# CTDL&GT

## CHUONG 4

Giảng viên: Vũ Văn Thỏa

# CHƯƠNG 4. CÂY NHỊ PHÂN

- Những khái niệm cơ bản
- Các thao tác trên cây nhị phân
- Các phương pháp pháp duyệt cây nhị phân
- Cây nhị phân tìm kiếm
- Một số ứng dụng của cây nhị phân
- Cây quyết định

# 4.1 Những khái niệm cơ bản

- *Cây là tập hợp các đối tượng, mỗi đối tượng gọi là nút và quan hệ giữa chúng được mô tả bởi các cạnh nối các nút.*

# 4.1.1 Định nghĩa cây

- **Định nghĩa 1**:

Cây là tập hợp hữu hạn các nút thỏa mãn:

- Có một nút gọi là gốc;
- Có quan hệ phân cấp “cha-con” giữa các nút.

# Định nghĩa 2 (đệ qui):

- Nếu T chỉ gồm 1 nút $\Rightarrow$ T là một cây với gốc là chính nút đó;

- Nếu $T_1$, …, $T_n$ ($n \geq 1$) là các cây có gốc tương ứng $R_1$, …, $R_n$ $\Rightarrow$ T là cây với gốc R được tạo thành bằng cách cho R thành nút cha của các nút $R_1$, …, $R_n$.

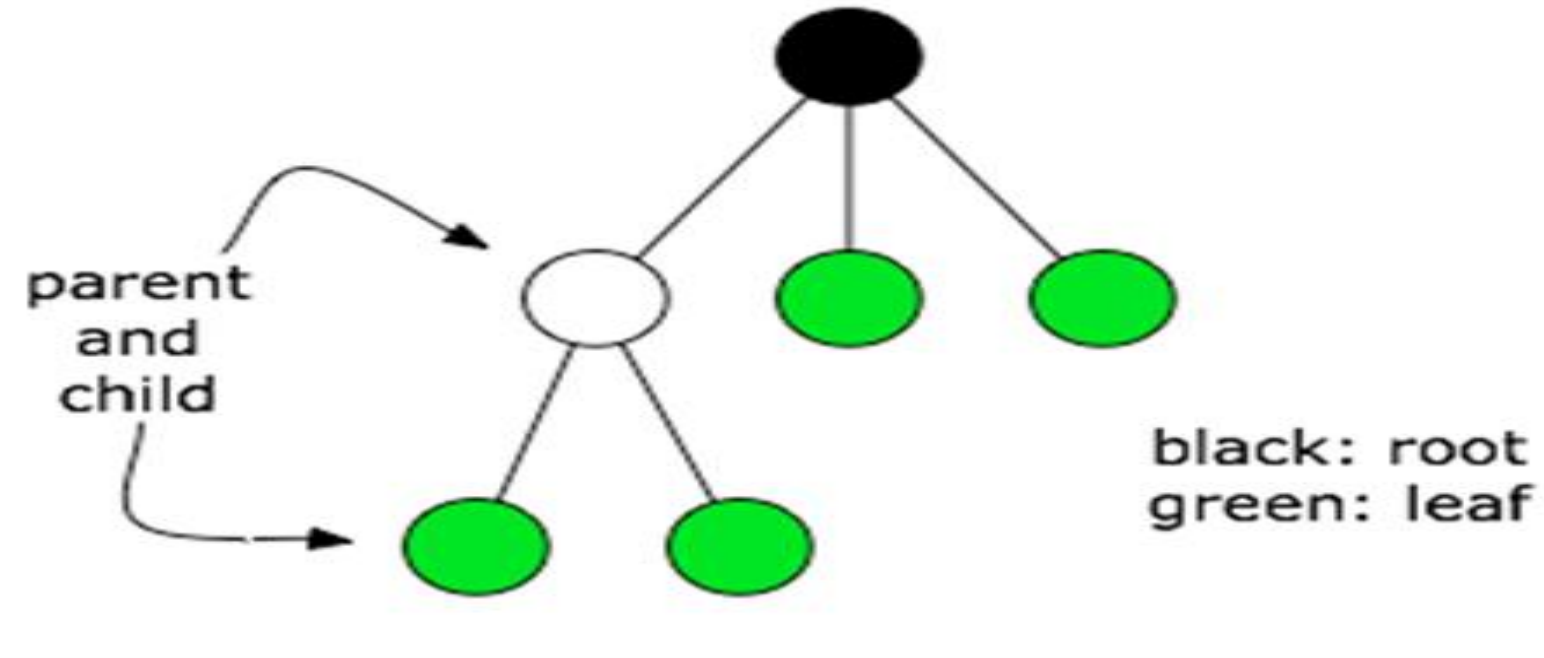
# Các ví dụ về cấu trúc cây

- Mục lục của một cuốn sách
- Cấu trúc thư mục trên đĩa máy tính.
- Cây phả hệ của các loài

# 4.1.2 Một số khái niệm

- **Nút gốc**: là nút của cây T không có cha.
- **Nút cha**: Nút R là cha của các nút $R_1$, ..., $R_n$.
- **Nút con**: Các nút $R_1$, …, $R_n$ gọi là nút con của R.
- **Bậc của một nút**: là số các nút con của nút đó.
- **Bậc của một cây T**: là bậc lớn nhất của các nút $\in$ T. T có bậc m $\Rightarrow$ T gọi là cây m-phân.

# Cấu trúc dữ liệu kiểu cây:



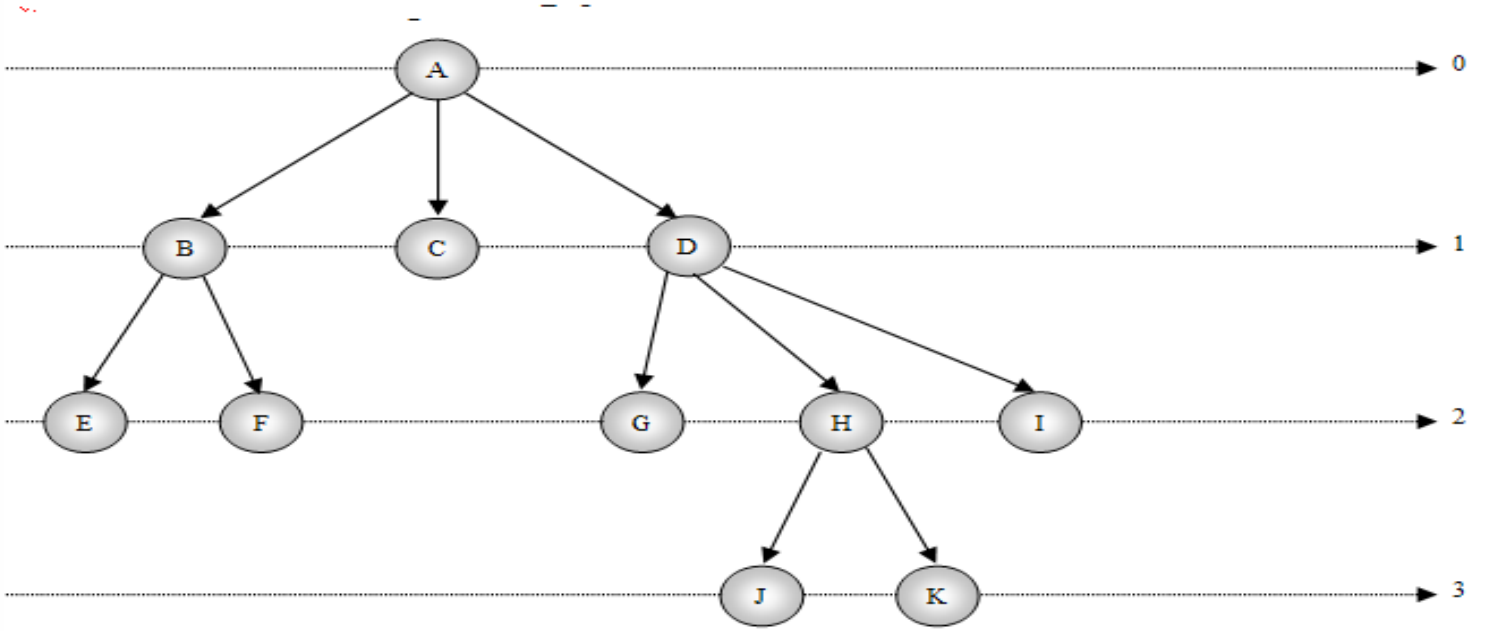
# 4.1.2 Một số khái niệm (Tiếp)

- **Nút lá:** là nút có bậc = 0 $\Rightarrow$ nút lá không có nút con.
- T là cây m-phân đầy đủ $\Leftrightarrow$ mỗi nút khác lá đều có đúng m con.
- **Nút nhánh (nút trong hay nút trung gian):** là nút vừa có con vừa có cha.

# 4.1.2 Một số khái niệm (Tiếp)

- **Đường đi**: Dãy các nút $R_1$, …, $R_k$, trong đó $R_i$ là cha của $R_{i+1}$ gọi là đường đi từ R1 đến Rk. Độ dài của đường đi từ $R_1$ đến $R_k$ bằng k-1.
- **Mức của nút**: là độ dài đường đi từ gốc đến nút (độ cao của nút) $\Rightarrow$ Nút gốc có mức 0.
- **Chiều cao của cây**: là mức lớn nhất của các nút $\in$ T.

# Các mức của cây



# 4.1.2 Một số khái niệm (Tiếp)

- **Cây con**: Mỗi cây có gốc tại nút a $\in$ T là một cây con của T.
- **Cây được sắp thứ tự**: là cây mà mỗi cây con được sắp theo một thứ tự nào đó.
- **Cây gán nhãn**: là cây mà mỗi đỉnh được gắn với một giá trị (nhãn)
- **Rừng**: là một danh sách các cây phân biệt:

$$F = (T_1, \ldots, T_n)$$

# 4.1.3 Cây nhị phân

- **Định nghĩa 1**:

- Cây nhị phân là cây mà mỗi nút có không quá 2 con;

- Phân biệt cây con bên trái và cây con bên phải.

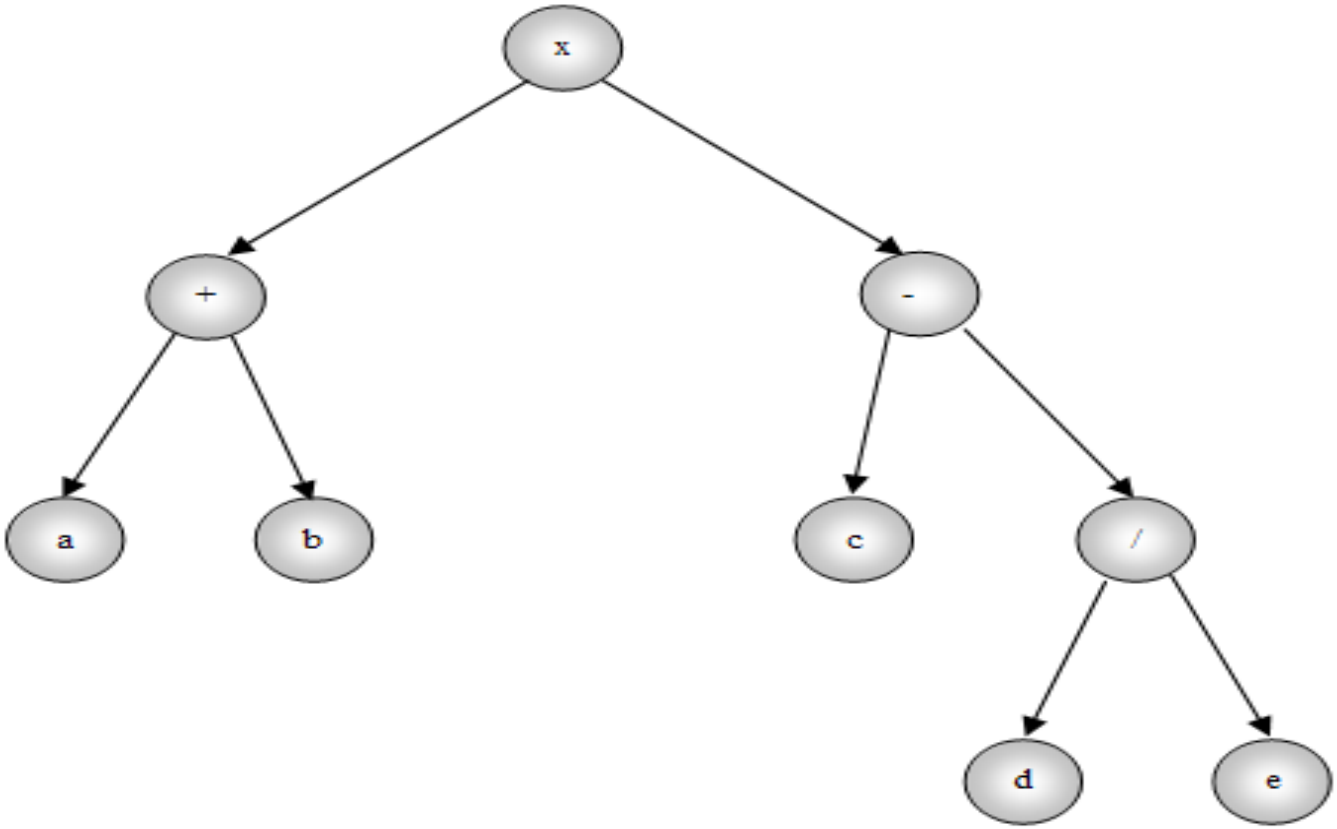
# Định nghĩa 2 (Đệ qui):

- Nếu T= $\varnothing \Rightarrow$ T là cây nhị phân.

- Nếu $T_1$ và $T_2$ là 2 cây nhị phân không có nút chung, r là nút không thuộc $T_1$ và $T_2$

$\Rightarrow$ T là cây nhị phân với gốc r và $T_1$ là cây con bên trái, $T_2$ là cây con bên phải.

# Cây nhị phân biểu diễn ( a+b) x (c-d/e )



# Tính chất 1:

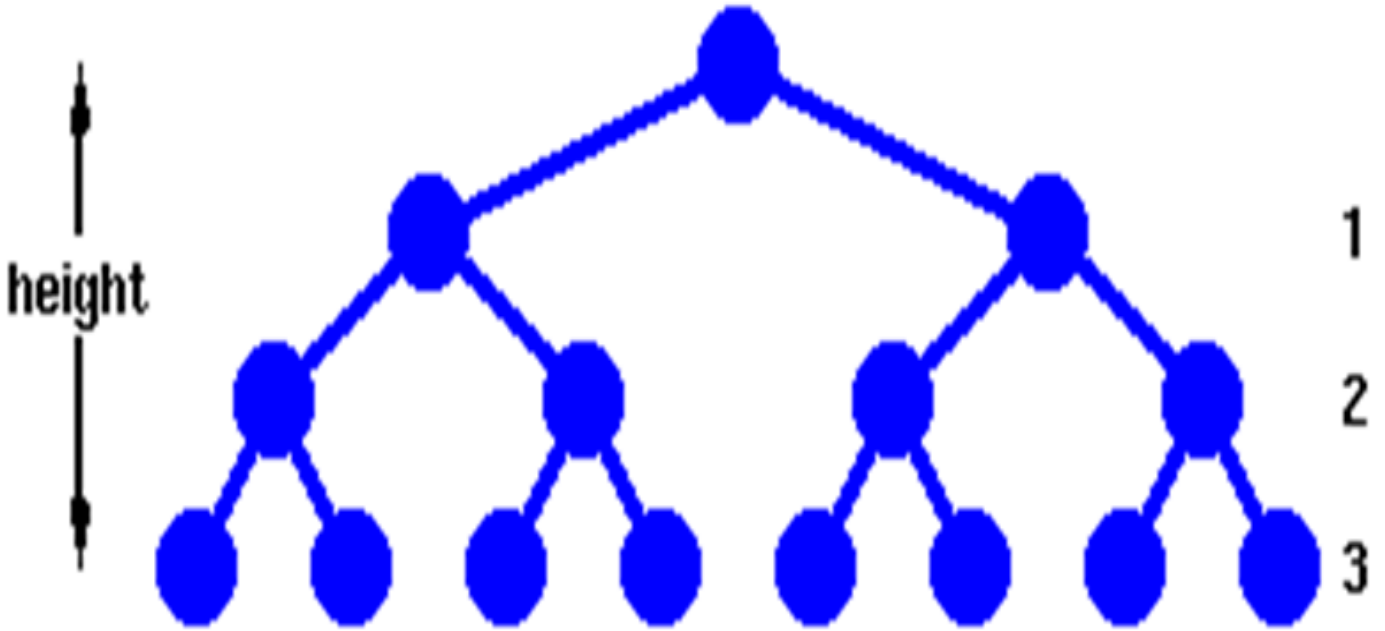
- Số lượng tối đa các nút trên mức i là 2^i và tối thiểu là 1 ( i ≥ 0)

- Số lượng tối đa các nút trên cây nhị phân có chiều cao h là 2^(h+1) -1 và tối thiểu là h+1 với h ≥ 0

# Cây nhị phân đầy đủ (hoàn chỉnh):

- Cây nhị phân đầy đủ là cây nhị phân mà mỗi mức i (i ≥ 0) đều có số nút tối đa 2^i

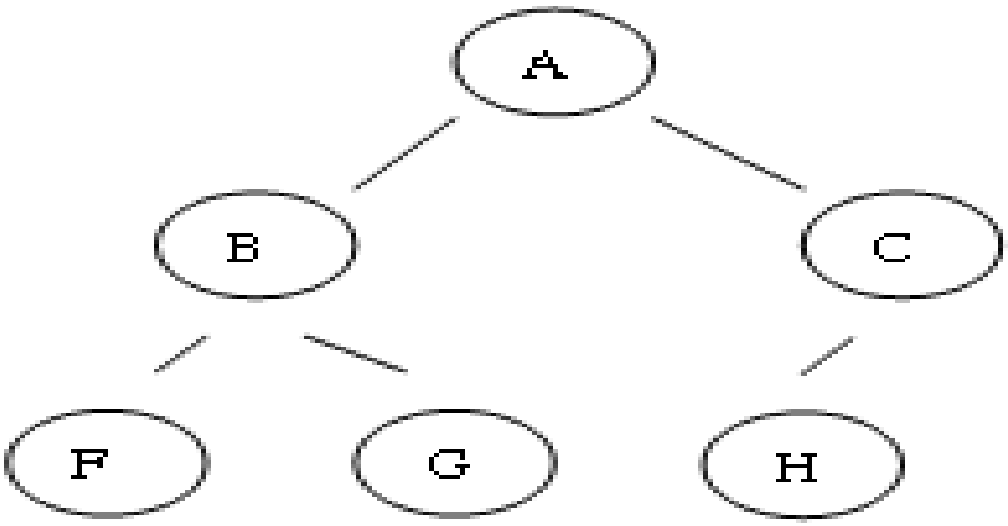
# Hình ảnh cây nhị phân đầy đủ:



# Cây nhị phân gần đầy:

- Cây nhị phân gần đầy (almost complete binary tree) độ cao h nếu ở các mức i < h có đầy đủ các nút, còn ở mức h thì đầy từ trái qua phải.

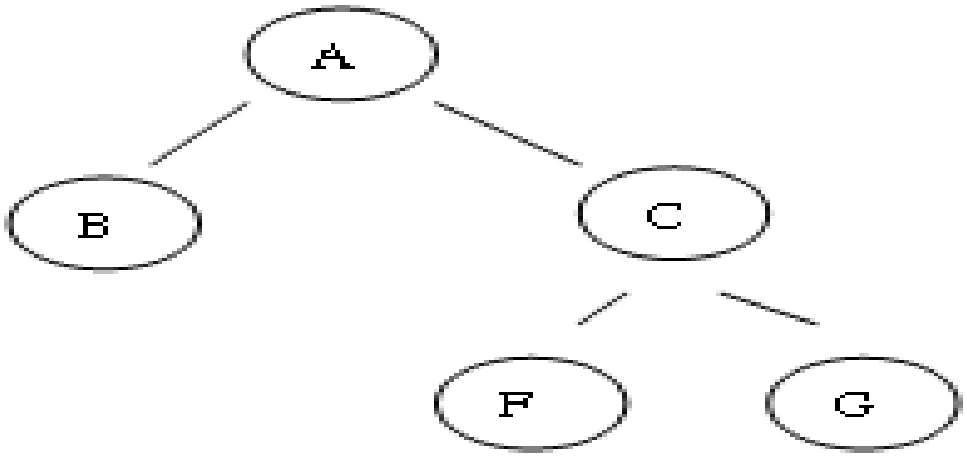
# Hình ảnh cây nhị phân gần đầy:



# Cây nhị phân đúng:

- Cây nhị phân đúng là cây nhị phân mà mỗi nút không phải lá đều có đúng 2 con.

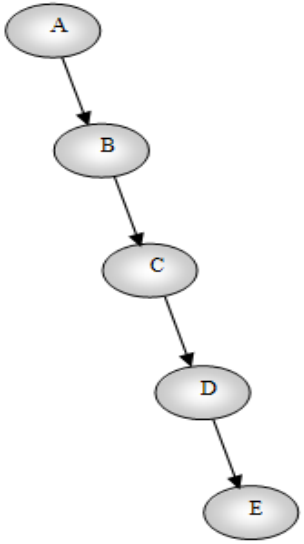
# Hình ảnh cây nhị phân đúng:

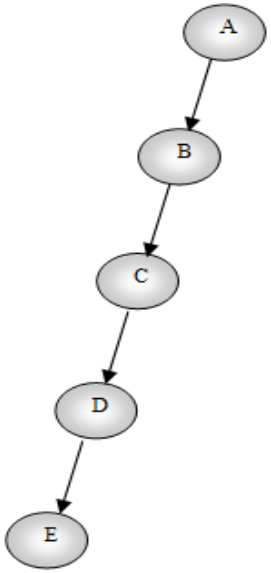


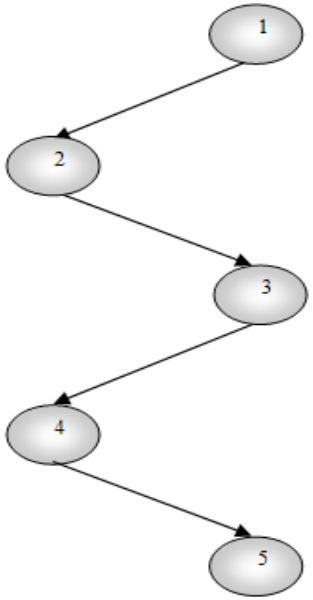
# Một số loại cây nhị phân suy biến:

**Cây nhị phân suy biến** :

cây lệch trái, cây lệch phải, cây dích dắc.



# Tính chất 2:

- Nếu số lượng nút là như nhau thì cây nhị phân suy biến có chiều cao lớn nhất, cây nhị phân đầy đủ có chiều cao nhỏ nhất.
- Cây nhị phân đầy đủ có n nút thì chiều cao của nó là h = [ log (n+1)] -1.

# 4.2 Các thao tác trên cây nhị phân

- **Biểu diễn cây**
- **Các thao tác**

# 4.2.1 Biểu diễn cây nhị phân

- Cài đặt bởi mảng (lưu trữ kế tiếp)

- Các nút của cây nhị phân được đánh số thứ tự từ 1 cho đến hết với số lượng ≤ max.

- Mỗi nút của cây nhị phân gồm 3 thành phần:
  - infor: mô tả thông tin gắn với mỗi nút.
  - left: số thứ tự của nút con bên trái. Nếu không có nút con bên trái thì left = 0.
  - right: số thứ tự của nút con bên phải. Nếu không có nút con bên phải thì right = 0.

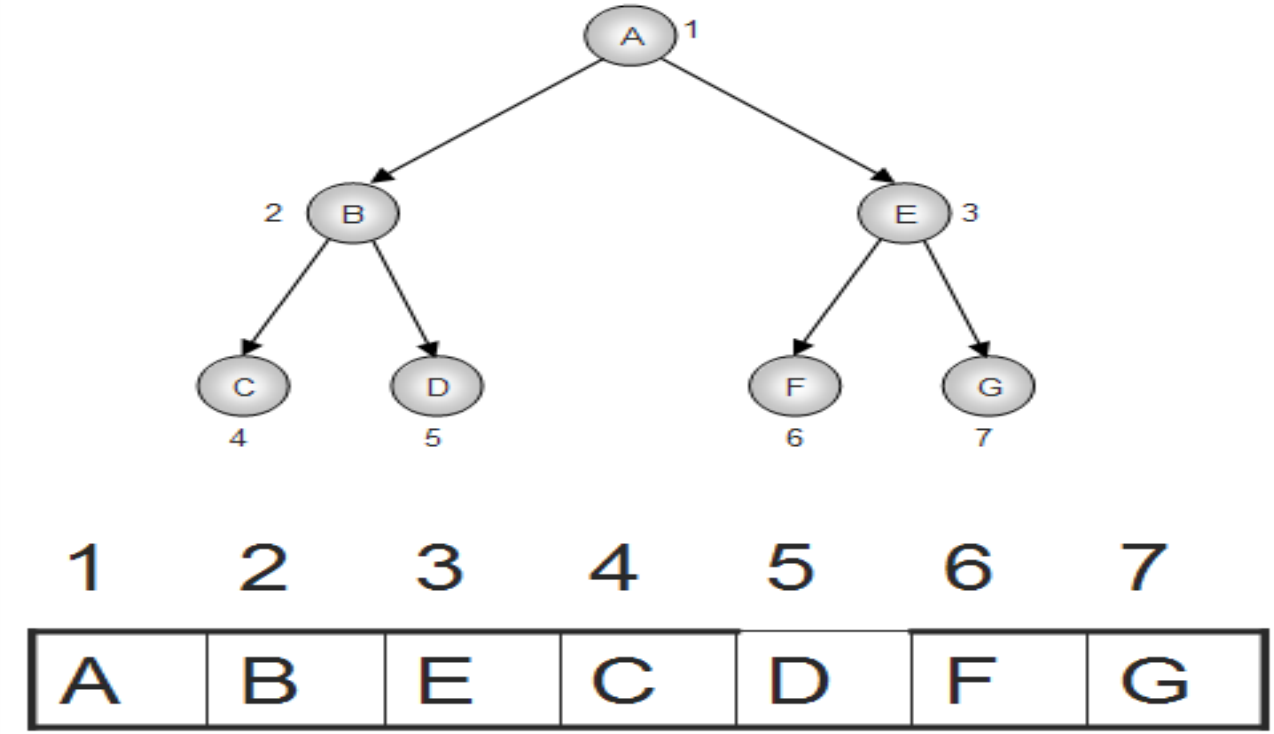
# Khai báo cấu trúc mảng của cây nhị phân:

```
#define  max  n
//Khai báo kiểu Item  (nếu cần)
struct node
{ Item  infor;
   int  left;
   int right;
 };
struct node  T[max];
```

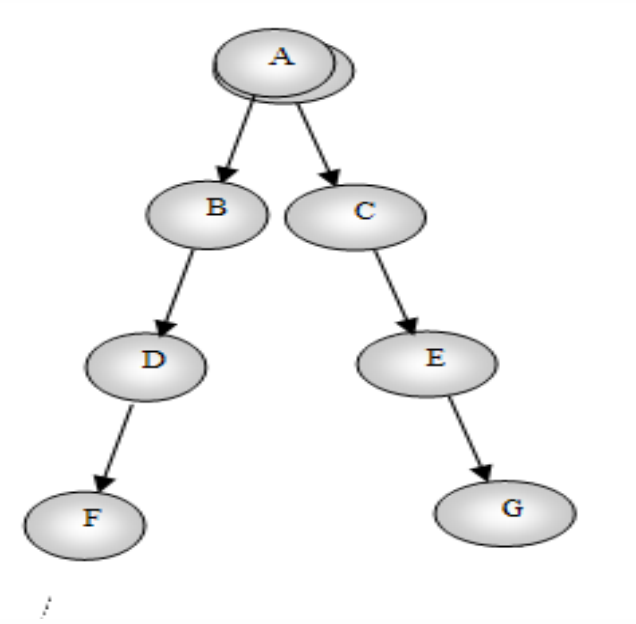
# Nhận xét 1:

- Thông thường, trong cây nhị phân, các nút được đánh số thứ tự theo " trên – dưới" và "trái – phải" $\Rightarrow$ Nút gốc có số thứ tự là 1. Với nút i thì: nút con trái của nó 2i và nút con phải là 2i+1; nút cha là [i/2].

# Biểu diễn mảng của cây nhị phân đầy đủ:



# Biểu diễn mảng của cây nhị phân khuyết:




| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | B | C | D | ∅ | ∅ | E | F | ∅ | ∅ | ∅ | ∅ | ∅ | ∅ | G |

# Nhận xét 2:

- Nếu cây nhị phân không đầy đủ thì cài đặt bằng mảng có thể gây ra lãng phí bộ nhớ

# Cài đặt bởi danh sách liên kết (lưu trữ móc nối):

**- Mỗi nút của cây nhị phân gồm 3 thành phần:**

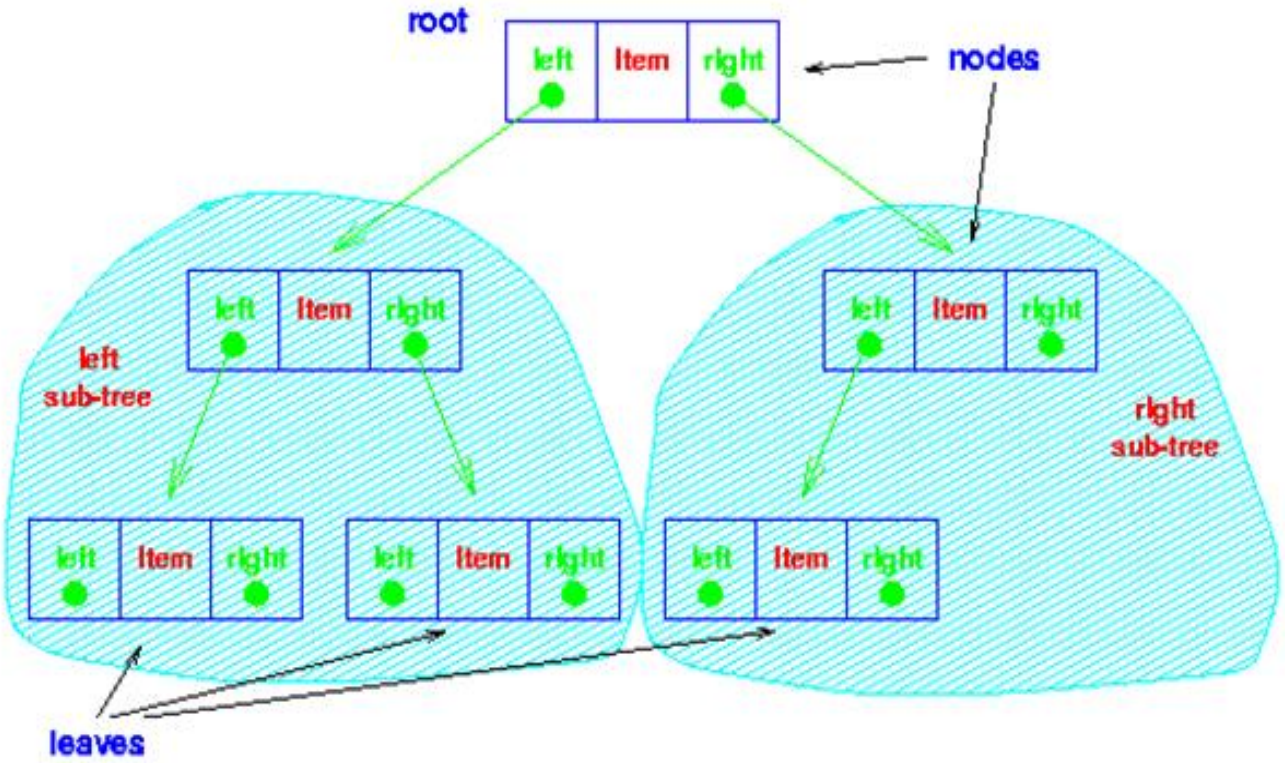
- infor: mô tả thông tin gắn với mỗi nút
- Con trỏ left chứa liên kết trỏ tới nút con trái hoặc NULL
- Con trỏ right chứa liên kết trỏ tới nút con phải hoặc NULL

**- Con trỏ root trỏ tới nút gốc của cây nhị phân**

# Khai báo cấu trúc cây nhị phân:

```
//Khai báo kiểu Item  (nếu cần)
struct node
{ Item  infor;
   struct node *left, *right;
 };
struct node  *root;
```

# Mô hình danh sách liên kết của cây nhị phân:



# 4.2.2 Các thao tác trên cây nhị phân

**1) Tìm cha của mỗi nút:**

- Hàm Parent(x) xác định cha của nút x. Nếu x là nút gốc thì không có cha nên Parent(x) sẽ nhận một giá trị đặc biệt nào đó hoặc bằng 0.
- **Thuật toán**: Duyệt các nút của cây.

- Nếu tồn tại nút y sao cho thành phần left hoặc thành phần Right của y là x thì Parent(x)= y.

- Nếu không tồn tại thì trả về giá trị đặc biệt nào đó hoặc bằng 0.

# 2) Tìm con trái của mỗi nút:

- Hàm LeftChild(x) xác định con bên trái của nút x. Nếu x là nút lá thì không có con nên LeftChild(x) sẽ nhận một giá trị đặc biệt nào đó.

- **Thuật toán**: Duyệt các nút của cây.

  - Nếu tồn tại nút y sao cho thành phần left của x là y thì leftChild(x)= y.

  - Nếu không tồn tại thì trả về giá trị đặc biệt nào đó.

# 3) Tìm con phải của mỗi nút

- Hàm RightChild(x) xác định con bên trái của nút x. Nếu x là nút lá thì không có con nên RightChild(x) sẽ nhận một giá trị đặc biệt nào đó.
- **Thuật toán**: Duyệt các nút của cây.

  - Nếu tồn tại nút y sao cho thành phần right của x là y thì RightChild(x)= y.

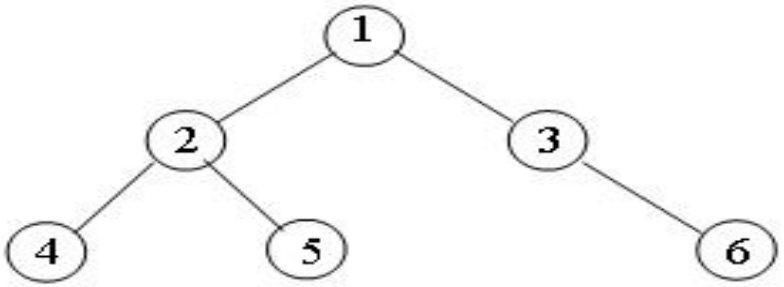
  - Nếu không tồn tại thì trả về giá trị đặc biệt nào đó.

# 4.3 Các phương pháp duyệt cây nhị phân

## 4.3.1 Duyệt cây theo thứ tự trước (Preorder traversal)

- Nếu cây T = $\varnothing \Rightarrow$ Dừng;
- Nếu cây T $\neq \varnothing$:
  - Thăm gốc
  - Duyệt cây con bên trái theo thứ tự trước
  - Duyệt cây con bên phải theo thứ tự trước

# Ví dụ 1: Cho cây




Thứ tự duyệt cây theo thứ tự trước:

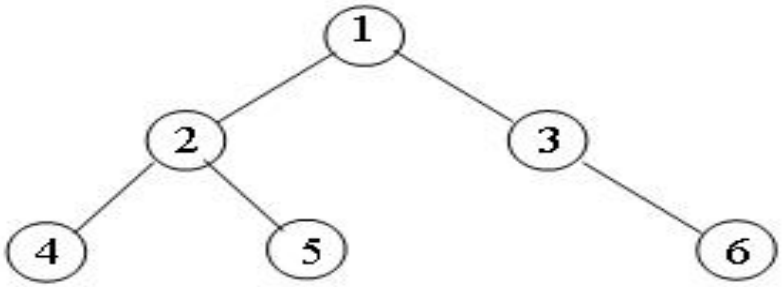
1, 2, 4, 5, 3, 6

# 4.3.2 Duyệt cây theo thứ tự giữa (Inorder traversal )

- Nếu cây T = $\varnothing \Rightarrow$ Dừng;
- Nếu cây T $\neq \varnothing$:

  - Duyệt cây con bên trái theo thứ tự giữa
  - Thăm gốc
  - Duyệt cây con bên phải theo thứ tự giữa

# Ví dụ 2: Cho cây




Thứ tự duyệt cây theo thứ tự giữa:

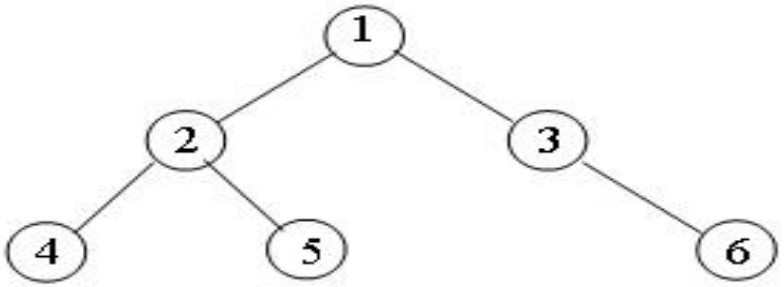
4, 2, 5, 1, 3, 6

# 4.3.3 Duyệt cây theo thứ tự sau (Postorder traversal)

- Nếu cây T = $\varnothing \Rightarrow$ Dừng;
- Nếu cây T $\neq \varnothing$:

  - Duyệt cây con bên trái theo thứ tự sau
  - Duyệt cây con bên phải theo thứ tự sau
  - Thăm gốc

# Ví dụ 3: Cho cây




Thứ tự duyệt cây theo thứ tự sau:

4, 5, 2, 6, 3, 1

# 4.4 Cây nhị phân tìm kiếm

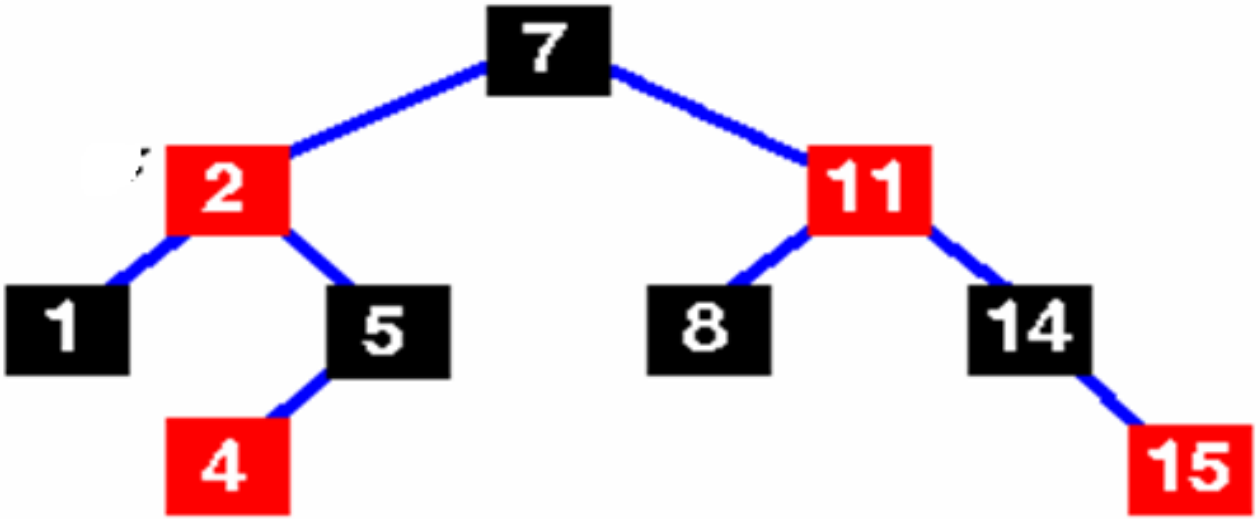
- Xét tập hợp các đối tượng D. Một thuộc tính của các phần tử $x \in D$ gọi là khóa (key) $\Leftrightarrow$ Với mọi $x \neq y \in D$ thì $key(x) \neq key(y)$.
- Giả thiết phần infor gắn với mỗi nút của cây nhị phân là khóa của một đối tượng nào đó

# 4.4.1 Định nghĩa

Cây nhị phân tìm kiếm là cây nhị phân hoặc rỗng, hoặc thỏa mãn các điều kiện sau:

- Khóa của các nút $\in$ cây con trái < khóa nút gốc
- Khóa nút gốc < khóa các nút $\in$ cây con phải.
- Cây con trái và cây con phải cũng là cây nhị phân tìm kiếm.

# Ví dụ: Cây nhị phân tìm kiếm



# 4.4.2 Cài đặt

```
//Khai báo kiểu Item  (nếu cần)
struct node
{ Item  key;
   struct node *left, *right;
 };
struct node  *root;
```

# 4.4.3 Các thao tác cơ bản

## 1) Xen một nút vào cây tìm kiếm

- **Input**: Cây nhị phân tìm kiếm với biến con trỏ root trỏ tới gốc của cây, k là khóa cho trước.
- **Output**: Cây nhị phân tìm kiếm có thêm một nút mới X với X->key = k.

# Cài đặt:

```
int insertT(struct node *root, item k)
{ struct node *X, *q;
   X= new node; X->key= k;    X->left= NULL;
    X->right= NULL;
   if  (root==NULL)   {root= X;  return 1;)}
    else   {q= root;
```

```
while (q!= NULL)
      if (k< q->key)
    { if (q->left!=NULL) q= q->left;  else {q-> left= X;
   return 1;}
     }  else if (k > q->key)
     {if (q->right!=NULL)q = q->right;   else {q->right = X;
   return 1;}
         } else return 0;
         }
}
```

# 2) Tìm kiếm

- **Input**: Cây nhị phân tìm kiếm với biến con trỏ root trỏ tới gốc của cây, k là khóa cho trước.
- **Output**: Nút X$\in$ cây nhị phân tìm kiếm có X->key = k.

# Cài đặt theo giải thuật đệ qui:

```
struct node *searchT(struct node *root, item k)
{if  (root==NULL) return NULL;
   else
if (k< root->key) return  searchT(root->left, k);
   else
if (k> root->key) return searchT(root->right, k);
   else return root;
}
```

# Cài đặt theo giải thuật lặp:

```
struct node *searchT(struct node *root, item k)
{int found= 0;  struct node *p;
 p= root;
while (p!= NULL) &&  (found==0))
  {if (k> p->key)     p= p->right;    else
        if (k< p->key) p= p->left;
           else found= 1;    }
if (found==0) return(NULL);    else   return p;
}
```

# 3) Loại bỏ một nút khỏi cây tìm kiếm

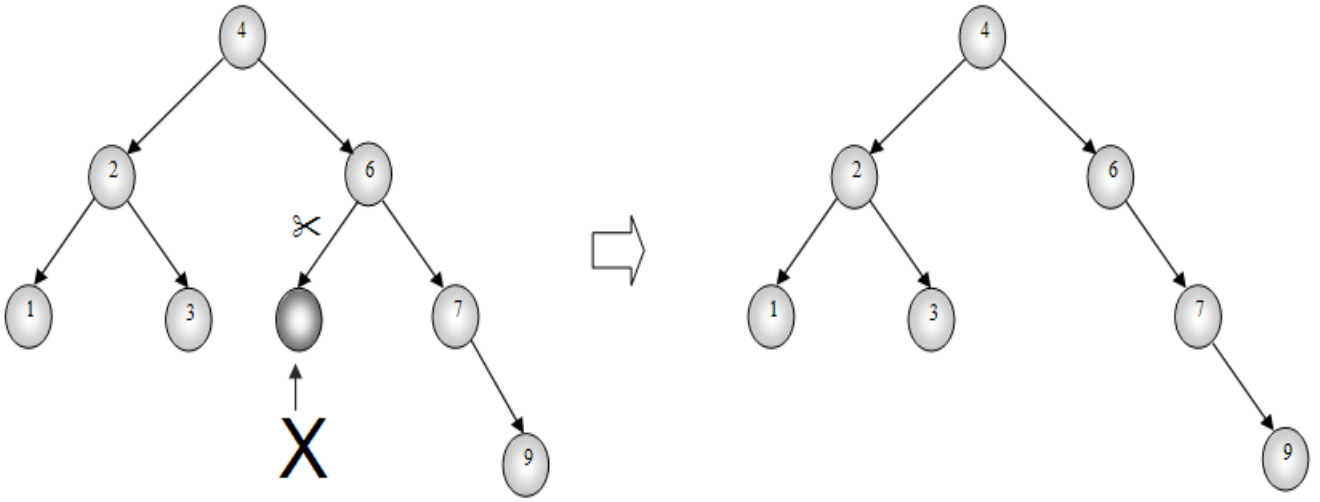
- **Input**: Cây nhị phân tìm kiếm với biến con trỏ root trỏ tới gốc của cây, k là khóa cho trước.
- **Output**: Cây nhị phân tìm kiếm bị loại một nút X với X->key = k.

- **Trường hợp 1**: X là nút lá

  $\Rightarrow$ khi xóa X không liên quan đến các nút khác

  $\Rightarrow$ cắt bỏ X khỏi cây

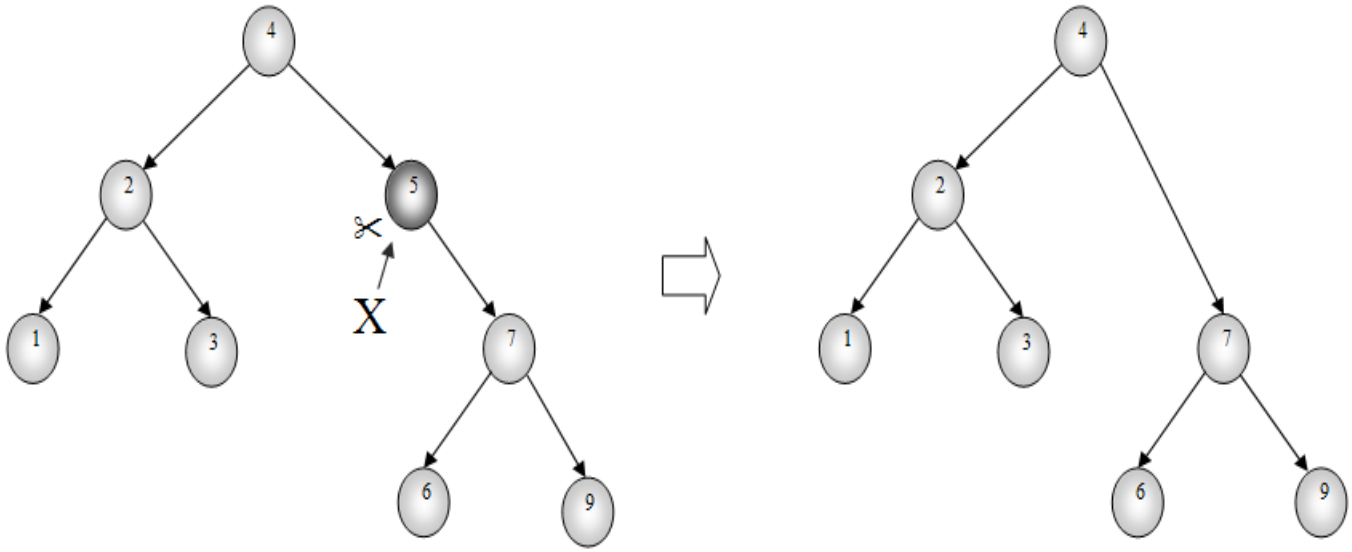
# Hình ảnh minh họa xóa bỏ một nút lá X :



**Trường hợp 2**: X là nút nửa lá (chỉ có 1 con)

$\Rightarrow$ khi xóa cần liên kết nút cha của X với nút con của X.

# Hình ảnh minh họa xóa bỏ một nút nửa lá X :

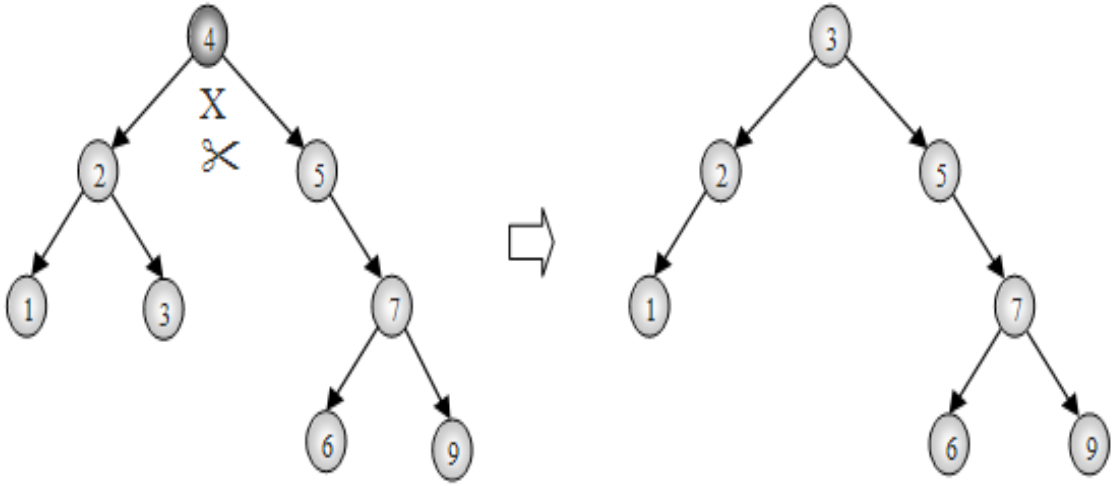


**Trường hợp 3**: X có đủ hai con

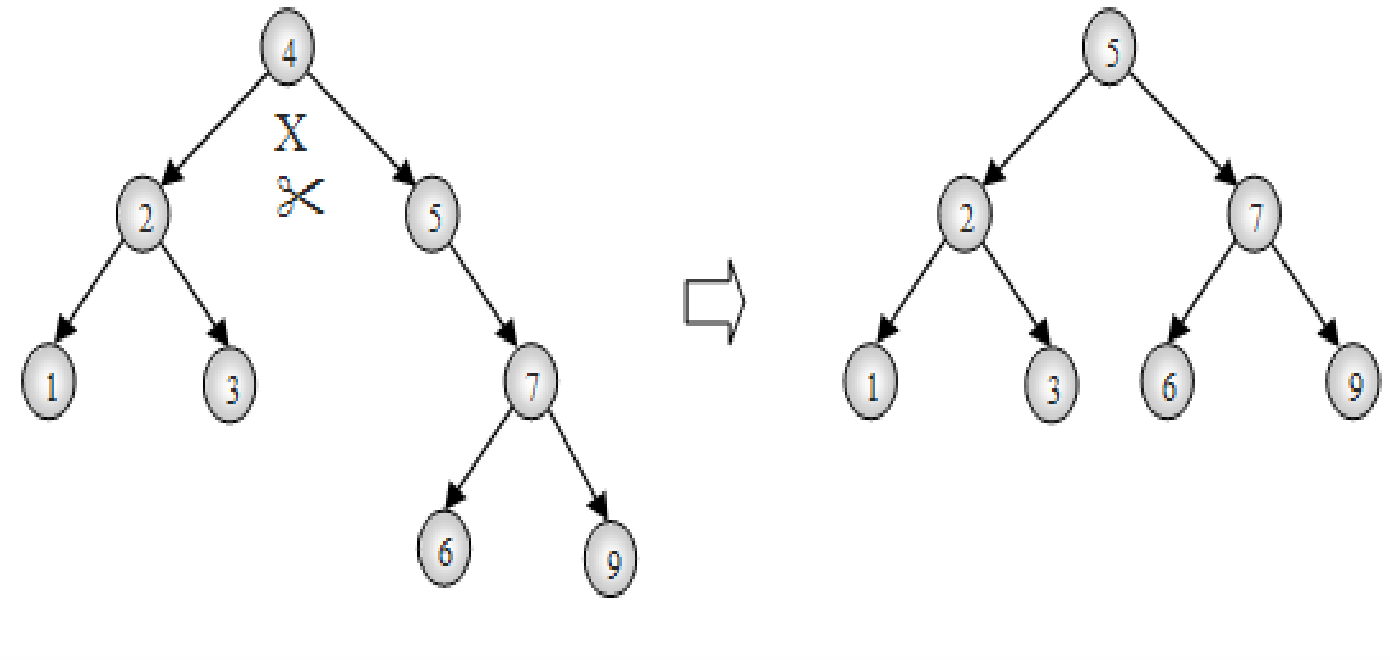
$\Rightarrow$ khi xóa X cần tìm nút D thay thế X:

- Chọn D là nút cực phải của cây con bên trái của X (hoặc nút cực trái của cây con bên phải);
- Thay thế X bởi D;
- Xóa D như trường hợp 1 hoặc 2;

# Nút con bên trái được chọn thay thế:



# Nút con bên phải được chọn thay thế:



# Cài đặt hàm xóa nút trỏ  bởi q:

```
void Del(struct node *q)
{ if (q-> left == NULL) //q là nút lá hoặc nửa lá
   { struct node *temp = q;
      q= temp -> right;  delete temp;
   } else
        if  (q -> right == NULL)
           { struct node *temp = q;
              q= temp -> left;     delete temp;
            } else
```

```
//q có hai con
   { // Tìm nút phải nhất của cây con trái
     struct node *temp = q->left;
     while (temp->right != NULL)
                    temp = temp->right;
     q->key= temp->key;
     Del(temp);}
}
```

# Cài đặt hàm loại khỏi cây gốc r một nút X có khóa là k:

```
void DeleteNode(struct node *r, Item k)
{ if (r != NULL)
   if (k < r-> key) DeleteNode(r->left, k);
     else
        if  (k > r -> key) DeleteNode(r->right, k);
          else Del(r);
}
```

# Cài đặt hàm xóa toàn bộ cây:

```
void ClearT(struct node *r)
  {  if (r != NULL)
       {  ClearT(r->left);
          ClearT(r->right);
          delete r;
        }
  }
```

# 4.4.4 Các loại cây nhị phân tìm kiếm

## 1) Cây AVL

Là cây nhị phân tìm kiếm mà tại mỗi node, độ cao của cây con trái và cây con phải sai khác nhau ≤ 1.

# a) Biểu diễn cây AVL:

```
struct node
        {       Item  key;
                int bal;
                struct node *left, *right;
        }
 struct node *root;
```

bal = 0: Hai cây con cao bằng nhau;

bal = -1: Cao bên trái;

bal = 1: Cao bên phải

# b) Xen một node có khóa k vào cây AVL

**Input:** Cây AVL, khóa k;

**Output**: Cây AVL mới có thêm node X có

X->key = k;

*Ký hiệu node gốc của cây con không cân bằng là node bất thường;*

# Giải thuật:

(1) Tạo node X có khóa k và xen vào cây tương tự cây NPTK;

(2) Có 3 trường hợp sau:

(2.1) Cây lệch trái (hoặc lệch phải) sau khi bổ sung vào cây con phải (hoặc trái) thì cây mới trở nên cân bằng $\Rightarrow$ Không phải làm gì cả ;

(2.2) Hai cây con có độ cao bằng nhau, sau khi bổ sung thì cây mới trở nên lệch trái (hoặc lệch phải) $\Rightarrow$ Không phải làm gì cả;

(2.3) Cây mới bị lệch $\Rightarrow$ chuyển (3);

(3) Cân bằng các node (bất thường) mà tại đó tính cân bằng bị phá vỡ (bal = ± 2):

(3.1) Node bổ sung làm tăng chiều cao của cây con trái của node con trái của node bất thường $\Rightarrow$ thực hiện phép quay phải:

- Đưa node con trái thay thế node bất thường;
- Đưa node bất thường thành con phải;
- Đưa cây con phải của node con trái thành cây con trái của node bất thường;

(3.2) Node bổ sung làm tăng chiều cao của cây con phải của node con phải của node bất thường

$\Rightarrow$ thực hiện phép quay trái:

- Đưa node con phải thay thế node bất thường;
- Đưa node bất thường thành con trái;
- Đưa cây con trái của node con phải thành cây con phải của node bất thường;

(3.3) Node bổ sung làm tăng chiều cao của cây con phải của node con trái của node bất thường $\Rightarrow$ thực hiện phép quay kép:

- Quay trái đối với cây con trái của node bất thường;
- Quay phải đối với cây con mới có gốc tại node bất thường;

(3.4) Node bổ sung làm tăng chiều cao của cây con trái của node con phải của node bất thường

$\Rightarrow$ thực hiện phép quay kép:

- Quay phải đối với cây con phải của node bất thường;
- Quay trái đối với cây con mới có gốc tại node bất thường;

# b) Hủy một node có khóa k của cây AVL

**Input:** Cây AVL, khóa k;

**Output**: Cây AVL mới đã hủy node X có X->key = k;

# Giải thuật:

(1) Xóa node X có X->key = k tương tự cây nhị phân tìm kiếm;

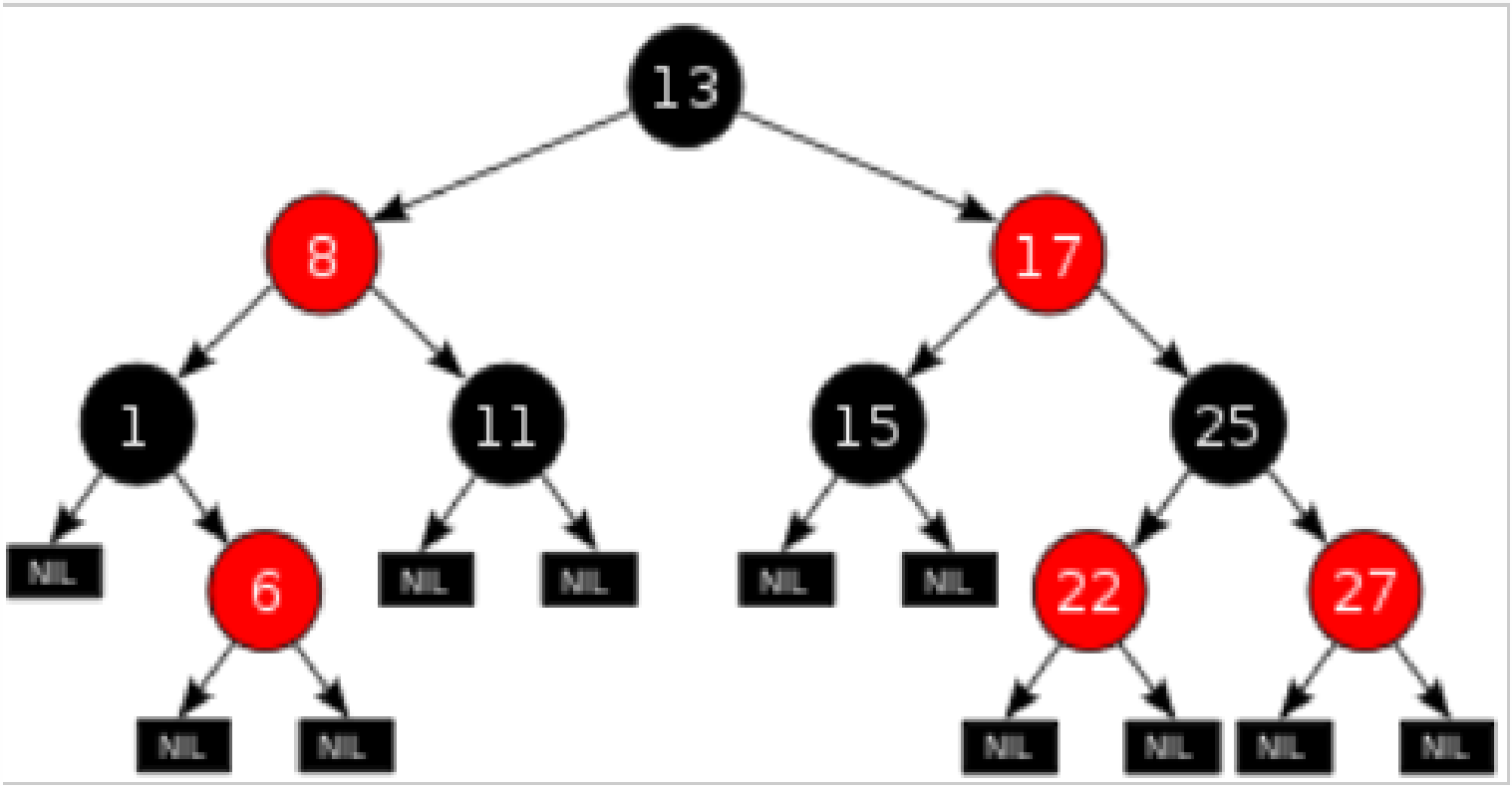
(2) Cân bằng lại cây nhận được;

# 2) Cây đỏ đen

**Định nghĩa** Cây đỏ đen là cây nhị phân tìm kiếm thỏa mãn:

- Mỗi nút hoặc là đỏ hoặc đen.
- Gốc là đen.
- Tất cả các lá là đen.
- Cả hai con của mọi nút đỏ là đen. (và suy ra mọi nút đỏ có nút cha là đen.)
- Tất cả các đường đi từ một nút đã cho tới các lá chứa một số như nhau các nút đen.

# Ví dụ cây đỏ - đen



# 4.5 Một số ứng dụng của cây nhị phân

## 4.5.1 Cây nhị phân biểu diễn biểu thức

**1) Định nghĩa**:

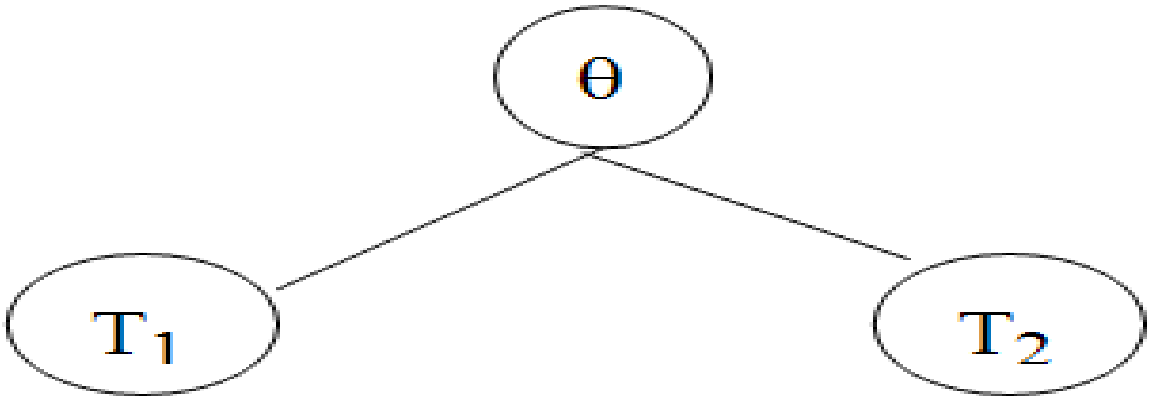
Cây biểu thức là cây nhị phân mà các nút gốc và nhánh chứa các phép toán, các nút lá chứa các toán hạng.

# 2) Cách xây dựng cây biểu thức

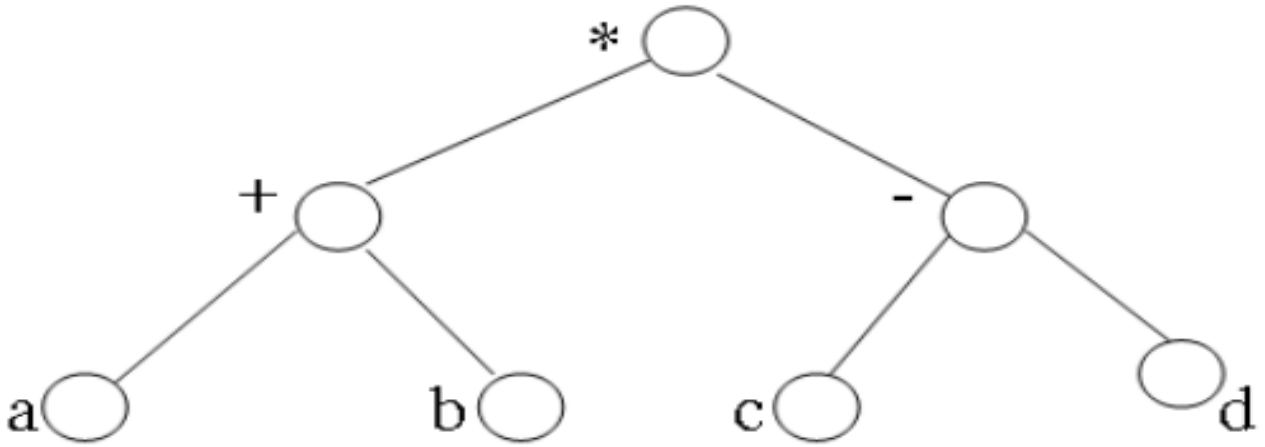
- Các phép toán hai ngôi (+, -, *, /, ^):

xét biểu thức E = ($E_1$) $\theta$ ($E_1$), trong đó $E_1$ được biểu diễn bởi cây nhị phân $T_1$, $E_2$ được biểu diễn bởi $T_2$ thì E được biểu diễn bởi cây có gốc chứa phép toán $\theta$, cây con bên trái $T_1$, cây con bên phải $T_2$.

# Minh họa cây biểu diễn E = (E1) θ (E2)



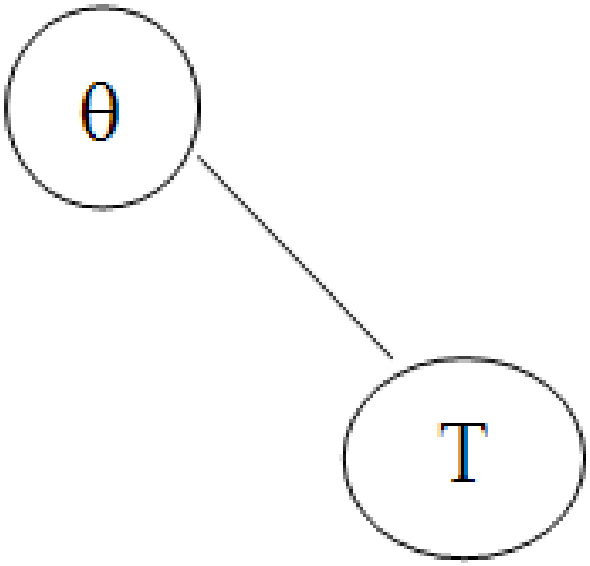
# Ví dụ cây biểu diễn E= (a + b)*(c - d)



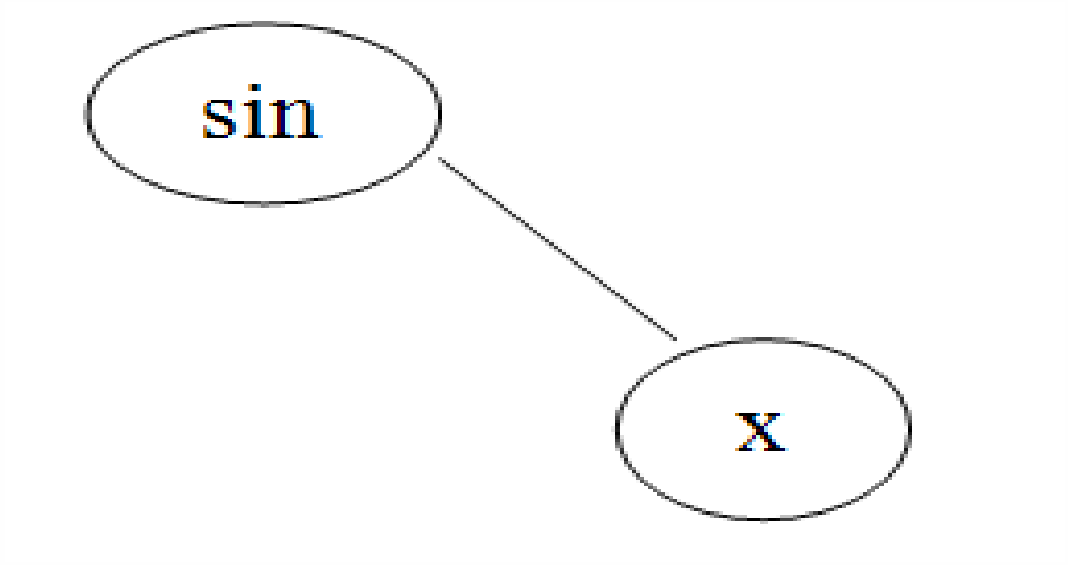
- Các phép toán 1 ngôi (sin x, …):

xét biểu thức $\theta(E)$, trong đó E được biểu diễn bởi cây T thì biểu thức được biểu diễn bởi cây có gốc chứa phép toán $\theta$, cây con bên trái rỗng, cây con bên phải T

# Minh họa cây biểu diễn θ(E)



# Ví dụ cây biểu diễn E= sin x



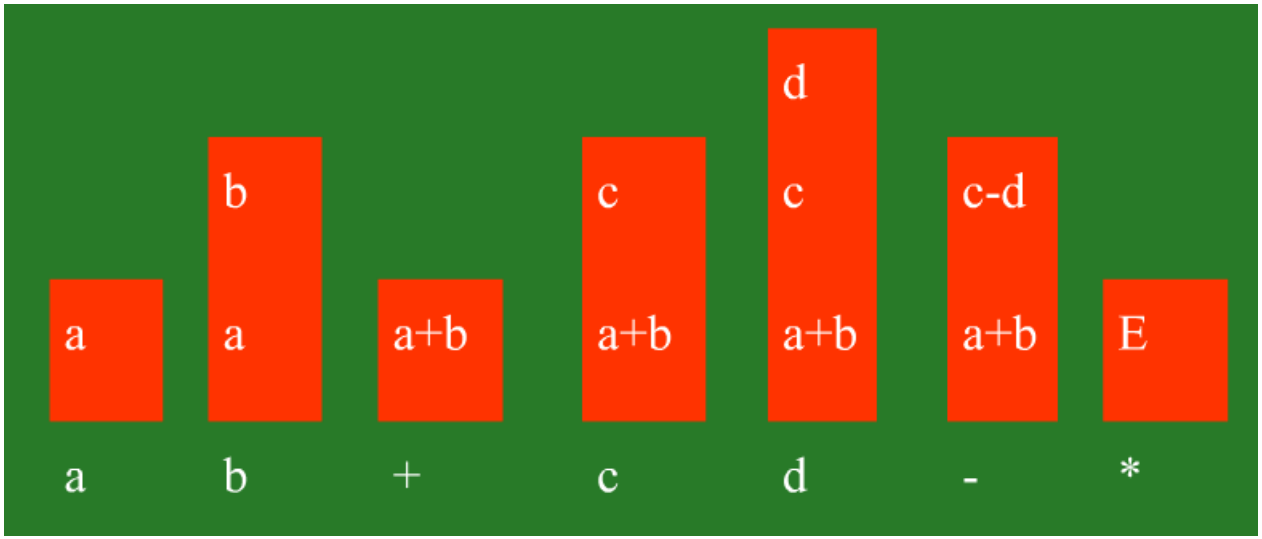
# 3) Duyệt cây biểu thức

- Duyệt theo thứ tự trước $\Rightarrow$ có biểu thức BaLan dạng tiền tố
- Duyệt theo thứ tự sau $\Rightarrow$ có biểu thức BaLan dạng hậu tố
- Duyệt theo thứ tự giữa $\Rightarrow$ có biểu thức BaLan dạng thông thường (trung tố)

# Ví dụ:

Với biểu thức E= (a + b)*(c - d) duyệt cây theo thứ tự sau có: a b + c d - *

# Stack tính E :



# 4.5.2 Mã Huffman

**1) Định nghĩa:**

Cho một bản tin M gồm các ký hiệu là phần tử của tập hợp hữu hạn A. Biết tần số xuất hiện các phần tử của A trong M. Bộ mã tiền tố của A sao cho độ dài mã của M ngắn nhất gọi là mã Huffman.

Cây nhị phân T với nút lá gán nhãn là ký hiệu $\in$ A, các cạnh gán 0 hoặc 1 biểu diễn mã tiền tố của A gọi là cây mã Huffman

# 2) Thuật toán tạo cây mã Huffman

**Input**: Bảng tần số xuất hiện các ký hiệu $\in$ A

**Output**: Cây nhị phân Huffman

(1) Tạo rừng T gồm n cây, mỗi cây chỉ gồm 1 đỉnh chứa tần số xuất hiện f(x), $x \in A$.

(2) Nếu T là cây thì dừng và xuất T.

(3) Chọn hai cây có gốc với giá trị nhỏ nhất nối thành cây với gốc mới nhận giá trị là tổng hai gốc. Cạnh trái gán nhãn 0, cạnh phải gán nhãn 1

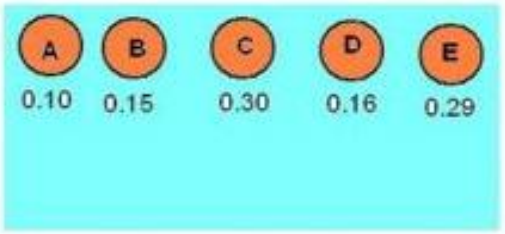
(4) Cây có gốc nhỏ hơn là con trái, cây kia là con phải và quay lại (2)

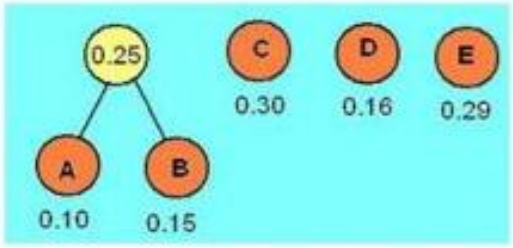
# Ví dụ

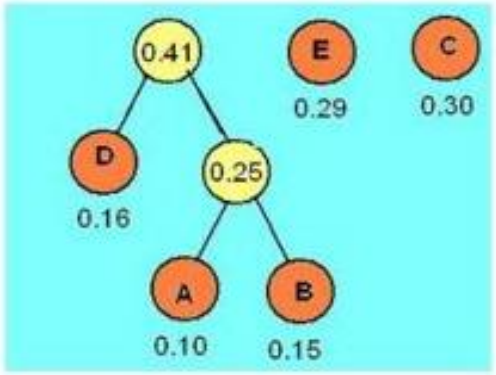
Cho bảng tần suất của 5 chữ cái A, B, C, D, E như sau

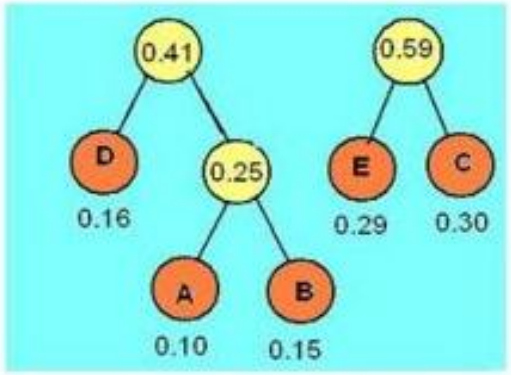
| A | B | C | D | E |
|---|---|---|---|---|
| 0,10 | 0,15 | 0,30 | 0,16 | 0,29 |

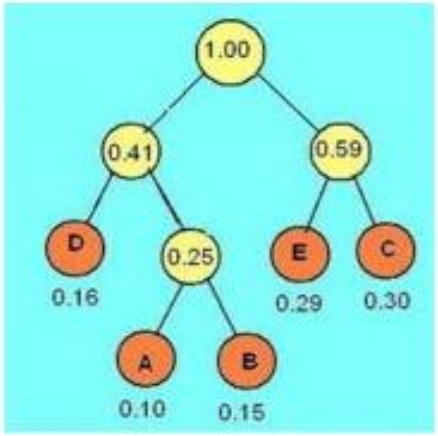
# Quá trình xây dựng cây Huffman như sau:



# ⇒ bộ mã tối ưu tương ứng là

| A | B | C | D | E |
|---|---|---|---|---|
| 010 | 011 | 11 | 00 | 10 |

# 3) Nén file bằng mã Huffman

Trong các bước trên, giả sử đã xây dựng được bộ mã Huffman của 256 ký hiệu có mã ASCII từ 0 đến 255 chứa trong mảng Code[1..256]. Việc nén file có thể phân tích sơ bộ như sau:

1. Đọc từng byte của file cần nén cho đến khi hết tệp,
2. Chuyển theo bộ mã thành xâu nhị phân,
3. Ghép với xâu nhị phân còn dư từ bước trước,
4. Nếu có đủ 8 bit trong xâu thì cắt ra tám bít đầu tiên ghi vào tệp nén,
5. Nếu đã hết tệp cần nén thì dừng

# 4.5.3 Sắp xếp kiểu vun đống (Heap Sort)

**Sử dụng cấu trúc cây nhị phân gần đầy:**

- Cây nhị phân gần đầy là cây tại các mức trừ mức cuối cùng, mỗi nút đều có đủ hai con; riêng mức cuối các nút đầy từ trái sang phải;

- Cây nhị phân gần đầy được mô tả bởi mảng sao cho con của nút thứ i là:

    + Con trái có thứ tự 2*i;

    + Con phải có thứ tự 2*i + 1;

# 1) Khái niệm đống

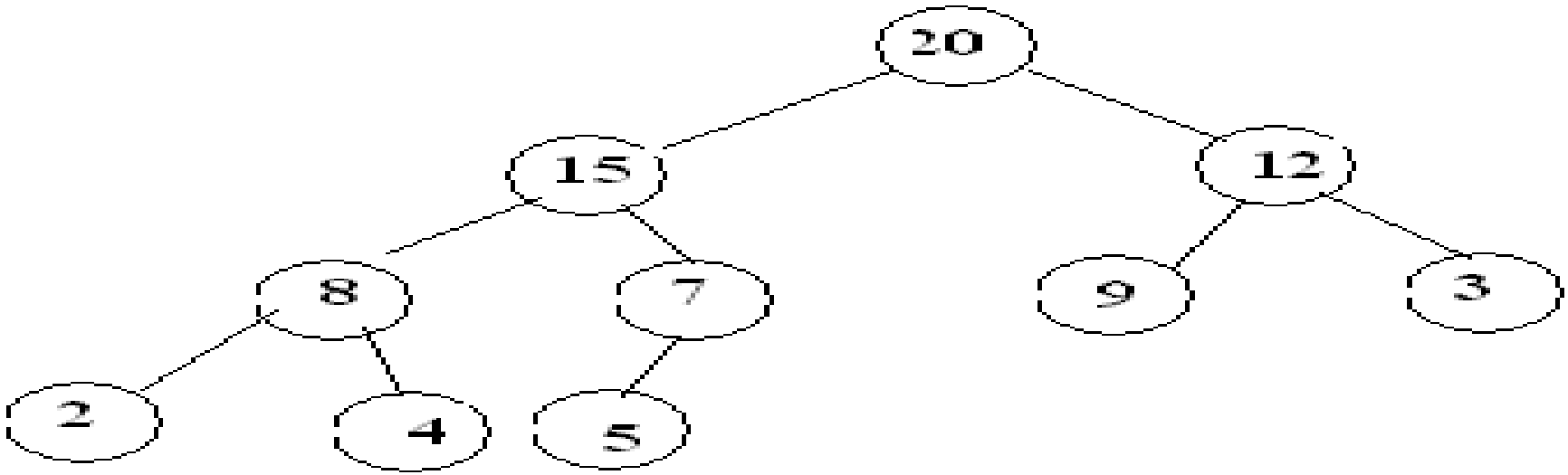
- Đống là cây nhị phân gần đầy sao cho khóa của nút cha > khóa của mọi con

  $\Rightarrow$ khóa của gốc là lớn nhất

# Ví dụ

Xét mảng K[] = (20, 15, 12, 8, 7, 9, 3, 2, 4, 5).

Có đống tương ứng là:



# 2) Phương pháp vun đống

- Thực hiện n/2 lần duyệt với thứ tự i = n/2 ngược đến 1:
- Với mỗi lần duyệt i biến đổi cây gốc i thành đống:

(1) key = K[i]:

(2) Xét con trái tại vị trí j = 2*i;

(3) Lặp khi j ≤ n:

3.1 Nếu con phải lớn hơn thì chọn j = j + 1;

3.2 Nếu key > K[j] thì key được gán cho cha của j: K[j/2] = key; return;

3.3 Nếu key < K[j]: K[j/2] = K[j]; j = 2*j; Quay lại 3.1;

(4) K[j/2] = key;

# Cài đặt:

```
void DongI(int i, int n)
{int key = k[i]; int j = 2*i;
while (j < = n)
   { if (j < n && k[j] < k[j+1])  j++;
     if (key > k[j]) {k[j/2] = key; return;}
   k[j/2] = k[j]; j= 2*j; }
   k[j/2] = key;
  }
```

# 3) Phương pháp sắp xếp

- **Pha** 1: Vun đống dãy ban đầu gồm các phần tử từ 1 đến n;
- **Pha 2**: Thực hiện n-1 lần duyệt với i= n-1 ngược đến 1:

  (1) Đổi chỗ hai phần tử ở đỉnh đống (thứ 1) và đáy đống (thứ i+1);

  (2) Vun đống dãy gồm các phần tử từ thứ 1 đến i;

# Cài đặt:

```
void Heap_Sort(int k[], int n)
{ int i;
   for (i = n/2; i >= 1; i--) DongI(i, n);//(1)
   for (i = n-1; i >= 1; i--)//(2)
   {int tmp = k[i+1]; k[i+1] = k[1]; k[1] = tmp;//(3)
         DongI(1, i);//(4)}
}
```

# 4) Đánh giá độ phức tạp

(1): O(nlogn);

(3): O(1);

(4): O(logn);

(2): O(n)*(O(1) + O(logn));

$\Rightarrow$ T(n) = O(nlogn);

# 4.6 Cây quyết định (Decision Tree)

Xét bài toán có nhiều trường hợp được biểu diễn lời giải bởi một cây và trong mỗi trường hợp cần lựa chọn lời giải tương ứng

$\Rightarrow$ Cây quyết định.

# Ví dụ:

**Bài toán:** Cho 8 đồng tiền vàng có hình dạng giống hệt nhau, nhưng có một đồng tiền giả có trọng lượng khác với các đồng tiền còn lại. Hãy xác định đồng tiền giả bằng cách dùng cân hai đĩa sao cho số lần cân là ít nhất?

# Giải:

**INPUT:**

Đánh số các đồng tiền đã cho là a, b, c, d, e, f, g, h;

**OUTPUT:**

x.H hoặc x.L với ý nghĩa đồng tiền giả x nặng hơn hoặc nhẹ hơn các đồng tiền thật;

$\Rightarrow$ Số lần cân ít nhất là 3

(1) Lần cân 1: a+b+c ? d+e+f

Nếu ? là > thì chuyển (2), nếu ? là = thì chuyển (3), nếu ? là < thì chuyển (4);

(2) Lần cân 2: a+d ? b+e

Nếu ? là > thì chuyển (2.1), nếu ? là = thì chuyển (2.2), nếu ? là < thì chuyển (2.3);

(2.1) Lần cân 3: a?b

Nếu ? là > thì a.H, nếu ? là = thì e.L;

(2.2) Lần cân 3: c?d

Nếu ? là > thì c.H, nếu ? là = thì f.L;

(2.3) Lần cân 3: b?a

Nếu ? là > thì b.H, nếu ? là = thì d.L;

(3) Lần cân 2:  g ? h

Nếu ? là > thì chuyển (3.1), nếu ? là < thì chuyển (3.2);

(3.1) Lần cân 3: g?a

Nếu ? là > thì g.H, nếu ? là = thì h.L;

(3.2) Lần cân 3: h?a

Nếu ? là > thì h.H, nếu ? là = thì g.L;

(4) Lần cân 2: a+d ? b+e

Nếu ? là > thì chuyển (4.1), nếu ? là = thì chuyển (4.2), nếu ? là < thì chuyển (4.3);

(4.1) Lần cân 3: a?b

Nếu ? là > thì b.L, nếu ? là = thì d.H;

(4.2) Lần cân 3: a?c

Nếu ? là > thì c.L, nếu ? là = thì f.H;

(4.3) Lần cân 3: b?a

Nếu ? là > thì a.L, nếu ? là = thì e.H;

# Ghi chú:

- Sử dụng mô hình cây $\Rightarrow$ Cây quyết định

# Thảo luận


THANK YOU