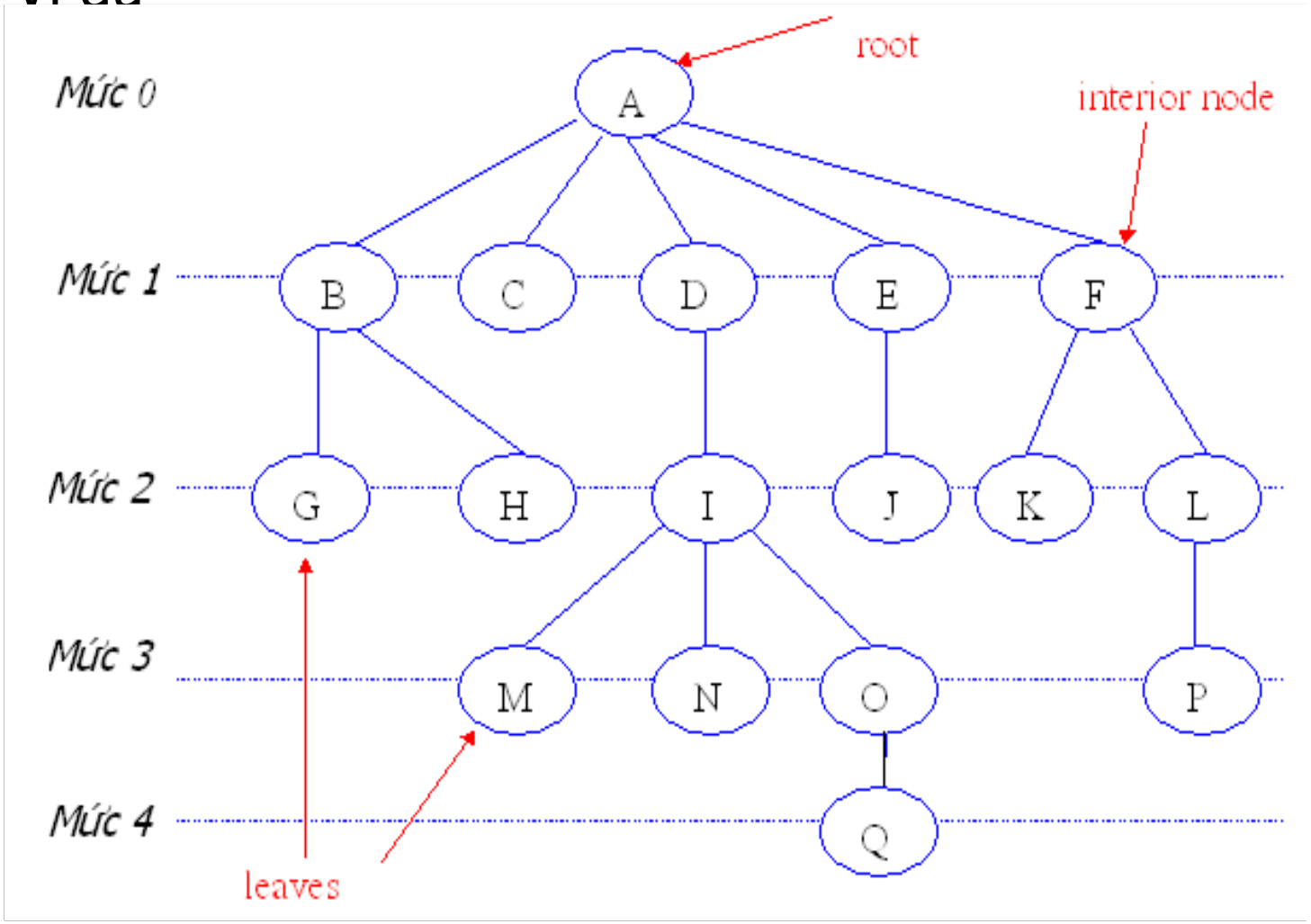
CÁC THUẬT NGỮ CƠ BẢN (2)

- Cây: cấu trúc đệ qui



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 , ko phải nút lá
  - 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)

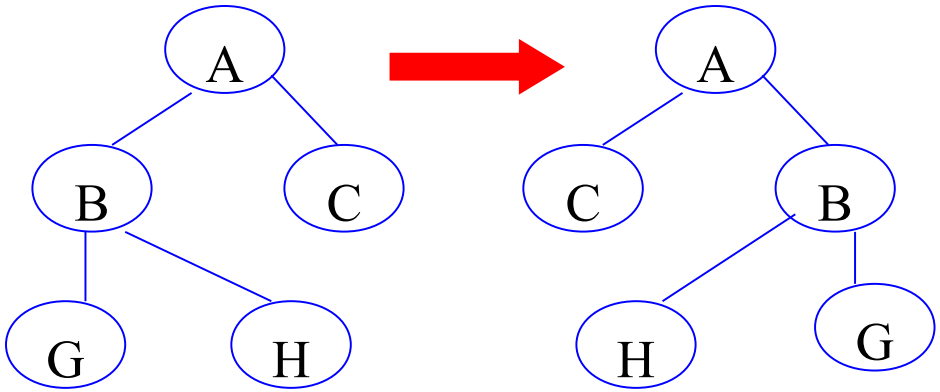
- Đường đi là một chuỗi các nút n1, n2, ..., nk trên cây sao cho n_i là nút cha của nút n_{i+1} (i=1..k-1)
  - VD: có đường đi A, D, I, O, Q
- Độ dài đường đi bằng số nút trên đường đi trừ 1
  - VD: độ dài đường đi A,D,I,O,Q = 5-1=4
- Chiều cao của 1 nút là độ dài đường đi từ nút đó đến nút lá xa nhất.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

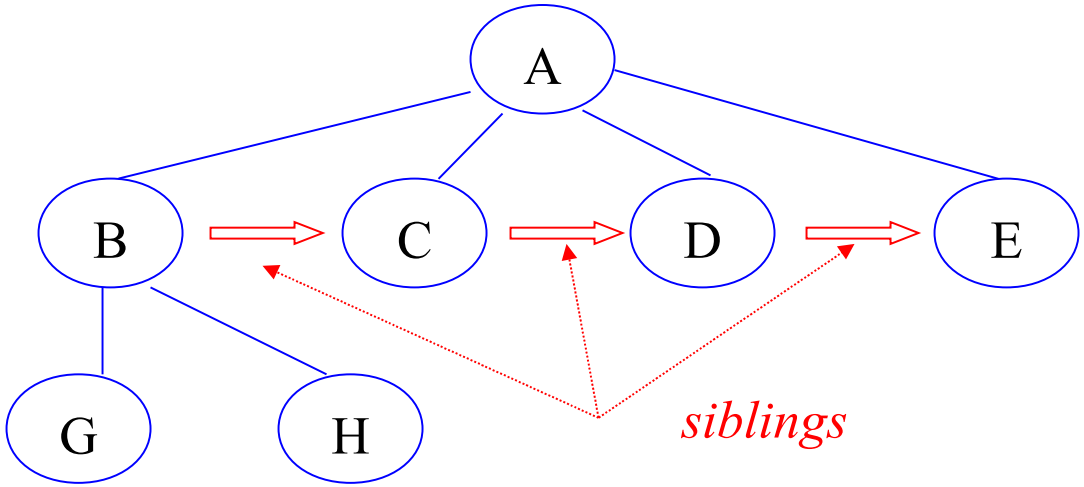
- Độ sâu của 1 nút là độ dài đường đi từ nút gốc đến nút đó, hay còn gọi là mức (level) của nút đó.
  - VD: I có độ sâu 2, E có độ sâu 1
    M, N, O, P có cùng mức 3

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 đó là 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 (siblings)
- Mở rộng: nếu n_i và n_k là hai nút anh em và nút n_i ở bên trái nút n_k thì các hậu duệ của nút n_i là bên trái mọi hậu duệ của nút n_k

CÁC THUẬT NGỮ CƠ BẢN (10)

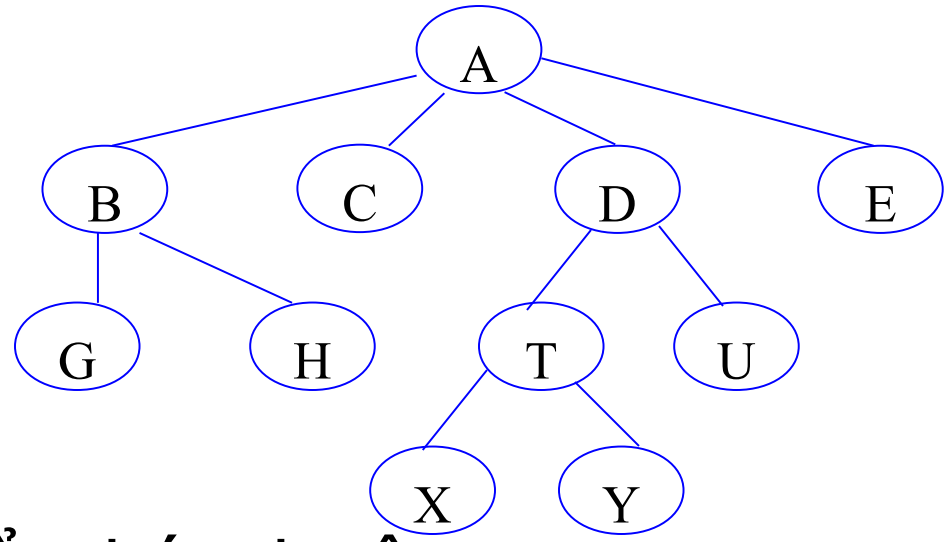
- Duyệt cây:
  - Quy tắc: đi qua lần lượt tất cả các nút của cây, mỗi nút đúng một lần
  - Danh sách duyệt cây: là danh sách liệt kê các nút theo thứ tự đi qua
  - Có 3 phương pháp duyệt tổng quát:
    - tiền tự (preorder)
    - trung tự (inorder)
    - hậu tự (posorder)

CÁC THUẬT NGỮ CƠ BẢN (11)

- Định nghĩa theo 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)

- Ví dụ

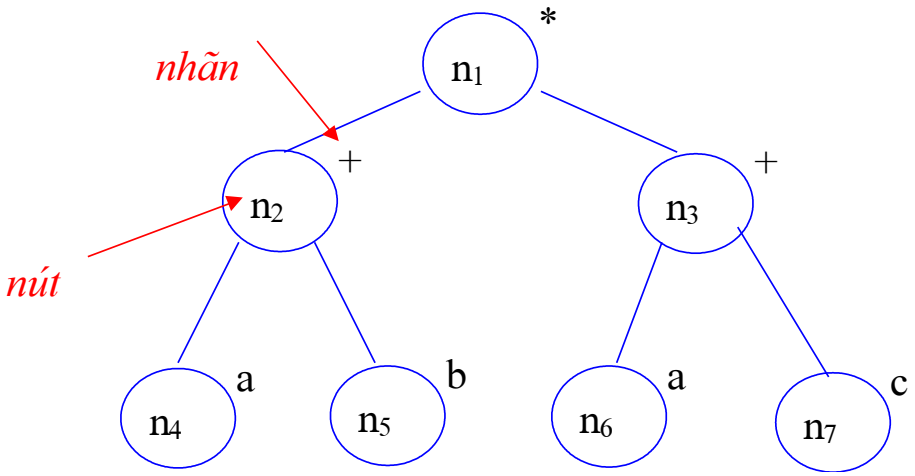


=>Các biểu thức duyệt:

- 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 (16)

- Cây có nhãn và cây biểu thức
  (labeled trees and expression trees)



- Lưu trữ kết hợp một nhãn (label) hoặc một giá trị 1(value) với một nút trên cây
- Nhãn: giá trị được lưu trữ tại nút đó, còn gọi là khóa của nút
- VD: (a+b)*(a+c)

CÁC PHÉP TOÁN CƠ BẢN TRÊN CÂY

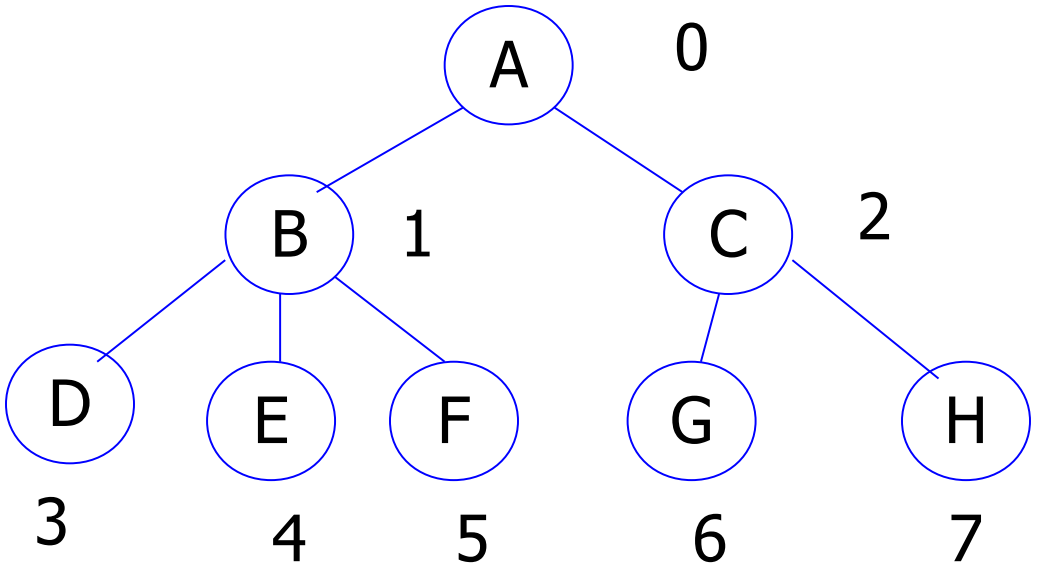
Tên hàm	Diễn giải
MAKENULL(*T*)	Tạo cây T rỗng
EMPTY(*T*)	Kiểm tra xem cây T có rỗng không?
ROOT(*T*)	Trả về nút gốc của cây T
PARENT(*n, T*)	Trả về cha của nút n trên cây T
LEFTMOST_CHILD(*n, T*)	Trả về con trái nhất của nút n
RIGHT_SIBLING(*n, T*)	Trả về anh em phải của nút n
LABEL(*n, T*)	Trả về nhãn của nút n
CREATEi(*v*, T_1, T_2,…,T_i)	Tạo cây mới có nút gốc n nhãn là v, và có i cây con. Nếu n=0 thì cây chỉ có một nút n

CÁC PHƯƠNG PHÁP CÀI ĐẶT CÂY

- CÀI ĐẶT CÂY BẰNG MẢNG
- CÀI ĐẶT CÂY BẰNG DANH SÁCH CÁC NÚT CON
- CÀI ĐẶT CÂY THEO PHƯƠNG PHÁP CON TRÁI NHẤT VÀ ANH EM PHẢI
- CÀI ĐẶT CÂY BẰNG CON TRỎ

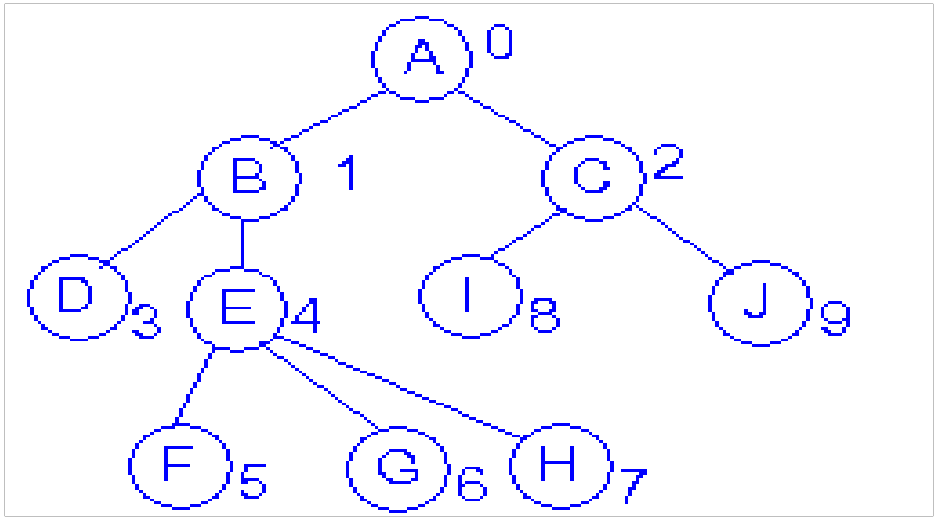
CÀI ĐẶT CÂY BẰNG MẢNG (1)

- Mô hình



	0	1	2	3	4	5	6	7 ← Maxnode
T_parent	-1	0	0	1	1	1	2	2
T_label	A	B	C	D	E	F	G	H

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 trên
Cha của nút trên cây
Chỉ số của mảng

Maxnode

Maxlength-1

CÀI ĐẶT CÂY BẰNG MẢNG (3)

- Khai báo

```
#define MAXLENGTH ... //chỉ số tối đa của mảng
#define NIL -1
typedef  ... DataType;
typedef int Node;
typedef struct {
   DataType Data[MAXLENGTH]; //nhãn (dliệu) của cây
   Node Parent[MAXLENGTH];     //nút cha của các nút
   int MaxNode; //Số nút thực sự trong cây
} Tree;
Tree T;
```

CÀI ĐẶT CÂY BẰNG MẢNG

- **Khởi tạo cây rỗng:**

```
void MakeNull_Tree (Tree *T) {
    (*T).MaxNode=0;
}
```

- **Kiểm tra cây rỗng**

```
int EmptyTree(Tree T) {
    return T.MaxNode == 0;
}
```

- **Xác định nút cha của nút trên cây**

```
Node Parent(Node n, Tree T) {
    if(EmptyTree(T)||(n>T.MaxNode-1))
                return NIL;
    else return T.Parent[n];
}
```

CÀI ĐẶT CÂY BẰNG MẢNG (5)

- **Đọc nhãn của nút trên cây**

```
DataType Label_Node(Node n,Tree T) {
    if(!EmptyTree(T)&&(n<=T.MaxNode-1))
            return T.Data[n];
}
```

- **Hàm trả về nút gốc trong cây**

```
Node Root(Tree T) {
  if (!EmptyTree(T)) return 0;
  else return NIL;
 }
```

CÀI ĐẶT CÂY BẰNG MẢNG (6)

- **Hàm trả về con trái nhất của một nút**

```
Node LeftMostChild(Node n, Tree T) {
   Node i; int found;
   if (n<0) return NIL;
   i=n+1; found=0;
   while ((i<=T.MaxNode-1) && !found)
         if (T.Parent[i]==n) found=1;
         else i=i+1;
   if (found) return i;
   else return NIL;
}
```

CÀI ĐẶT CÂY BẰNG MẢNG (7)

- **Hàm xác định anh em phải của một nút**

```
Node RightSibling(Node n,Tree T) {
    Node i,parent;  int found;
    if (n<0) return NIL;
    parent=T.Parent[n];
    i=n+1; found=0;
    while ((i<=T.MaxNode-1) && !found)
        if (T.Parent[i]==parent) found=1;
        else i=i+1;
    if (found) return i;
    else return NIL;
}
```

CÀI ĐẶT CÂY BẰNG MẢNG (8)

- **Thủ tục duyệt tiền tự**

```
void PreOrder(Node n,Tree T) {
  if (n != NIL) {
    Node i;
    printf("...", Label_Node(n,T));
    i=LeftMostChild(n,T);
    while (i!=NIL) {
       PreOrder(i,T);
       i=RightSibling(i,T);
    }
  }
}
```

CÀI ĐẶT CÂY BẰNG MẢNG (9)

- **Thủ tục duyệt trung tự**

```
void InOrder(Node n, Tree T) {
  if (n!= NIL) {
     Node i;
     i=LeftMostChild(n, T);
     if (i!=NIL) InOrder(i,T);
     printf("... ",Label_Node(n,T));
     i=RightSibling(i,T);
     while (i!=NIL) {
        InOrder(i,T);
        i=RightSibling(i,T);
     }
  }
}
```

CÀI ĐẶT CÂY BẰNG MẢNG (10)

- **Thủ tục duyệt hậu tự**

```
void PostOrder(Node n,Tree T) {
  if (n!=NIL) {
    Node i;
    i=LeftMostChild(n,T);
    while (i!=NIL) {
      PostOrder(i,T);
      i=RightSibling(i,T);
    }
    printf("... ",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 một nút
  - Hiển thị danh sách duyệt cây theo các phương pháp duyệt tiền tự, trung tự, hậu tự

BÀI TẬP (2)

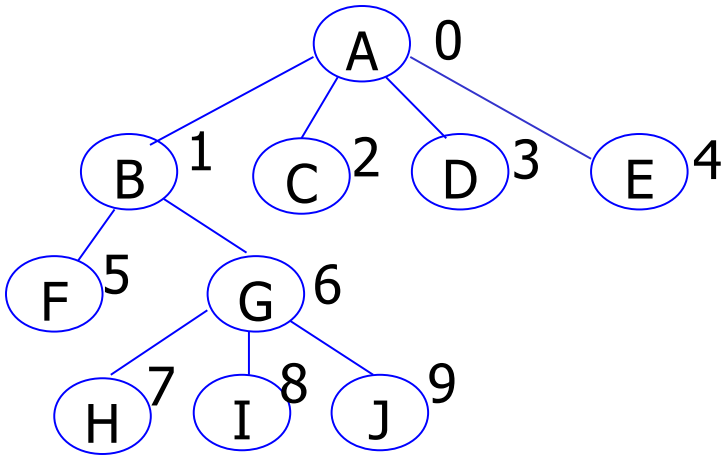
```
void ReadTree(Tree *T)
{ int i;
  MakeNull_Tree(T);
  do{ printf("Nhap so nut ");
      scanf("%d",&(T->MaxNode));
  }while ((T->MaxNode<1) ||
          (T->MaxNode>MAXLENGTH));

  printf("Nhap nhan cua nut goc "); fflush(stdin);
  scanf("%c",&(*T).Data[0]);
 (*T).Parent[0]=NIL; // nut goc khong co cha
  for (i=1;i<=(*T).MaxNode-1;i++){
     printf("Nhap cha cua nut %d ",i);
      scanf("%d",&(*T).Parent[i]);
     printf("Nhap nhan cua nut %d ",i);
     fflush(stdin);
     scanf("%c",&(*T).Data[i]);
   }
```

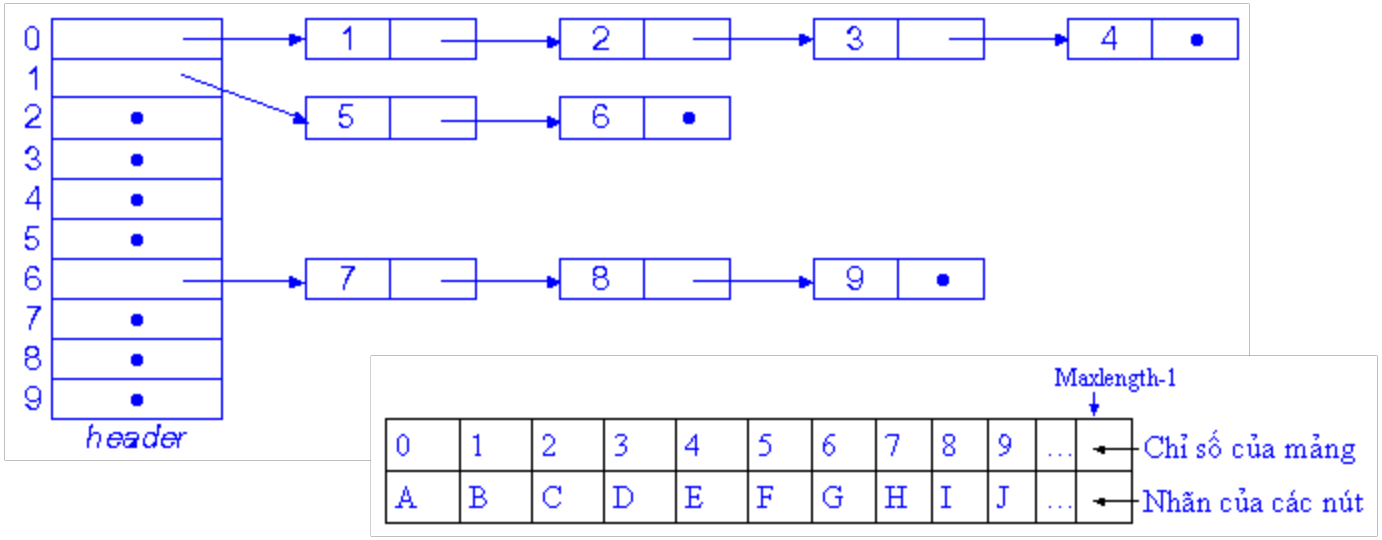
BÀI TẬP (3)

```
void main(){
  printf("Nhap du lieu cho cay tong quat\n");
  ReadTree(&T);
  printf("Danh sach duyet tien tu cua cay la\n");
  PreOrder(Root(T),T);
  printf("\nDanh sach duyet trung tu la\n");
  InOrder(Root(T),T);
  printf("\nDanh sach duyet hau tu cua cay la\n");
  PostOrder(Root(T),T);
}
```

CÀI ĐẶT CÂY BẰNG DS CÁC NÚT CON (1)

- Minh họa





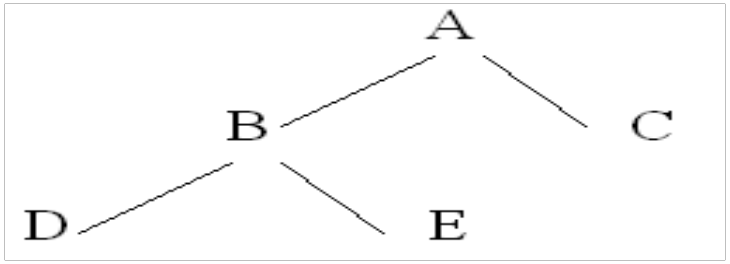
CÀI ĐẶT CÂY BẰNG DS CÁC NÚT CON (2)

- 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 ….. . LabelType
typedef ….. .  LIST;
typedef struct  {
    LIST header[maxlength];
    LabelType  labels[maxlength];
    node  root;
}TREE;
```

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

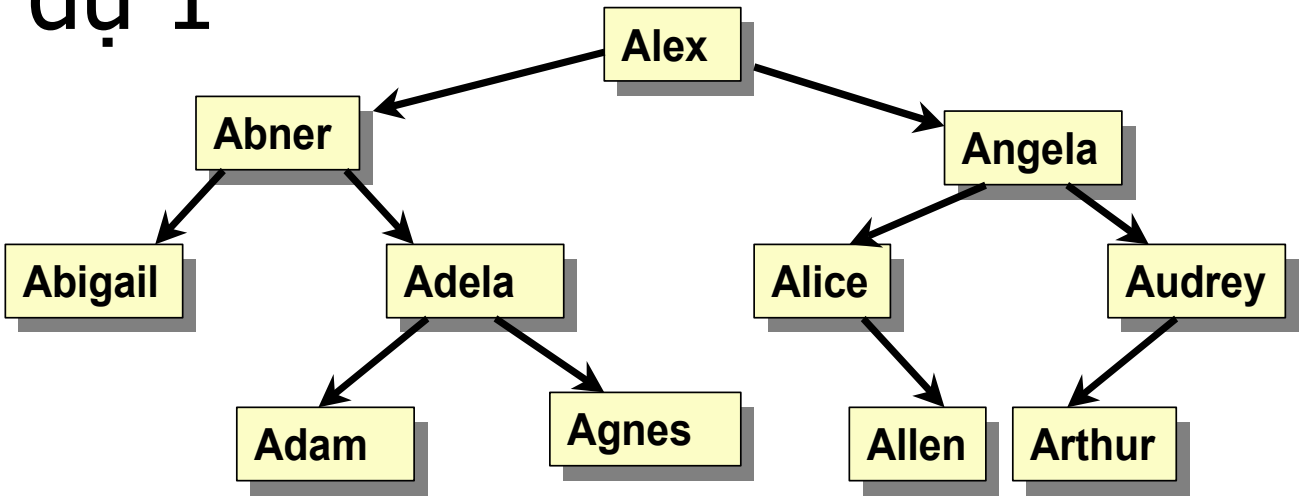
- Ví dụ



	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	

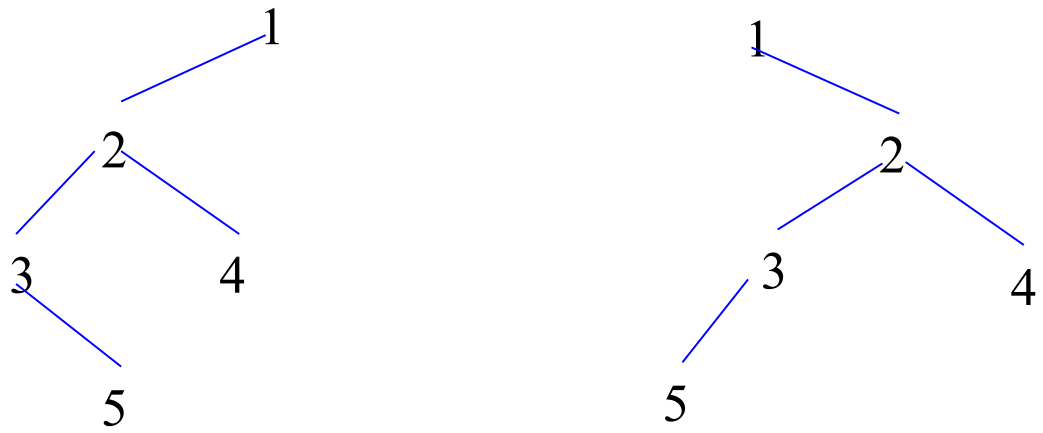
CÂY NHỊ PHÂN (1)

- Định nghĩa
  - Là cây rỗng hoặc 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): duyệt nút gốc, duyệt tiền tự con trái, duyệt tiền tự con phải.
  - Trung tự (LNR): duyệt trung tự con trái, duyệt nút gốc, duyệt trung tự con phải.
  - Hậu tự (LRN): duyệt hậu tự con trái, duyệt hậu tự con phải, duyệt nút gốc.

CÀI ĐẶT CÂY NHỊ PHÂN (1)

- Khai báo

```
typedef … TData;
typedef struct TNode* TNodeType;
struct TNode {
        TData data;
        TNodeType left,right;
};
typedef TNodeType TTree;
```

- Tạo cây rỗng

```
void MakeNullTree(TTree *T) {
     (*T)=NULL;
}
```

- Kiểm tra cây rỗng

```
int EmptyTree(TTree T) {
     return T==NULL;
}
```

CÀI ĐẶT CÂY NHỊ PHÂN (1)

- **Xác định con trái**

```
TTree LeftChild(TTree n) {
    if (n!=NULL) return n->left;
    else return NULL;
}
```

- **Xác định con phải**

```
TTree RightChild(TTree n) {
    if (n!=NULL) return n->right;
    else return NULL;
}
```

- **Kiểm tra xem một nút có phải là lá không?**

```
int IsLeaf(TTree n) {
   if(n!=NULL)
     return(LeftChild(n)==NULL)&&(RightChild(n)==NULL);
   else return NULL;
}
```

CÀI ĐẶT CÂY NHỊ PHÂN (1)

- Duyệt tiền tự

```
void PreOrder(TTree T) {
  if (T!= NULL) {
      printf("...",T->data);
      //if (LeftChild(T)!=NULL)
      PreOrder(LeftChild(T));
      //if(RightChild(T)!=NULL)
      PreOrder(RightChild(T));
  }
}
```

CÀI ĐẶT CÂY NHỊ PHÂN (1)

- Duyệt trung tự

```
void InOrder(TTree T) {
  if (T!= NULL) {
    //if (LeftChild(T)=!NULL)
    InOrder(LeftChild(T));
    printf("... ",T->data);
    //if(RightChild(T)!=NULL)
    InOrder(RightChild(T));
  }
}
```

CÀI ĐẶT CÂY NHỊ PHÂN (1)

- **Duyệt hậu tự**

```
void PostOrder(TTree T) {
  if (T!= NULL) {
    //if(LeftChild(T)!=NULL)
    PostOrder(LeftChild(T));
    //if(RightChild(T)!=NULL)
    PostOrder(RightChild(T));
    printf("... ",T->data);
  }
}
```

CÀI ĐẶT CÂY NHỊ PHÂN (1)

- **Xác định số nút trong cây**

```
int nb_nodes(TTree T) {
   if(EmptyTree(T)) return 0;
   else return 1 + nb_nodes(LeftChild(T))+
                   nb_nodes(RightChild(T));
}
```

CÀI ĐẶT CÂY NHỊ PHÂN (1)

- **Tạo cây mới từ hai cây có sẵn**

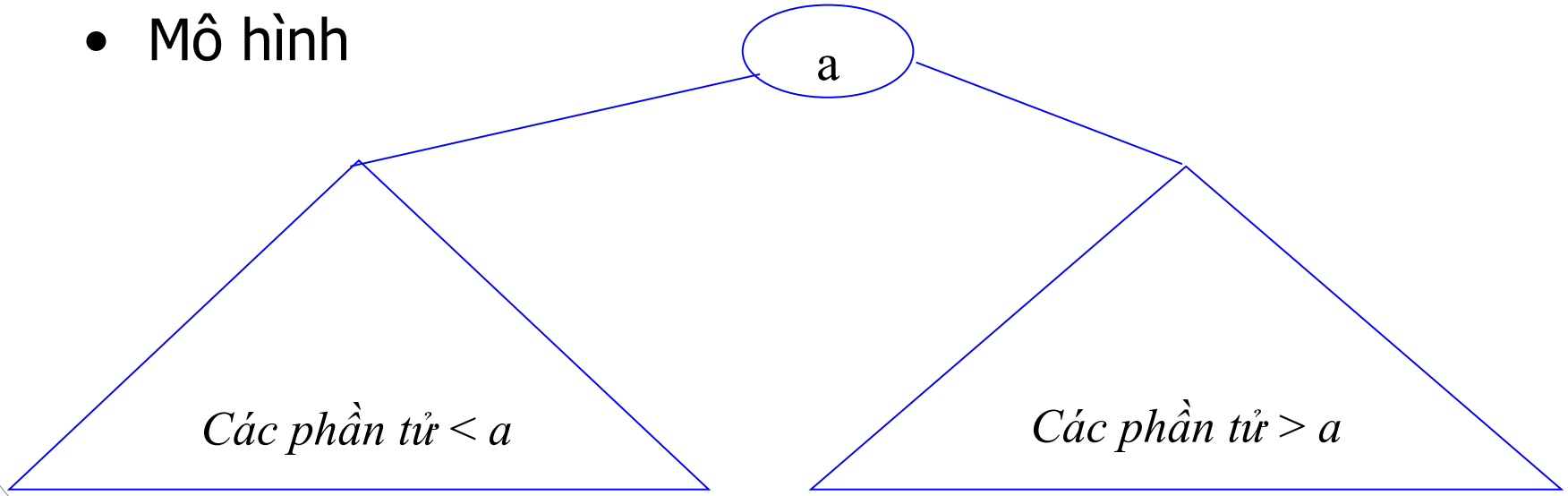
```
TTree Create2(TData v,TTree l,TTree r) {
      TTree N;
      N=(TNodeType)malloc(sizeof(struct TNode));
      N->data=v;
      N->left=l;
      N->right=r;
      return N;
}
```

CÂY TÌM KIẾM NHỊ PHÂN (Binary search tree-BST)

- Định nghĩa

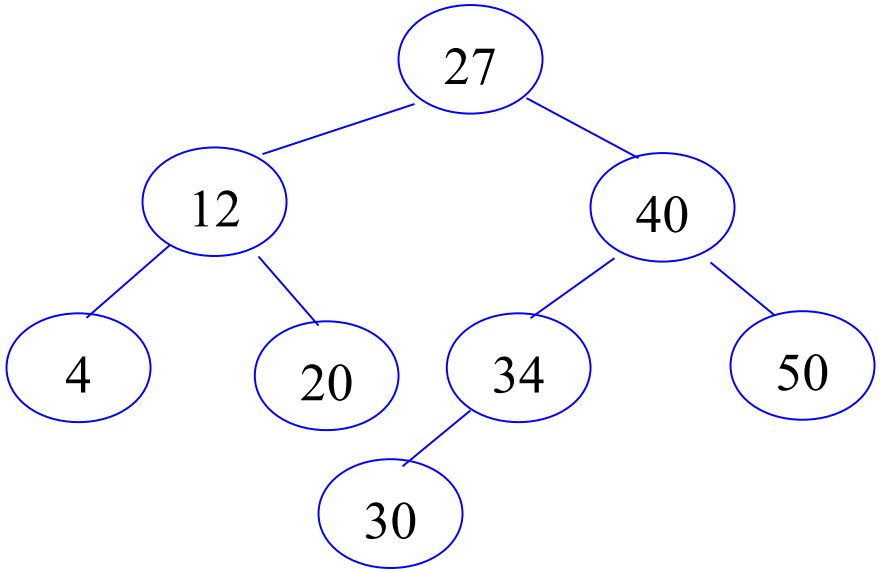
  Cây BST là cây nhị phân mà khóa tại mỗi nút lớn hơn khóa của tất cả các nút thuộc cây con bên trái và nhỏ hơn khóa của tất cả các nút thuộc cây con bên phải.

- Mô hình



CÂY TÌM KIẾM NHỊ PHÂN

- Ví dụ



- Nhận xét
  - Trên cây BST không có 2 nút cùng khóa.
  - Cây con của 1 cây BST là 1 cây tìm kiếm nhị phân.
  - Duyệt trung tự tạo thành dãy khóa có giá trị tăng: 4, 12, 20, 27, 30, 34, 40, 50

CÀI ĐẶT CÂY BST

- Khai báo

```
typedef ... KeyType;
typedef struct Node* NodeType;
struct Node {
    KeyType key;
    NodeType left, right;
};
typedef NodeType Tree;
```

CÀI ĐẶT CÂY BST

- Tìm kiếm một nút có khoá x
  - Bắt đầu từ nút gốc ta tiến hành các bước sau:
    - 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 nhãn của nút hiện hành: tìm X trên cây con bên trái
    - Nếu X lớn hơn nhãn của nút hiện hành: tìm X trên cây con bên phải

CÀI ĐẶT CÂY BST

```
Tree Search(KeyType x, Tree Root){
   if (Root == NULL) return NULL;//không tìm thấy x
  else if (Root->key == x) // tìm thấy khoá x
          return Root;
        else if (Root->key < x)
          return Search(x,Root->right);
        else
          return Search(x,Root->left);
}
```

CÀI ĐẶT CÂY BST

- Thêm một nút có khoá x vào cây

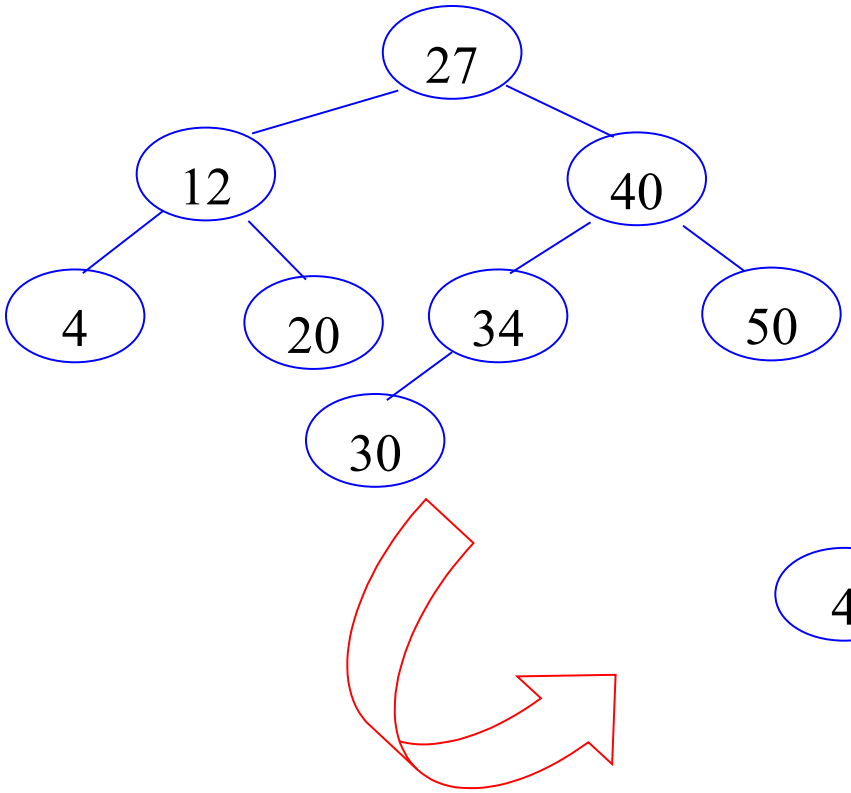
  Muốn thêm 1 nút có khóa X vào cây BST, trước tiên ta phải tìm kiếm xem đã có X trên cây chưa.

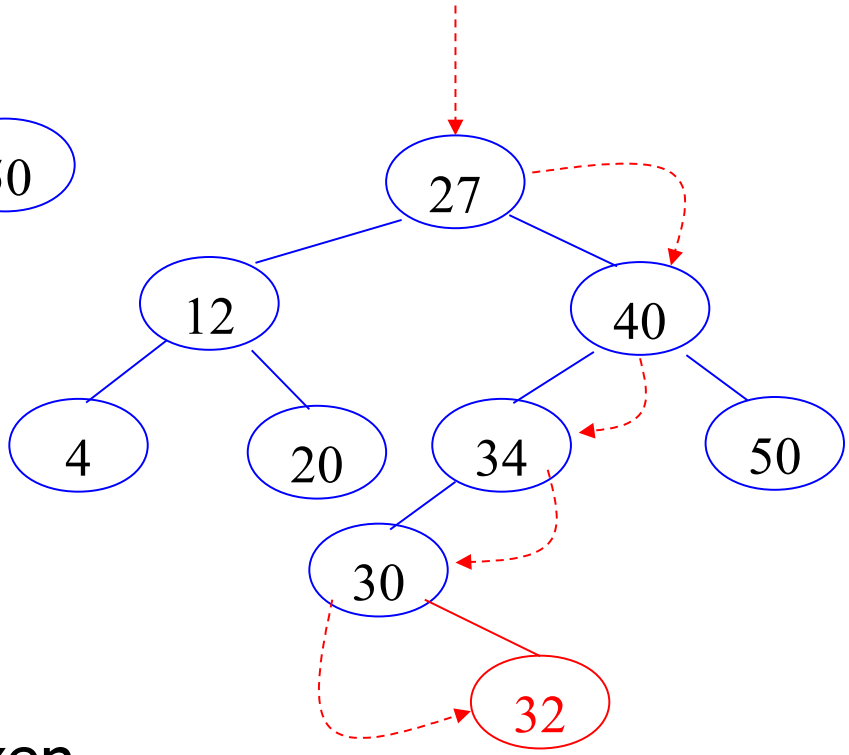
  Nếu có thì giải thuật kết thúc, nếu chưa thì ta mới thêm vào. Việc thêm vào không làm phá vỡ tính chất cây BST.

  - 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 nhãn của nút hiện hành: xen X vào cây con bên trái
  - Nếu X lớn hơn nhãn của nút hiện hành: xen 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

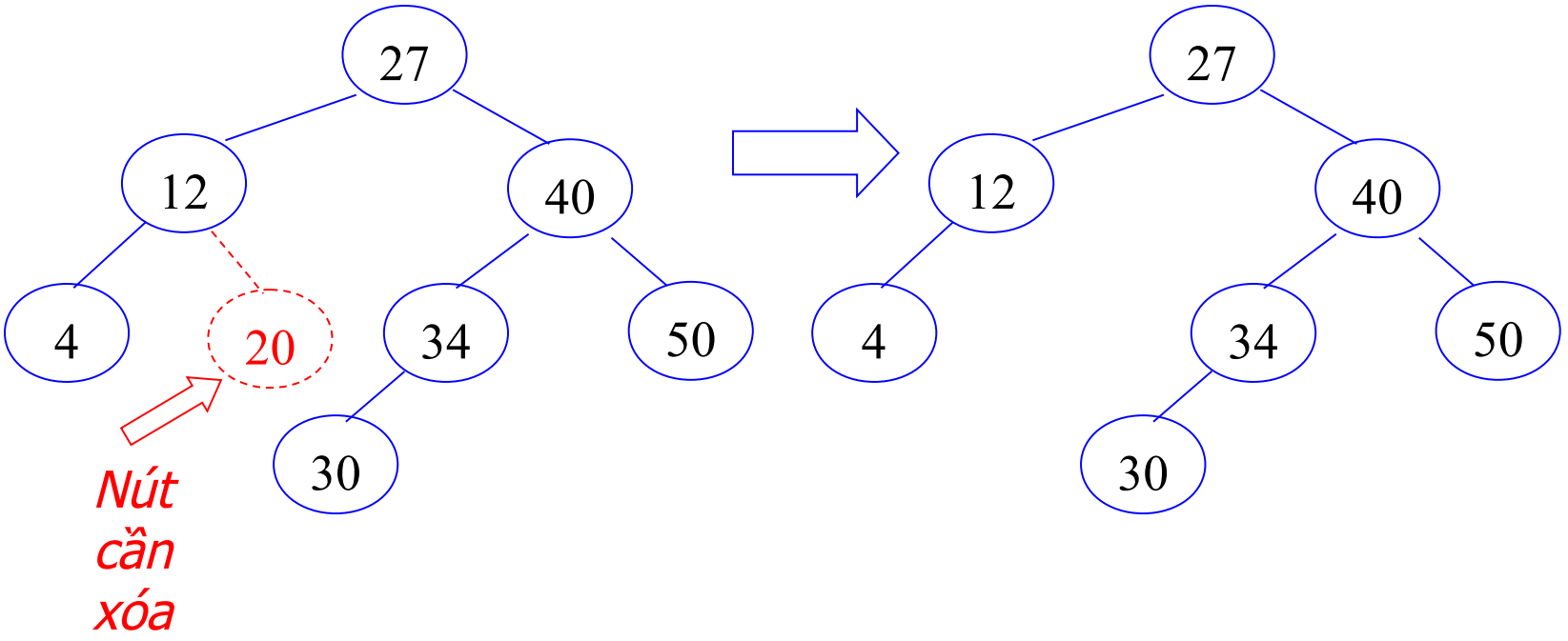
```
void InsertNode(KeyType x, Tree *Root) {
    if (*Root == NULL) {
        (*Root)=(NodeType)malloc(sizeof(struct Node));
        (*Root)->key = x;
        (*Root)->left = NULL;
        (*Root)->right = NULL;
    }
    else if (x < (*Root)->key)
         InsertNode(x, &((*Root)->left));
    else if (x > (*Root)->key)
         InsertNode(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

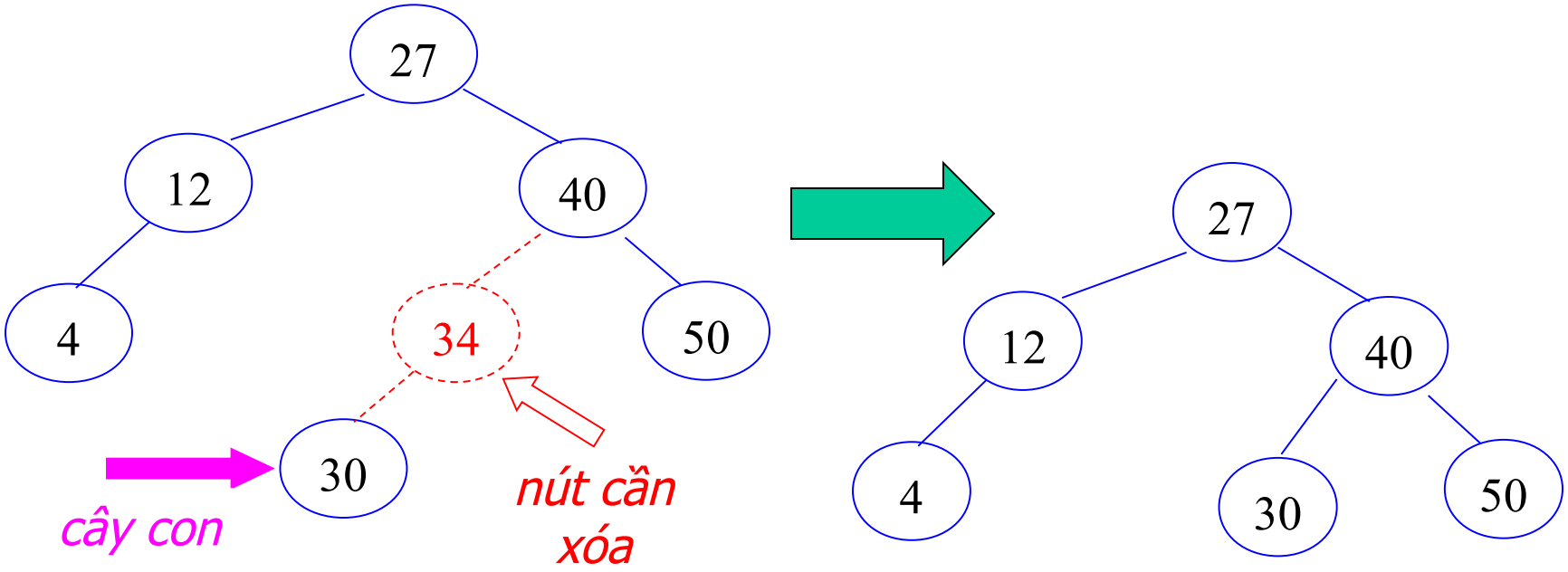
CÀI ĐẶT CÂY BST

- 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

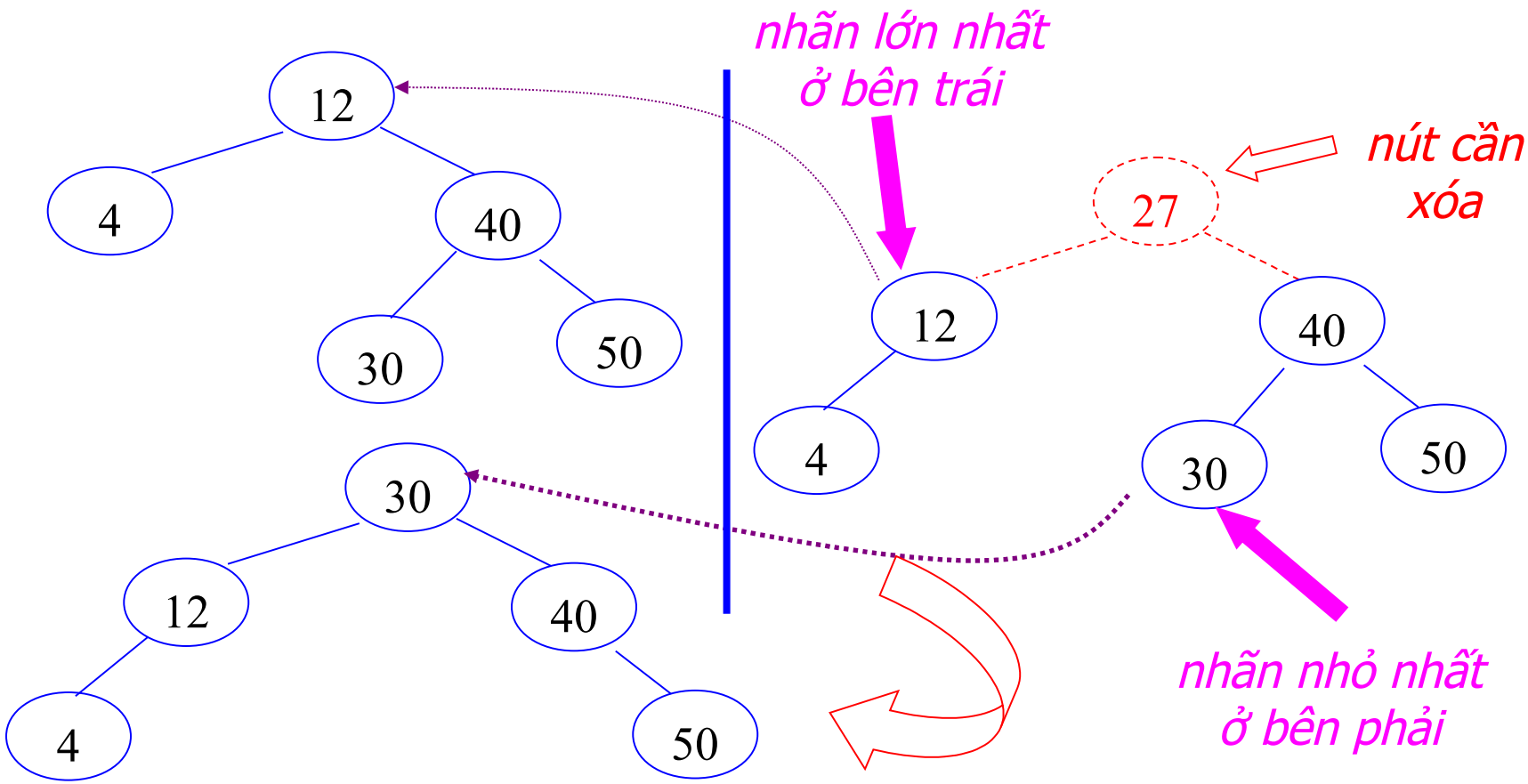


CÀI ĐẶT CÂY BST

- Trường hợp 3
  - N có hai cây con: thay nút này bởi
    - Nút có nhãn lớn nhất của cây con bên trái, hoặc
    - Nút có nhãn nhỏ nhất của cây con bên phải

CÀI ĐẶT CÂY BST

- Ví dụ: Xoá nút có nhãn 27



CÀI ĐẶT CÂY BST

```
void DeleteNode(KeyType x, Tree *Root) {
      if ((*Root) != NULL)
        if (x < (*Root)->key)
            DeleteNode(x, &((*Root)->left));
        else if (x > (*Root)->key)
            DeleteNode(x, &((*Root)->right));
        else if ((*Root)->left == NULL)
            (*Root) = (*Root)->right;
        else if ((*Root)->right == NULL)
            (*Root) = (*Root)->left;
        else
            (*Root)->key = DeleteMin(&((*Root)->right));
}
```

CÀI ĐẶT CÂY BST

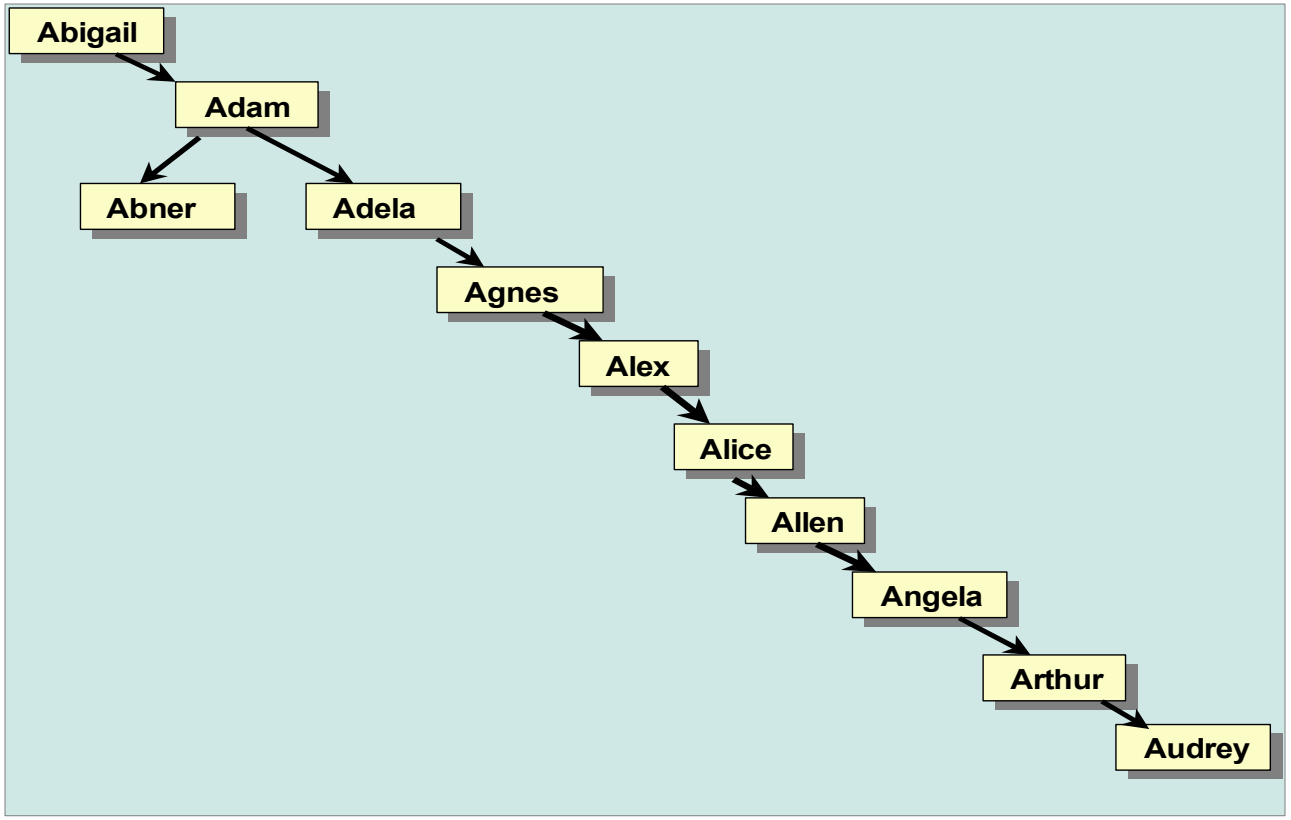
```
KeyType DeleteMin(Tree *Root) {
     KeyType k;
     if ((*Root)->left == NULL) {
          k=(*Root)->key;
          (*Root) = (*Root)->right;
          return k;
     }
     else
          DeleteMin(&((*Root)->left));
}
```

KIẾN THỨC BỔ SUNG (1)

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

KIẾN THỨC BỔ SUNG (2)

- Bên dưới là một cây TKNP phân "không cân bằng"



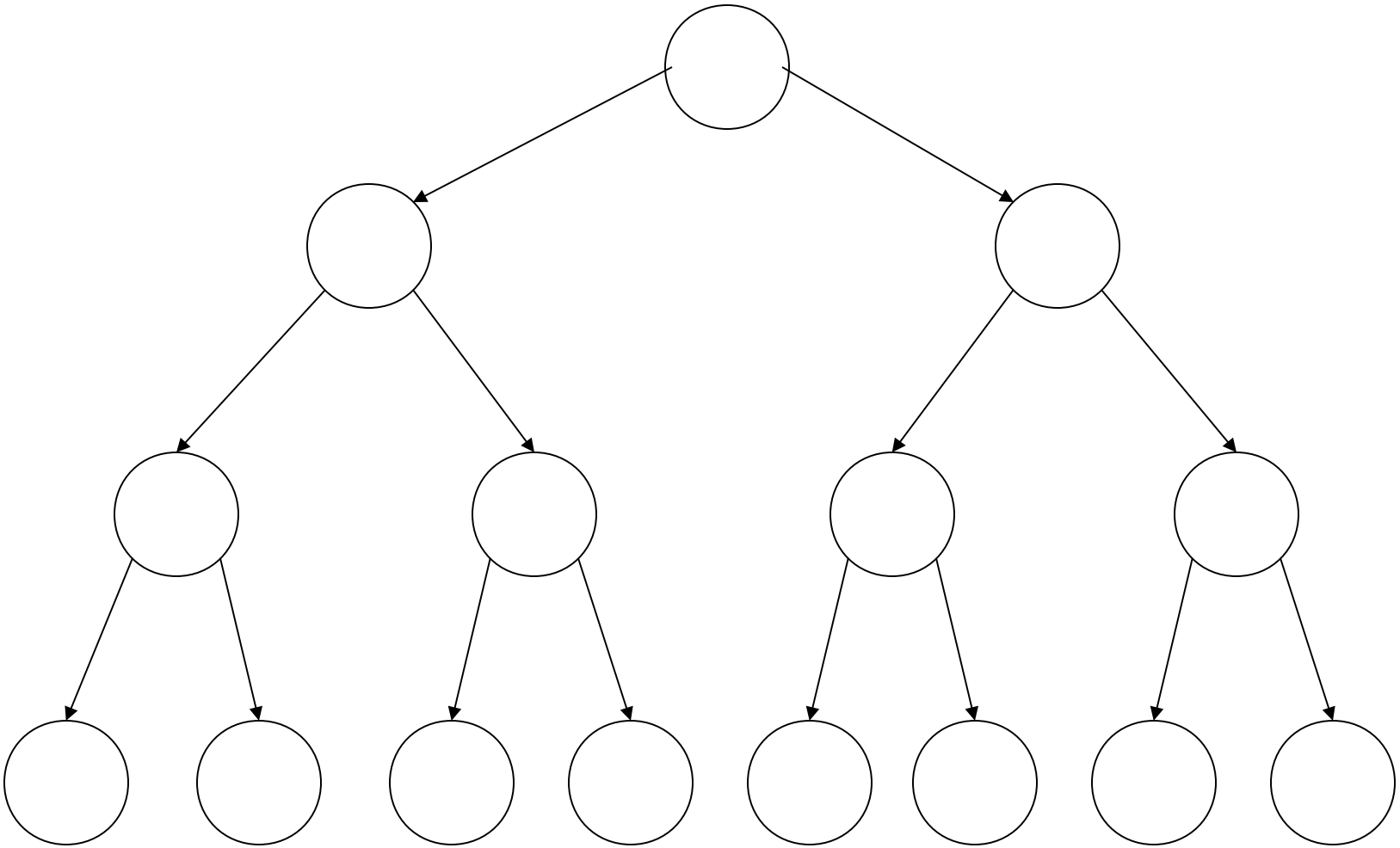
CÂY NHỊ PHÂN ĐẦY ĐỦ (1)

(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)

- Ví dụ -Một cây nhị phân đầy đủ



CÂY NHỊ PHÂN ĐẦY ĐỦ (3)

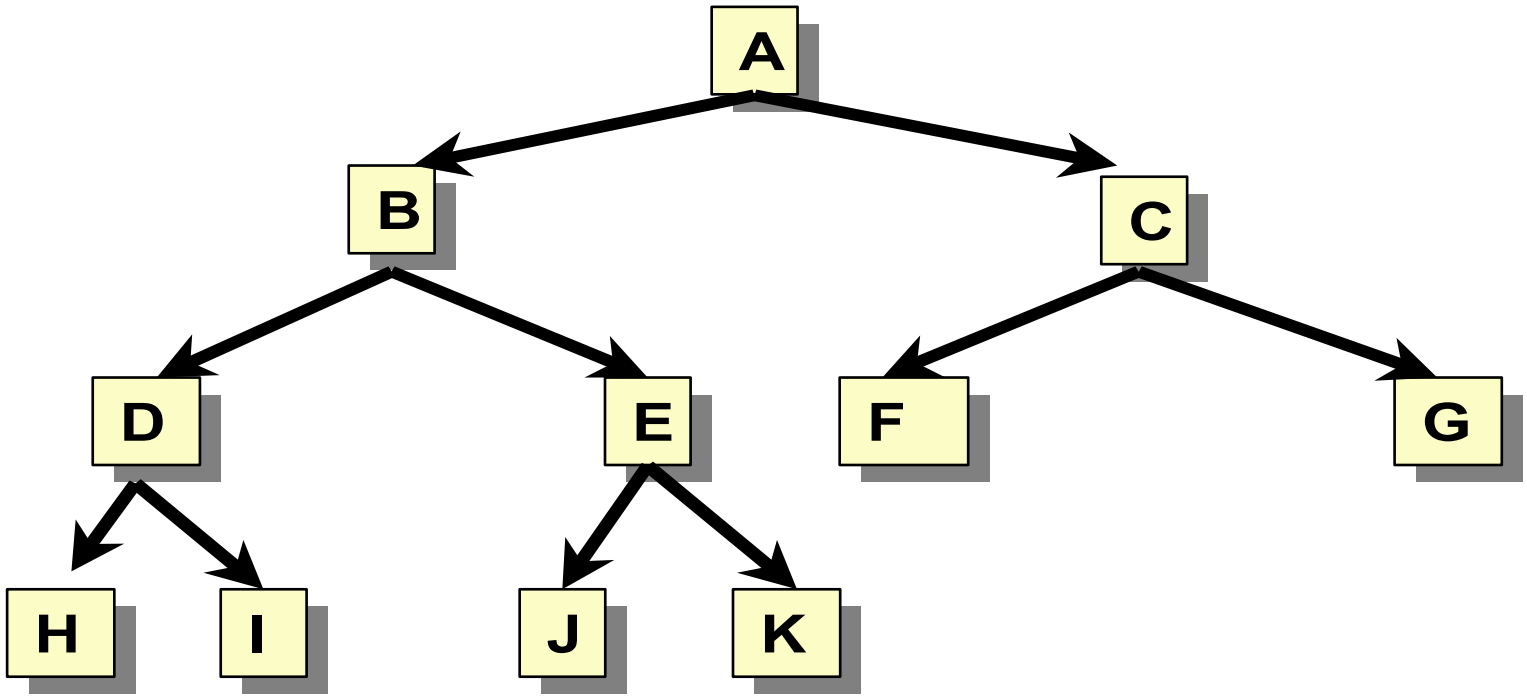
- **Câu hỏi về cây nhị phân đầy đủ:**
  - **Một cây nhị phân đầy đủ chiều cao h sẽ có bao nhiêu nút lá?**
  - **Một cây nhị phân đầy đủ chiều cao h sẽ có tất cả bao nhiêu nút?**

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



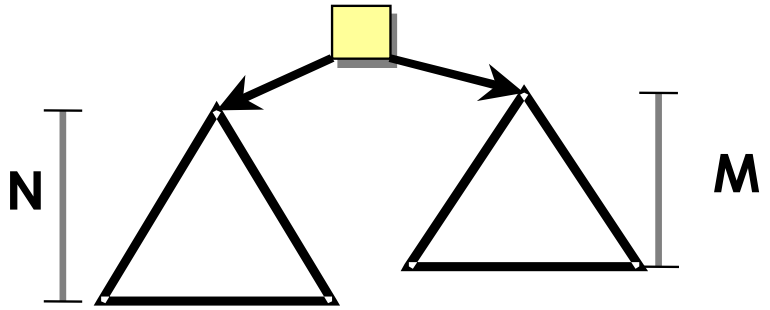
CÂY NHỊ PHÂN HOÀN CHỈNH (3)

- 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 $\lfloor \log n \rfloor$

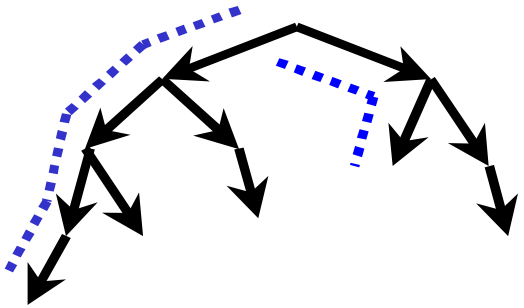
CÂY NHỊ PHÂN CÂN BẰNG VỀ CHIỀU CAO (Height-balanced Binary Tree)

- 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)