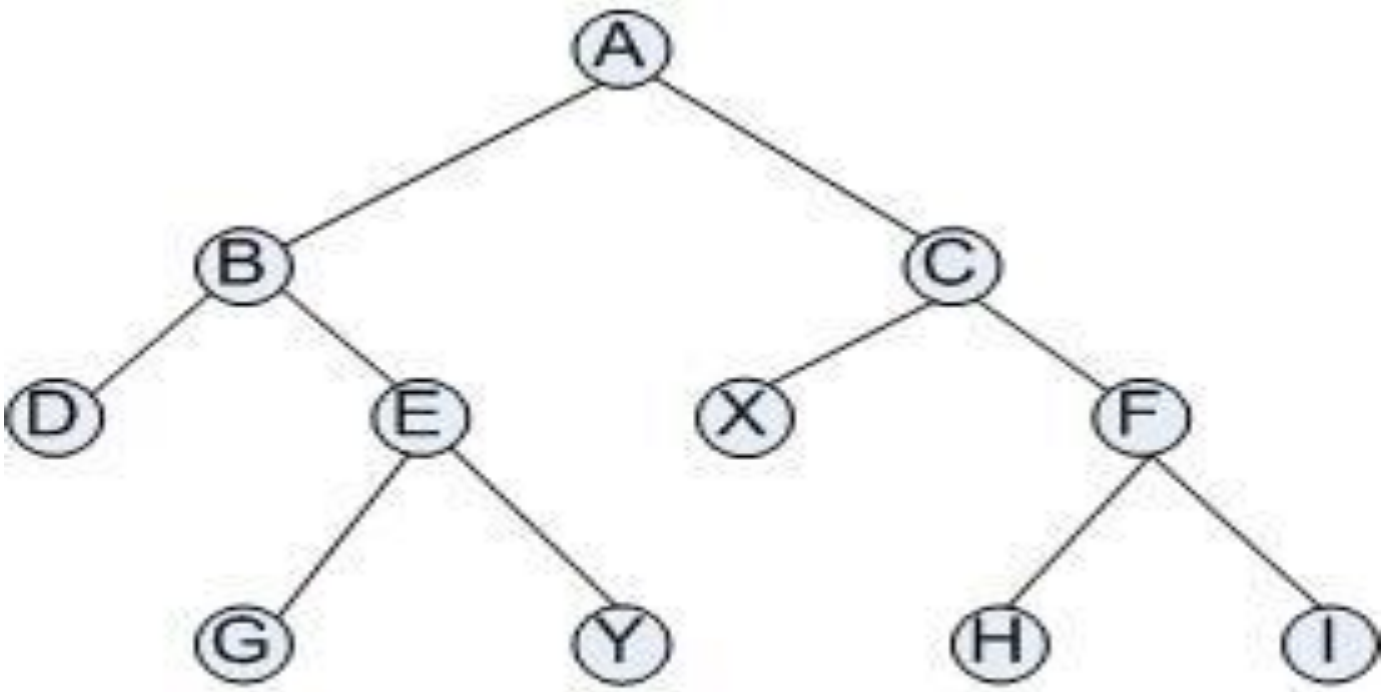
CHƯƠNG 4 . CÂY NHỊ PHÂN

1. Cây nhị phân tổng quát
2. Cây nhị phân tìm kiếm

ThS.Nguyễn Thúy Loan

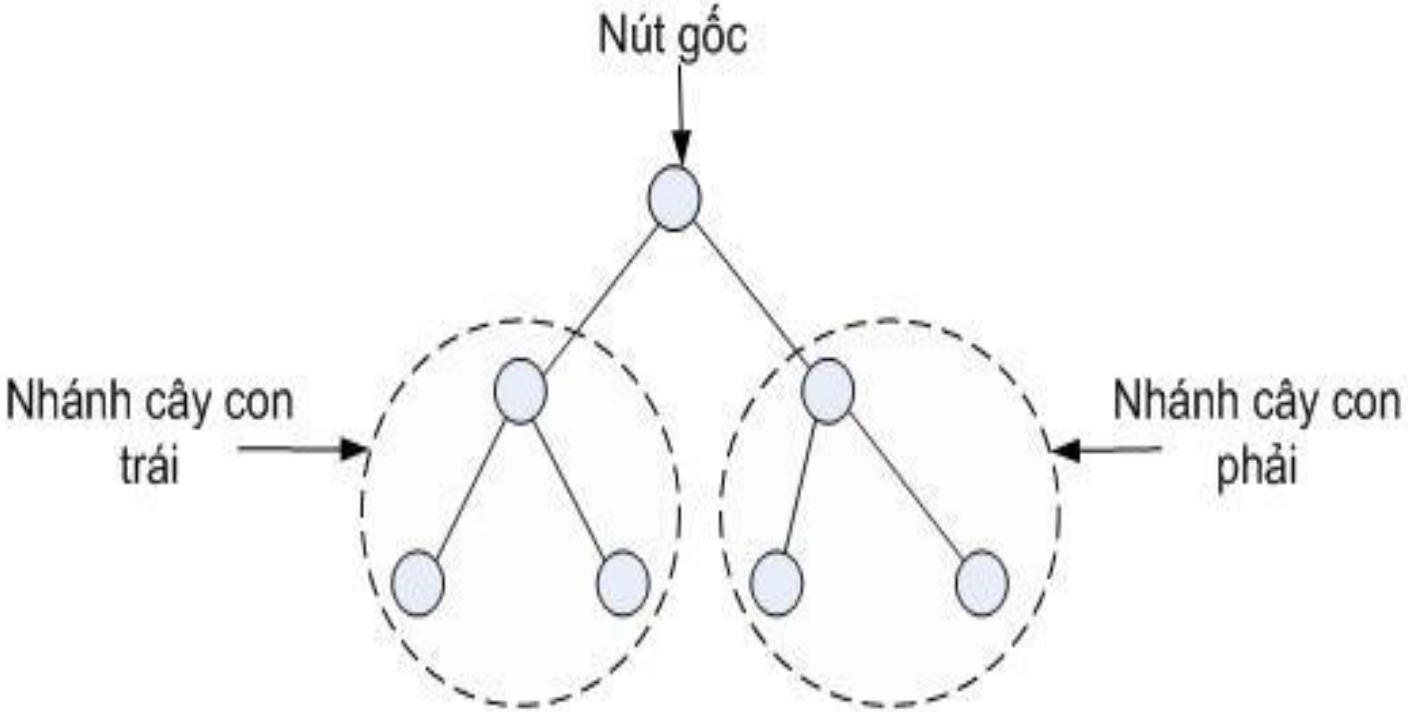
Cây nhị phân đúng *(strictly binary tree)*



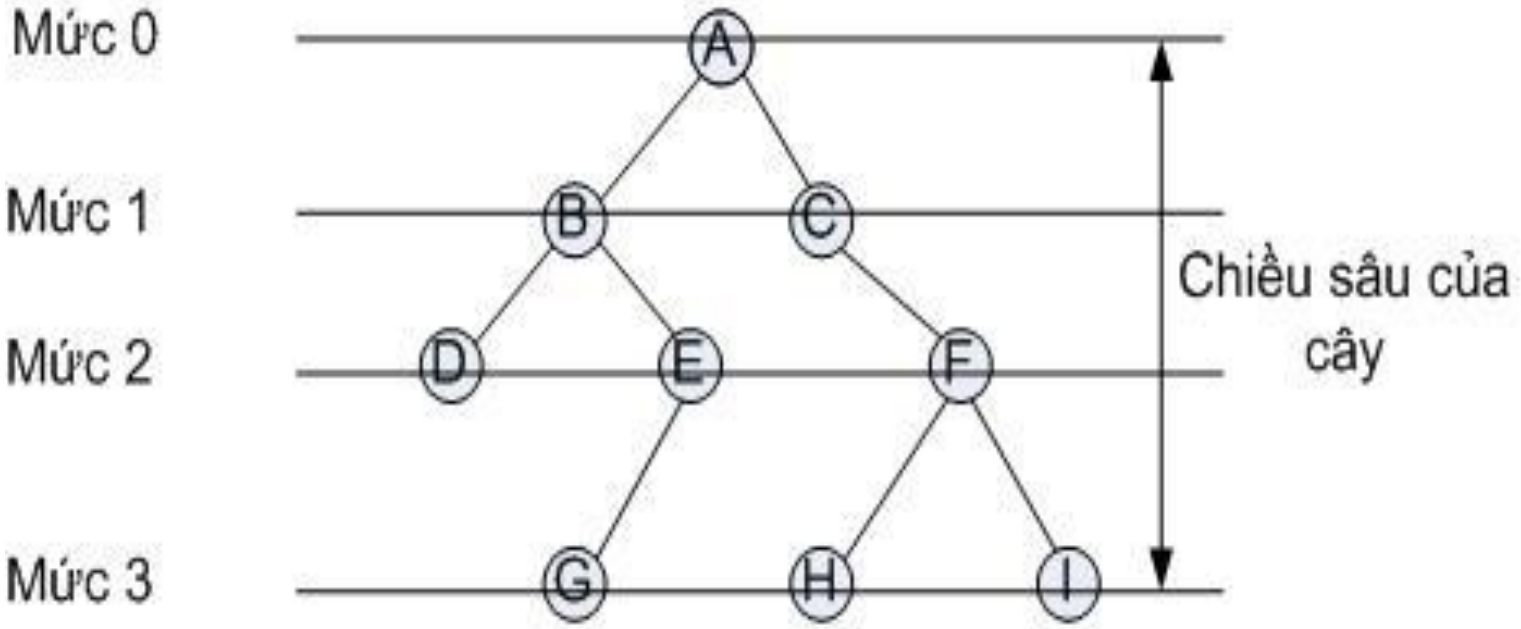
Hình: Cây nhị phân đúng

Giới thiệu về cây nhị phân

- Cây nhị phân là một cấu trúc gồm một tập hữu hạn các nút cùng kiểu dữ liệu (tập nút này có thể rỗng) và được phân thành 3 tập con:
  - Tập con thứ nhất có một nút gọi là nút gốc (root)
  - Hai tập con còn lại tự thân hình thành hai cây nhị phân là nhánh cây con bên trái (left subtree) và nhánh cây con bên phải (right subtree) của nút gốc. Nhánh cây con bên trái hoặc bên phải cũng có thể là cây rỗng.



Hình vẽ mô tả cây nhị phân

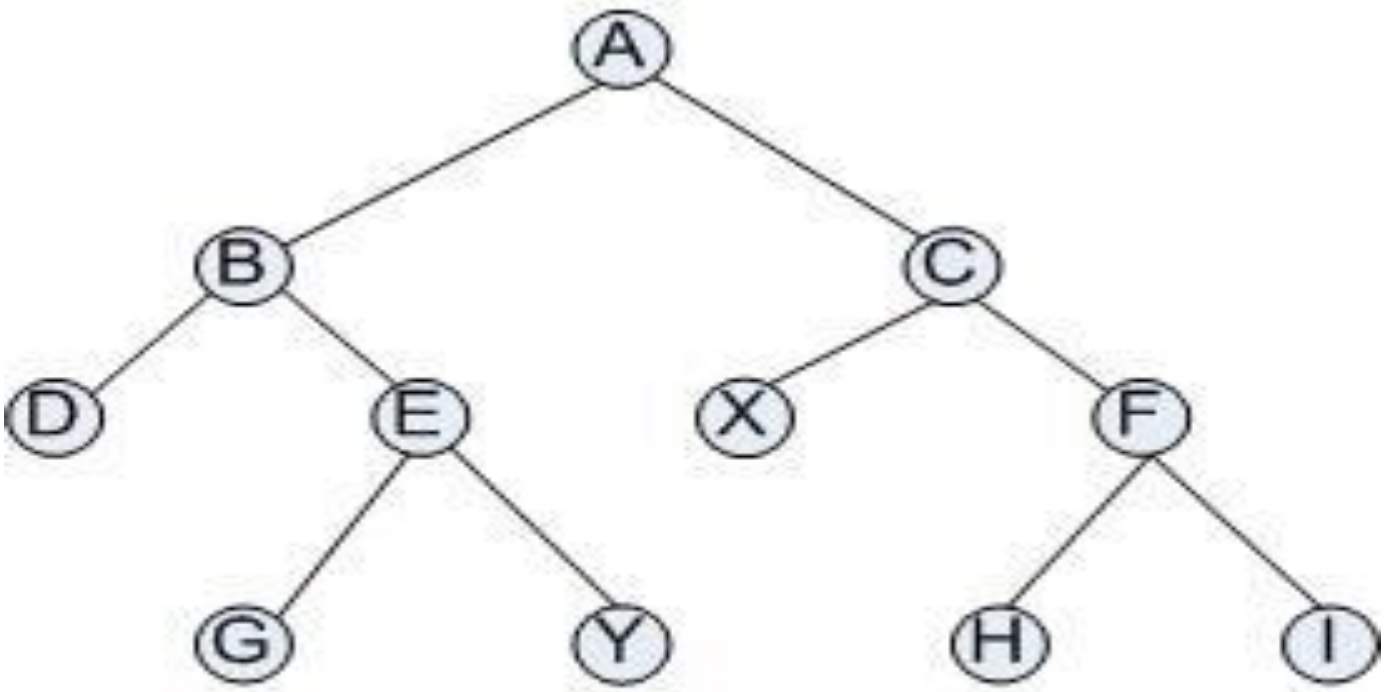


Hình: Khái niệm trên cây nhị phân

Cây nhị phân đúng *(strictly binary tree)*

- Một cây nhị phân gọi là cây nhị phân đúng nếu nút gốc và tất cả các nút trung gian đều có đủ hai nút con.
- Nếu cây nhị phân đúng có n nút lá thì cây này sẽ có tấc cả 2n - 1 nút.

Cây nhị phân đúng *(strictly binary tree)*



Hình: Cây nhị phân đúng

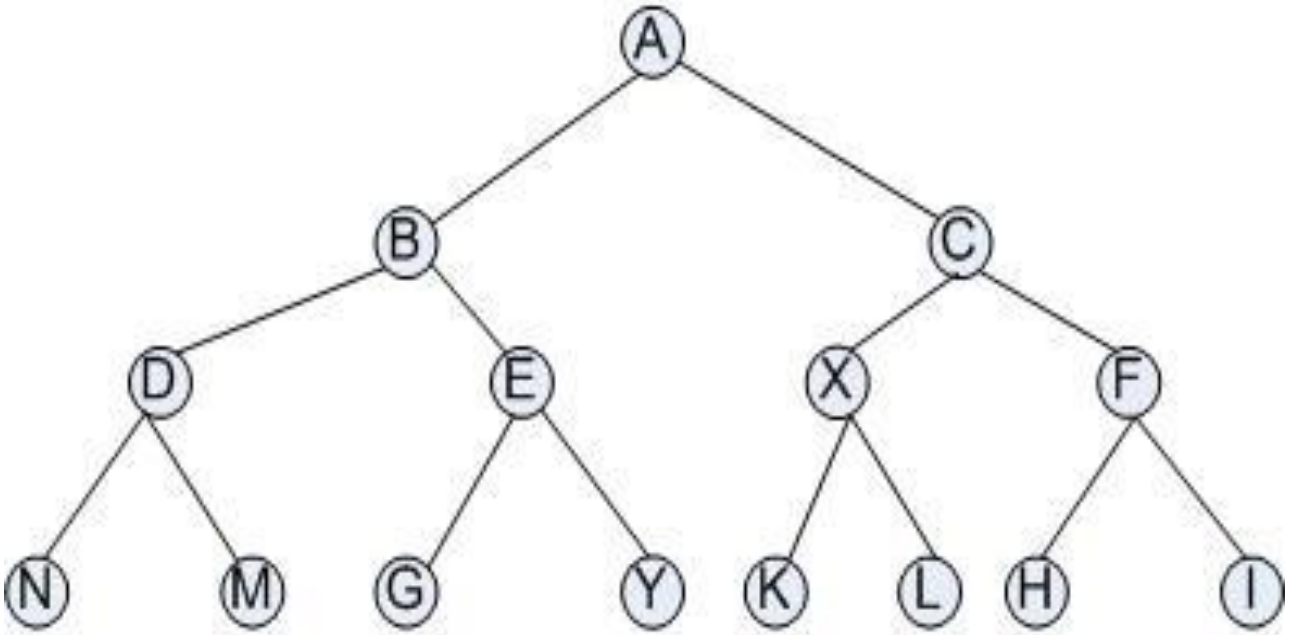
Cây nhị phân đầy *(complete binary tree)*

❖ Một cây nhị phân được gọi là cây nhị phân đầy với chiều sâu d thì:

- Trước tiên nó phải là cây nhị phân đúng.
- Tất cả các nút lá đều có mức là d.

❖ Cây nhị phân đầy là cây nhị phân có số nút tối đa ở mỗi mức.

Cây nhị phân đầy *(complete binary tree)*



Hình: Cây nhị phân đầy

Ba phép duyệt cây nhị phân

1. Duyệt cây nhị phân theo thứ tự trước (NLR- Node Left Right). Đầu tiên thăm nút gốc, sau đó đến duyệt cây con bên trái, sau đó duyệt cây con bên phải.

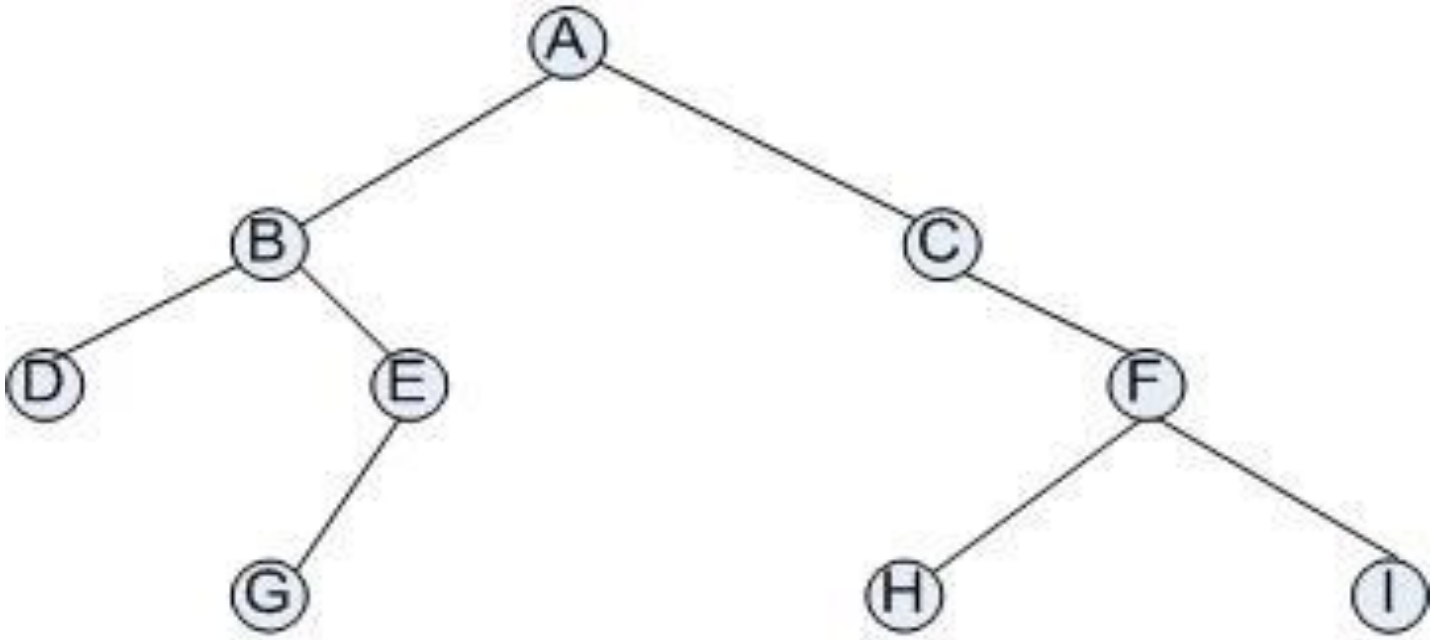
2. Duyệt cây theo thứ tự giữa (LNR):

Đầu tiên duyệt qua nhánh cây con bên trái, sau đó thăm nút gốc, cuối cùng duyệt cây con bên phải.

3. Duyệt cây theo thứ tự sau (LRN):

Đầu tiên, duyệt nhánh cây con bên trái, sau đó duyệt nhánh cây con bên phải, cuối cùng thăm nút gốc.

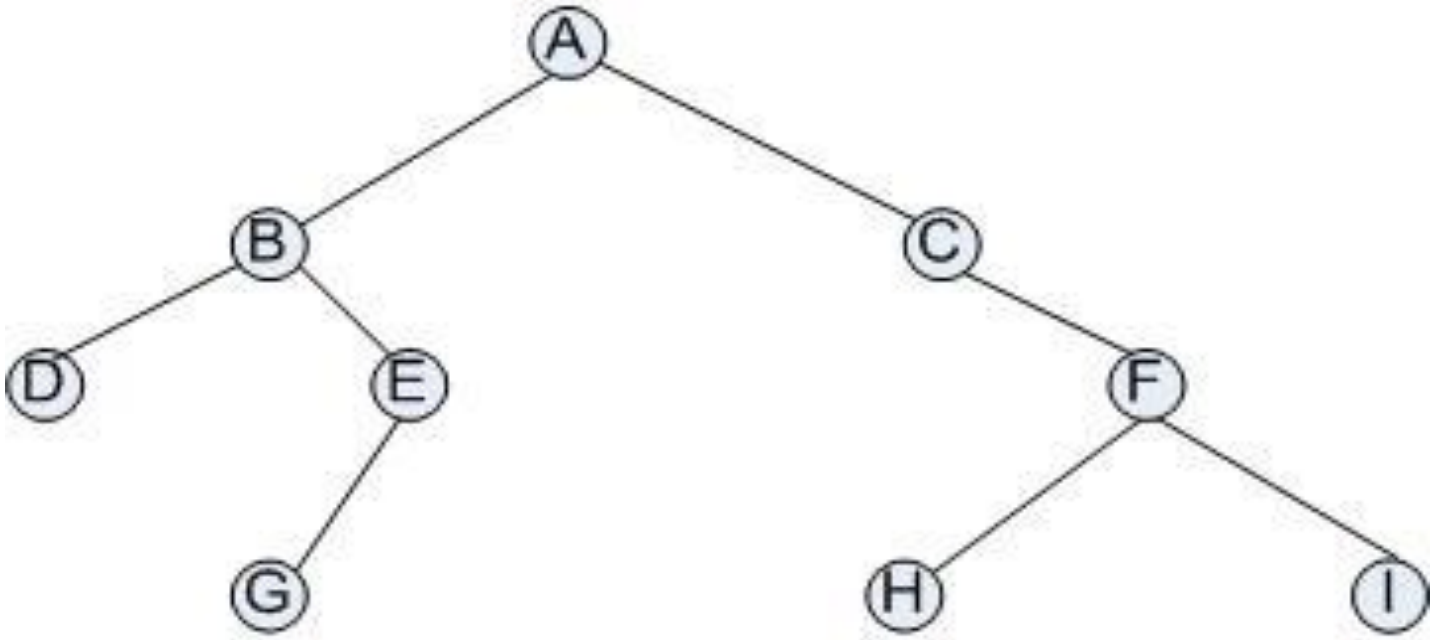
Ba phép duyệt cây nhị phân



Hình: Ba phép duyệt cây nhị phân

Nếu duyệt cây trên theo thứ tự NLR

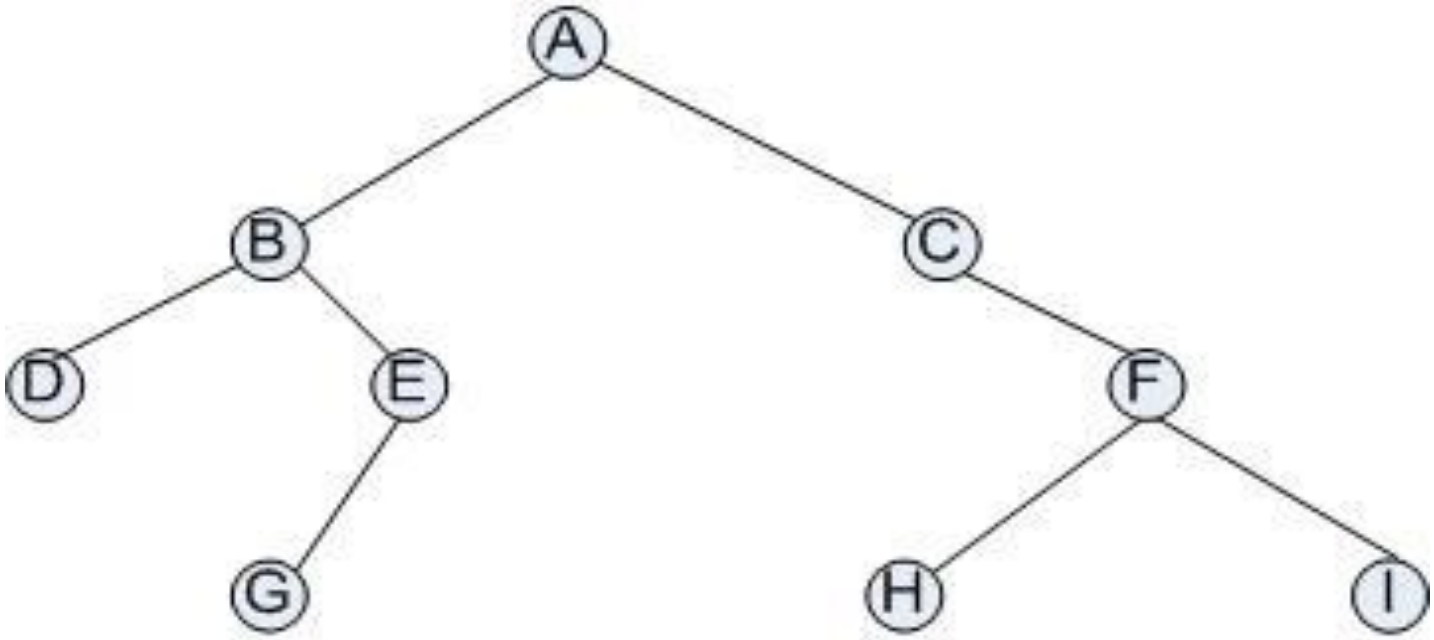
thì thứ tự các nút sẽ là: **A B D E G C F H I**



Hình: Ba phép duyệt cây nhị phân

Nếu duyệt cây trên theo thứ tự LNR

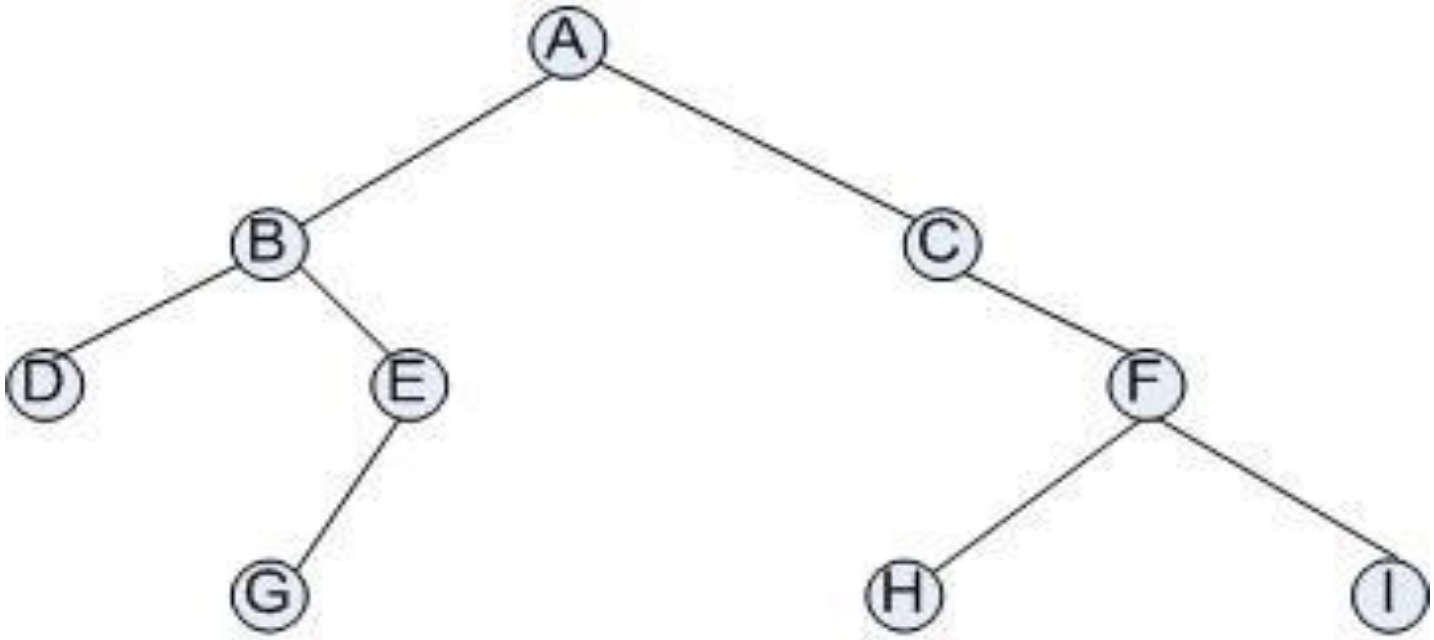
Thì thứ tự các nút sẽ là: **D B G E A C H F I**



Hình: Ba phép duyệt cây nhị phân

Nếu duyệt cây trên theo thứ tự LRN

Thì thứ tự các nút sẽ là: **D G E B H I F C A**



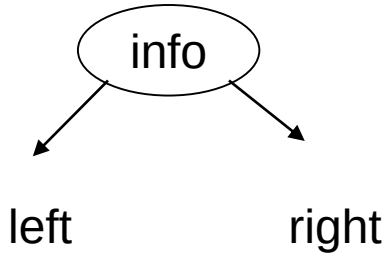
Hình: Ba phép duyệt cây nhị phân

Cấu trúc dữ liệu trên cây nhị phân

Mỗi nút trên cây nhị phân tổng quát là một mẩu tin có các trường như sau:

- Trường info: chứa nội dung của nút.
- Trường left là con trỏ dùng để chỉ nút con bên trái.
- Trường right là con trỏ dùng để chỉ nút con bên phải.

```
typedef struct node
{     int info;
      struct node*left;
      struct node *right;
};
typedef    node * TREE;
```

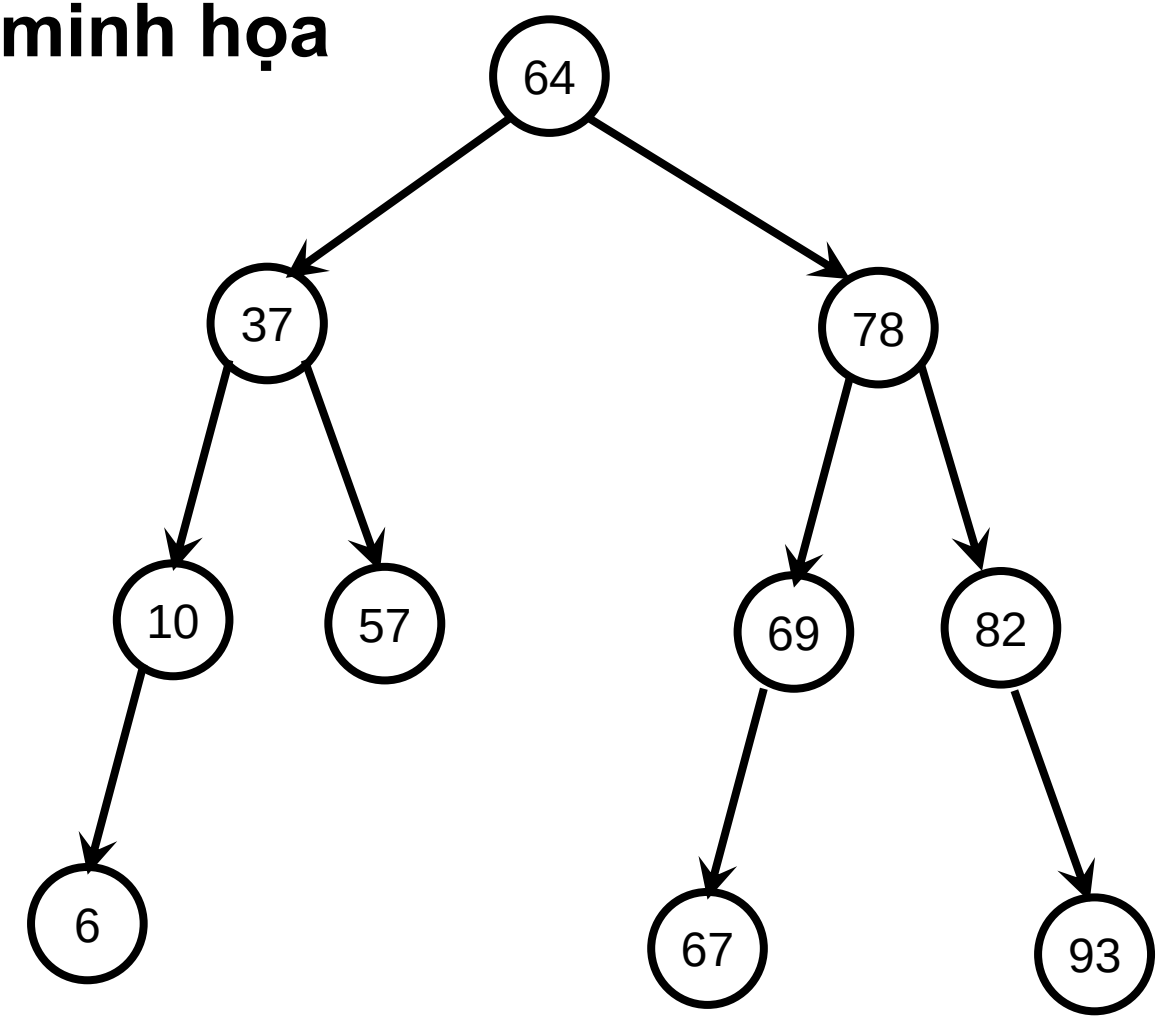


CÂY NHỊ PHÂN TÌM KIẾM
(BINARY SEARCH TREE)

ThS.Nguyễn Thúy Loan

CÂY NHỊ PHÂN TÌM KIẾM

Hình ảnh minh họa



CÂY NHỊ PHÂN TÌM KIẾM

1. Cây nhị phân tìm kiếm là cây nhị phân hoặc bị rỗng, hoặc tất cả các nút trên cây có nội dung thoả mãn các điều kiện sau:
   - Nội dung của tất cả các nút thuộc nhánh cây con bên trái đều nhỏ hơn nội dung của nút gốc.
   - Nội dung của tất cả các nút thuộc nhánh cây con bên phải đều lớn hơn nội dung của nút gốc.
   - Cây con bên trái và cây con bên phải tự thân cũng hình thành hai cây nhị phân tìm kiếm.

So sánh cây BST với các cấu trúc khác

- Với danh sách kề (Mảng) : Tác vụ thêm , xoá trên danh sách kề không hiệu quả vì chúng phải dời chổ nhiều lần các nút trong danh sách. Tuy nhiên, nếu danh sách kề là có thứ tự thì tác vụ tìm kiếm trên danh sách thực hiện rất nhanh bằng phương pháp tìm kiếm nhị phân, tốc độ tìm kiếm tỉ lệ với O(log n).

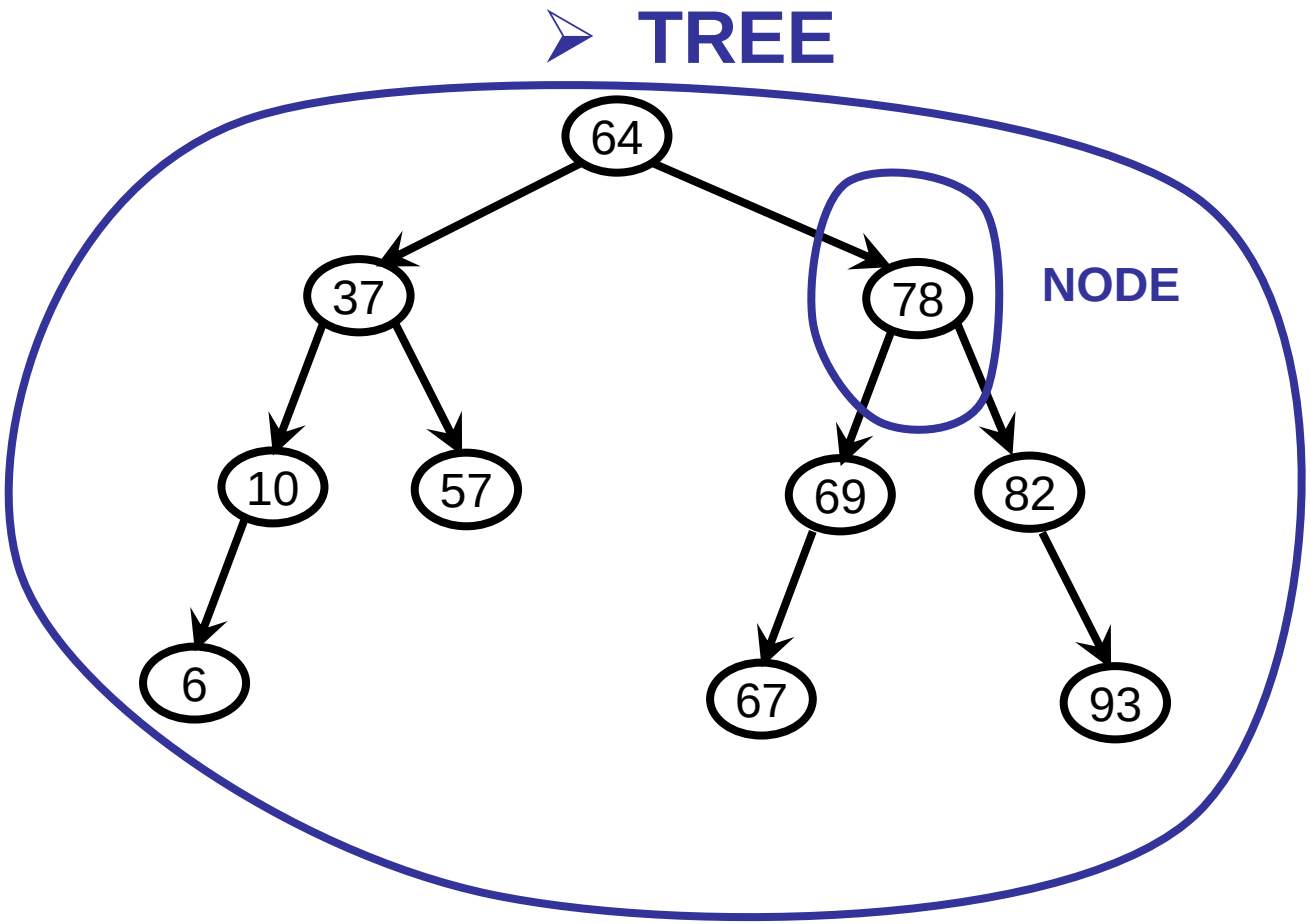
So sánh cây BST với các cấu trúc khác

- Với danh sách liên kết: Tác vụ thêm nút, xoá nút trên danh sách liên kết rất hiệu quả, lúc này chúng ta không phải dời chỗ các nút mà chỉ hiệu chỉnh một vài liên kết cho phù hợp. Nhưng tác vụ tìm kiếm trên danh sách liên kết không hiệu quả vì thường dùng phương pháp tìm kiếm tuyến tính dò từ đầu danh sách. Tốc độ tìm kiếm tỉ lệ với O(n).

So sánh cây BST với các cấu trúc khác

- Cây nhị phân tìm kiếm: Là cấu trúc dung hoà được 2 yếu tố trên: việc thêm nút hay xoá nút trên cây khá thuận lợi và thời gian tìm kiếm khá nhanh. Nếu cây nhị phân tìm kiếm là cân bằng thì thời gian tìm kiếm là O(log n), với n là số phần tử trên cây.

Hình ảnh minh họa



2. CẤU TRÚC DỮ LIỆU CỦA CÂY PHÂN TÌM KIẾM

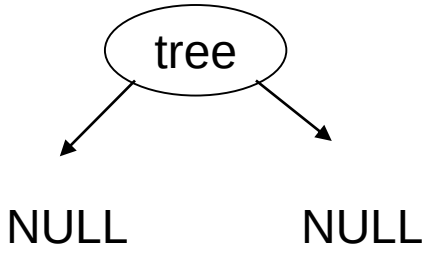
```
1.struct node
2.{
3.     KDL info;
4.     struct node *pLeft;
5.     struct node *pRight;
6.};
7.typedef struct node NODE;
8.typedef NODE *TREE;
```

- KDL là kiểu dữ liệu của đối tượng được lưu trong node của cây nhị phân tìm kiếm.

3. KHỞI TẠO CÂY PHÂN TÌM KIẾM

- Khái niệm: Khởi tạo cây nhị phân tìm kiếm là tạo ra cây nhị phân rỗng không chứa node nào hết.
- Định nghĩa hàm:

```
1.void Init(TREE &t)
2.{
3.      t = NULL;
4.}
```



4. KIỂM TRA CÂY NHỊ PHÂN RỖNG

- Khái niệm: Cây nhị phân rỗng là cây không chứa một node nào. Hàm sẽ trả về giá trị 1 nếu cây nhị phân rỗng. Ngược lại hàm trả về giá trị 0.
- Định nghĩa hàm

```
11.int IsEmpty(TREE t)
12.{
13.     if (t==NULL)
14.          return 1;
15.     return 0;
16.}
```

5. TẠO NODE CHO CÂY NHỊ PHÂN TÌM KIẾM

- Khái niệm: Tạo node cho cây nhị phân tìm kiếm là xin cấp phát bộ nhớ có kích thước bằng kích thước của KDL NODE để chứa thông tin biết trước.

```
NODE* GetNode(KDL x)
{    NODE *p = new NODE;
     if(p==NULL)
          return NULL;
     else
     {     p->info=x;
           p->pLeft = NULL;
           p->pRight= NULL;
     }
     return p;
}
```

6. THÊM NODE CHO CÂY NHỊ PHÂN TÌM KIẾM

- Khái niệm: Thêm một node vào trong cây nhị phân tìm kiếm là thêm thông tin vào cây sao cho tính chất của cây nhị phân tìm kiếm không bị vi phạm.

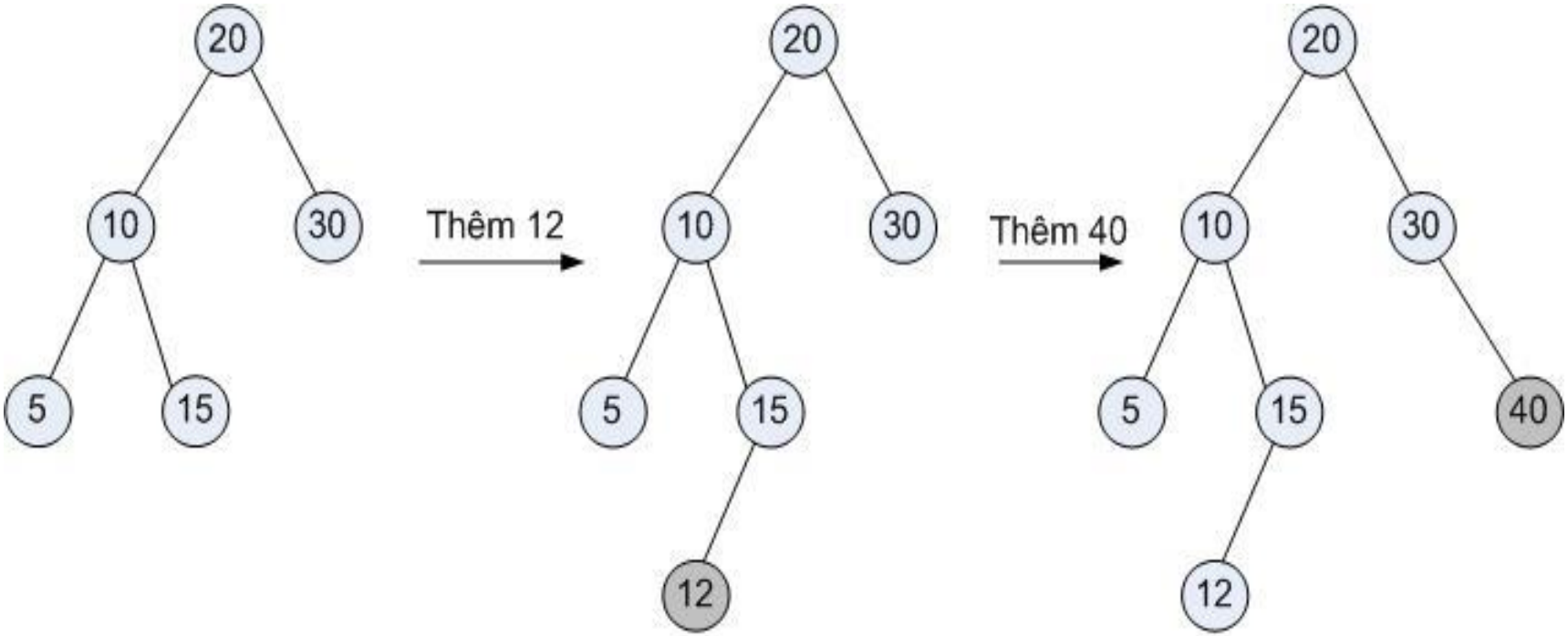
6. THÊM NODE CHO CÂY NHỊ PHÂN TÌM KIẾM

Giá trị trả về:

Hàm thêm một node vào trong cây nhị phân tìm kiếm trả về một trong 3 giá trị -1, 0, 1 như sau:

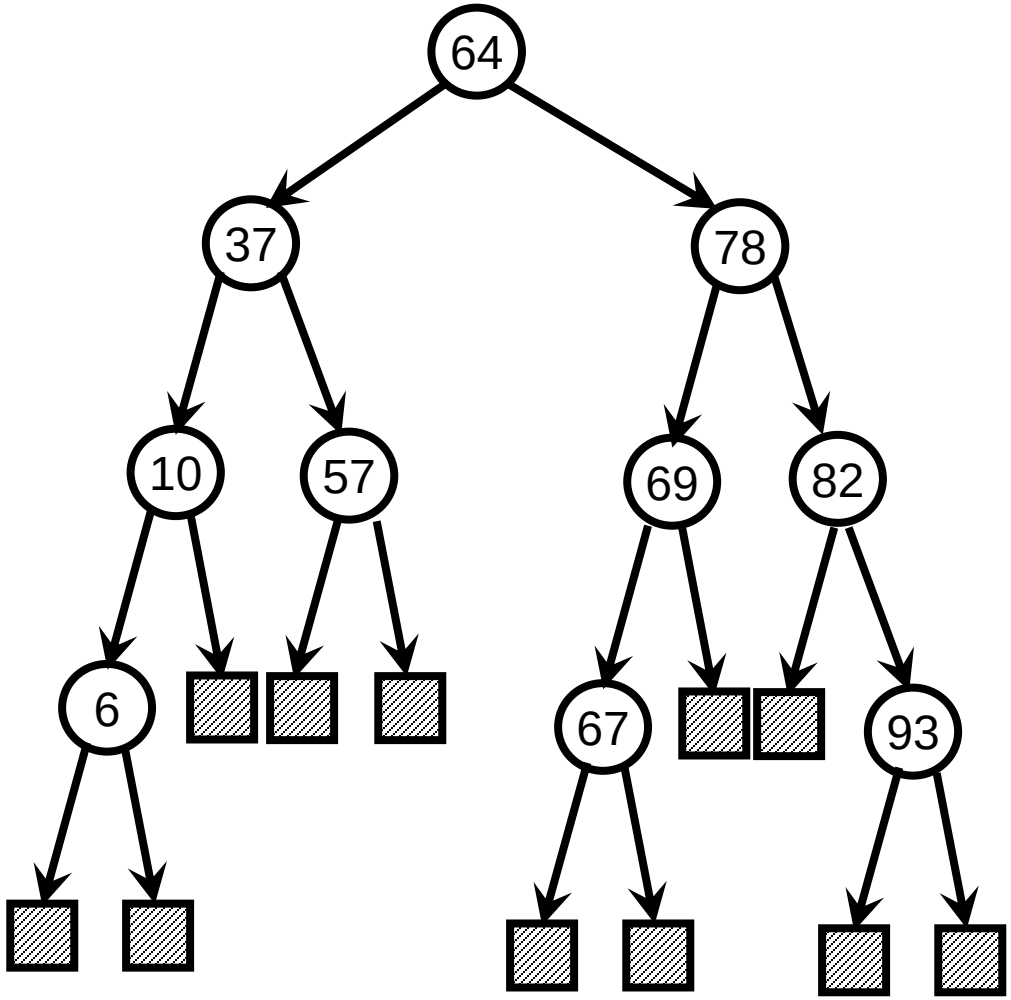
- **Giá trị 1**: Thêm thành công
- **Giá trị 0**: Trùng với khoá một node đã có sẵn trong cây.
- **Giá trị -1**: Không đủ bộ nhớ.

6. THÊM NODE CHO CÂY NHỊ PHÂN TÌM KIẾM



Hình minh hoạ tác vụ thêm vào trên cây nhị phân tìm kiếm

6. THÊM NODE CHO CÂY NHỊ PHÂN TÌM KIẾM



6. THÊM NODE CHO CÂY NHỊ PHÂN TÌM KIẾM

//Thêm một node vào cây nhị phân tìm kiếm các số nguyên.

- Định nghĩa hàm

```
11.int InsertNode(TREE &t,int x)
12.{
13.
14.}
```

6. THÊM NODE CHO CÂY NHỊ PHÂN TÌM KIẾM

```
11.int InsertNode(TREE &t,int x)
12.{
13.   if(t!=NULL)
14.   {
15.      if(t->info < x)
16.         return InsertNode(t->pRight,x);
17.      if(t->info > x)
18.         return InsertNode(t->pLeft,x);
19.      return 0;
20.   }
21.   t = GetNode(x);
22.   if(t==NULL)
23.      return -1;
24.   return 1;
25.}
```

7. NHẬP CÂY NHỊ PHÂN TÌM KIẾM

// Nhập dãy số nguyên lưu trên cây NPTK

```
void Input(TREE &t)
{    int x;
     Init(t);
     printf( " Nhan số 0 de thoat ");
     do
     {     printf(" nhap x:");
           scanf("%d",&x);
           if( x!=0 )
                InsertNode(t,x)
            else
                printf (" ket thuc nhap ");
     }while(x!= 0);
}
```

Duyệt cây nhị phân

Ba phép duyệt cây nhị phân

1. Duyệt cây nhị phân theo thứ tự trước (NLR- Node Left Right). Đầu tiên thăm nút gốc, sau đó đến duyệt cây con bên trái, sau đó duyệt cây con bên phải.

2. Duyệt cây theo thứ tự giữa (LNR):

Đầu tiên duyệt qua nhánh cây con bên trái, sau đó thăm nút gốc, cuối cùng duyệt cây con bên phải.

3. Duyệt cây theo thứ tự sau (LRN):

Đầu tiên, duyệt nhánh cây con bên trái, sau đó duyệt nhánh cây con bên phải, cuối cùng thăm nút gốc.

8. DUYỆT CÂY NHỊ PHÂN TÌM KIẾM

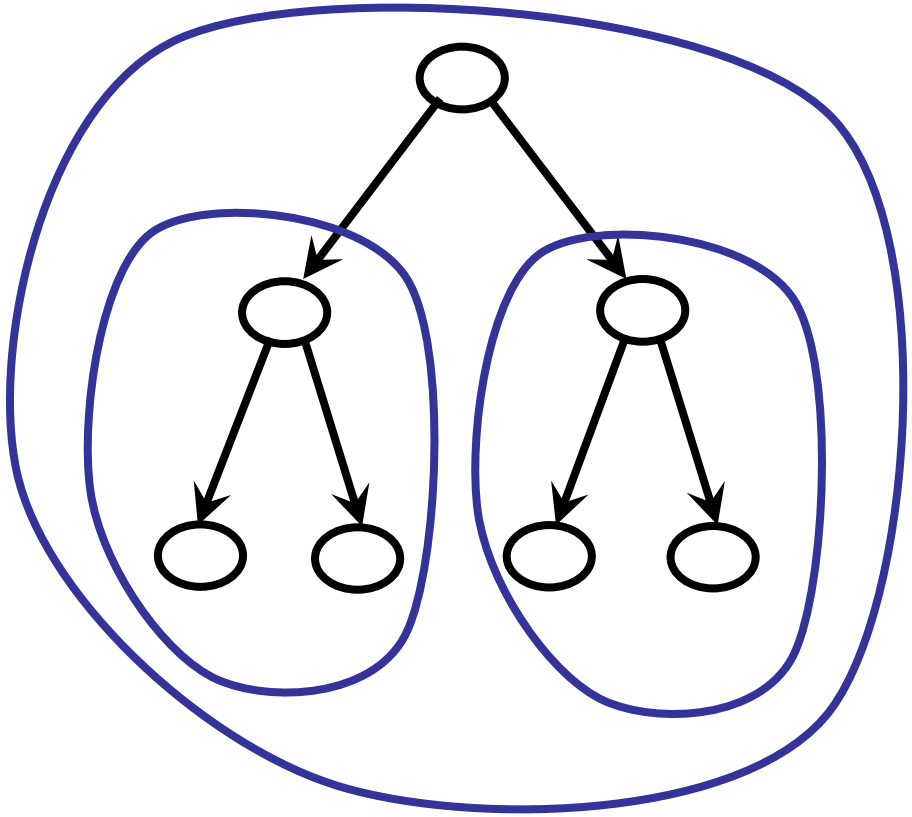
- Khái niệm: Duyệt cây nhị phân tìm kiếm là thăm qua tất cả các node trong cây mỗi node một lần
- Định nghĩa hàm trừu tượng

```
1.KDL Process(TREE t)
2.{    if(t==NULL)
3.         return ...
4.     ...Process(t->pLeft);
5.     ...
6.     ...Process(t->pRight);
7.     return ...
8.}
```

Hàm xuất tất cả các node trong cây nhị phân tìm kiếm các số nguyên theo kiểu LNR

```
1.void Xuat(TREE t)// LNR
2.{
3.      if(t==NULL)   return;
4.      Xuat(t->pLeft);
5.      printf("%4d",t->info);
6.      Xuat(t->pRight);
7.      return;
8.}
```

TÍNH TỔNG CÁC GIÁ TRỊ DƯƠNG TRÊN CÂY



Chương trình

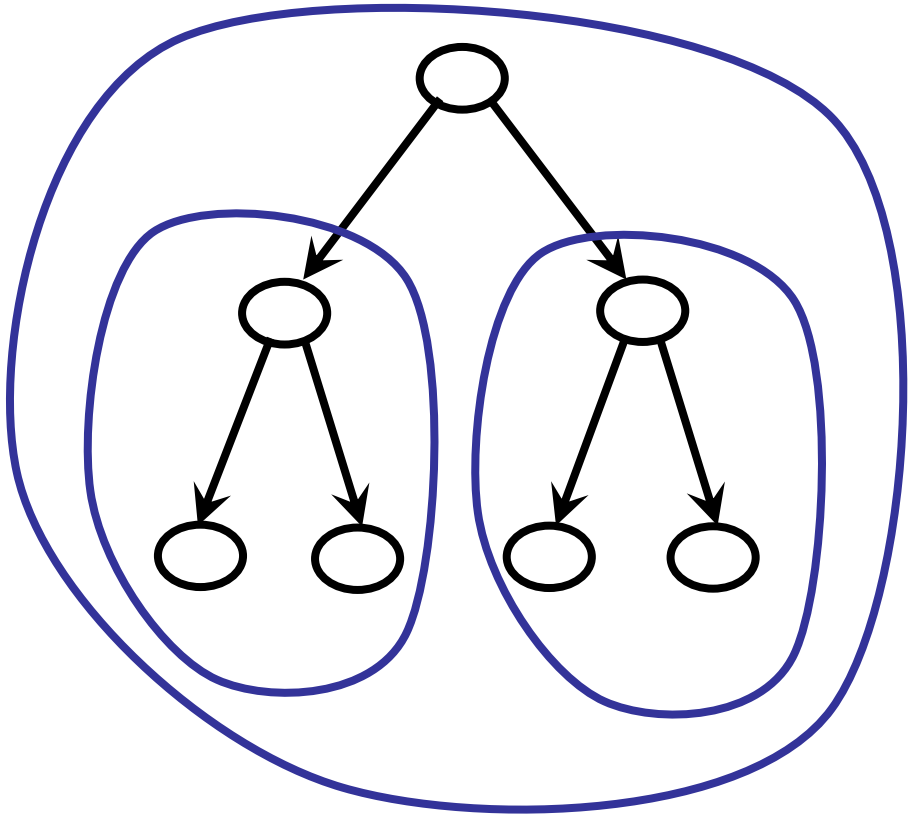
```
83.long Tong(TREE t)
84.{
85.     if(t==NULL) return 0;
86.     long a=Tong(t->pLeft);
87.     long b=Tong(t->pRight);
88.     return (a+b+t->info);
89.
90.}
```

Chương trình

```
83.long Tongchan(TREE t)
84.{
85.     if(t==NULL) return 0;
86.    long a=Tongchan(t->pLeft);
87.     long b=Tongchan(t->pRight);
88.     if (t->info%2==0)
89.                 return (a+b+t->info);
90.      return a+b;
91.}
```

```
83.long TongDuong(TREE t)
84.{
85.     if(t==NULL) return 0;
86.     long a=TongDuong(t->pLeft);
87.     long b=TongDuong(t->pRight);
88.     if (t->info>0)
89.                 return (a+b+t->info);
90.     return a+b;
91.}
```

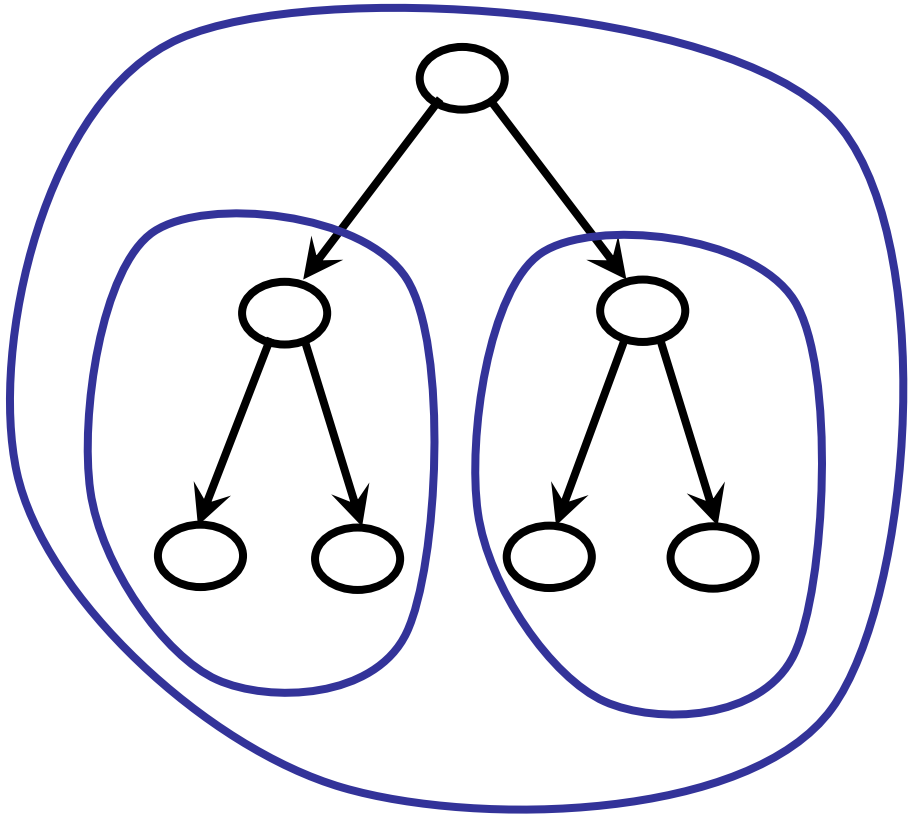
Chương trình



Chương trình

```
83.int demnut(TREE t)
84.{
85.     if(t==NULL) return 0;
86.     int a=demnut(t->pLeft);
87.     int b=demnut(t->pRight);
88.     return (a+b+1);
89.
90.}
```

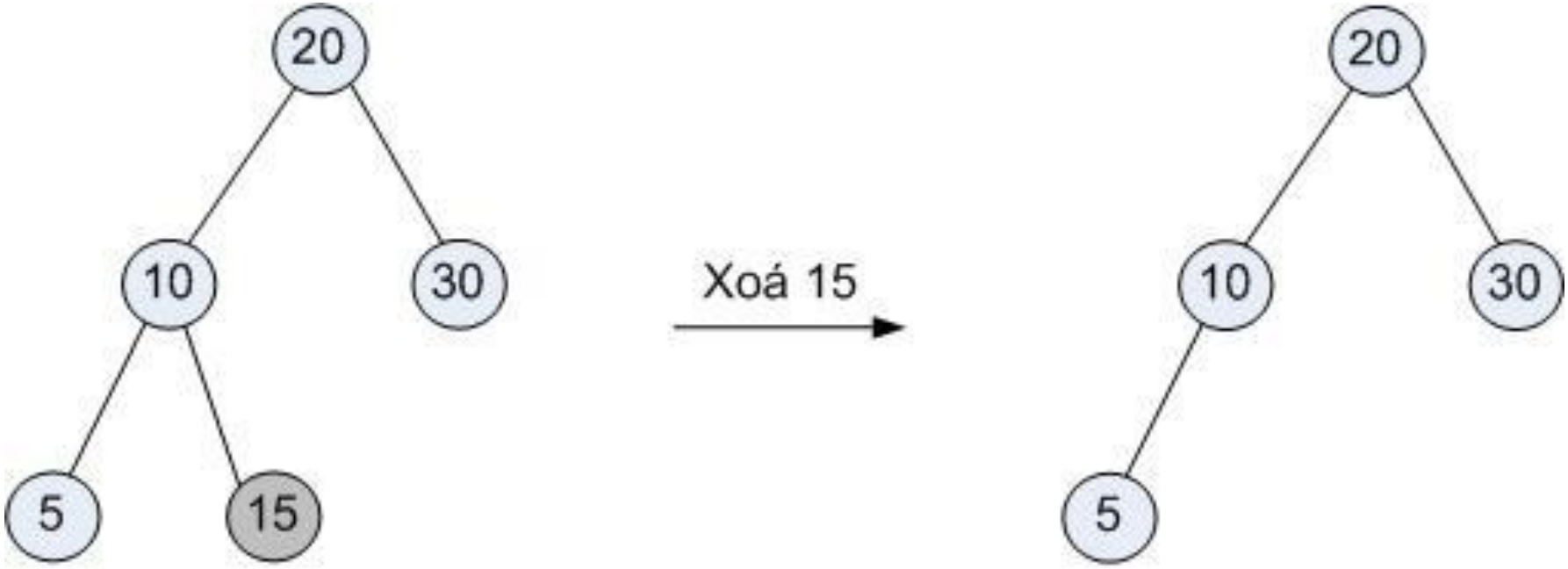
Chương trình



Chương trình

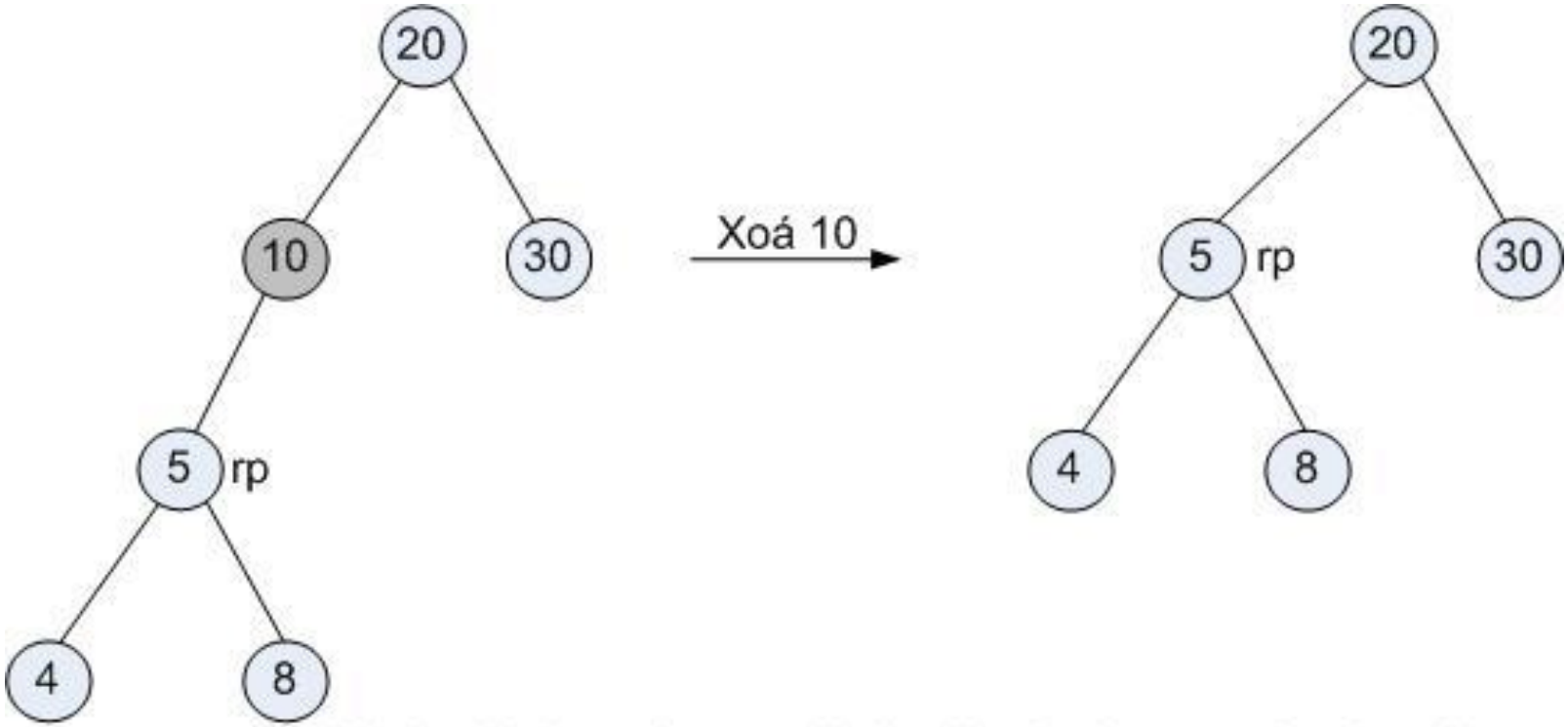
```
83.int chieucao(TREE t)
84.{
85.     if(t==NULL) return 0;
86.     int a=chieucao(t->pLeft);
87.     int b=chieucao(t->pRight);
88.     if (a>b)
89.                 return (a+1);
90.      return b+1;
91.}
```

Xóa phần tử trên cây



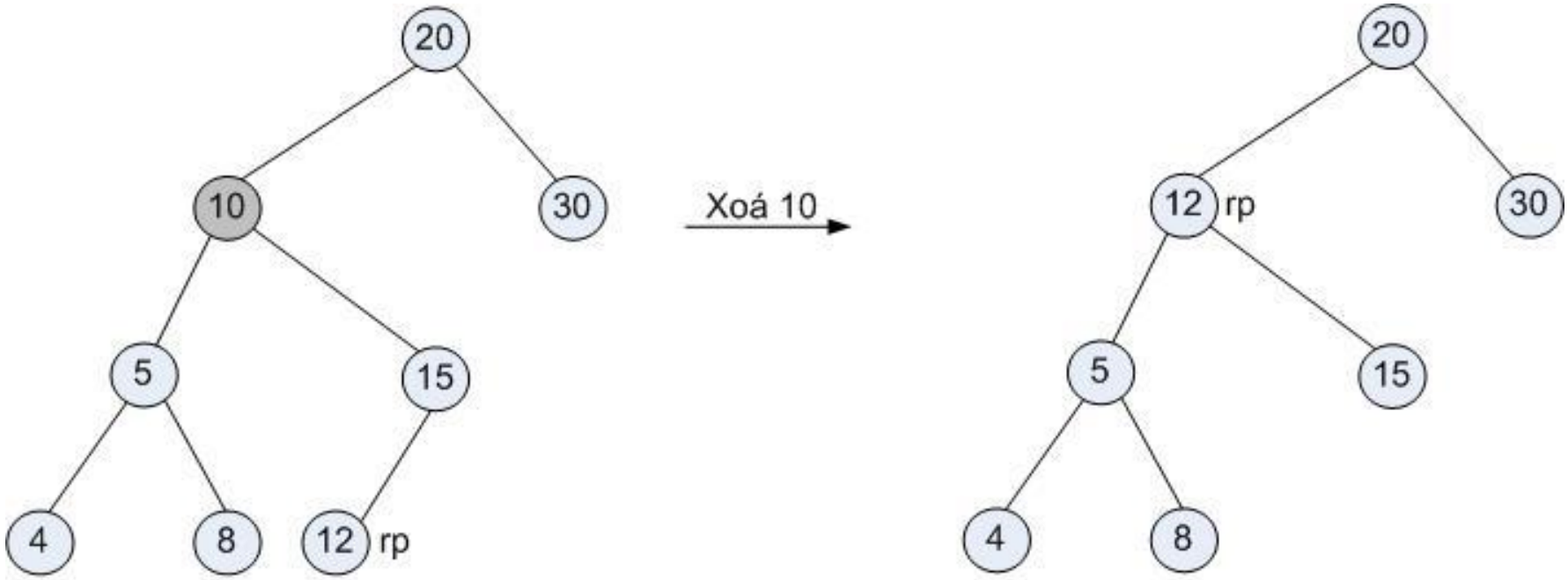
Hình minh hoạ tác vụ xoá nút lá trên cây nhị phân tìm kiếm

Xóa phần tử trên cây



Hình minh hoạ tác vụ xoá nút có một cây con trên cây nhị phân tìm kiếm

Xóa phần tử trên cây



Hình minh hoạ tác vụ xoá nút có hai cây con trên cây nhị phân tìm kiếm

Xóa phần tử trên cây

Có 3 vị trí để xóa

1. X nằm ở nút lá
2. X chỉ có 1 nút con (bên trái hoặc phải)
3. X có đủ 2 nút con

Trường hợp 1 : chỉ đơn giản hủy x vì nó không móc nối đến phần tử nào khác .

Trường hợp 2: trước khi hủy x, ta móc nối cha của x với con duy nhất của nó.

Xóa phần tử trên cây

Trường hợp 3 : ta không thể hủy trực tiếp do x có đủ 2 con, do đó thay vì hủy x, ta sẽ tìm một phần tử thế mạng y.

Có 2 phần tử thỏa mãn yêu cầu

1.Phần tử trái nhất trên cây con phải . (chọn để cài)

2. Phần tử phải nhất trên cây con trái .

Xóa phần tử trên cây

```
int delnode ( tree & t , int x )
{
    if ( t == NULL )    return 0;
    if ( t -> info > x)  return   delnode ( t -> pleft , x ) ;
    if ( t -> info < x )  return  delnode ( t -> pright , x ) ;
    else        // tìm thấy nút x cần xóa
    {     tree p = t ;
          if ( t -> pleft  == NULL )  t = t -> pright ;
          else
                if ( t -> pright == NULL)  t = t -> pleft ;
                else
```

Xóa phần tử trên cây

```
                else
                {
                        tree  q = t -> pright ;
                        searchnode ( p , q ) ;
// tìm phần tử cực trái bên nhánh phải thế mạng
                }
          delete p ;
        }
        return 0;
}
```

Tìm phần tử q thay thế cho p

```
void  searchnode ( tree &p , tree  &q )
{
    if ( q -> pleft  != NULL)
        searchnode ( p , q -> pleft ) ;
    else
    {
        p -> info = q -> info ;
        p = q ;
        q = q -> pright ;
    }
}
```

THU HỒI BỘ NHỚ

- Vấn đề: Định nghĩa hàm thu hồi tất cả các bộ nhớ đã cấp phát cho cây nhị phân tìm kiếm các số nguyên.
- Định nghĩa hàm trừu tượng

```
1.void RemoveAll(TREE &t)
2.{
3.     if(t==NULL)  return;
4.     RemoveAll(t->pLeft);
5.     RemoveAll(t->pRight);
6.     delete t;
7.}
```