LY TU TRONG COLLEGE

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

GIẢNG VIÊN: NGUYỄN VÂN GIANG

LY TU TRONG COLLEGE

CHƯƠNG 1: MỞ ĐẦU

Mục tiêu:

Sau khi học xong người học hiểu được:

- Mục đích, vị trí, vai trò của môn học;
- Khái niệm cấu trúc dữ liệu, giải thuật và các yêu cầu chung khi xây dựng cấu trúc dữ liệu và giải thuật.

LY TU TRONG COLLEGE

1.1. CẤU TRÚC DỮ LIỆU (Data Structure)

- CTDL là cách thức tổ chức dữ liệu sao cho phản ánh đúng thực tế bài toán.
- CTDL là tập hợp các biến và các kiểu dữ liệu được liên kết với nhau.

Ví dụ: CTDL của chương trình quản lý SV

```
typedef struct SV
{     char MaSV[10];
      char TenSV[50];
      int NS;
      float Diem;
};
SV a[100];
```

- Một cấu trúc dữ liệu tốt phải thỏa:

+ Phản ánh đúng thực tế

+ Phù hợp với các thao tác xử lý trên dữ liệu đó

+ Tiết kiệm tài nguyên của hệ thống (thời gian,bộ nhớ):

LY TU TRONG COLLEGE

1.2. GIẢI THUẬT (Algorithms)

- Giải thuật là trình tự các thao tác xử lý dữ liệu để giải quyết bài toán.
- Gọi n là kích thước của dữ liệu đầu vào, ta ký hiệu T(n) là hàm biểu diễn thời gian thực hiện của giải thuật.
- Mức độ tăng lên của T(n) khi n tăng lên gọi là độ phức tạp tính tóan của giải thuật và được biểu điễn bằng hàm O (gọi là Big O).
- Các lớp độ phức tạp tính toán phổ biến:

$$O(1) < O(\log n) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(2^n)$$

1.3. MỐI QUAN HỆ GIỮA CTDL VÀ GIẢI THUẬT

Data structures + Algorithms = Programs

- Để xác định được giải thuật phù hợp cần phải biết nó tác động đến những loại dữ liệu nào

- Khi chọn lựa một cấu trúc dữ liệu cũng cần phải hiểu rõ những thao tác nào sẽ được tác động lên nó

BÀI TẬP

Viết chương trình DanhSachSoNguyen.cpp thực hiện các công việc sau:

a. Nhập mảng số nguyên

b. Xuất mảng số nguyên

c. Cho phép người dùng nhập một số x và kiểm tra xem trong mảng có số x hay không ?

Chỉ số i	0	1	2	3	4	5	6	7 (n-1)
Giá trị a[i]	7	2	0	9	8	1	4	3

LY TU TRONG COLLEGE

CHƯƠNG 2: TÌM KIẾM VÀ SẮP XẾP
BÀI 1: TÌM KIẾM (SEARCH)

Mục tiêu:

Sau khi học xong người học hiểu được:

- Trình bày được ý tưởng của phương pháp tìm kiếm tuyến tính, tìm nhị phân;
- Mô phỏng và đánh giá được giải thuật của phương pháp tìm kiếm tuyến tính, tìm nhị phân;
- Cài đặt được các giải thuật tìm kiếm trên mảng.
- Vận dụng được các giải thuật tìm kiếm trong lập trình các ứng dụng

1. Bài toán tìm kiếm

INPUT:

- Dãy (a) gồm n phần tử $a_0, a_1, \ldots, a_{n-1}$.
- x là giá trị của phần tử cần tìm

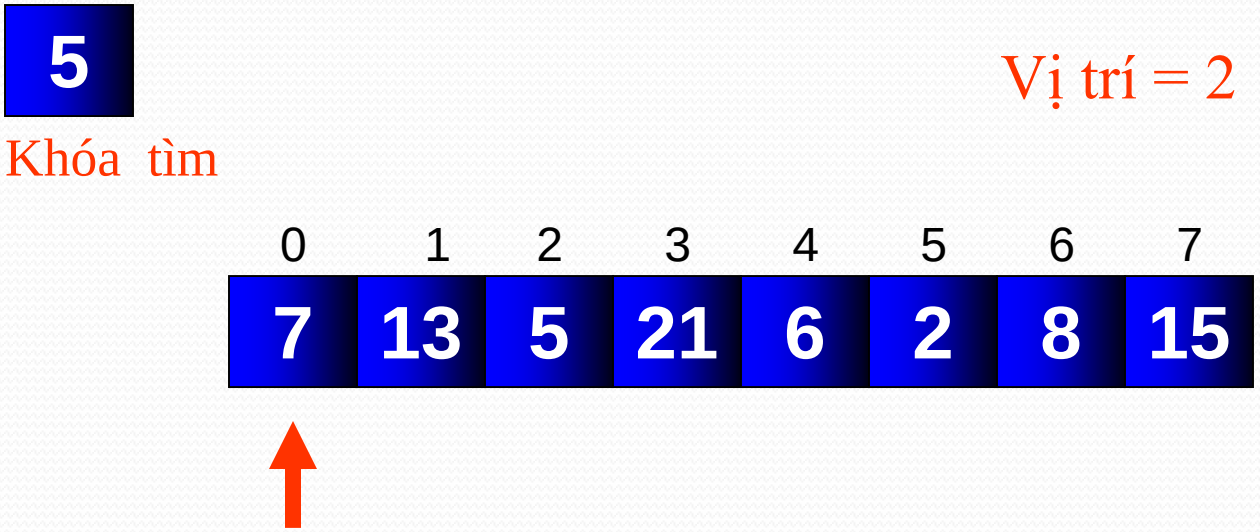
OUTPUT:

k ($0 \le k \le n-1$) nếu $a_k = x$, (k là vị trí xuất hiện của x trong dãy (a))

- 1 nếu trong dãy không có phần tử nào bằng x

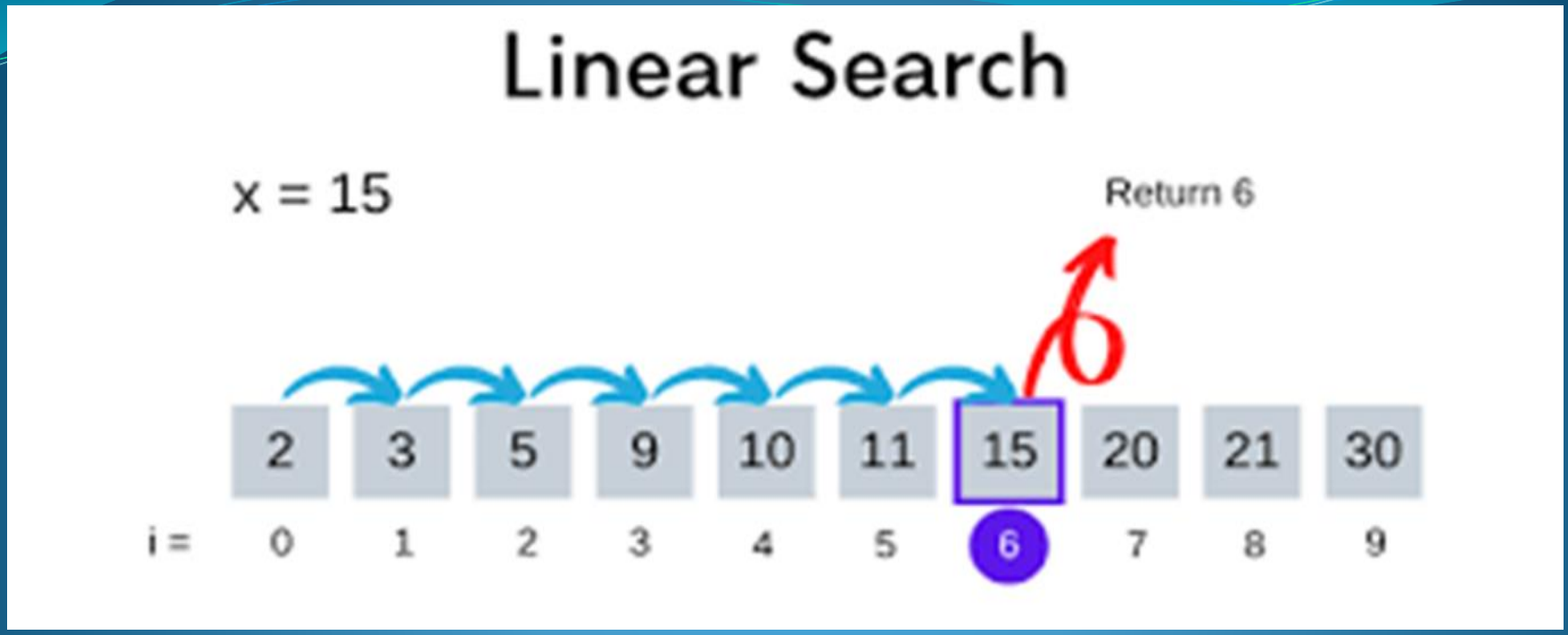
2. Tìm tuyến tính (Linear Search)

2.1. Ý tưởng của giải thuật



Tìm thành công

Số lần so sánh: 3



Lần lượt so sánh x với các a[i] (i=0,1,2,…,n-1)

- Nếu có a[i] = x thì dừng và trả về i
- Nếu không có a[i] nào bằng x thì trả về -1

i	0	1	2	3	4	5	6	7(n-1)
a[i]	12	2	8	5	1	6	8	15

Lưu ý : Giải thuật tìm tuyến tính luôn trả về vị trí xuất hiện đầu tiên của x trong dãy số.

2.2. Cài đặt

```
int linearSearch (int a[], int n, int x)
{
      int i = 0;
      while (i<n&&a[i]!=x)
            i++;
      if( i==n)return -1;
      return i;
}
```

LY TU TRONG COLLEGE

2.3. Thuật toán cải tiến

```
int LinearSearch (int a[], int n, int x)
{
      int i = 0;
      a[n]=x;
      while (a[i]!=x)
            i++;
      if( i==n)return -1;
      return i;
}
```

LY TU TRONG COLLEGE

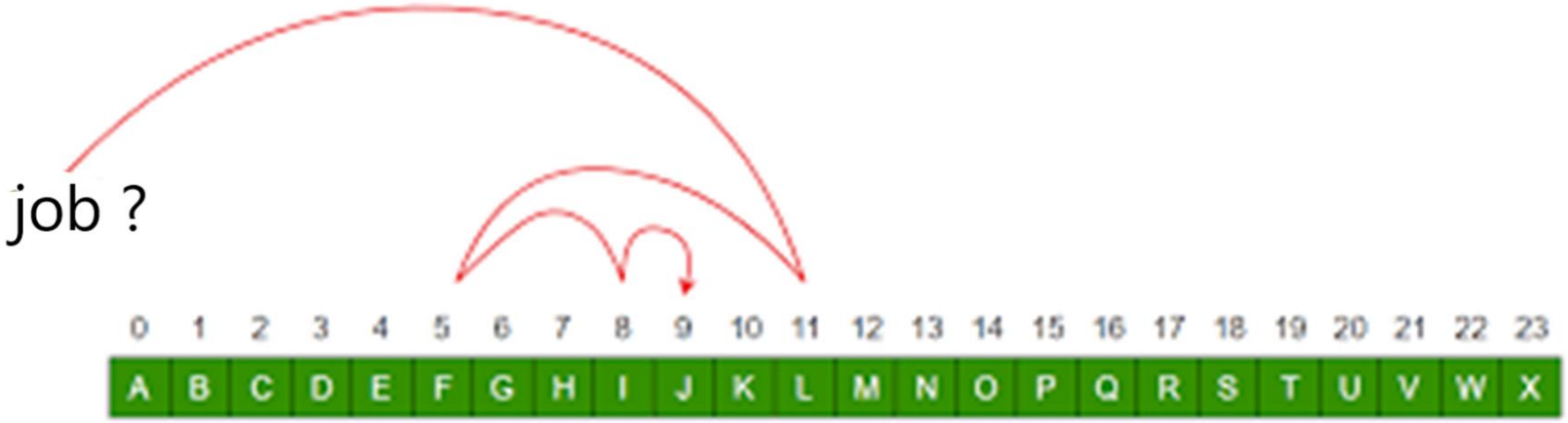
Đánh giá giải thuật:

Độ phức tạp của giải thuật là O(n)

Trường hợp	Số lần so sánh	Ghi chú
Tốt nhất	1	Phần tử đầu tiên có giá trị x
Xấu nhất	n + 1	Không có x trong mảng
Trung bình	(n+1)/2	Giả sử xác suất các phần tử trong mảng nhận giá trị x là như nhau.

3. Tìm nhị phân (Binary Search)

3.1. Ý tưởng của giải thuật



Điều kiện áp dụng: Dãy (a) đã được sắp thứ tự tăng dần

Xét dãy: a[l], a[l+1],…,a[r] (l ≤ r)

- Gọi m=(l+r)/2 và so sánh a[m] với x :
  + Nếu a[m]=x: Trả về m
  + Nếu a[m]<x: Thực hiện lại giải thuật với l=m+1
  + Nếu a[m]>x: Thực hiện lại giải thuật với r=m-1
- Giải thuật dừng khi l> r và trả về -1

3.2. Cài đặt:

```
int binarySearch(int a[],int n,int x)
{
      int l=0,r=n-1,m;
      do
      {
            m=(l+r)/2;
            if (x==a[m])return m;
            if (x<a[m]) r=m-1;
            else l=m+1;
      }while (l<=r);
      return -1;
}
```

Trường hợp	Số lần so sánh	Ghi chú
Tốt nhất	1	Phần tử giữa của mảng có giá trị x
Xấu nhất	$\log_2 n$	Không có x trong mảng
Trung bình	$\log_2 n/2$	Giả sử xác suất các phần tử trong mảng nhận giá trị x là như nhau

BÀI TẬP

Viết tiếp chương trình **DanhSachSoNguyen.cpp**, thực hiện các công việc sau :

d. Cài đặt giải thuật tìm tuyến tính cải tiến và gọi hàm này để tìm số x có trong mảng không ? Nếu có thì trả về vị trí của số x, nếu không có thì xuất câu thông báo.

e. Cài đặt hàm tìm nhị phân và gọi hàm này để tìm số x có trong mảng không ? Nếu có thì trả về vị trí của số x, nếu không có thì xuất câu thông báo.

f. So sánh kết quả tìm kiếm của 2 hàm nếu x=7 và (a) : 1, 3, 7, 7, 8, 9,11. Giải thích kết quả.

LY TU TRONG COLLEGE

BÀI 2: SẮP XẾP (SORT)

Mục tiêu:

Sau khi học xong người học có khả năng:

- Trình bày được ý tưởng của phương pháp sắp xếp Chèn trực tiếp và Phân hoạch;
- Mô phỏng và đánh giá được giải thuật của phương pháp sắp xếp Chèn trực tiếp và Phân hoạch;
- Cài đặt được các giải thuật sắp xếp trên mảng.
- Vận dụng được các giải thuật sắp xếp trong lập trình các ứng dụng

LY TU TRONG COLLEGE

1. Bài toán sắp xếp

Giả sử cần sắp xếp tăng dần

INPUT:

- Dãy (a) gồm n phần tử a_0, a_1,..., a_{n-1}.

OUTPUT:

- Dãy (a) thỏa : $a_0 \leq a_1 \leq ... \leq a_{n-1}$

2. Phương pháp đổi chỗ trực tiếp (Interchange Sort)

Bước	0	1	2	3	4	5	6
Bước 0	4	6	5	1	2	9	3
Bước 1	1	6	5	4	2	9	3
Bước 1	1	5	6	4	2	9	3
Bước 1	1	4	6	5	2	9	3
Bước 2	1	2	6	5	4	9	3
Bước 2	1	2	5	6	4	9	3
Bước 2	1	2	4	6	5	9	3
Bước 3	1	2	3	6	5	9	4
Bước 3	1	2	3	5	6	9	4
Bước 4	1	2	3	4	6	9	5
Bước 5	1	2	3	4	5	9	6
Kết quả	1	2	3	4	5	6	9

Trình bày rút gọn:

Bước	0	1	2	3	4	5	6
Bước 0	4	6	5	1	2	9	3
Bước 1	1	6	5	4	2	9	3
Bước 2	1	2	6	5	4	9	3
Bước 3	1	2	3	6	5	9	4
Bước 4	1	2	3	4	6	9	5
Bước 5	1	2	3	4	5	9	6
Kết quả	1	2	3	4	5	6	9

2.2. Ý tưởng của giải thuật:

Giải thuật gồm n-1 bước (từ bước i= 0 đến bước i= n-2).

Ở bước 0, ta phải sắp xếp sao cho a[0] sẽ là phần tử nhỏ nhất dãy a[0],…,a[n-1].

Ở bước 1, ta phải sắp xếp sao cho a[1] sẽ là phần tử nhỏ nhất dãy a[1],…,a[n-1]

Tổng quát:

Ở bước i, ta phải sắp xếp sao cho a[i] sẽ là phần tử nhỏ nhất dãy a[i],…,a[n-1], bằng cách :

So sánh a[i] với các a[j] (i+1 ≤ j≤n-1), nếu thấy a[i]>a[j] thì hóan vị a[i] với a[j].

Giải thuật gồm n-1 bước (bước i= 0,... n-2).

Ở bước i, ta phải sắp xếp sao cho a[i] sẽ là phần tử nhỏ nhất dãy a[i],…,a[n-1], bằng cách :

So sánh a[i] với các a[j] (i+1 ≤ j≤n-1), nếu thấy a[i]>a[j] thì hóan vị a[i] với a[j].

2.3. Cài đặt:

```
void interchangeSort(int a[], int n)
{
    for (int i=0; i<=n-2;i++)
        for (int j=i+1;j<=n-1 ;j++)
            if (a[i]>a[j])
                swap(a[i],a[j]);
}
```

LY TU TRONG COLLEGE

```
void interchangeSort(int a[], int n)
{
      for (int i=0; i<=n-2;i++)
            for (int j=i+1;j<=n-1 ;j++)
                  if (a[i]>a[j])
                        swap(a[i],a[j]);
}
```

3. Phương pháp phân hoạch (Quick Sort)

3.1. Ý tưởng của giải thuật

a [left] ---------- x ----------a[right]

a[i]<x a[j]>x

left ----------→ i

j←----------right

Giả sử ta cần sắp tăng dần dãy a[l],...,a[r]

Bước 1: Chọn x=a[(l+r)/2] làm mốc, i=l, j=r

Bước 2:

Trong khi a[i]<x thì tăng i

Trong khi a[j]>x thì giảm j

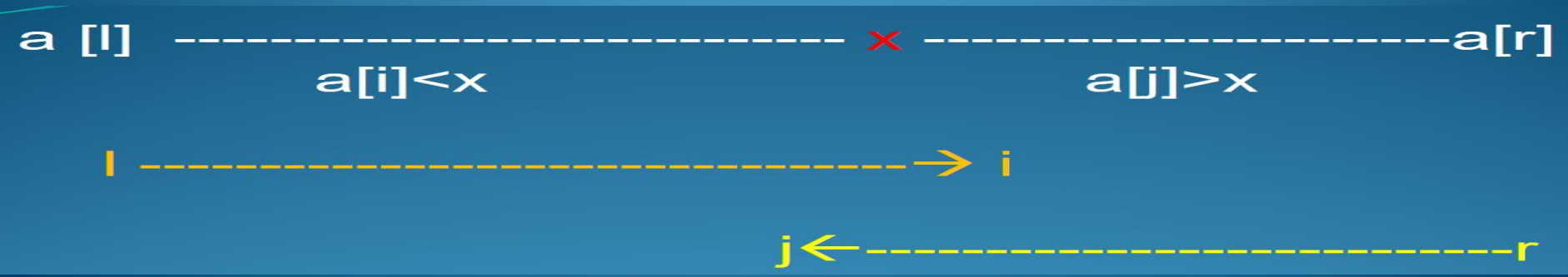
Nếu i<=j thì hóan vị a[i], a[j] rồi tăng i, giảm j

Nếu i<=j thì quay lại bước 2, ngược lại qua bước 3

Bước 3:

Nếu l<j thì thực hiện lại giải thuật với r=j

Nếu i<r thì thực hiện lại giải thuật với l=i



Ví dụ 1: Sắp dãy sau tăng dần bằng Quick Sort: 2, 9, 5, 4, 7, 6, 8

	-1	*0*	*1*	*2*	*3*	*4*	*5*	*6*
l=0, r=6, *x=4*		i **2**	i **9**	**5**	j **(4)**	j **7**	j **6**	j **8**
		2	j **(4)**	i j **5**	**9**	j **7**	j **6**	j **8**
l=0, *r=1, x=2*	j	i j **(2)**	i j **4**					
l=2, *r=6, x=7*				i **5**	i **9**	**(7)**	j **6**	j **8**
				5	j **6**	i j **(7)**	i **9**	**8**

	0	1	2	3	4	5	6
l=2, r=3, *x=5*		j	i j **(5)**	i j **6**			
l=5, r=6, *x=9*						i **(9)**	j **8**
						j **8**	i **(9)**
Kết quả	2	4	5	6	7	8	9

LY TU TRONG COLLEGE

3.3. Cài đặt:

```
void quickSort(int a[],int l,int r)
{
     int  x=a[(l+r)/2],i=l,j=r;
     do
     {
          while (a[i]<x) i++;
          while (a[j]>x) j--;
          if (i<=j)
          {    swap(a[i],a[j]); i++;j--;}
     }while (i<=j);
     if (l<j) quickSort(a,l,j);
     if (i<r) quickSort(a,i,r);
}
```

LY TU TRONG COLLEGE

Đánh giá giải thuật:

Độ phức tạp của giải thuật là O (nlogn)

BÀI TẬP

Bổ sung chương trình `DanhSachSoNguyen.cpp`, thực hiện các công việc sau bằng 2 phương pháp : Interchange Sort và Quick Sort

g. Sắp xếp danh sách tăng dần.

h. Sắp xếp danh sách giảm

CHƯƠNG 3: CÂY
Bài 1: CẤU TRÚC CÂY

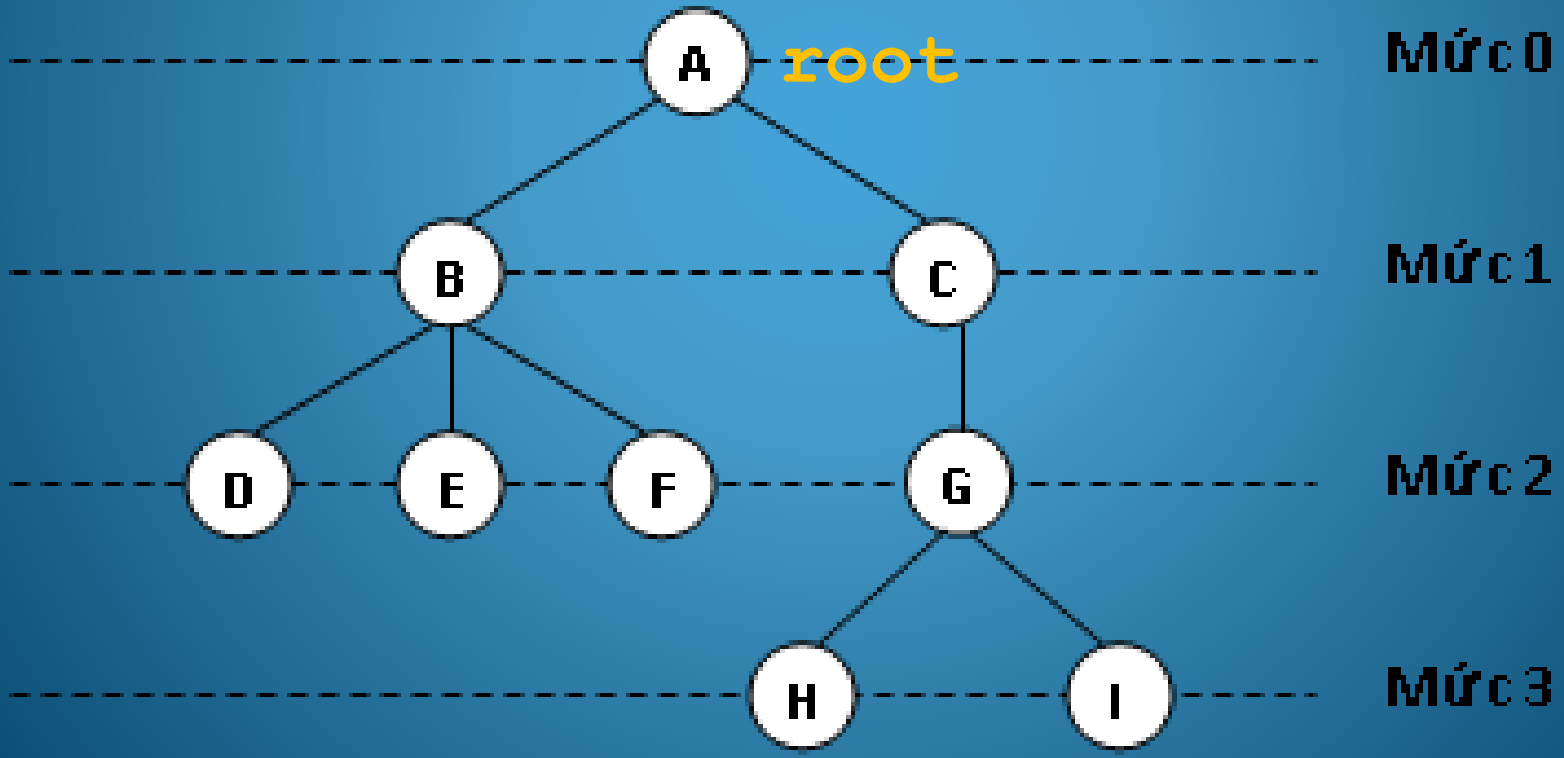
Mục tiêu:

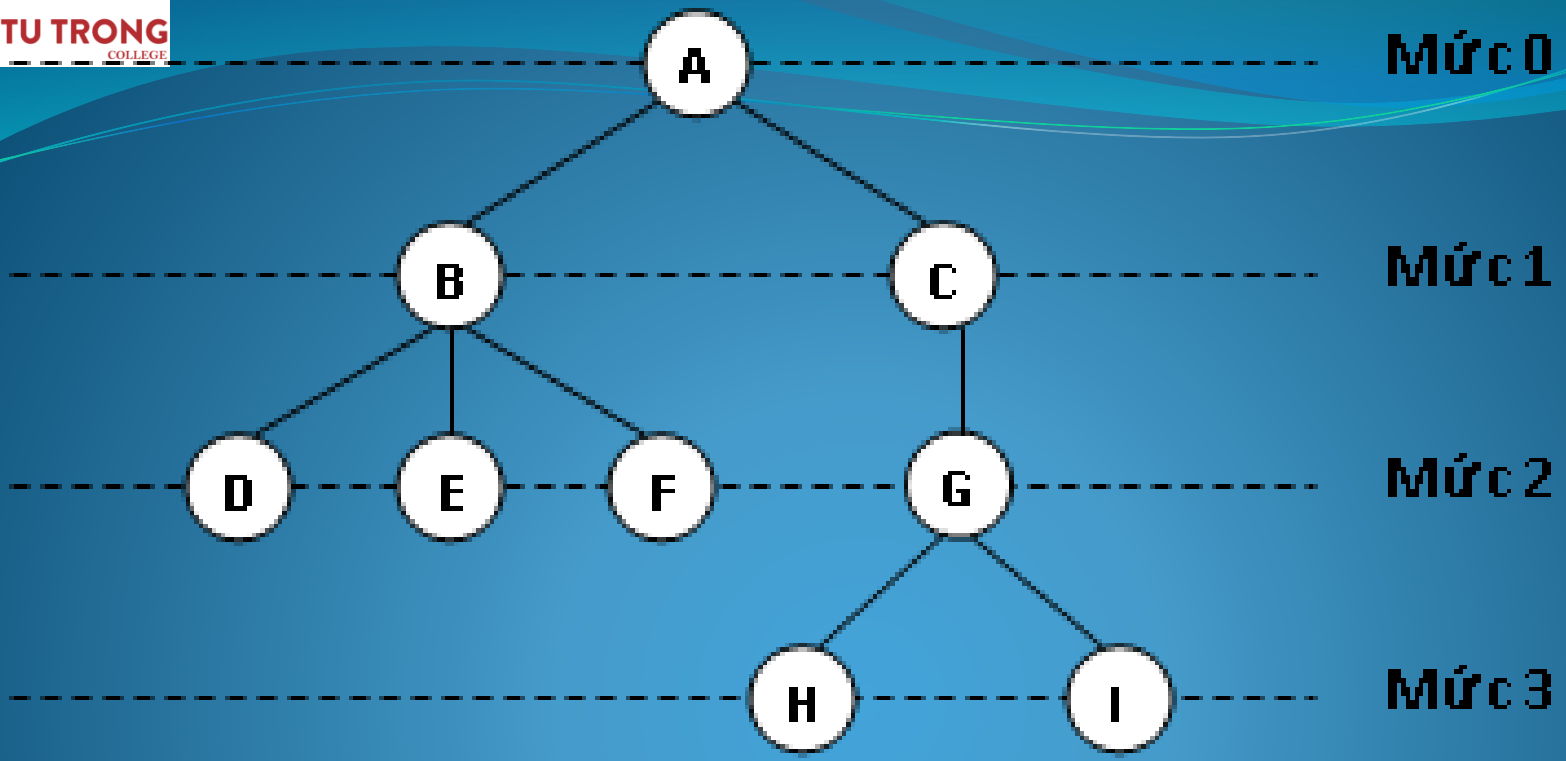
Sau khi học xong người học có khả năng:

- Phát biểu được các khái niệm về cây và cây nhị phân;
- Chuyển đổi được cây bất kỳ về cây nhị phân;

1. Định nghĩa

Cây (tree) là một tập hợp các phần tử gọi là node. Trong đó có một node đặc biệt được gọi là gốc, các node còn lại được chia thành những tập rời nhau $T_1, T_2, ..., T_n$ Trong đó T_i cũng là một cây

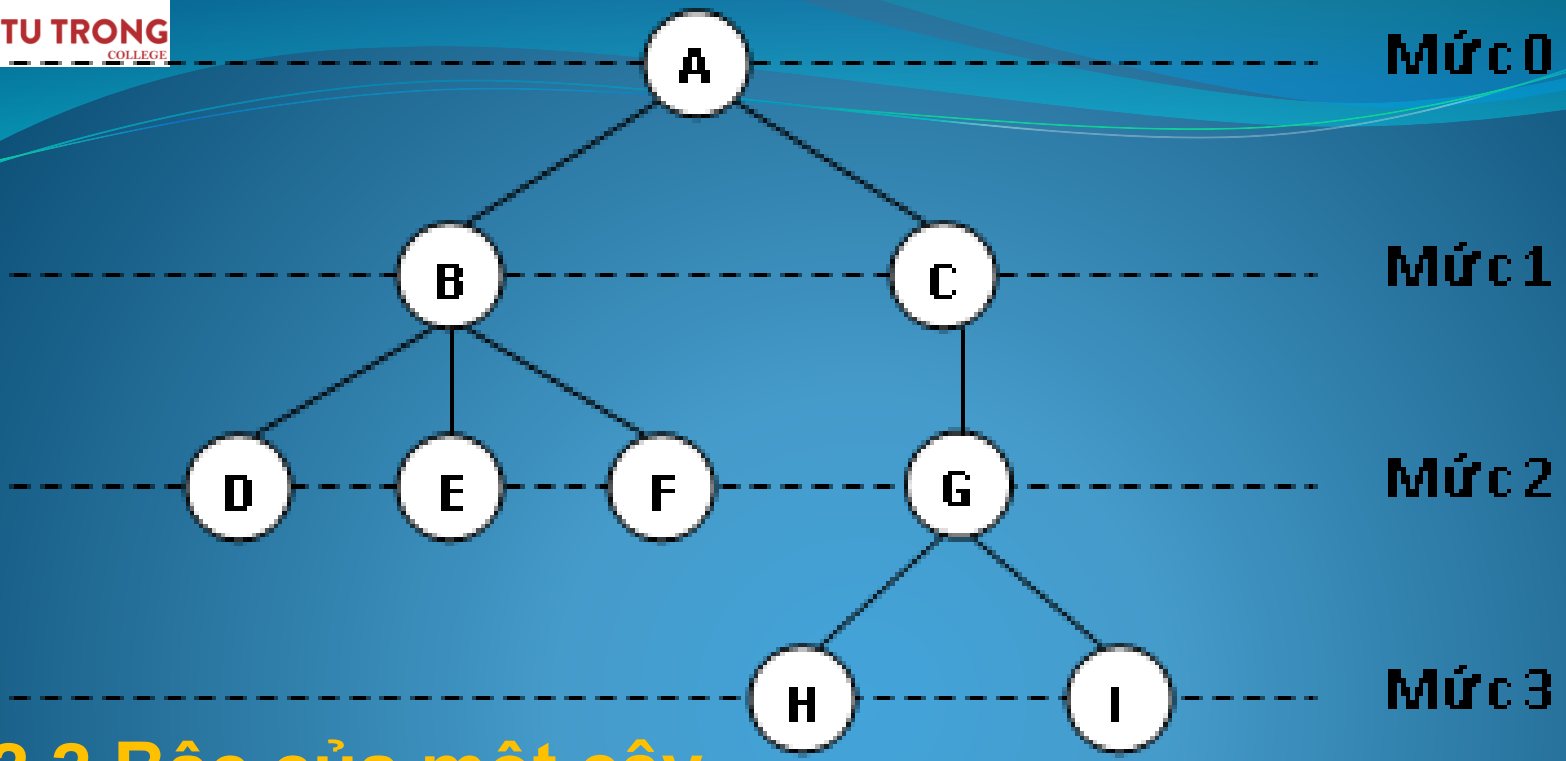




2. Một số khái niệm cơ bản

2.1. Bậc (degree) của một node

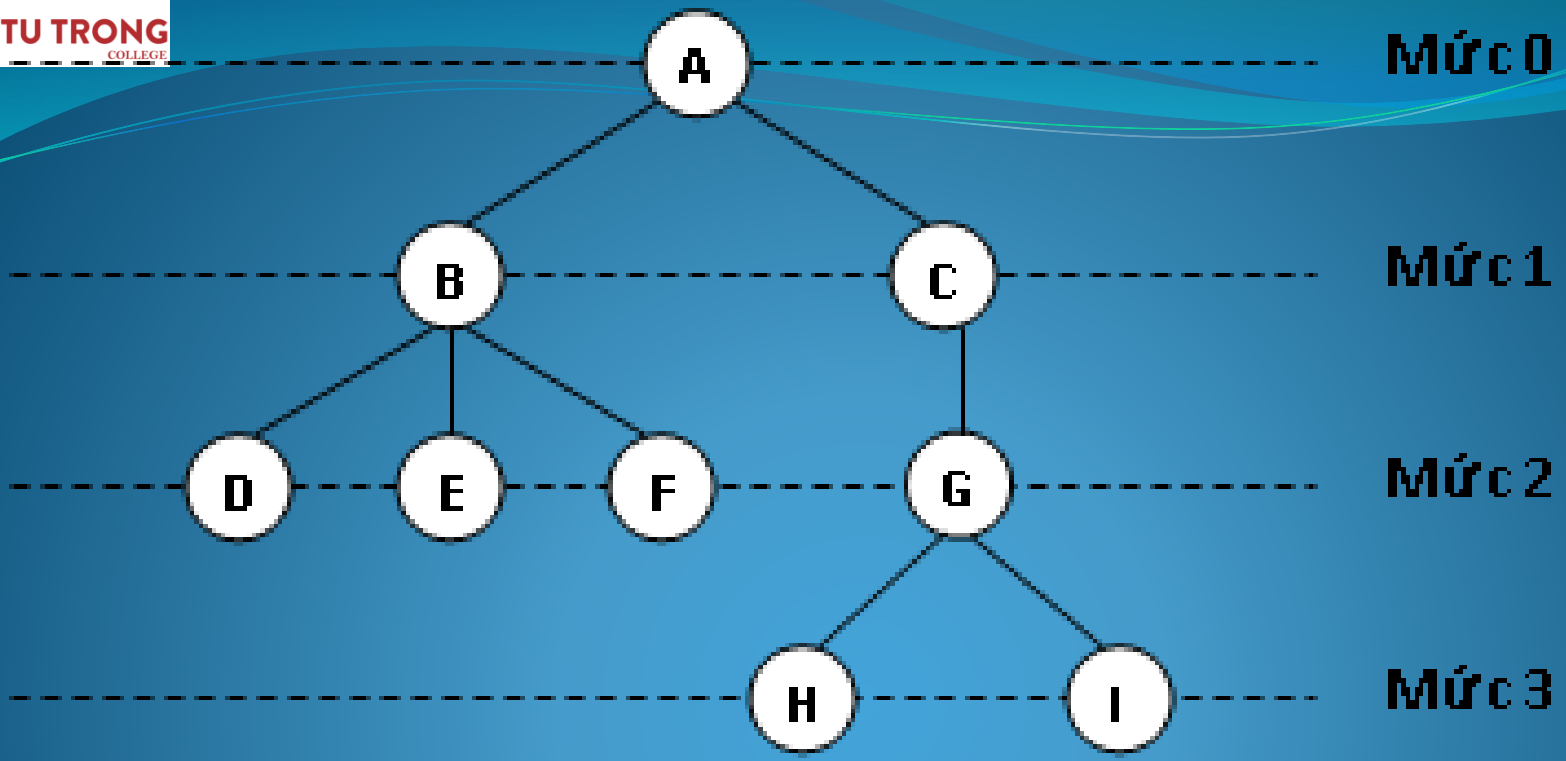
Là số cây con của node đó



2.2.Bậc của một cây

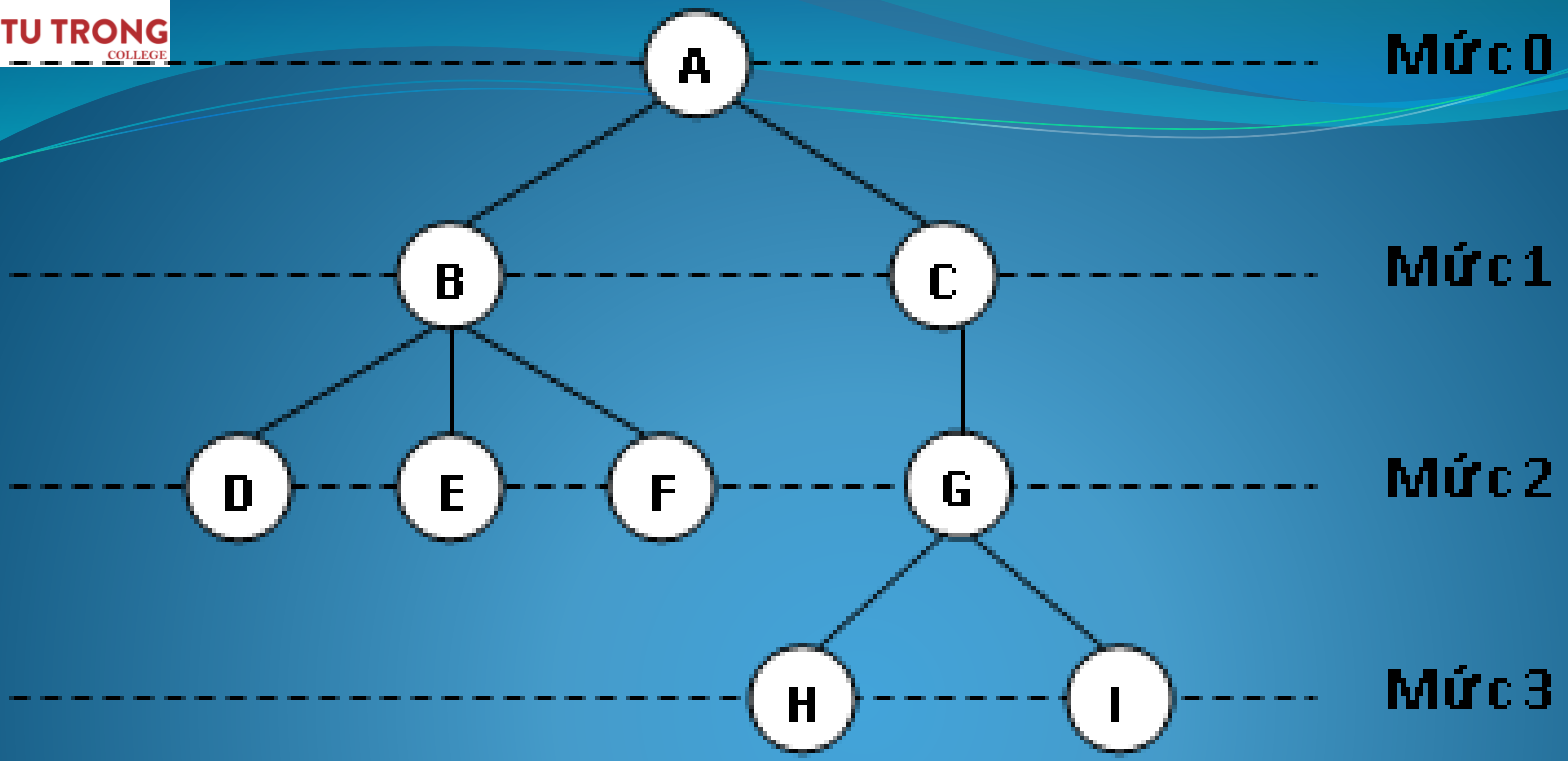
Là bậc lớn nhất của các node trong cây.

Cây có bậc n thì gọi là cây n-phân.



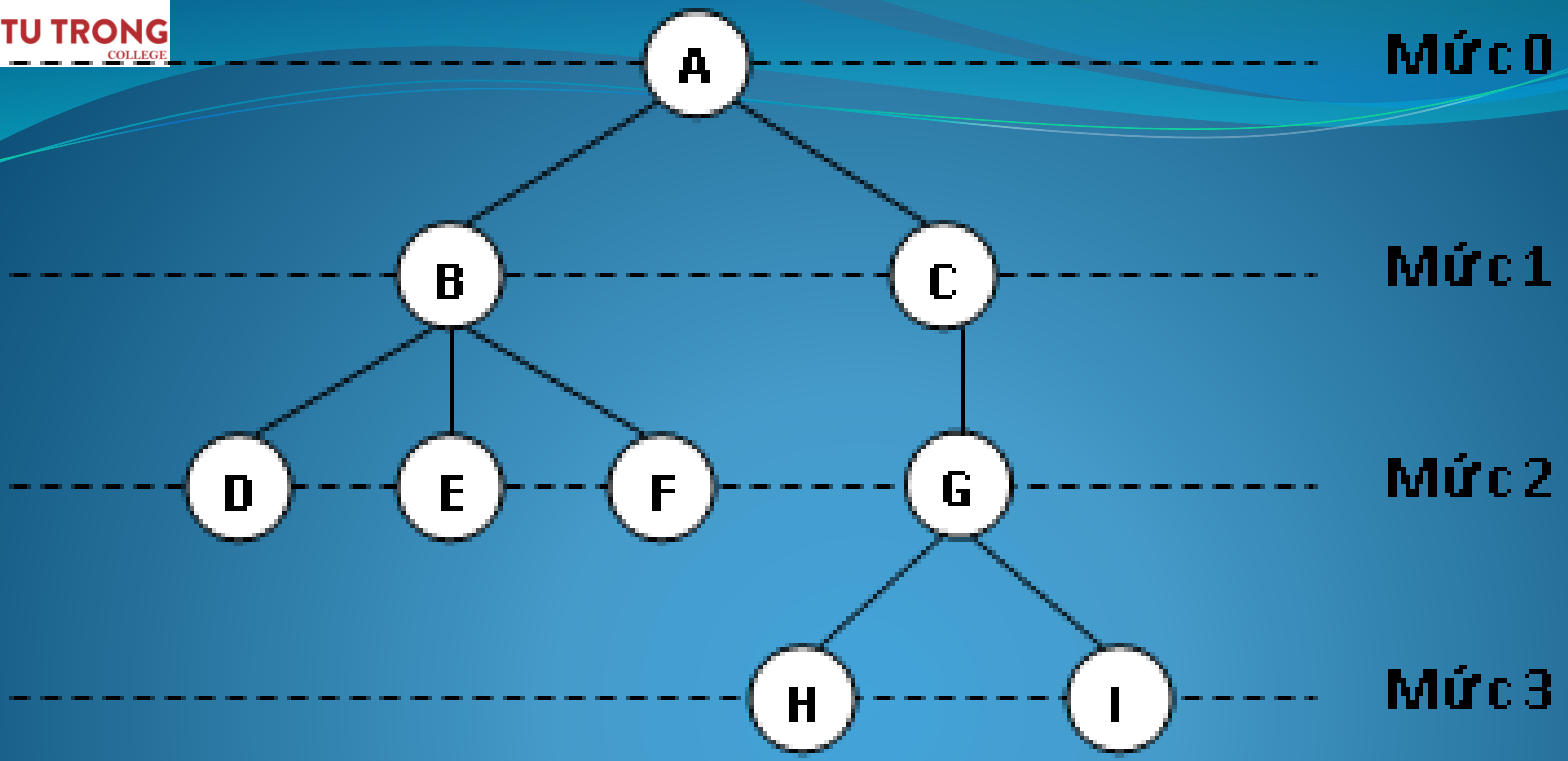
2.3. Node lá (leaf)

Là node có bậc bằng 0.



2.4. Node nhánh hay node trung gian (interior)

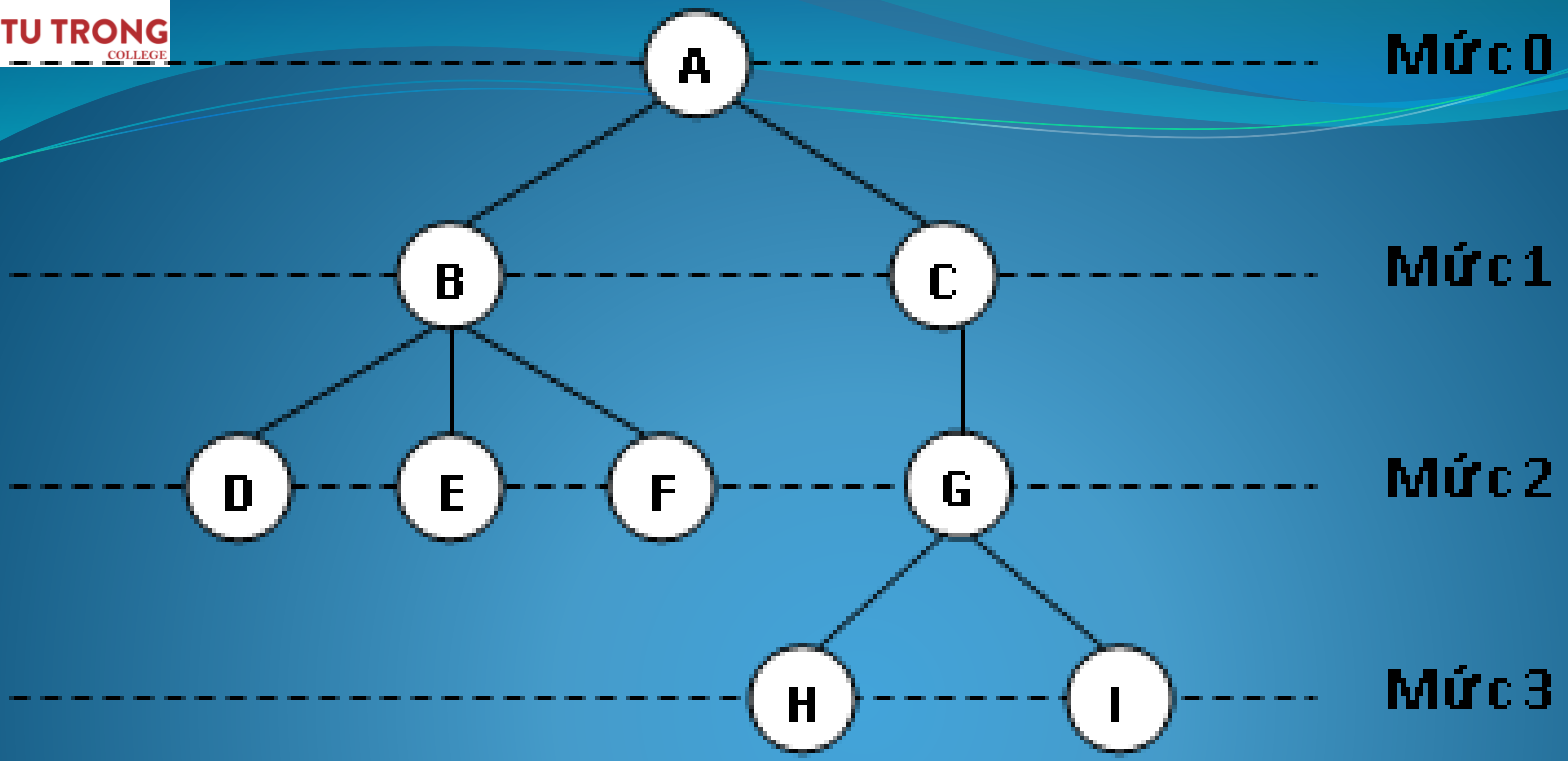
Là node có bậc khác 0 và không phải là node gốc .



2.5. Mức (level) của một node

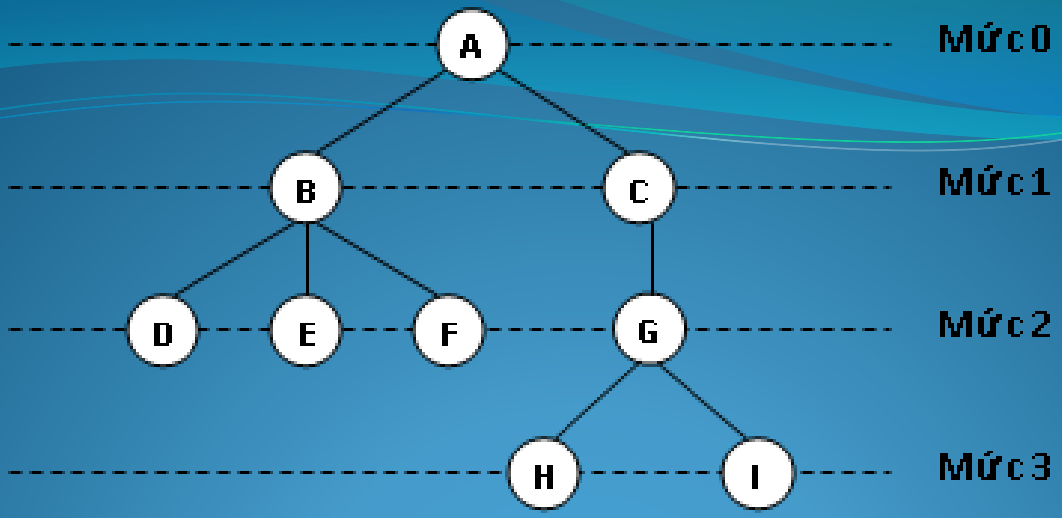
Mức (level) của node gốc là 0

Mức của node khác gốc bằng mức của node gốc của cây con chứa nó cộng 1.



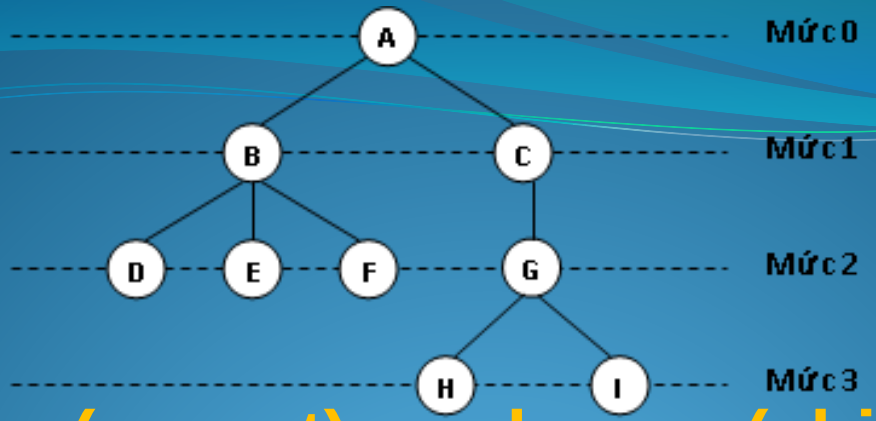
2.6. Chiều cao của một cây (height)

Là số mức lớn nhất của node có trên cây đó cộng thêm 1



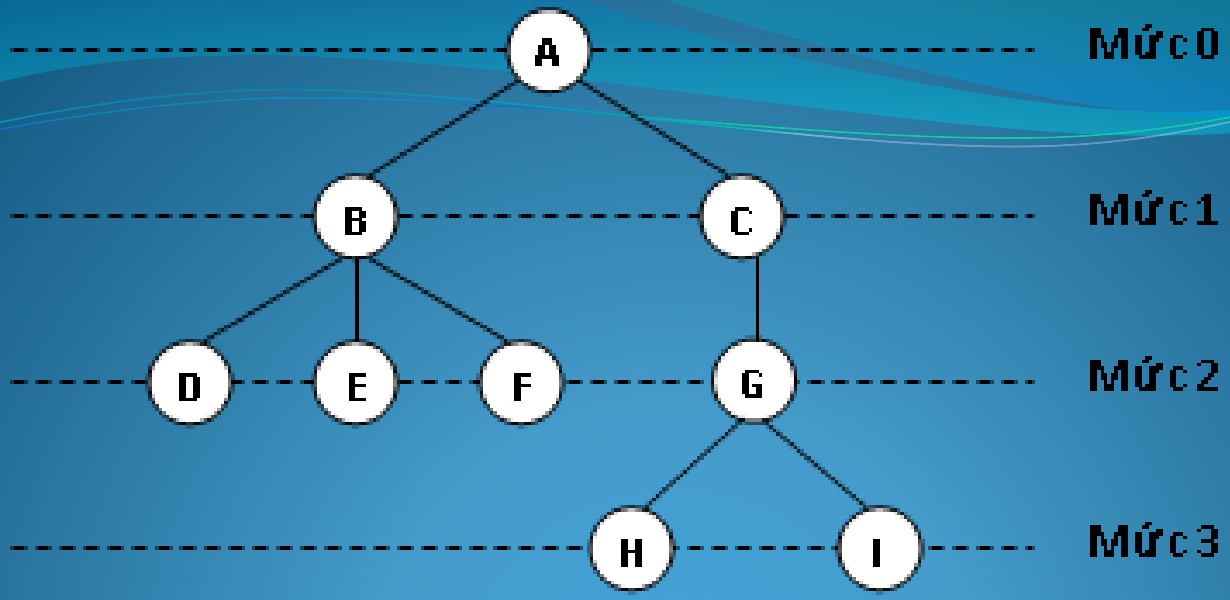
2.7. Node trước , node sau

Node A gọi là node trước của node B nếu cây con có gốc là A chứa node B. Và lúc đó, node B được gọi là node sau của node A.



2.8. Node cha (parent), node con (child)

Nếu A là node trước của node B, và mức của node B bằng mức của node A cộng với 1, thì node A được gọi là node cha của node B, và node B được gọi là node con của node A.



2.9. Chiều dài đường đi từ gốc đến node x (path length)

Bằng số nhánh từ node gốc đến node x, nghĩa là bằng mức của node x.

2.10. Chiều dài đường đi trong (internal path length) của cây

Là tổng chiều dài đường đi của tất cả các node trong cây : $P^I = \sum_{i=1}^{n} d_i$, trong đó:

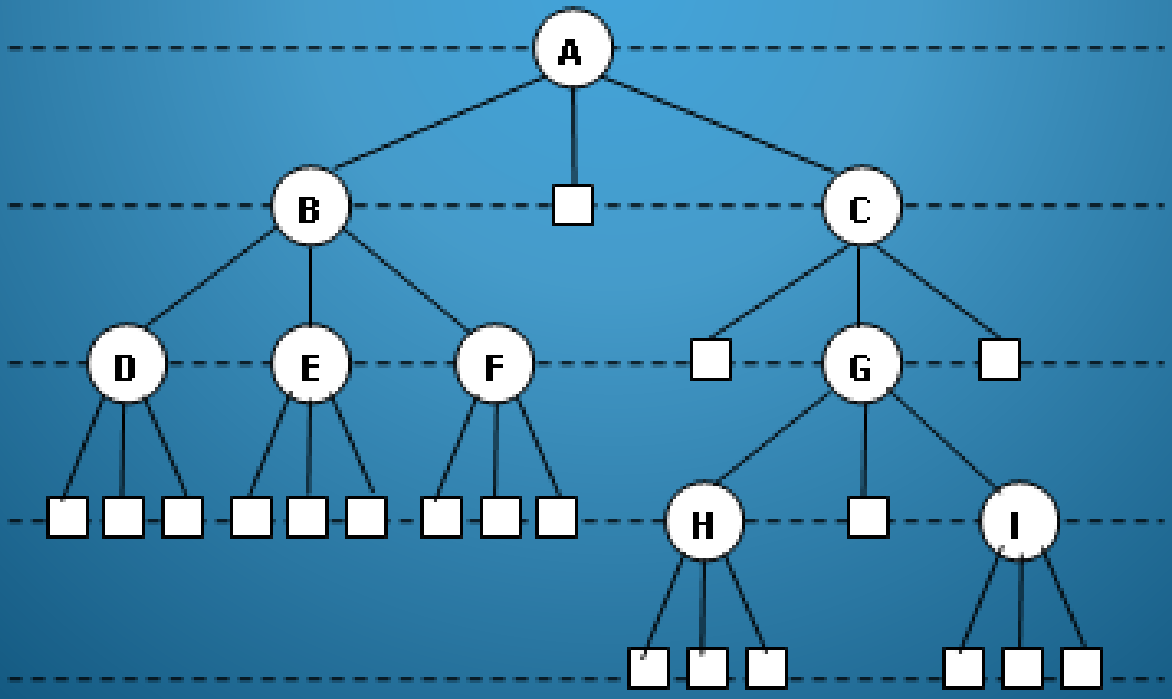
d_i: chiều dài đường đì của node thứ i

n: số node trong cây

Chiều dài đường đi trong trung bình của cây : $P_A^I = \frac{P^I}{n}$

2.11. Chiều dài đường đi ngòai (external path length) của cây

Là tổng chiều dài của tất cả các node đặc biệt trong cây (là những node giả được thêm vào cây sao cho bậc của tất cả các node bằng với bậc của cây)



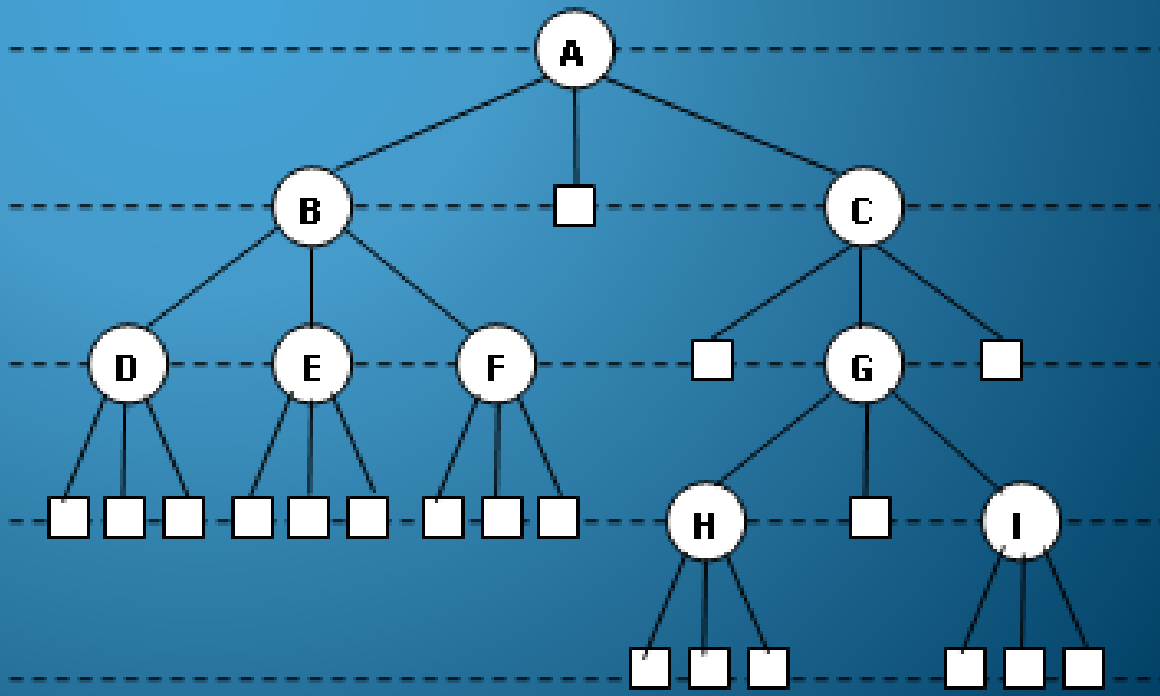
$P^E = \sum_{i=1}^{m} d'_i$, trong đó:

d'_i: chiều dài đường đi của node thứ i

m: số node giả trong cây

Chiều dài đường đi ngoài trung bình của cây

$$P_A^E = \frac{P^E}{m}$$



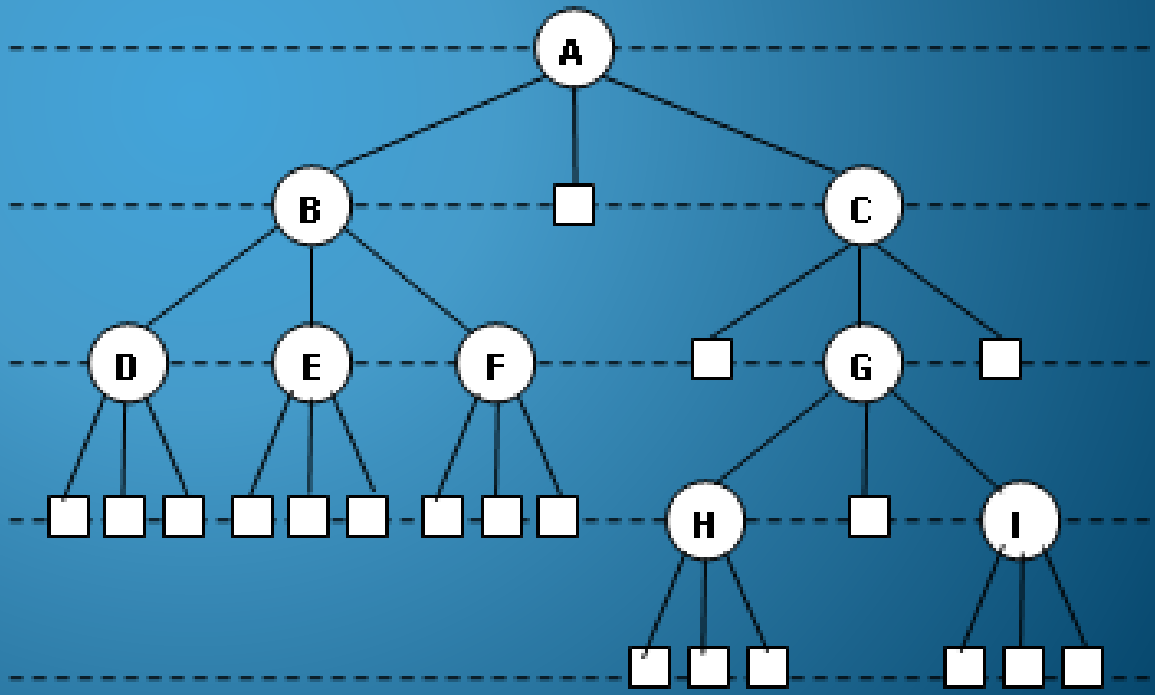
$P^E = \sum_{i=1}^{m} d'_i$, trong đó:

d'_i: chiều dài đường đi của node thứ i

m: số node giả trong cây

Chiều dài đường đi ngoài trung bình của cây

$P_A^E = \frac{P^E}{m}$



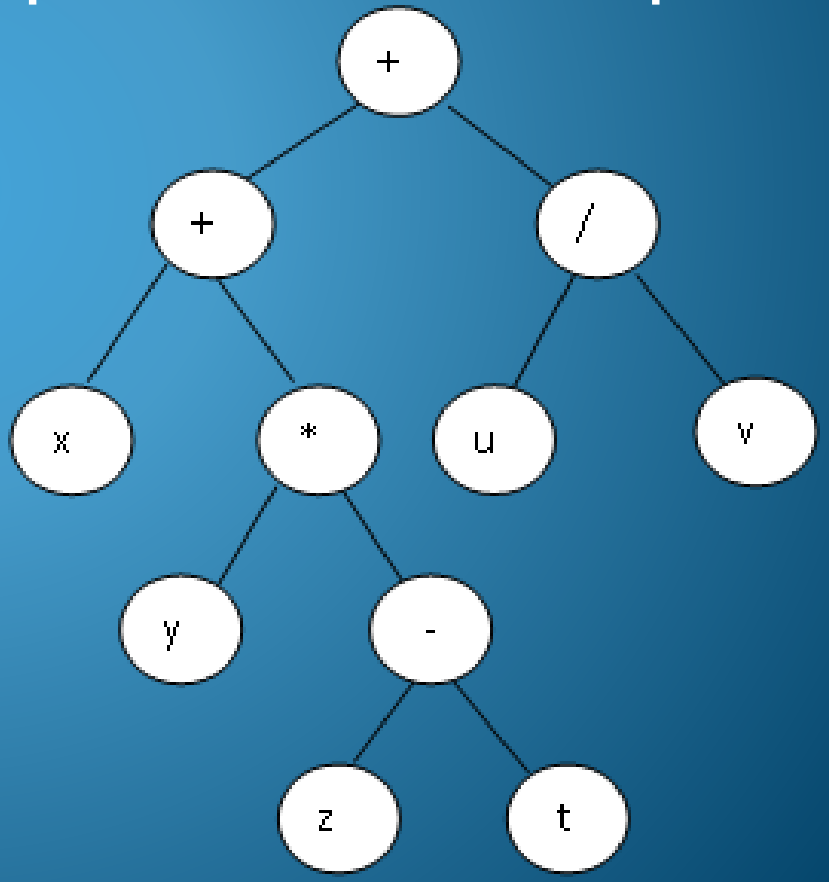
3. Cây nhị phân (Binary Tree)

3.1. Định nghĩa

Là cây mà mỗi node có tối đa 2 cây con

Ví dụ cây nhị phân biểu diễn một biểu thức tóan học:

x+y*(z-t)+u/v

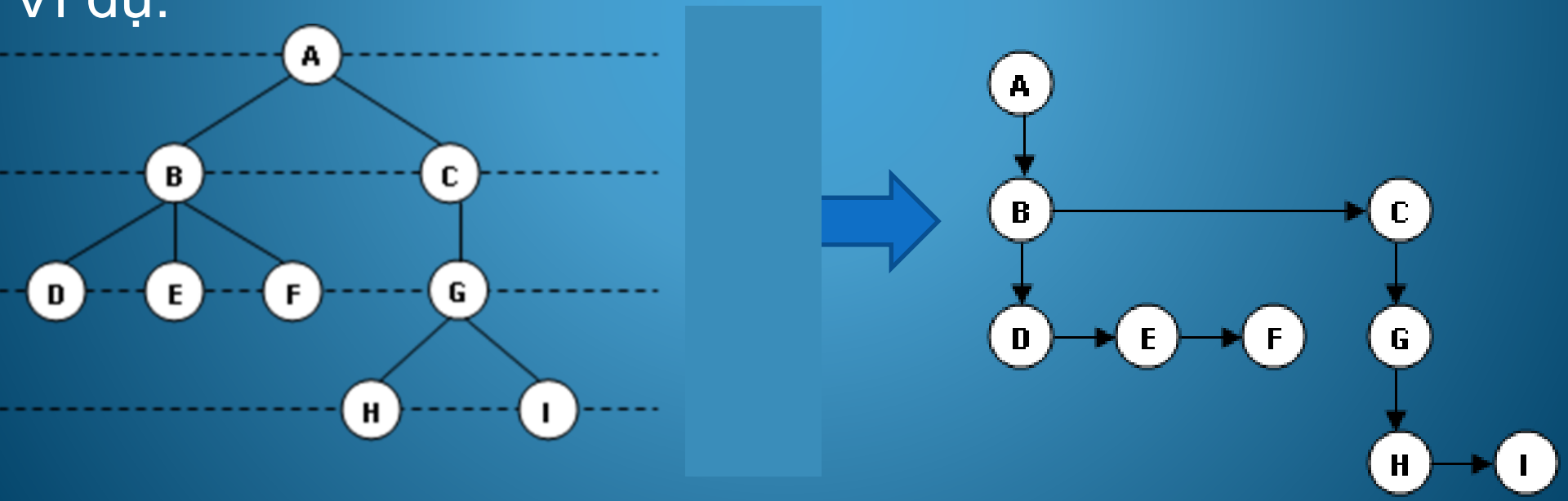


3.2. Cách chuyển từ cây bất kỳ về cây nhị phân

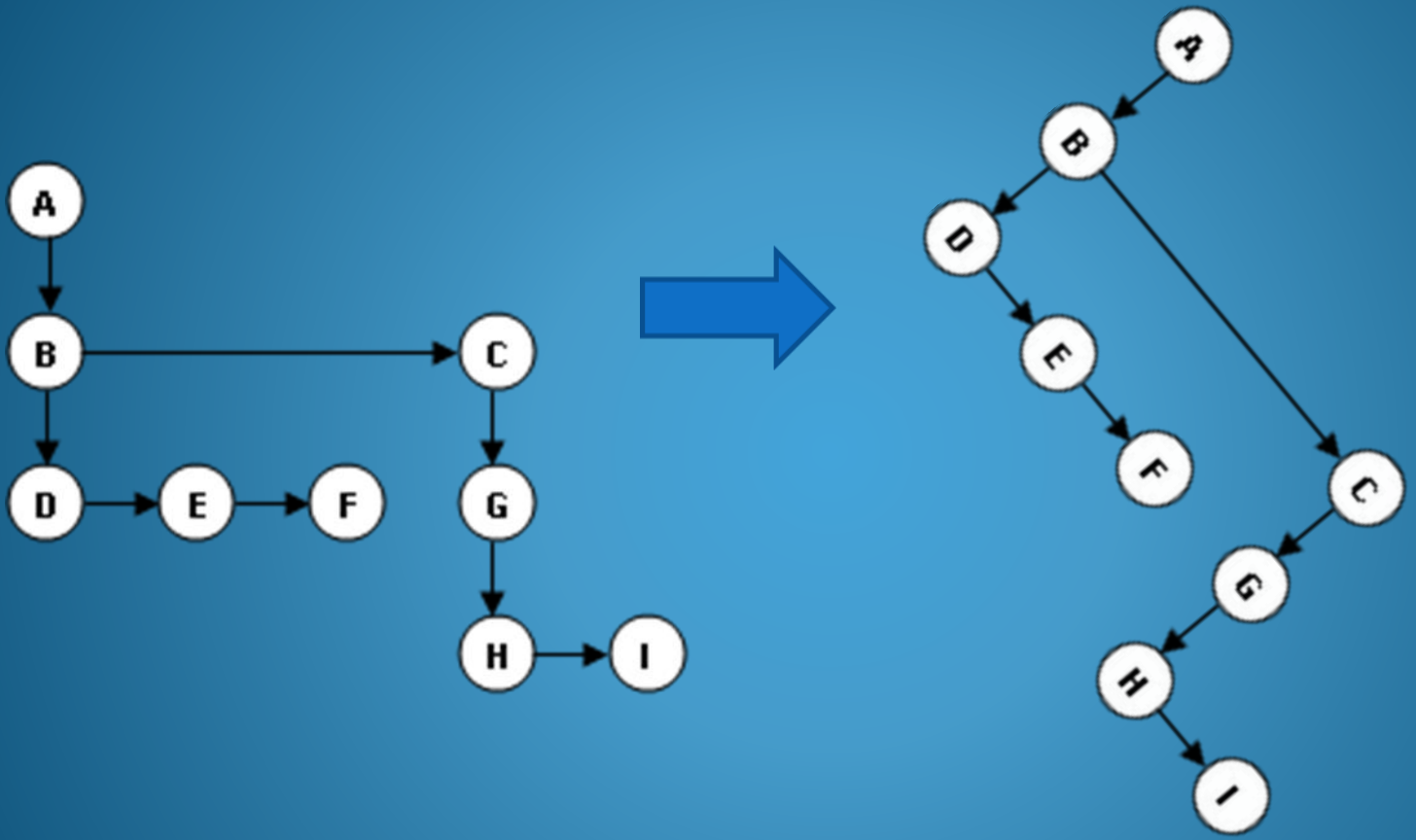
Bước 1: Vẽ lại cây theo quy ước

- Các node có cùng node cha sẽ cùng nằm trên cùng một đường ngang.
- Các node con ở đường phía dưới

Ví dụ:



Bước 2: Xoay hình qua bên phải 45 độ



Bài 2: CÂY NHỊ PHÂN TÌM KIẾM

Mục tiêu:

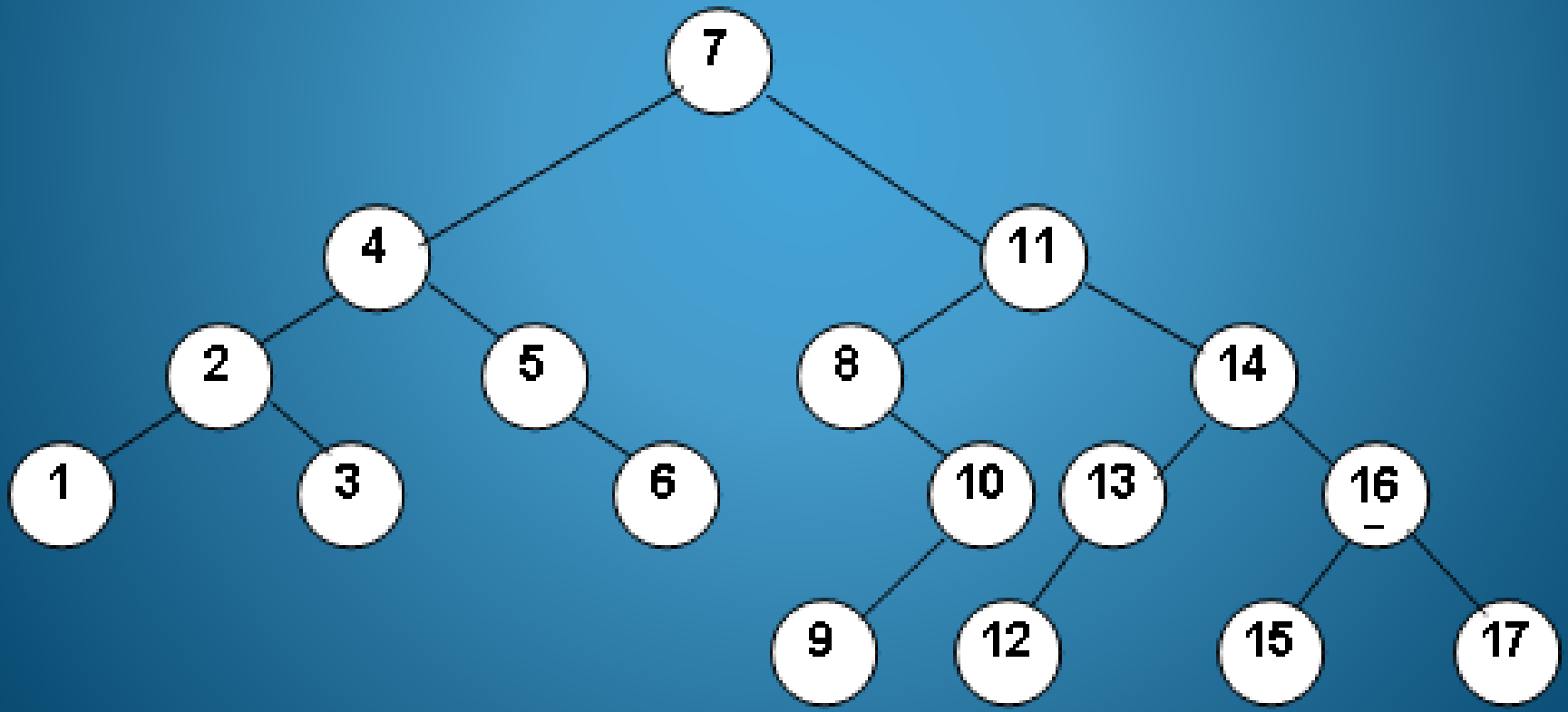
Sau khi học xong người học có khả năng:

- Phát biểu được định nghĩa cây nhị phân tìm kiếm;
- Hiểu và cài đặt được cây nhị phân tìm kiếm và các thao tác trên cây nhị phân tìm kiếm;

LY TU TRONG COLLEGE

1. Định nghĩa

Cây nhị phân tìm kiếm (BST) là cây nhị phân thỏa : Tại mỗi node, giá trị của node này lớn hơn giá trị của các node của cây con bên trái và nhỏ hơn giá trị của các node của cây con bên phải.

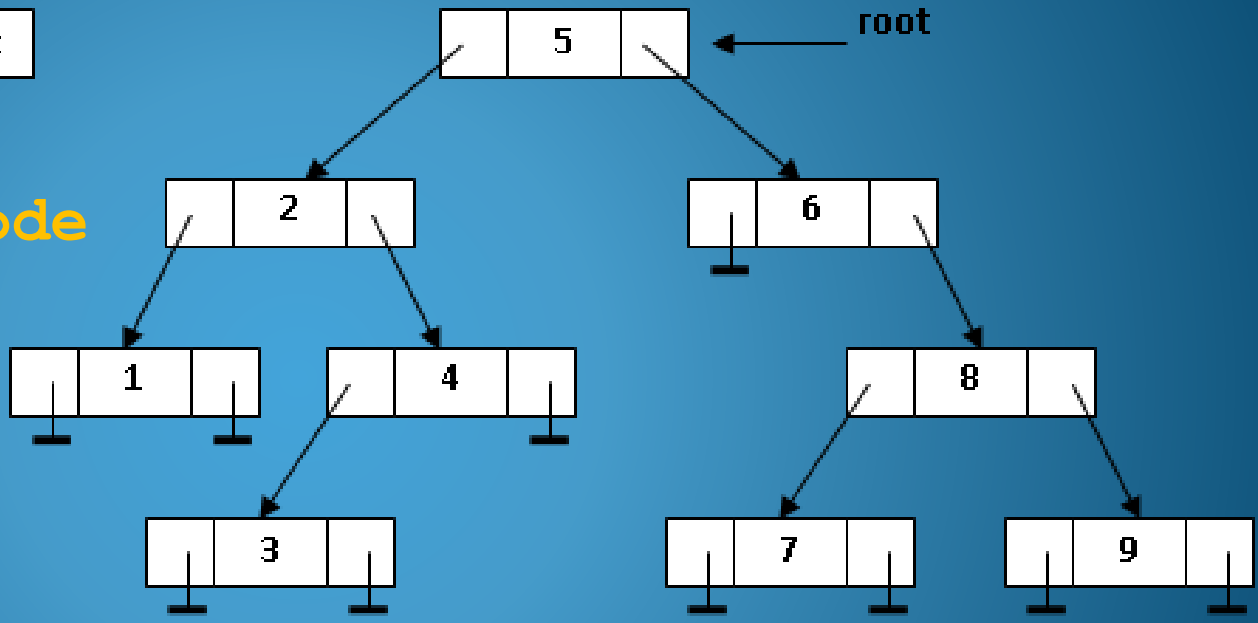


2. Một số thao tác trên cây nhị phân tìm kiếm

Khai báo cây

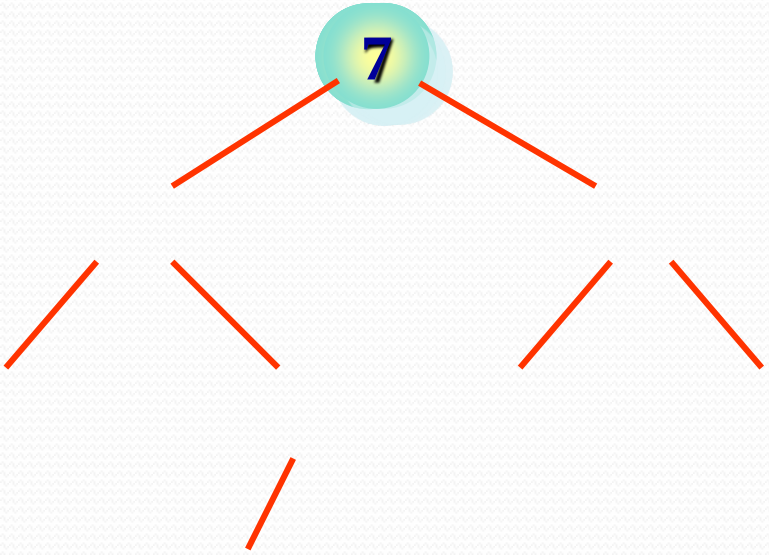
node

```
typedef struct node
{
   int data;
   node* left;
   node* right;
};
node* root;
```



2.2. Thêm một nút vào cây nhị phân tìm kiếm

7	36	3	1	6	4	15	40



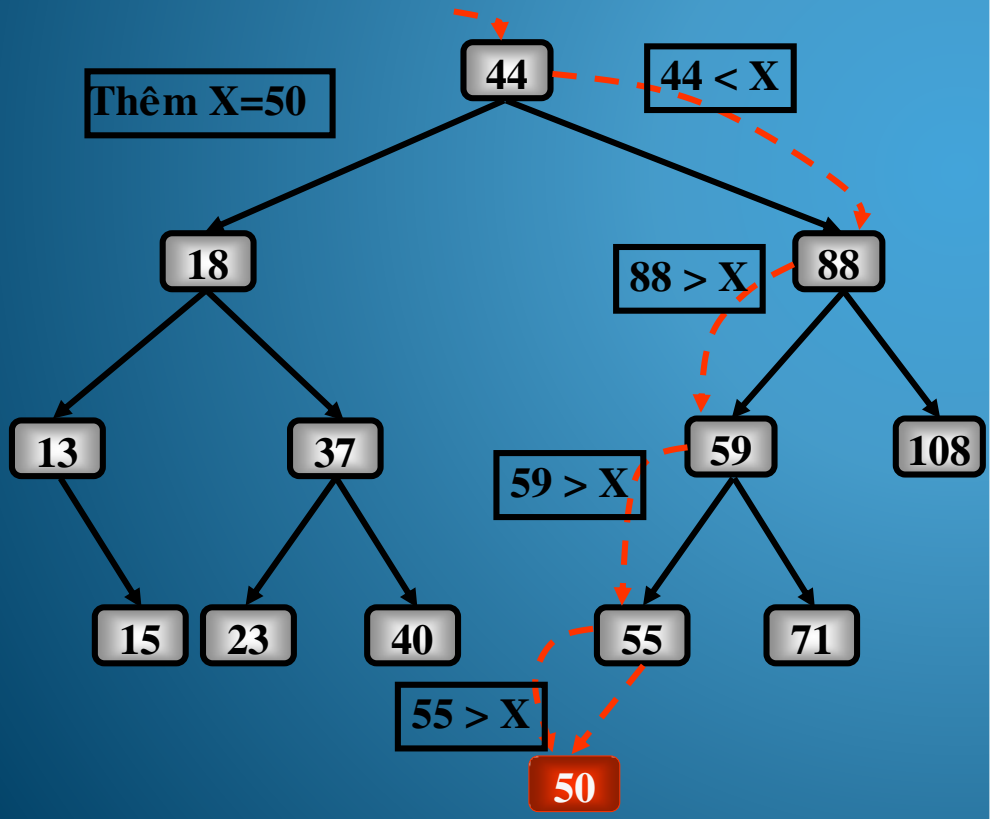
- Nếu node cần thêm nhỏ hơn node đang xét thì thêm về bên trái
- Ngược lại thì thêm về bên phải

Hàm `InsertTree(root,x)` cho phép chèn một node có khóa x vào cây có gốc root và trả về giá trị :

–1 khi không đủ bộ nhớ,

0 khi số x đã có trong cây

1 khi chèn thành công.

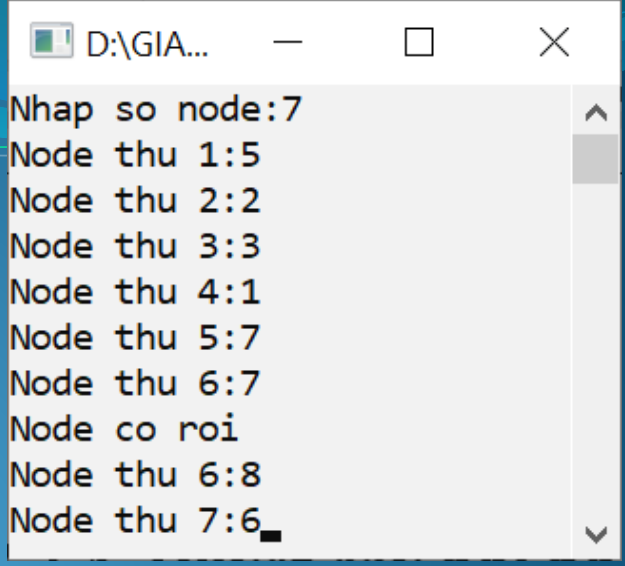



```
int InsertTree(tree  &root, int  x)
{
    if (root!=NULL)
    {
        if (root->data==x) return 0;
        if (root->data >x)
            return InsertTree(root->left,x);
        return InsertTree(root->right,x);
     }
    else
    {
         root=new node;
         if (root==NULL) return -1;
         root->data=x;
         root->left=NULL;
         root->right=NULL;
         return 1;
    }
}
```

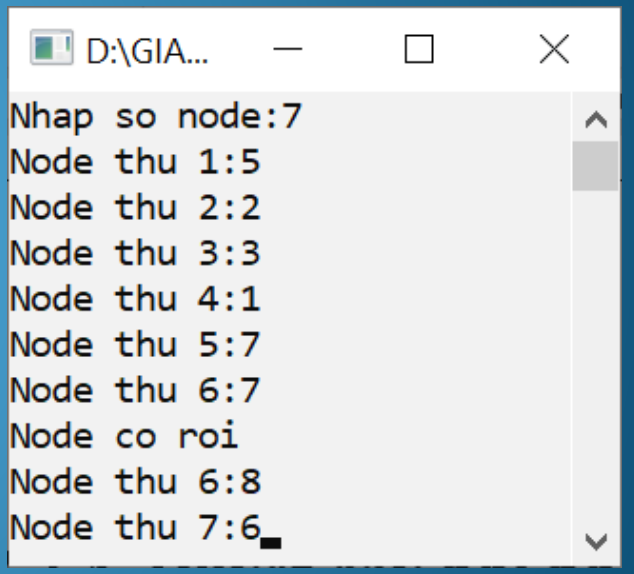
LY TU TRONG COLLEGE

2.2. Tạo cây nhị phân tìm kiếm

```
void createTree(node* &root)
{
  int x,n,t;
  printf("Nhap n = ");
  scanf("%d",&n);
  for (int i=1;i<=n;i++)
  {
     scanf("%d",&x);
     t=insertTree(root,x);
```

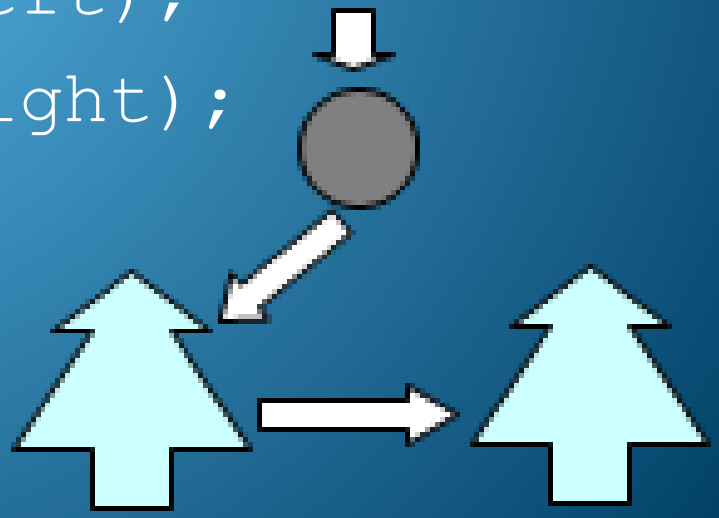


```
if (t==-1)
{
    printf("Bo nho day!\n");
    getch(); exit(1);
}
else
    if (t==0)
    {
        printf("Node co roi!\n");i--;
    }
}
```



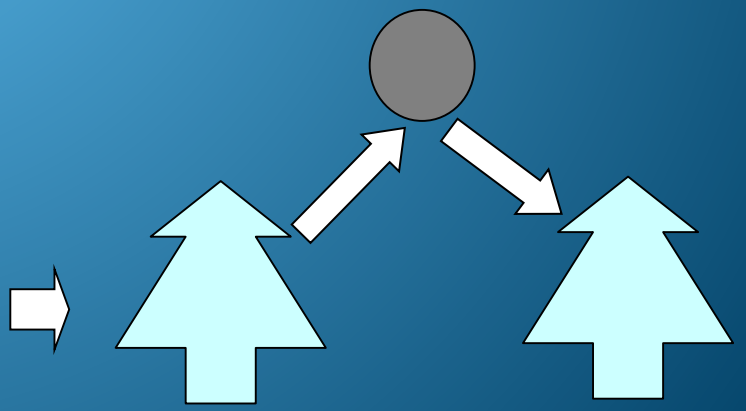
Duyệt theo thứ tự trước (PreOder) : Node-Left-Right

```
void traverseNLR(node* root)
{
    if(root!=NULL)
    {
        printf("%d\t",root->data);
        traverseNLR(root->left);
        traverseNLR(root->right);
    }
}
```



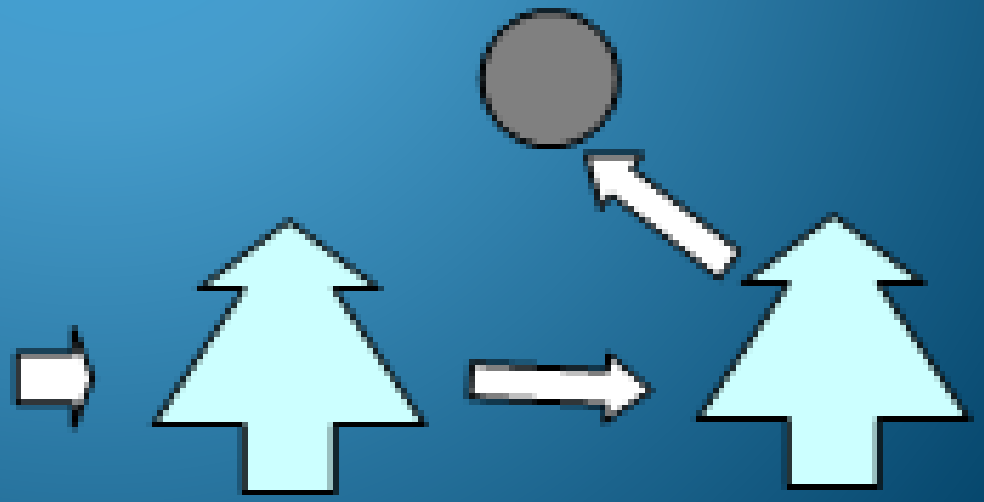
Duyệt theo thứ tự giữa (InOder) : Left-Node-Right

```
void traverseLNR(node* root)
{
    if(root!=NULL)
    {
        traverseLNR(root->left);
        printf("%d\t",root->data);
        traverseLNR(root->right);
    }
}
```

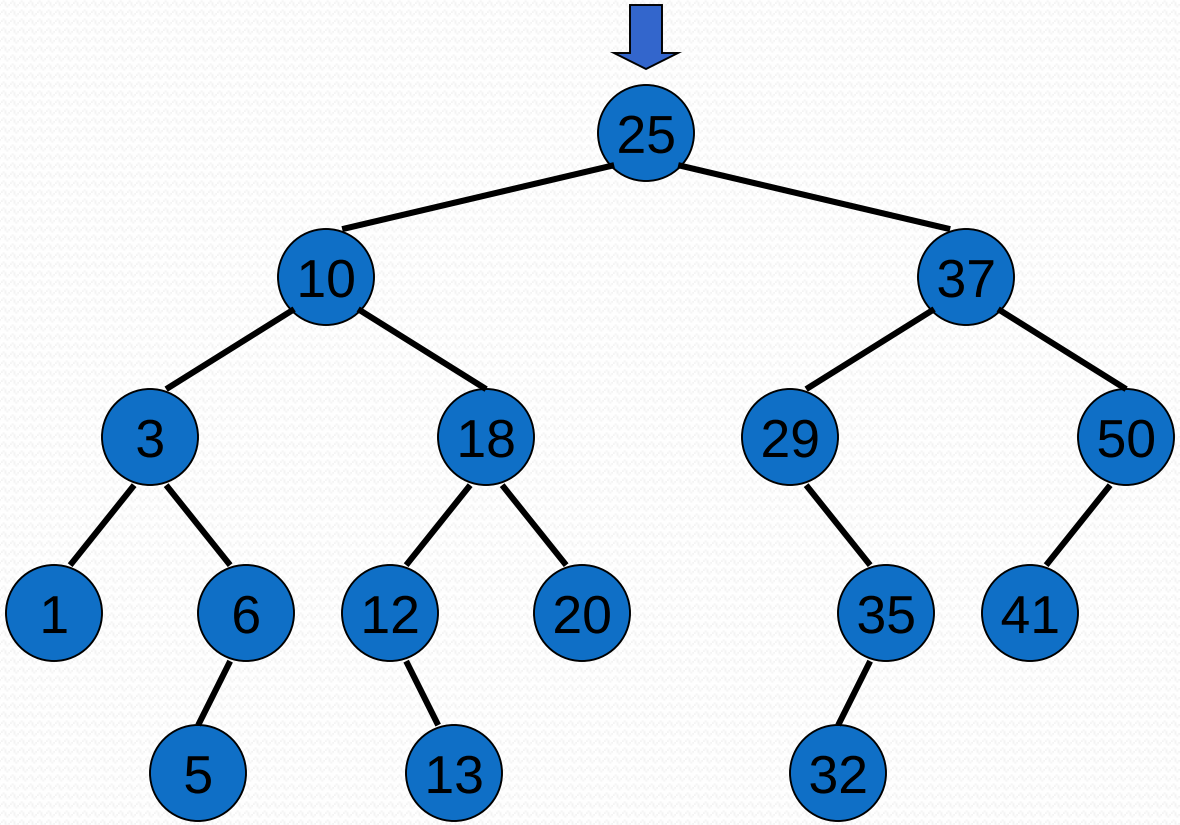


Duyệt theo thứ tự sau (PostOder) : Lef-Rightt-Node

```
void traverseLRN(node* root)
{
    if(root!=NULL)
    {
        traverseLRN(root->left);
        traverseLRN(root->right);
        printf("%d\t",root->data);
    }
}
```



2.3.. Tìm phần tử có khóa x trong cây nhị phân tìm kiếm

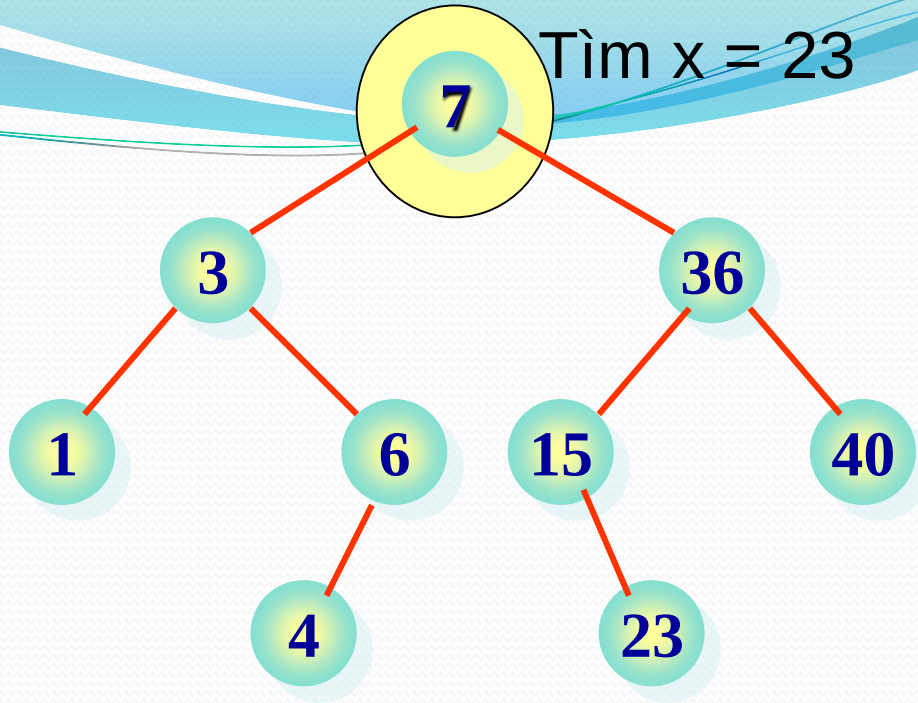


Tìm kiếm 13

Tìm thấy

Số lần so sánh: 5

Hàm **searchTree(root,x)** trả về địa chỉ của node x (nếu không tìm thấy trả về NULL)



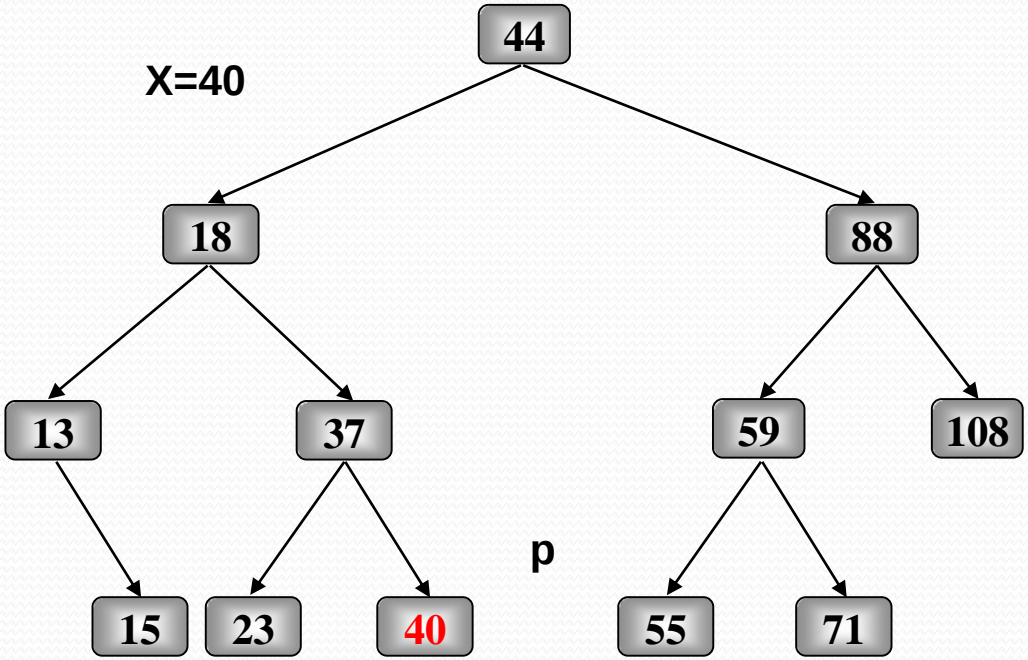

```
node* searchTree(node* root,int x)
{
   while (root!=NULL)
   {
      if (root->data==x) return root;
      if (root->info>x) root=root->left;
      else root=root->right;
   }
   return NULL;
}
```

2.5. Xóa phần tử có khóa x trong cây nhị phân tìm kiếm

2.5.1. Ý tưởng của giải thuật: Gọi p là node cần xóa

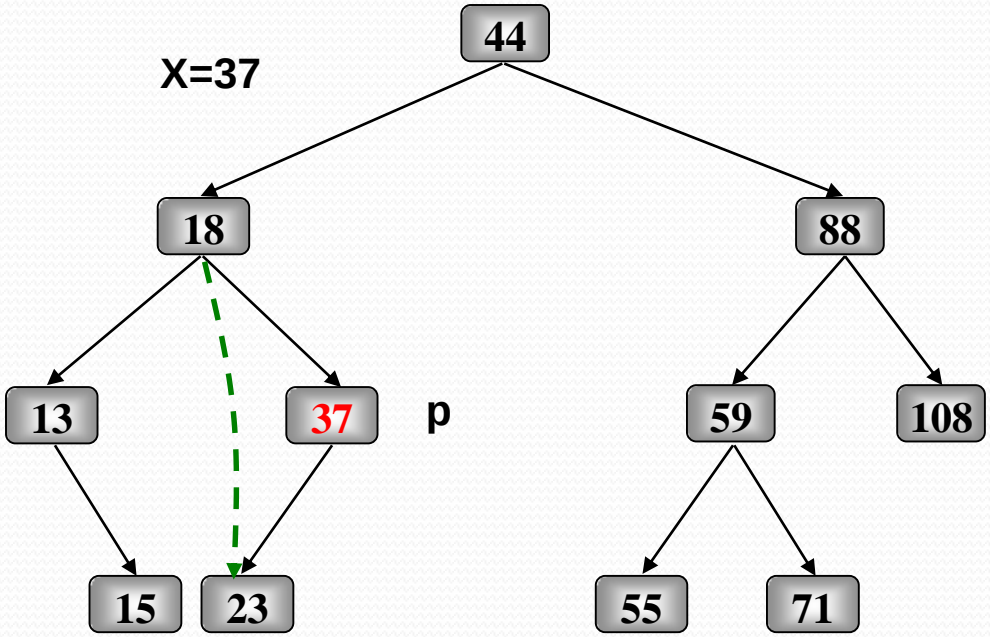
Trường hợp 1: p là node lá

- Cho node cha của p thay vì trỏ vào p thì trỏ vào NULL
- Hùy vùng nhớ giành cho node p



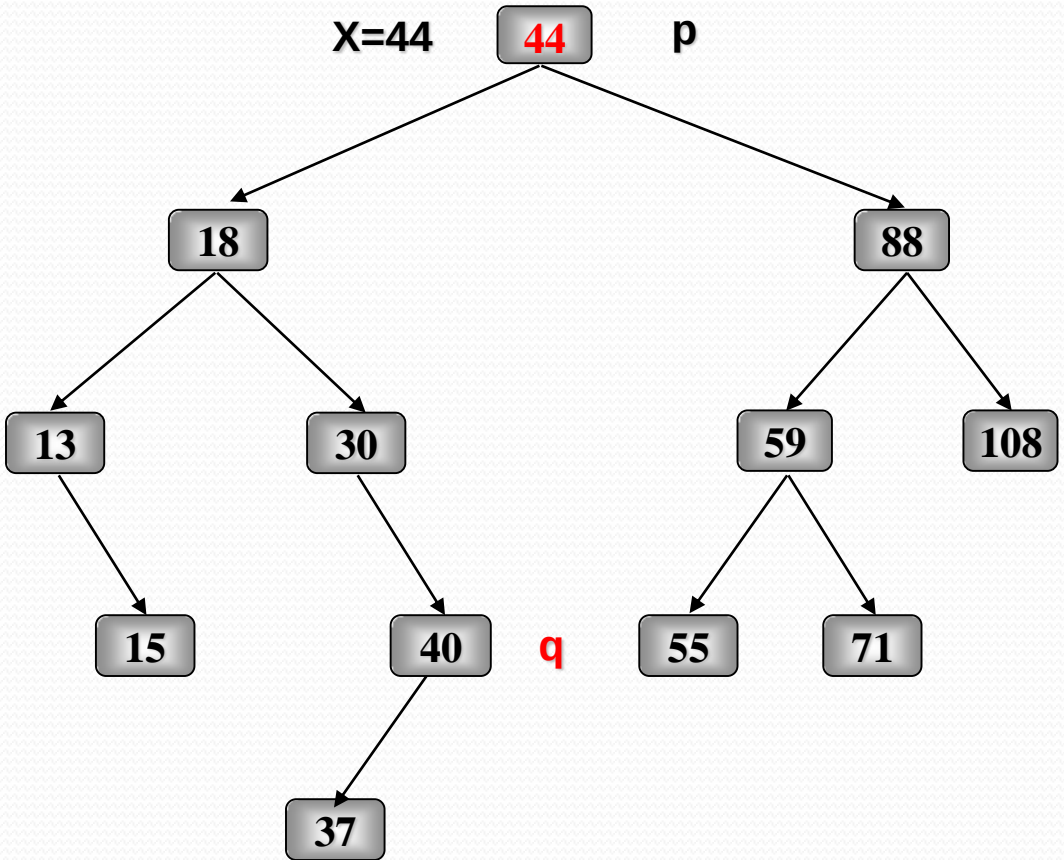
Trường hợp 2: p có 1 con

- Cho node cha của p thay vì trỏ vào p thì trỏ vào con của p
- Hùy vùng nhớ giành cho node p



Trường hợp 3: p có 2 con

- Tìm node q là node thế mạng cho q (là node cực phải của cây con trái)
- Chép giá trị của q vào p
- Xóa node q



2.5.2. Cài đặt

Hàm `removeNode(node *&root, int x)`:

– Trả về 1 nếu hủy thành công và trả về 0 nếu không tìm thấy node có khóa x.

Hàm `searchStandFor(node* &p,node* &q)`:

- Tìm node cực phải q trong cây con gốc q
- Chép giá trị của q vào p
- Cho node cha của q trỏ vào con trái của q
- Gán p=q (để ra ngoài hủy node p)

LY TU TRONG COLLEGE

```
int removeNode(node* &root,int x)
{
      if (root==NULL)    return 0;
      if (root->data< x) return RemoveNode(root->right,x);
      if (root->data> x)  return RemoveNode(root->left,x);
      node *p=root;
      if (root->right==NULL) root=root->left;          //TH12
      else
            if (root->left==NULL)  root=root->right;//TH12
            else
                  searchStandFor(p,root->left);        //TH3
      delete p;
      return 1;
}
```

```
void searchStandFor(node* &p,node* &q)
{
   if (q->right!=NULL)
      SearchStandFor(p,q->right);        //Tìm node cực phải
                                          của cây con trái của p
      else                          //Tìm thấy  q là node cực phải
      {
        p->data=q->data;             //Chép giá trị của q vào p
        p=q;                                        //Gán p=q
        q=q->left;                        //Cho node cha của q
                                        trỏ vào node con trái của q
   }
}
```

LY TU TRONG COLLEGE

2.6. Xóa cây

```
void removeTree(node* &root)
{
    if (root!=NULL)
    {
        removeTree(root->left);
        removeTree(root->right);
        delete root;
    }
}
```

BÀI TẬP TẬP MẪU

1. Hàm in ra màn hình tất cả các node của cây có giá trị chẵn.

```
void inNodeChan(node* root)
{
      if(root!=NULL)
      {
            if (root->data%2==0)
                  printf("%d\t", root->data);
            traverseNLR(root->left);
            traverseNLR(root->right);
      }
}
```

BÀI TẬP TẬP MẪU

2. Hàm đếm số node của cây có giá trị chẵn

```
int demNodeChan(node* root)
{
   if(root==NULL) return 0;
   if (root->data%2==0)
      return 1+ demNodeChan(root->left)+
                demNodeChan(root->right);
   return demNodeChan(root->left)+
                demNodeChan(root->right);
}
```

LY TU TRONG COLLEGE

BÀI TẬP

7.4. Hãy viết chương trình cho phép người dùng nhập một cây nhị phân tìm kiếm và thực hiện các công việc sau:

a. In các node lá trong cây.

b. In các node lá của cây theo thứ tự giảm dần.

c. Đếm số node bậc 1 trong cây.

d. Tính tổng giá trị các node bậc 2.

e. Tính giá trị trung bình của các node có khoá nhỏ hơn x và lớn hơn y .

f. Tính chiều cao của cây.

g. In ra tất cả các nút ở mức thứ k của cây.

h. In các node của cây theo từng mức, mỗi mức in trên 1 dòng.

i. Tính chiều dài đường đi trong của cây.

j.Tính chiều dài đường đi ngoài của cây.