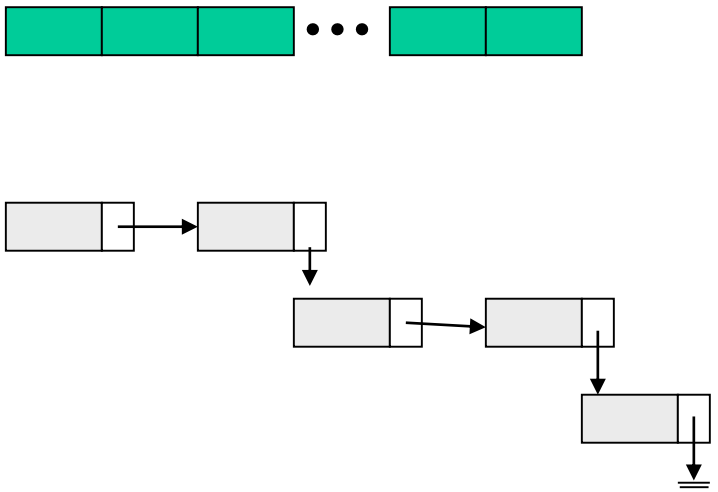
CẤU TRÚC DỮ LIỆU & GIẢI THUẬT

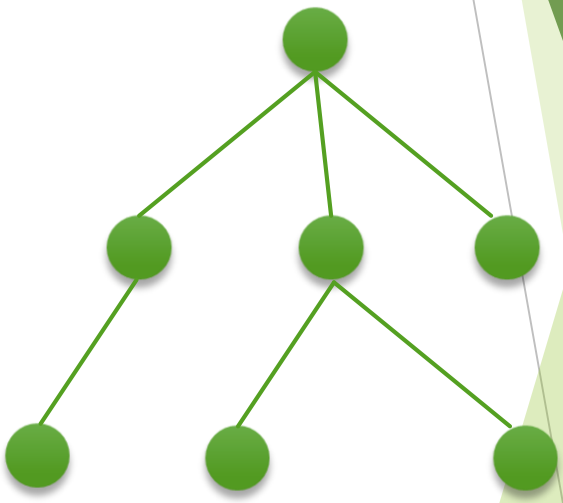
CÂY NHỊ PHÂN

Nội dung

1. Cấu trúc cây tổng quát
2. Khái niệm
3. Đặc điểm
4. Định nghĩa cấu trúc dữ liệu
5. Các lưu ý khi cài đặt
6. Các thao tác xử lý

Cấu trúc cây?

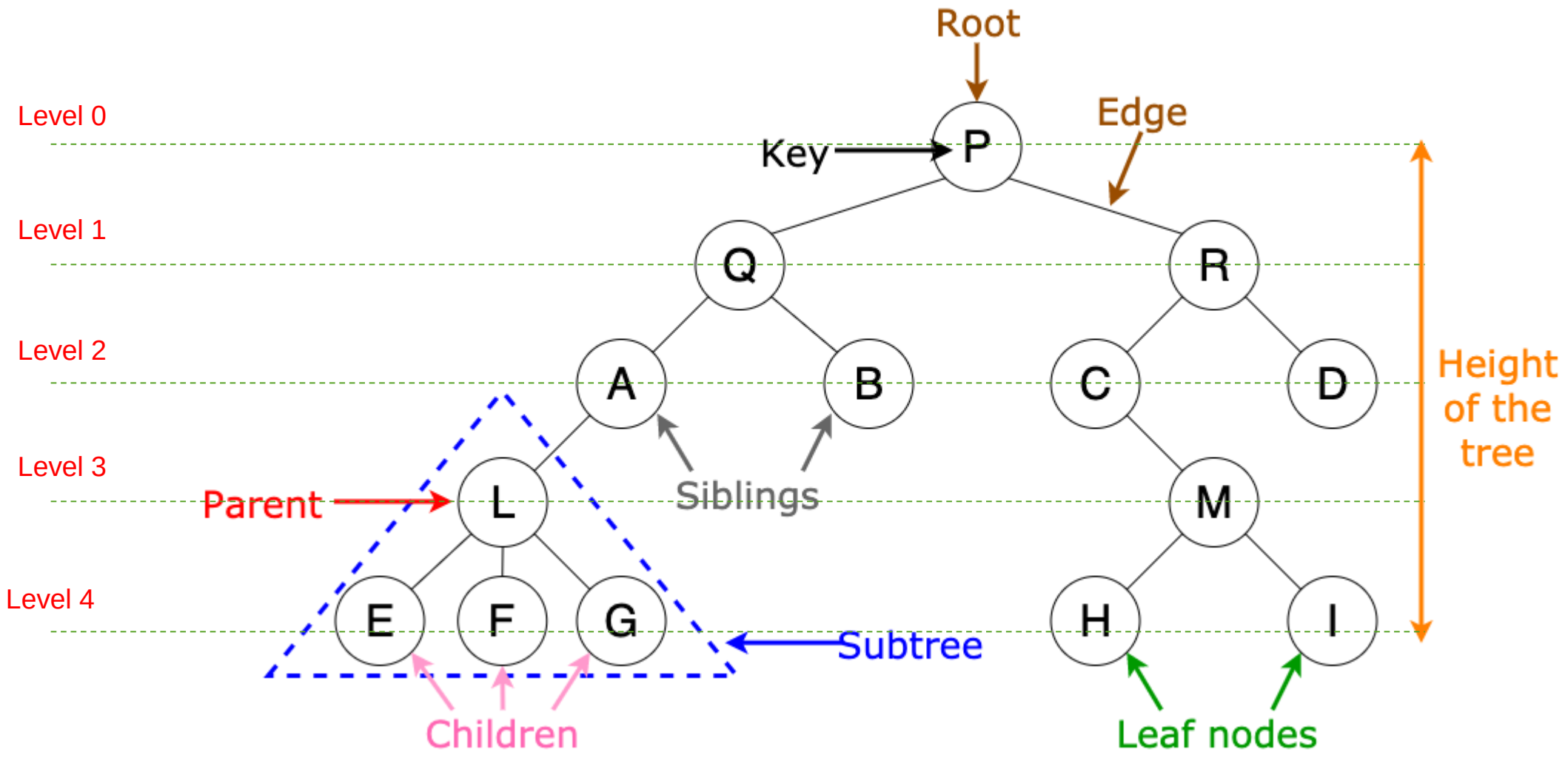




Cấu trúc cây tổng quát

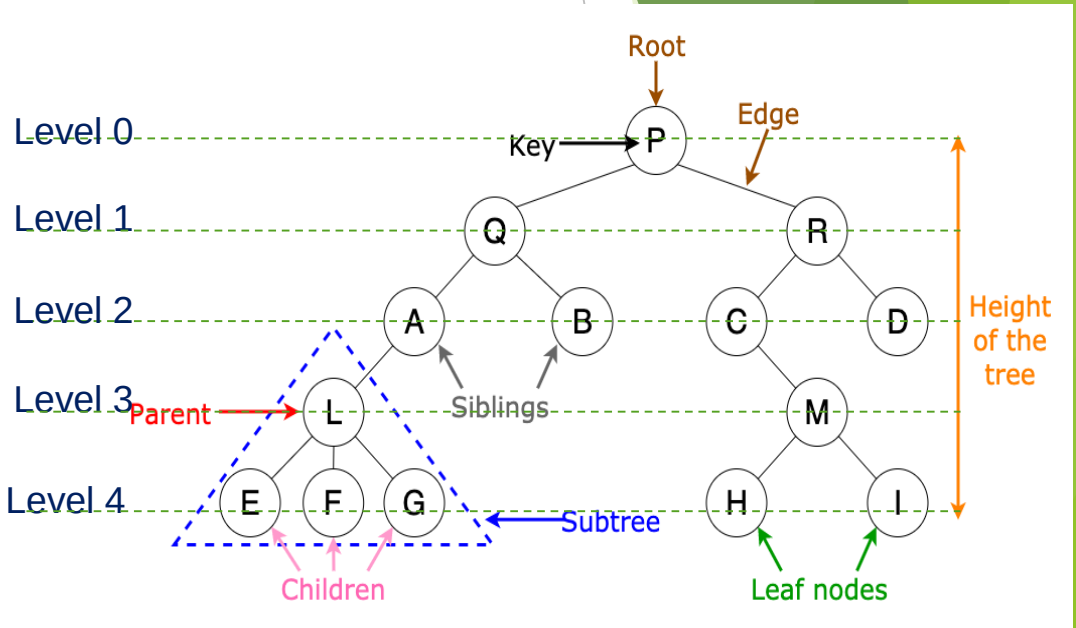
- Tập hợp các nút và cạnh nối các nút
- Có một nút gọi là gốc
- Quan hệ one-to-many giữa các nút
- Có duy nhất một đường đi từ gốc đến một nút
- Các loại cây:
  - *Nhị phân: mỗi nút có {0,1, 2} nút con*
  - *Tam phân: mỗi nút có {0,1,2,3} nút con*
  - *n-phân: mỗi nút có {0,1,..,n} nút con*

Cấu trúc cây tổng quát



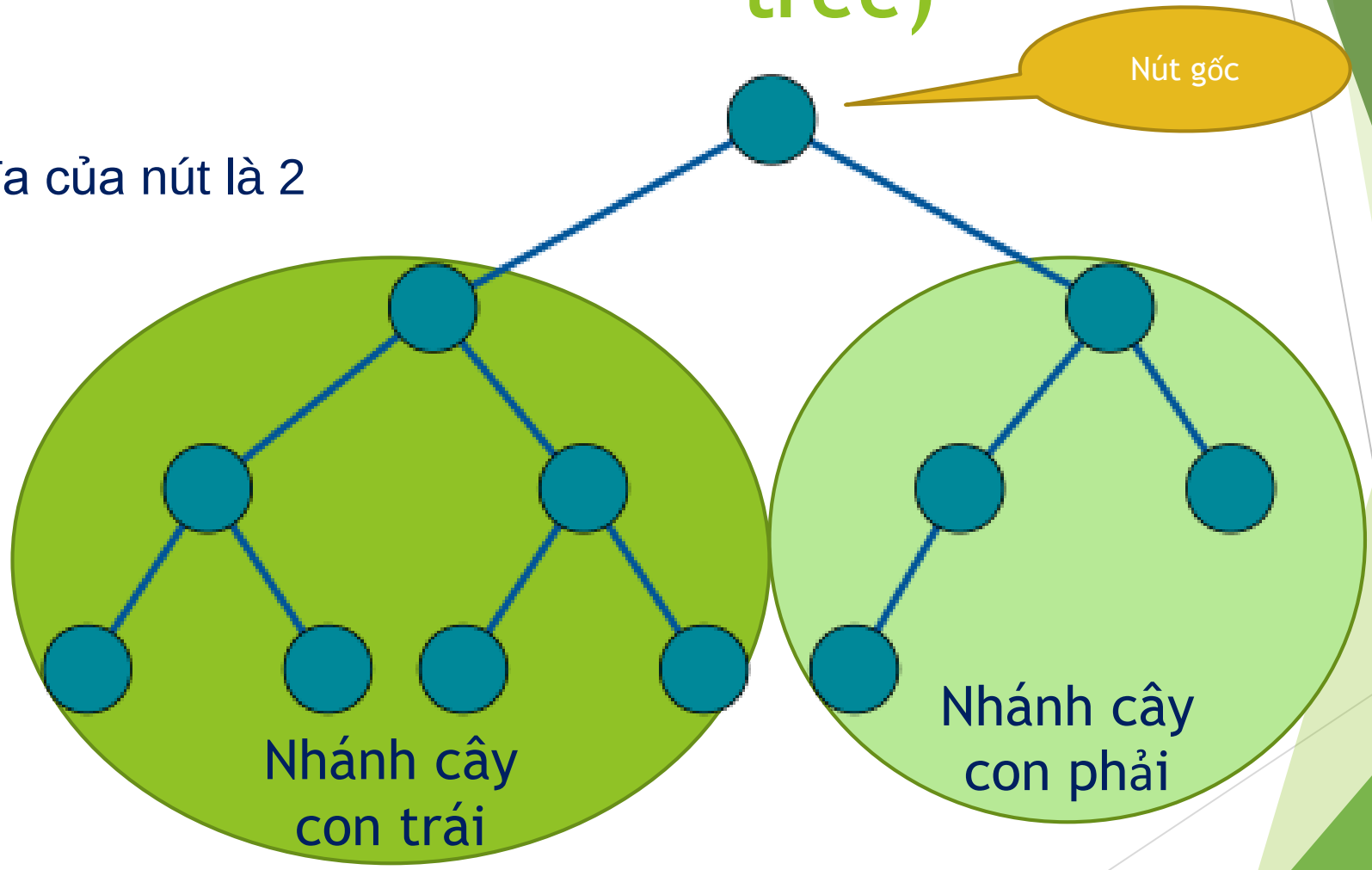
Cấu trúc cây tổng quát

- **Nút gốc** : không có nút cha
- **Nút lá** : không có nút con
- **Nút trong** : không phải nút lá và nút gốc
- **Bậc của nút** : số nút con của nút đó
- **Bậc của cây** : bậc lớn nhất của các nút
- **Mức của nút:**
  - Nút gốc có mức = 0
  - Các nút khác nút gốc có mức = mức của nút cha + 1
- **Độ cao cây:** là mức lớn nhất của các nút lá + 1



Cây nhị phân (Binary tree)

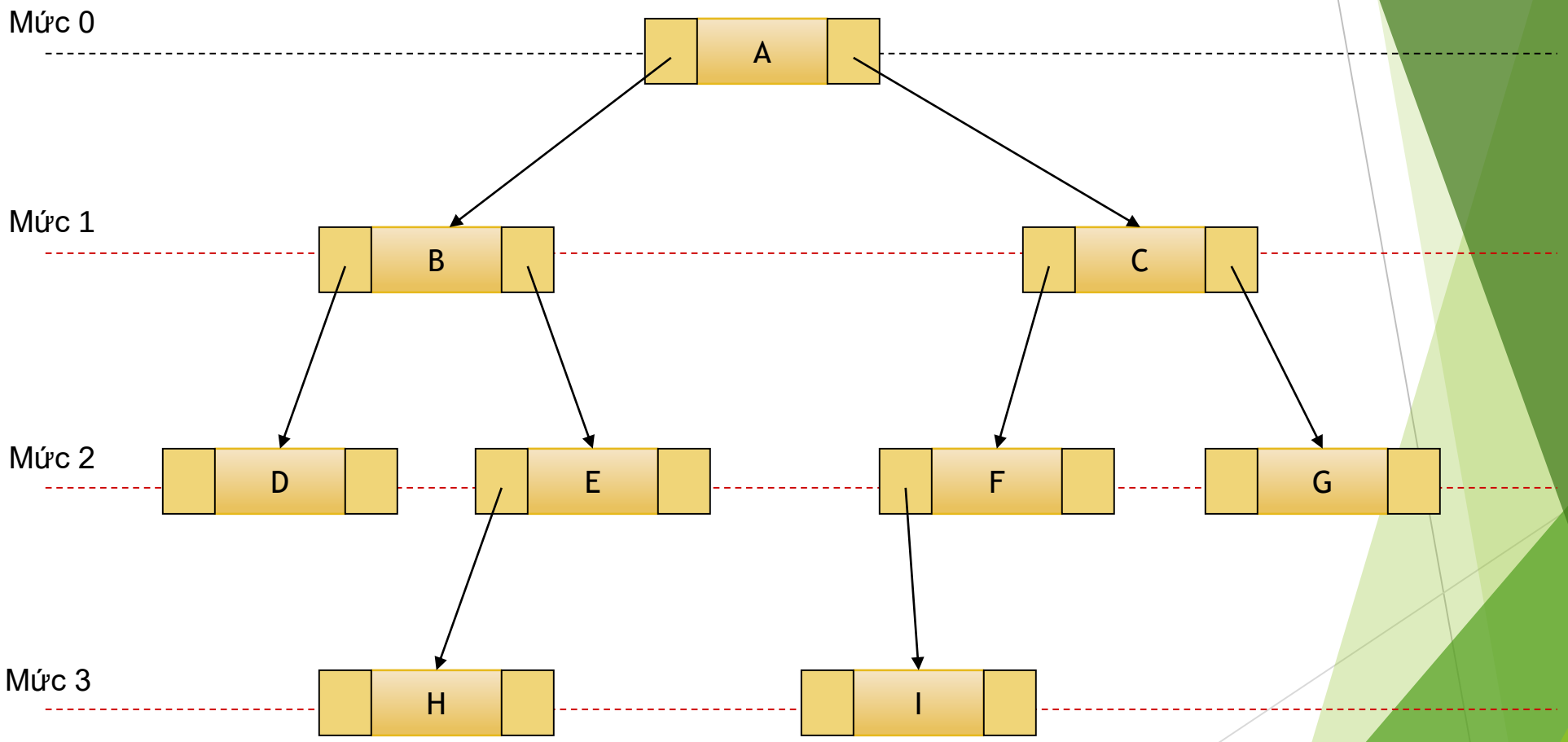
Bậc tối đa của nút là 2



Đặc điểm của cây nhị phân

- Số nút ở mức $I \leq 2^{I-1}$
- Số nút ở mức lá $\leq 2^{h-1}$, với h là chiều cao của cây
- Chiều cao của cây $h \geq \log_2 N$, với N là số nút trong cây

Mô tả dữ liệu cây nhị phân



Mô tả dữ liệu cây nhị phân

- Cây nhị phân là một cấu trúc bao gồm các phần tử (node) được kết nối với nhau theo quan hệ "cha-con" với mỗi cha có tối đa 2 con
- Mỗi nút gồm các thông tin:

  - Dữ liệu lưu trữ: *info*
  - Liên kết tới cây con trái của nút (nếu có): *left*
  - Liên kết tới cây con phải của nút (nếu có): *right*

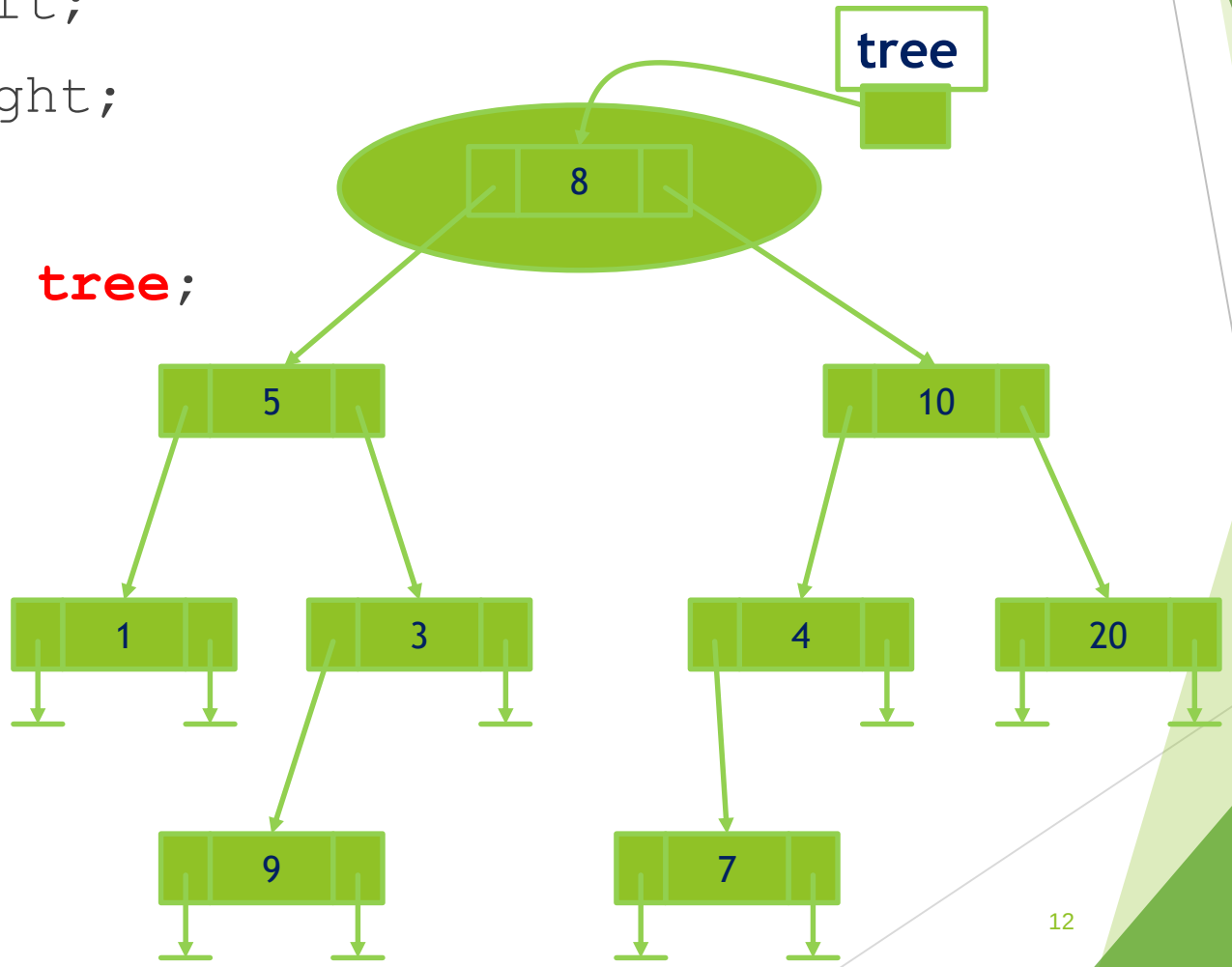
Các tác vụ trên cây nhị phân

1. Khởi tạo cây
2. Kiểm tra rỗng
3. Tạo nút
4. Thêm trái/ phải
6. Xóa trái/ phải
7. Duyệt cây: PreOrder, InOrder, PostOrder
8. Tìm kiếm
9. Xóa cây

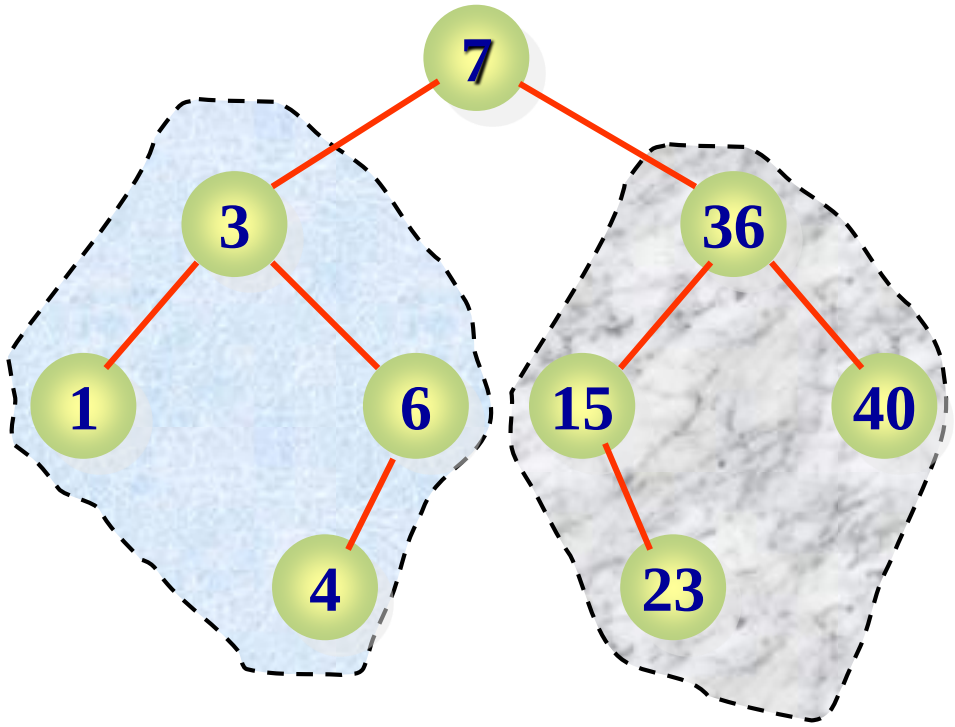
Khai báo CTDL cây nhị phân

```
typedef struct tagNode
{
   DataType   info;
   struct tagNode *left;
   struct tagNode *right;
}*PtrNode, Node;
```

Khai báo cây: PtrNode **tree**;



Cây nhị phân tìm kiếm
Binary search tree (BST)



Cây nhị phân tìm kiếm số nguyên

- Là 1 cây nhị phân
- Giá trị khóa của một node luôn lớn hơn giá trị khóa của các node cây con trái và nhỏ hơn giá trị khóa các node cây con phải

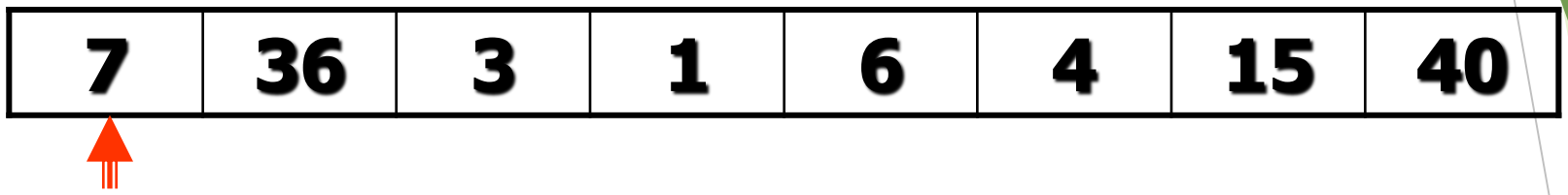
➔Giá trị khóa min nằm ở nút trái nhất

➔Giá trị khóa max nằm ở nút phải nhất

Các thao tác cơ bản trên BST

1. Tạo cây
2. Duyệt cây
3. Cho biết các thông tin của cây
4. Tìm kiếm
5. Xoá node trên cây

Tạo cây

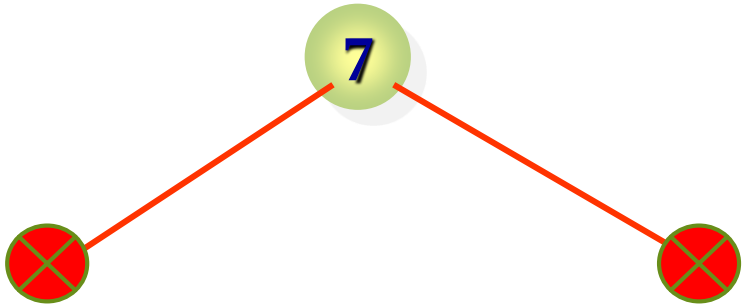


Xuất phát từ gốc

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

Tạo cây

7	36	3	1	6	4	15	40

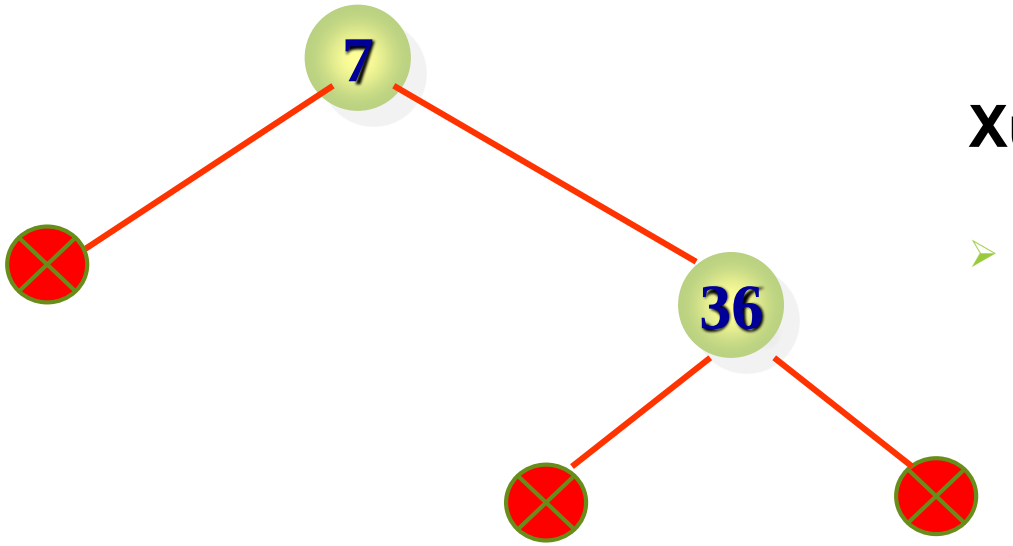


Xuất phát từ gốc

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

Tạo cây

7	36	3	1	6	4	15	40

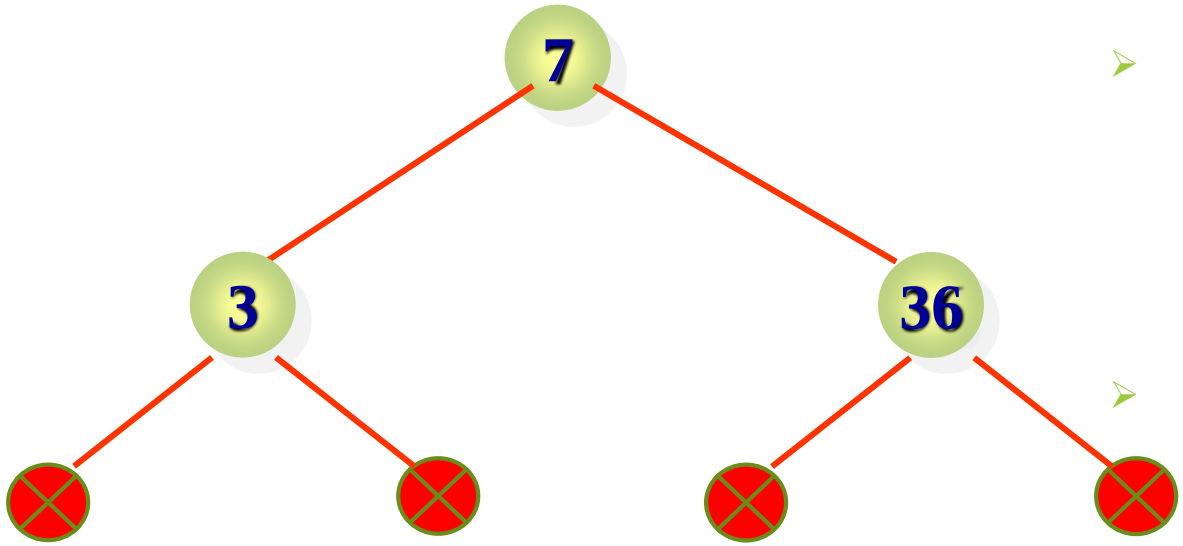


Xuất phát từ gốc

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

Tạo cây

7	36	3	1	6	4	15	40

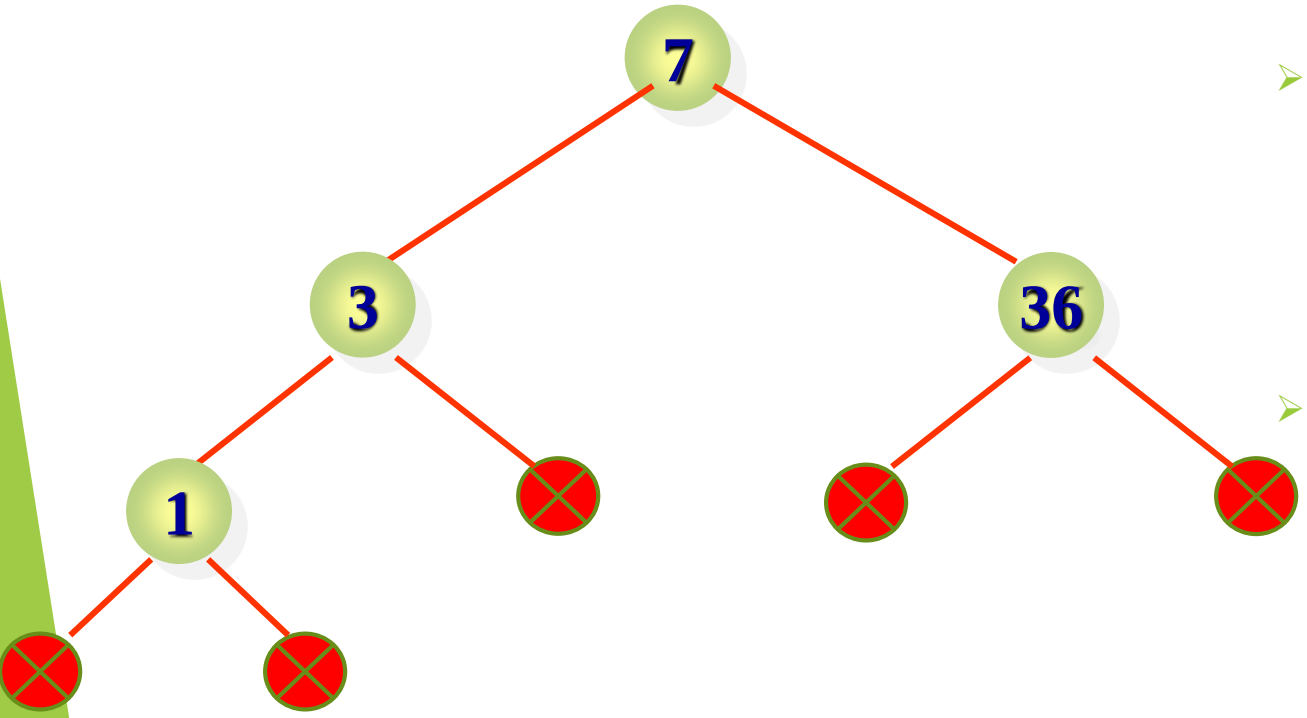


Xuất phát từ gốc

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

Tạo cây

7	36	3	1	6	4	15	40



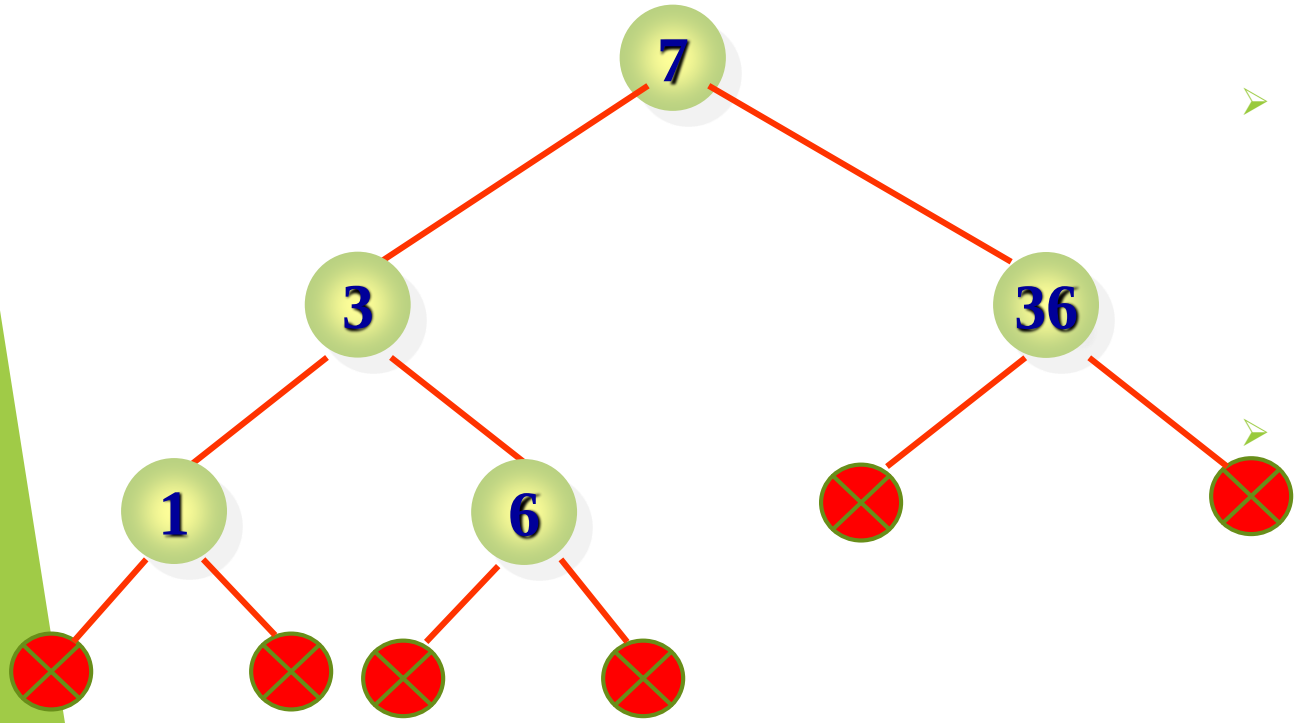
Xuất phát từ gốc

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

Tạo cây

7	36	3	1	6	4	15	40

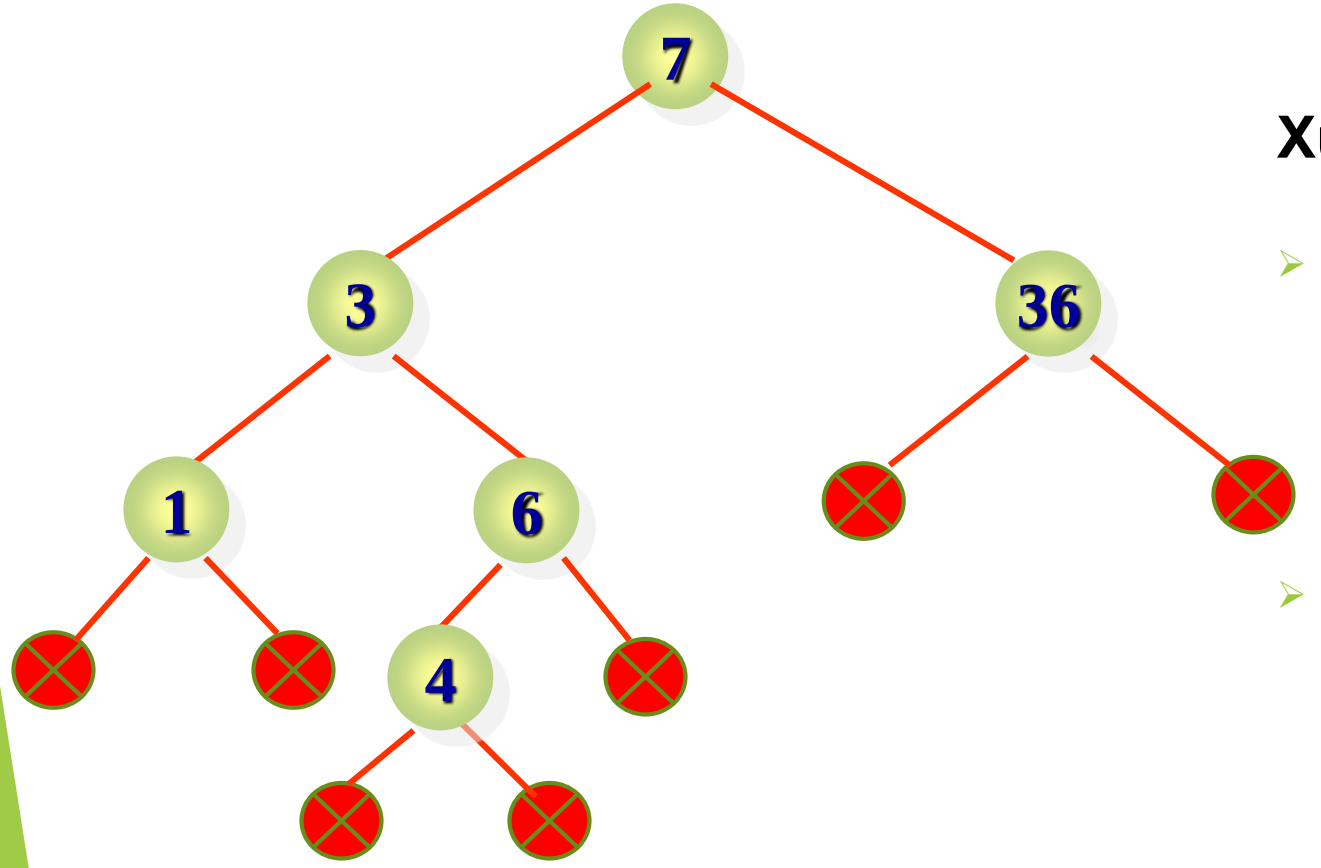
Xuất phát từ gốc



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

Tạo cây

7	36	3	1	6	4	15	40

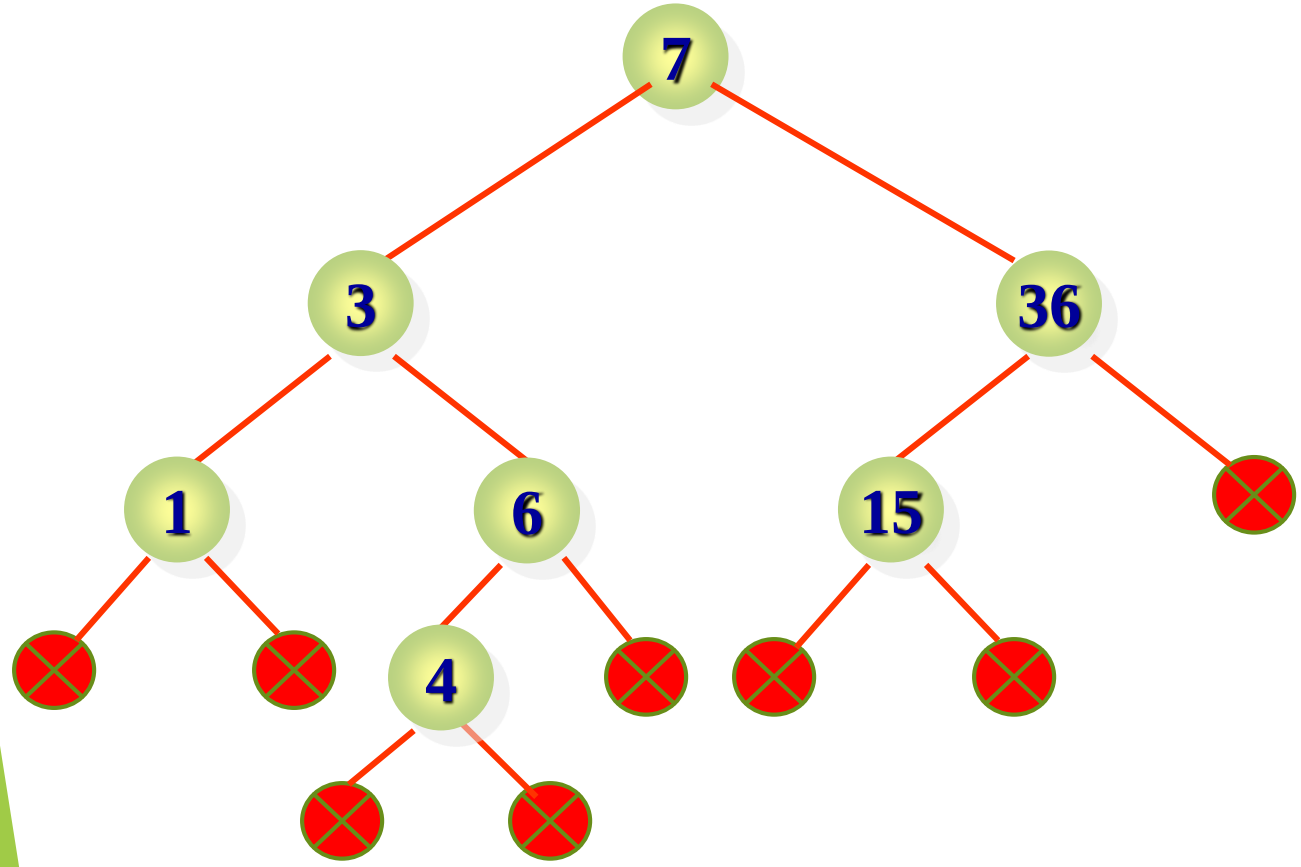


Xuất phát từ gốc

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

Tạo cây

7	36	3	1	6	4	15	40

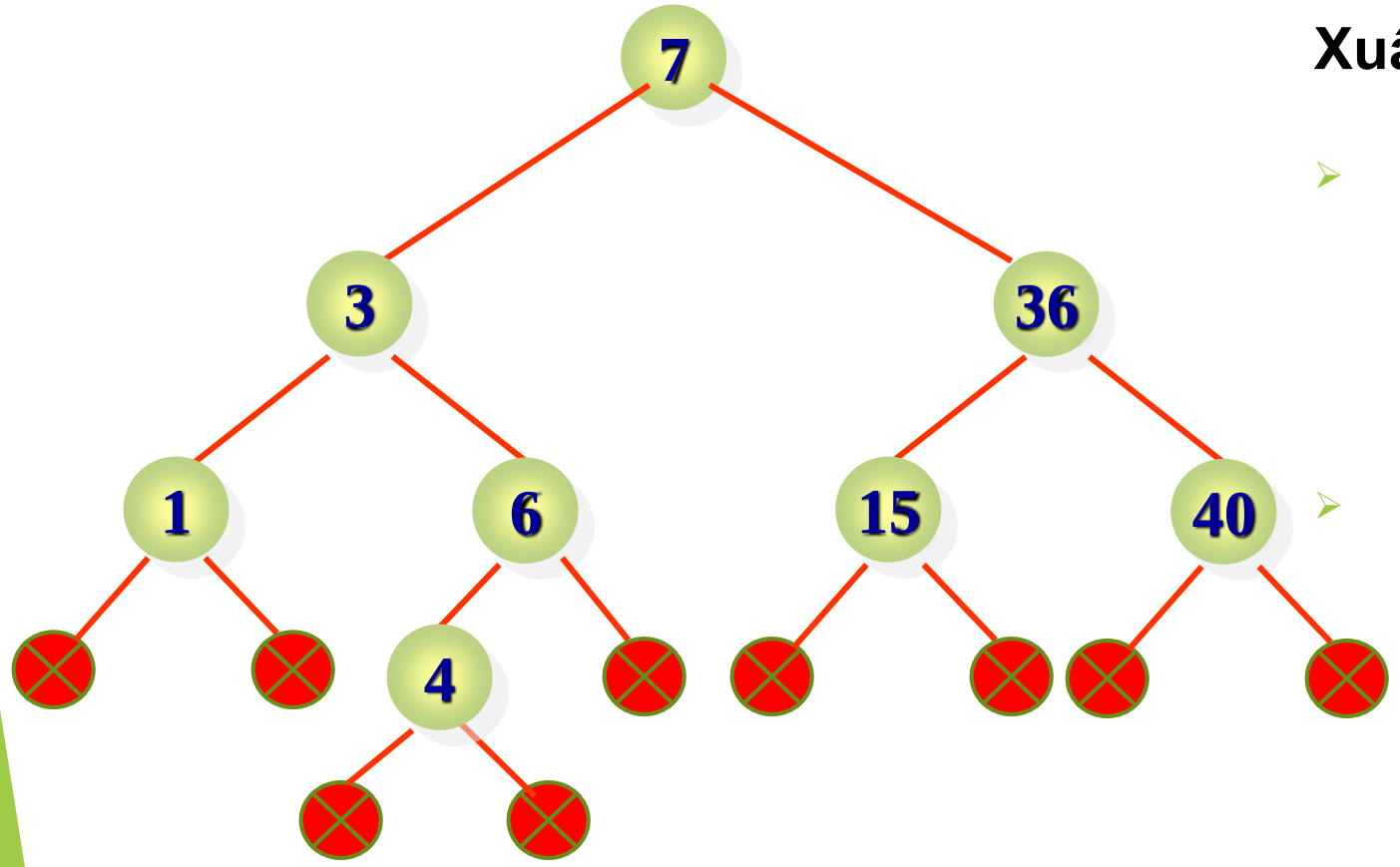


Xuất phát từ gốc

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

Tạo cây

7	36	3	1	6	4	15	40



Xuất phát từ gốc

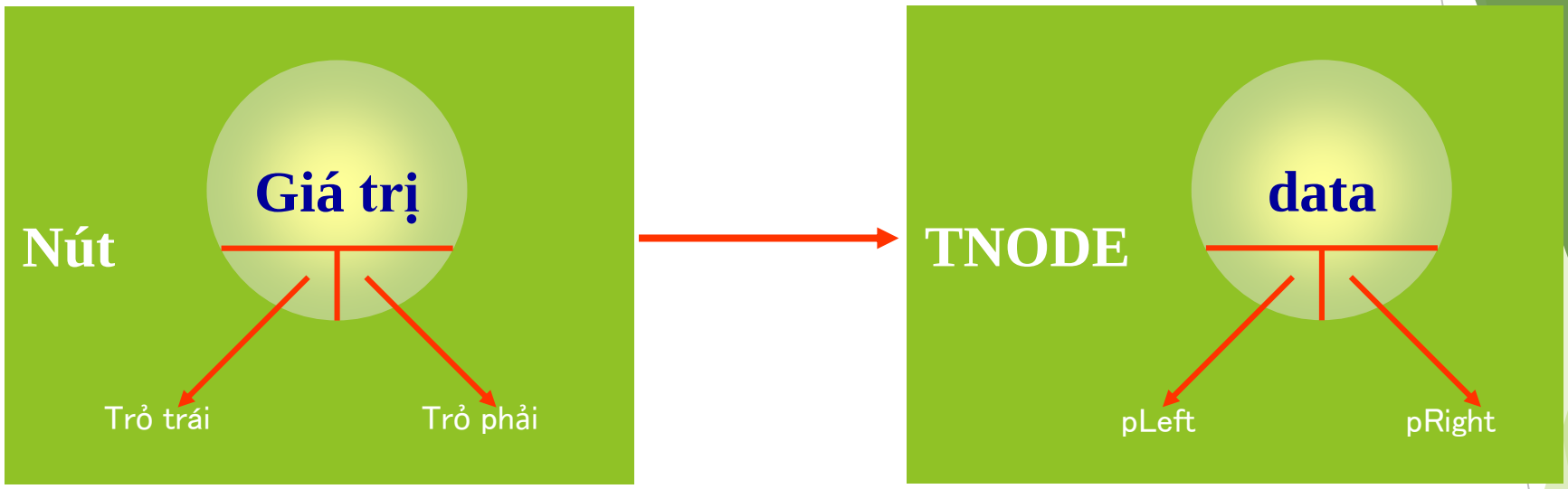
- Nếu node cần thêm nhỏ hơn node đang xét thì thêm về bên trái
- Ngược lại thì thêm về bên phải

Bài tập

Vẽ cây nhị phân tìm kiếm số nguyên theo thứ tự từ trái sang phải cho dãy số sau:

15; 30; 46; 3; 21; 7; 9; 10; 29; 34; 13; 5; 37; 18

Định nghĩa cấu trúc BST




```
typedef struct tagNode
{
   DataType data;
   struct tagNode *pLeft, *pRight;
}Node, *PtrNode;
```

VD: khai báo node chứa giá trị nguyên

```
typedef struct tagNode
{
   int data;
   struct tagNode *pLeft, *pRight;
}Node, *PtrNode;
```

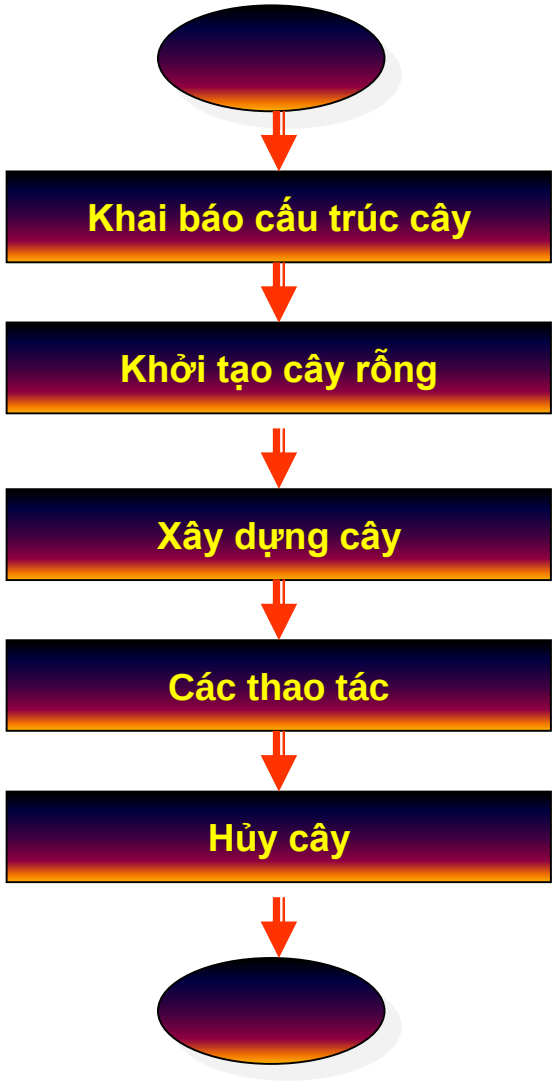
Các lưu ý khi cài đặt

Bước 1: Khai báo kiểu dữ liệu biểu diễn cây

Bước 2: Xây dựng hàm đưa dữ liệu (nhập) vào cây

Bước 3: Xây dựng các thao tác duyệt, tìm kiếm, huỷ, …

Cấu trúc chương trình



Giải thuật chèn 1 node vào BST

- **Input:** Cây BST t và Node x
- **Ouput:**
  - 0: Trùng giá trị (không chèn)
  - -1: Lỗi cấp phát bộ nhớ
  - 1: Chèn thành công x vào cây t
- B1: Nếu cây rỗng thì

  tree = x

  return 1
- B2: Thực hiện tìm kiếm giá trị x
  - Nếu giá trị x = giá trị của t thì return 0
  - Nếu giá trị x < giá trị của t thì thêm nút lá x bên trái của t
  - Ngược lại, thêm nút lá x bên phải của t

Hàm thêm một phần tử vào cây

```
int InsertNode (PtrNode &t, int x){
   if(t!=NULL){
            if(x==t->data)    return 0; //Có giá trị trùng
       else{
            if(x<t->data)            InsertNode(t->pLeft, x);
            else                     InsertNode(t->pRight, x);
       }
   }
   else{ //Chèn node vào cây
            t = new Node;
            if(t==NULL) return -1; //Không cấp phát được bộ nhớ
            t->data=x;
            t->pLeft=t->pRight=NULL;
            return 1; //Thêm thành công
   }
}
```

Bài tập

1. Viết hàm tạo cây nhị phân số nguyên từ các giá trị nhập vào từ bàn phím.

   Quá trình nhập cho đến khi gặp giá trị trùng hoặc hết bộ nhớ

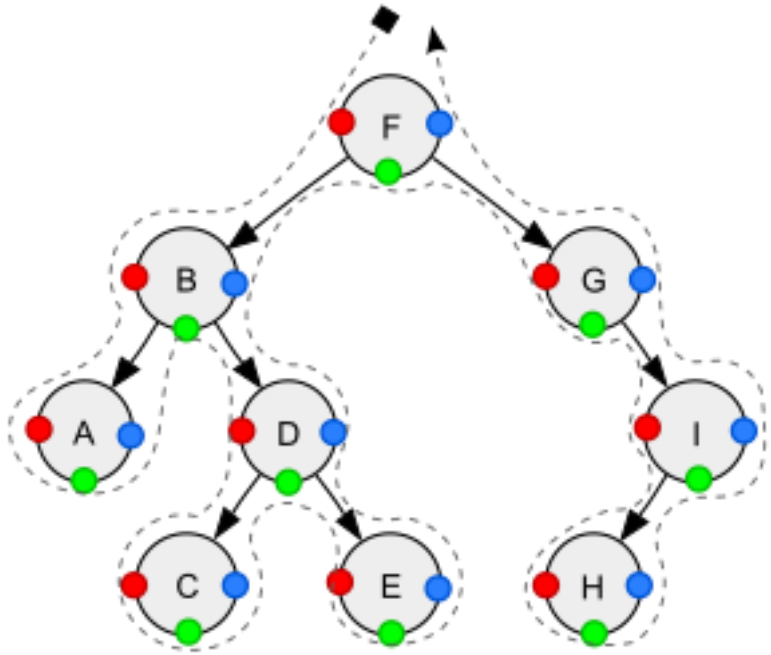
2. Viết hàm tạo cây nhị phân số nguyên từ dãy số nguyên cho trước

Duyệt cây

Thứ tự trước (**N**LR)

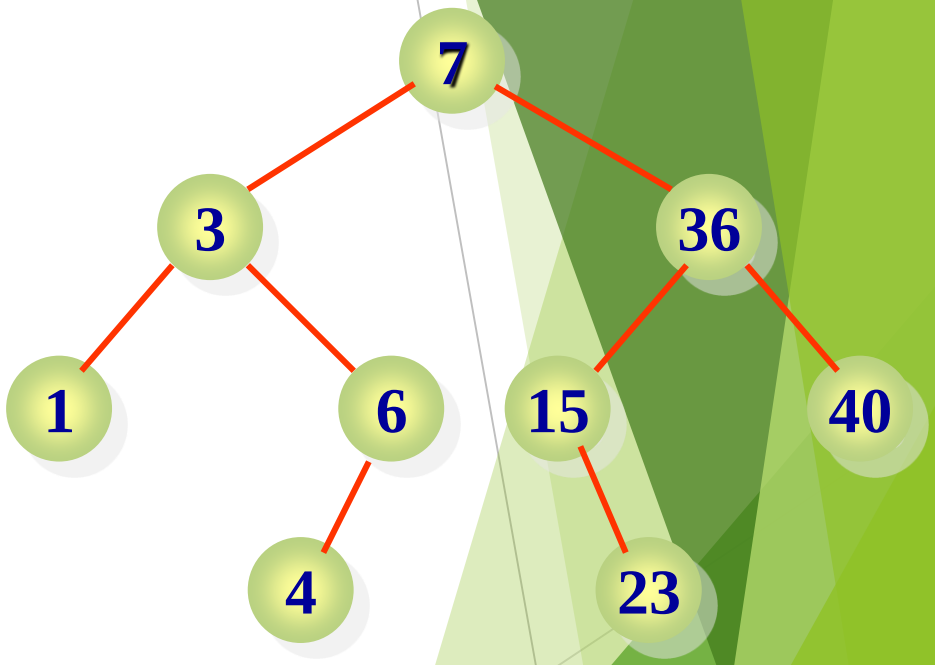
Thứ tự giữa (L**N**R)

Thứ tự sau (LR**N**)



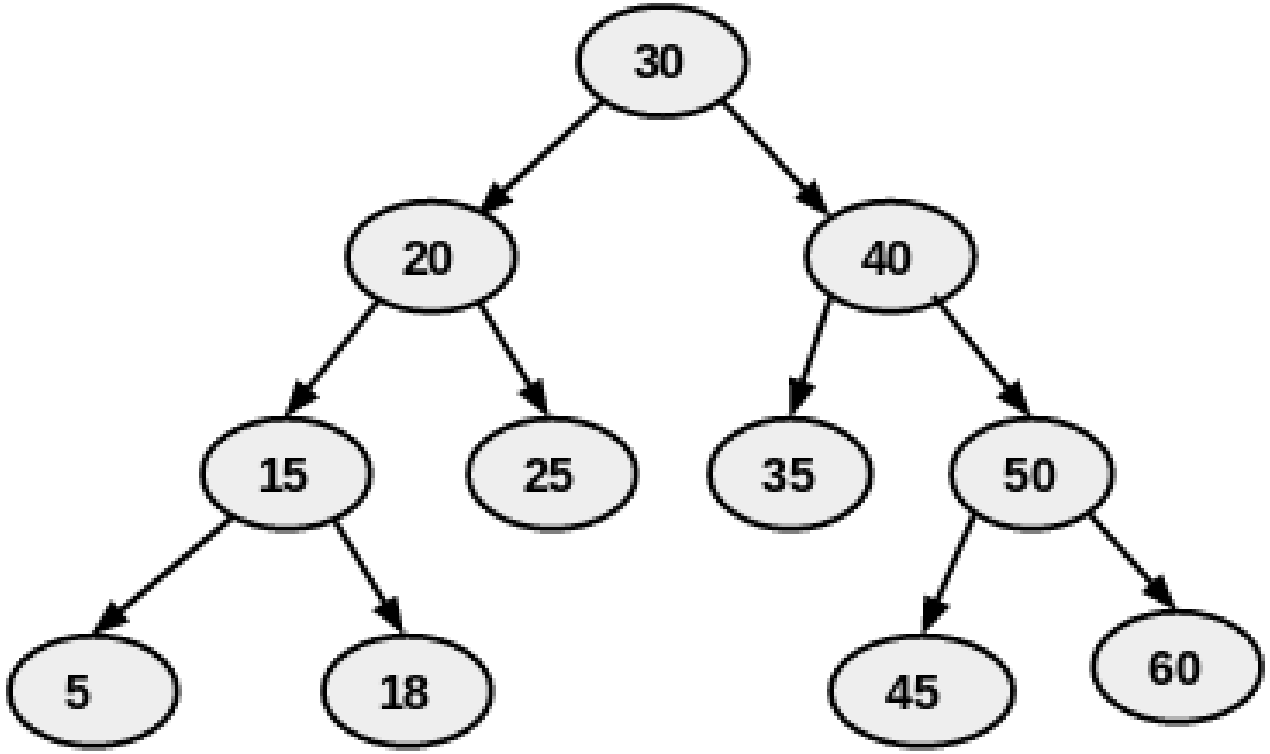
Duyệt cây

Bước	Kết quả duyệt theo thứ tự NLR								
1	7	L7	R7						
2		3	L3	R3	R7				
3			1	R3	R7				
4				6	L6	R7			
5					4	R7			
6						36	L36	R36	
7							15	R15	R36
8								23	R36
9									40
KQ	7	3	1	6	4	36	15	23	40



Bài tập

DUYỆT CÂY BST THEO THỨ TỰ TRƯỚC



Hàm duyệt NLR

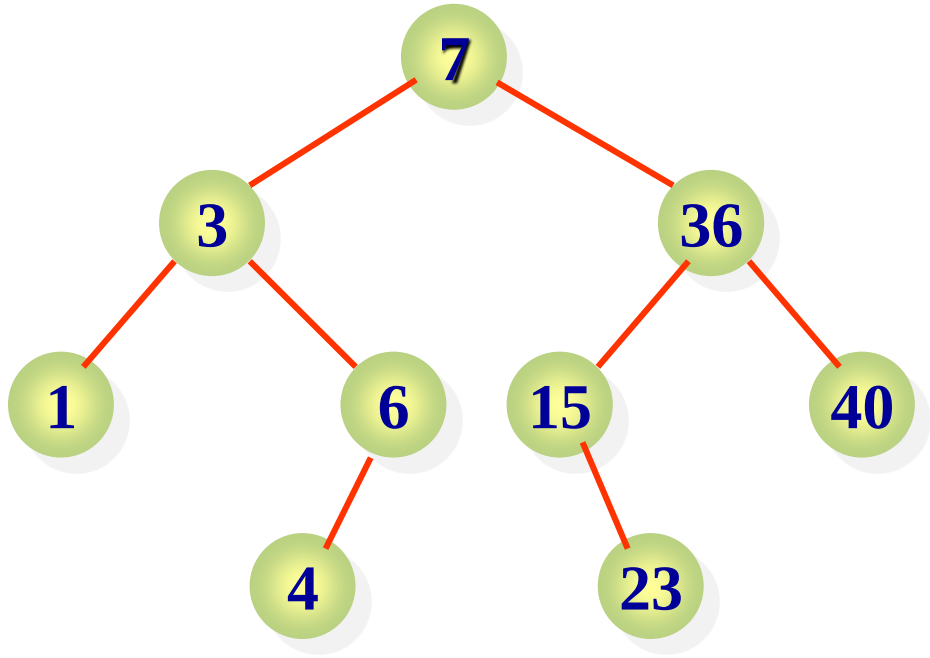
Tại node t đang xét, nếu khác rỗng thì

- In giá trị của t
- Duyệt cây con bên trái của t theo thứ tự NLR
- Duyệt cây con bên phải của t theo thứ tự NLR

```
void NLR (Tree t)
{
   if(t!=NULL)
   {
      printf("%d ", t->data);
      NLR(t->pLeft);
      NLR(t->pRight);
   }
}
```

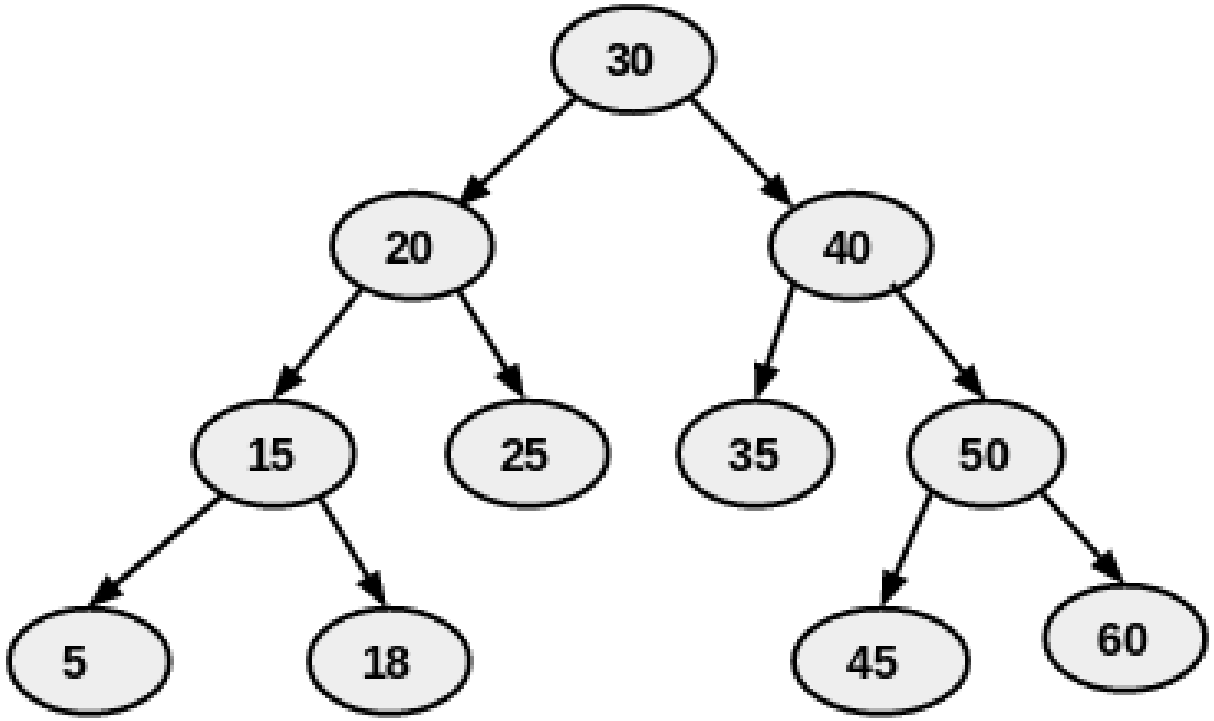
Duyệt LNR

Bước	Kết quả duyệt theo thứ tự LNR								
1	L7	7	R7						
2	L3	3	R3	7	R7				
3	1	3	R3	7	R7				
4		3	R3	7	R7				
5			L6	6	7	R7			
6			4	6	7	R7			
7				6	7	R7			
8					7	R7			
9						L36	36	R36	
10						15	R15	36	R36
11							23	36	R36
12								36	R36
13									40
KQ	1	3	4	6	7	15	23	36	40



Bài tập

DUYỆT CÂY BST THEO THỨ TỰ GIỮA



Hàm duyệt LNR

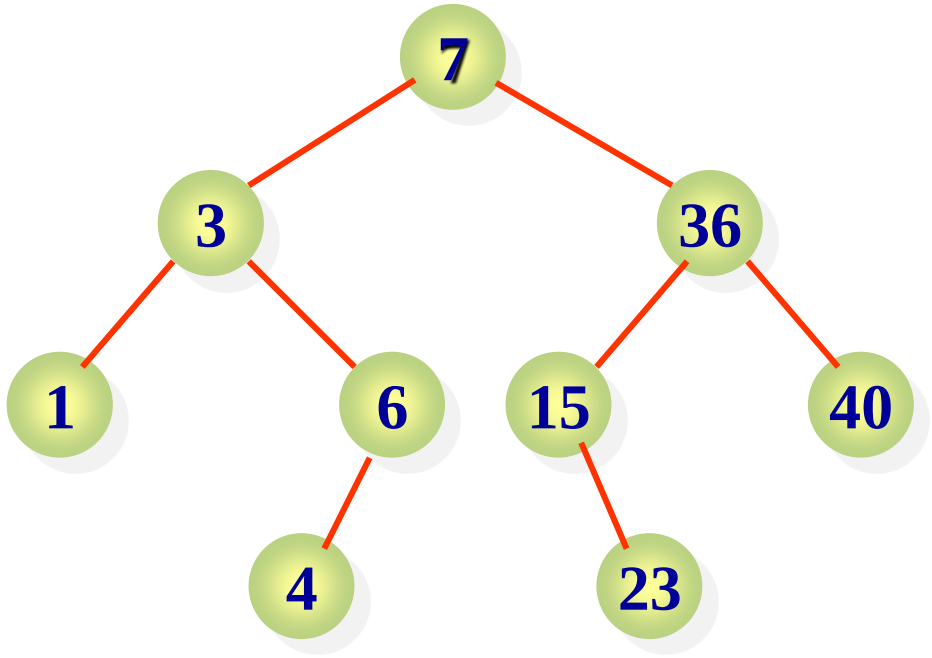
Tại node t đang xét, nếu khác rỗng thì

- Duyệt cây con bên trái của t theo thứ tự LNR
- In giá trị của t
- Duyệt cây con bên phải của t theo thứ tự LNR

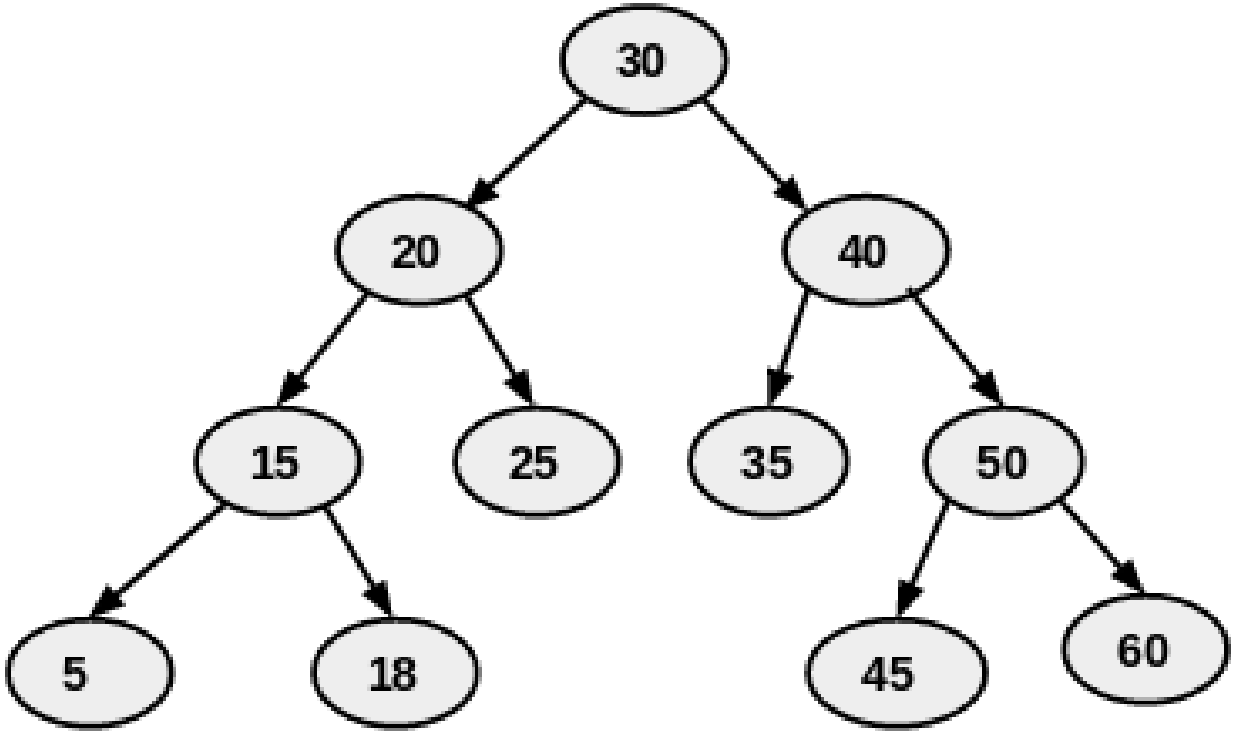
```
void LNR (Tree t)
{
   if(t!=NULL)
   {
       LNR(t->pLeft);
       printf("%d ", t-
   >data);
       LNR(t->pRight);
   }
}
```

Duyệt LRN

Bước	Kết quả duyệt theo thứ tự LRN								
1	L7	R7	7						
2	L3	R3	3	R7	7				
3	1	R3	3	R7	7				
4		L6	6	3	R7	7			
5		4	6	3	R7	7			
6			6	3	R7	7			
7				3	R7	7			
8					L36	R36	36	7	
9					R15	15	R36	36	7
10					23	15	R36	36	7
11						15	R36	36	7
12							40	36	7
13								36	7
14									7
KQ	1	4	6	3	23	15	40	36	7



Bài tập



DUYỆT CÂY BST THEO THỨ TỰ SAU

Hàm duyệt LRN

Tại node t đang xét, nếu khác rỗng thì

- Duyệt cây con bên trái của t theo thứ tự LRN
- Duyệt cây con bên phải của t theo thứ tự LRN
- In giá trị của t

```
void LRN (Tree t)
{
   if(t!=NULL)
   {
      LRN(t->pLeft);
      LRN(t->pRight);
      printf("%d ", t->data);
   }
}
```

Kiểm tra kết quả duyệt

Kiểm tra kết quả duyệt bằng cách xây dựng lại cây nhị phân từ kết quả duyệt

Tạo cây từ kết quả duyệt NLR

- Chọn giá trị đầu tiên làm node gốc
- Lần lượt đưa các giá trị còn lại từ trái sang phải vào cây theo nguyên tắc tạo cây

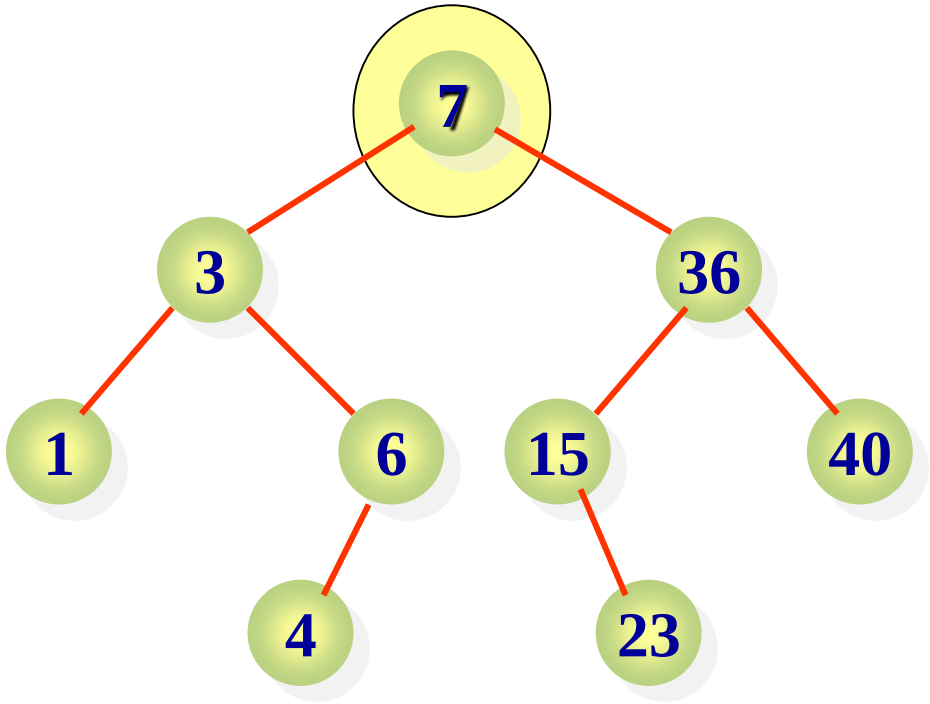
Tạo cây từ kết quả duyệt LRN

- Chọn giá trị cuối cùng làm node gốc
- Lần lượt đưa các giá trị còn lại từ phải sang trái vào cây theo nguyên tắc tạo cây

Các thao tác tìm kiếm

1. Tìm x
2. Tìm min
3. Tìm min của cây con bên phải
4. Tìm max
5. Tìm max của cây con bên trái

Ví dụ tìm x = 23



Bài tập

Cho cây nhị phân tìm kiếm, mỗi node có giá trị nguyên, hãy định nghĩa các hàm sau:

- Tìm node có giá trị x
- Tìm node có giá trị lớn nhất
- Tìm node có giá trị nhỏ nhất của cây con phải

Bài tập

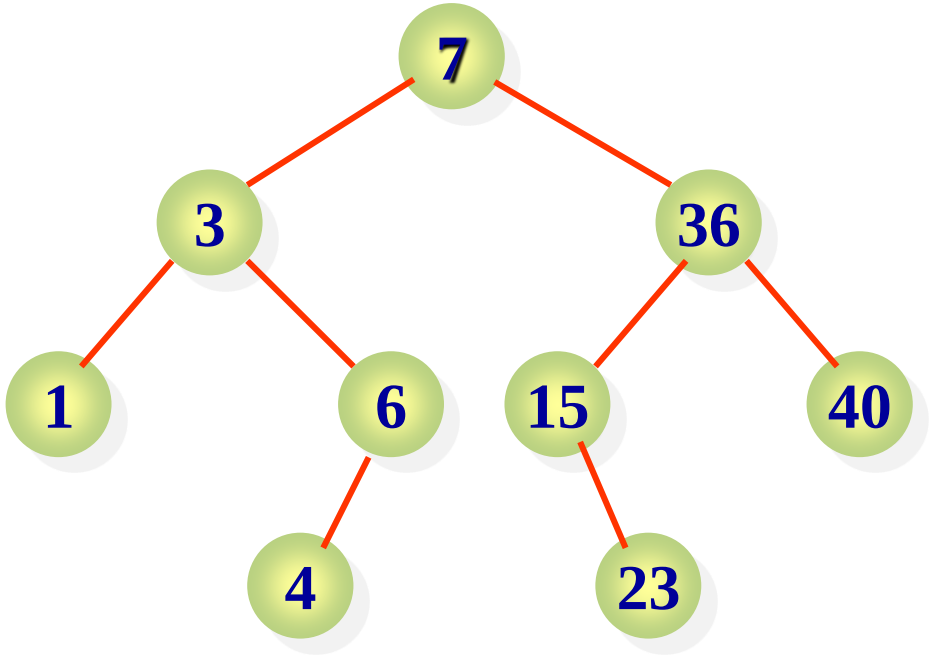
Cho cây nhị phân tìm kiếm, mỗi node có giá trị nguyên, hãy định nghĩa các hàm sau:

1. In ra các node có giá trị chẵn
2. Đếm số node của cây
3. Tính độ cao của cây

Bài tập

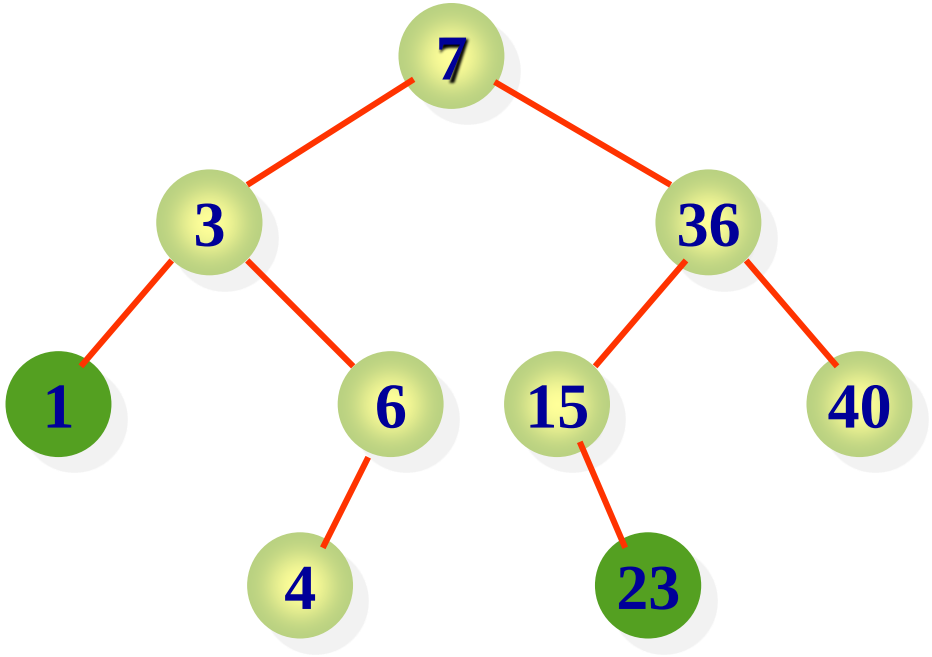
5. Đếm số node lá (node bậc 0)
6. Đếm số node có 1 cây con (node bậc 1)
7. Đếm số node chỉ có 1 cây con phải
8. Đếm số node có 1 cây con trái
9. Đếm số node 2 cây con (node bậc 2)
10. In các node trên từng mức của cây
11. Cho biết độ dài đường đi từ gốc đến node x

Xóa node trên cây



1. Node lá
2. Node có 1 cây con
3. Node có 2 cây con

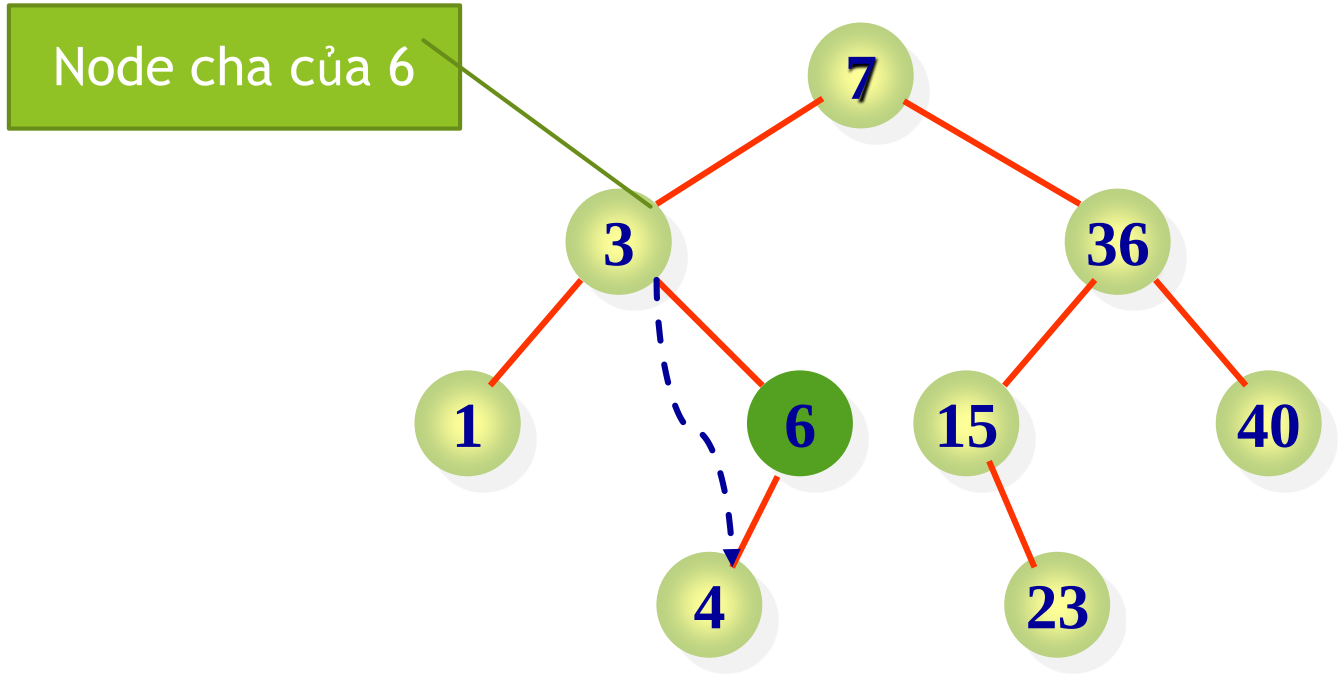
Xóa node lá



Xóa 1

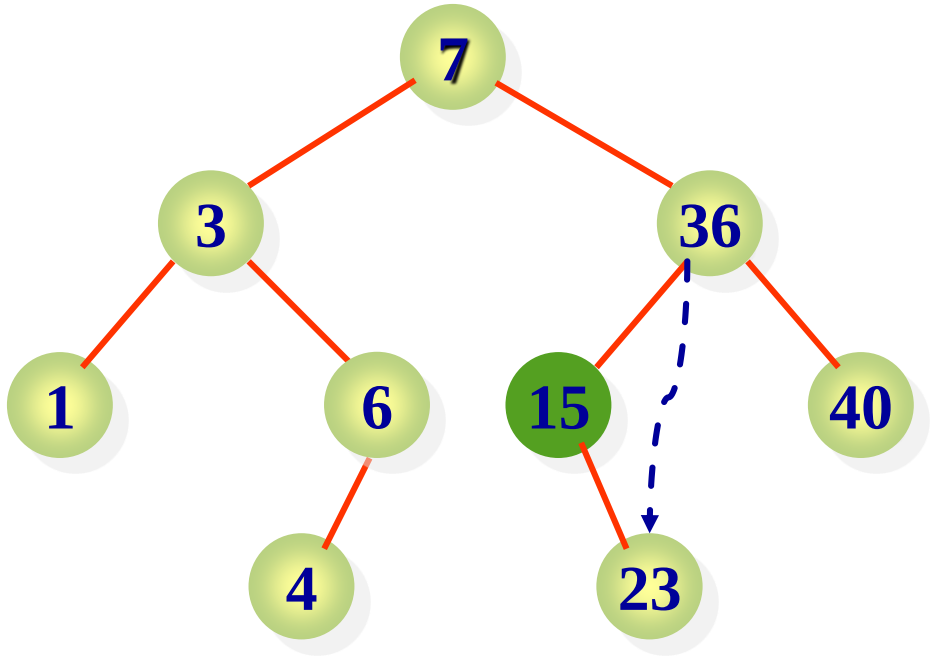
Xóa 23

Xóa node 1 cây con



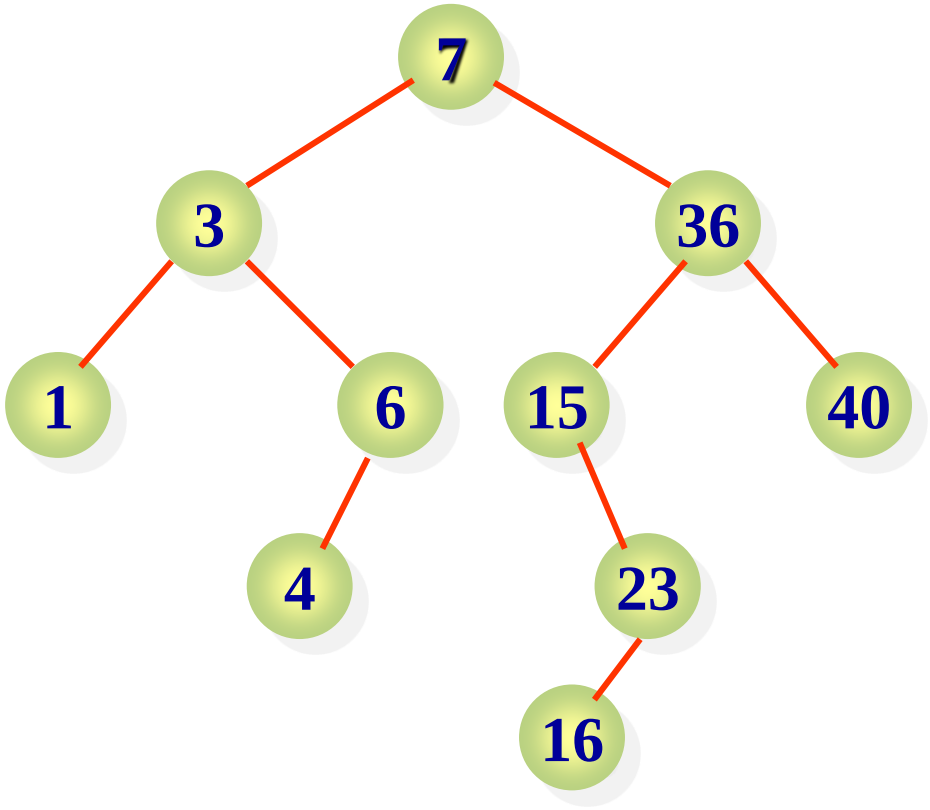
Xóa 6

Xóa node 1 cây con



Xóa 15

Xóa node có 2 cây con



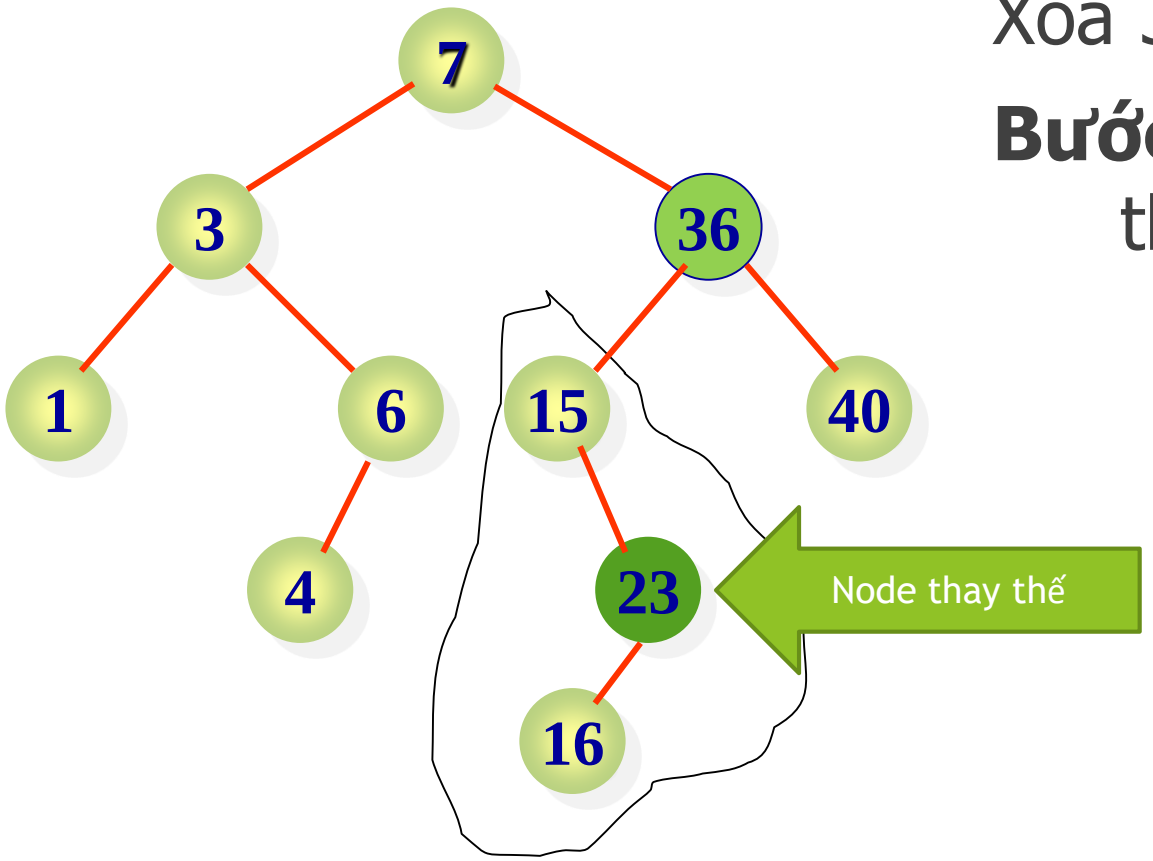
Bước 1: Tìm node thay thế

- ***Cách 1:*** *Tìm node trái nhất của cây con phải*
- ***Cách 2:*** *Tìm node phải nhất của cây con trái*

Bước 2: Thay giá trị của node thay thế vào node cần xóa

Bước 3: Xóa node thay thế

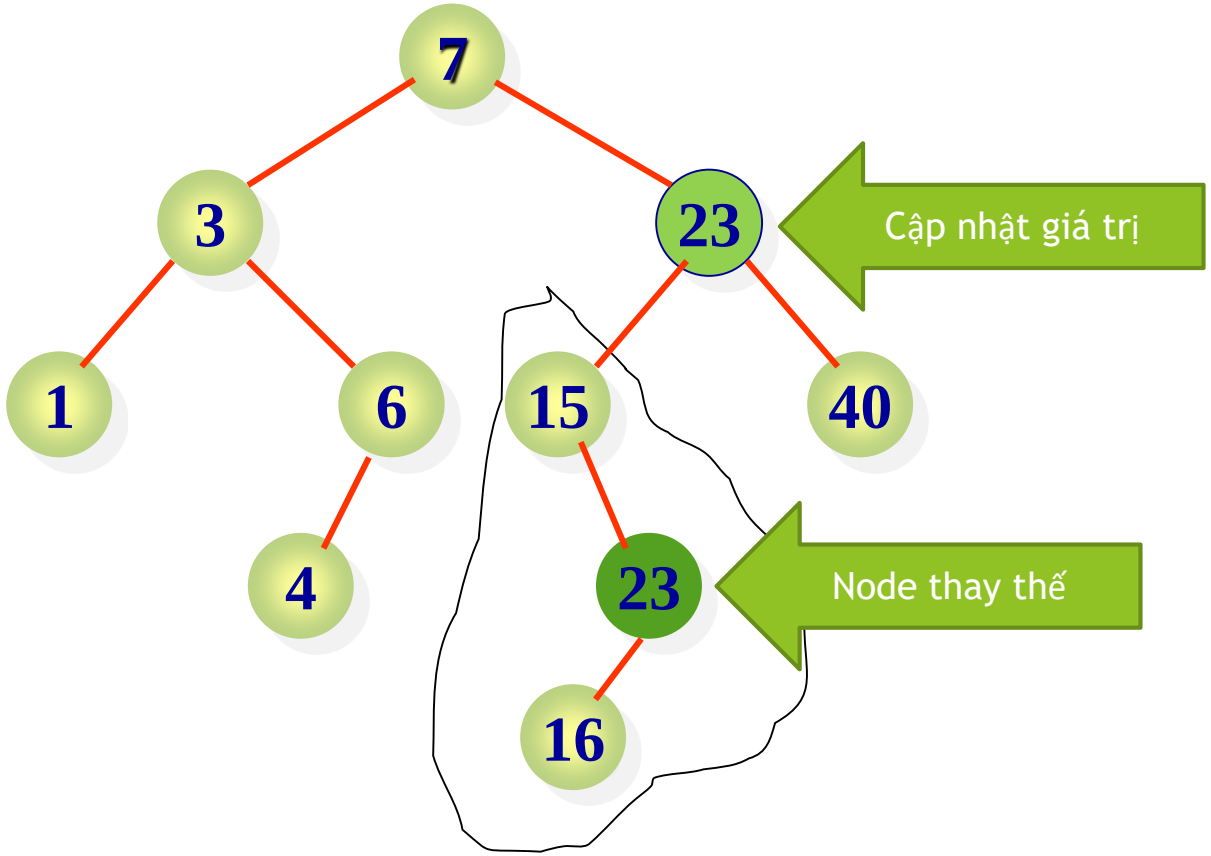
Xóa node 2 cây con



Xóa 36

Bước 1: Tìm node thay thế

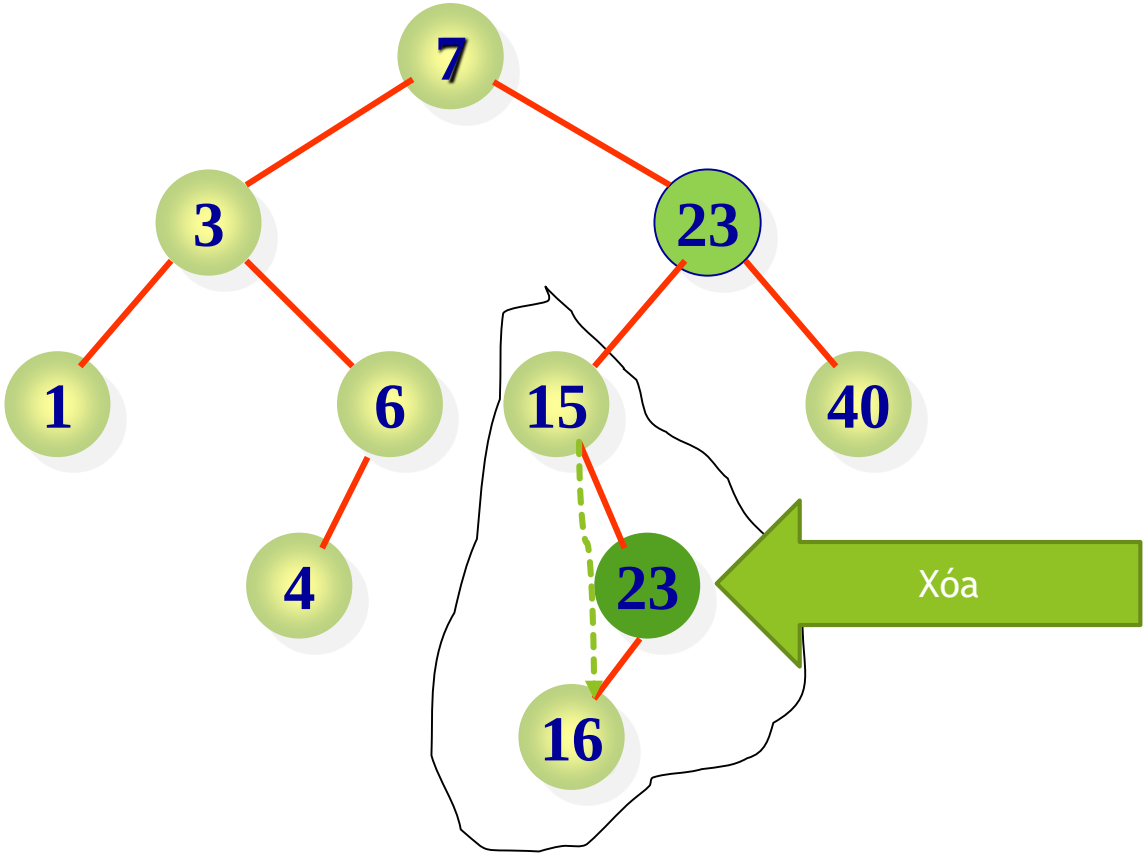
Xóa node 2 cây con



Xóa 36

Bước 2: Thay giá trị node thay thế cho node cần xóa

Xóa node 2 cây con

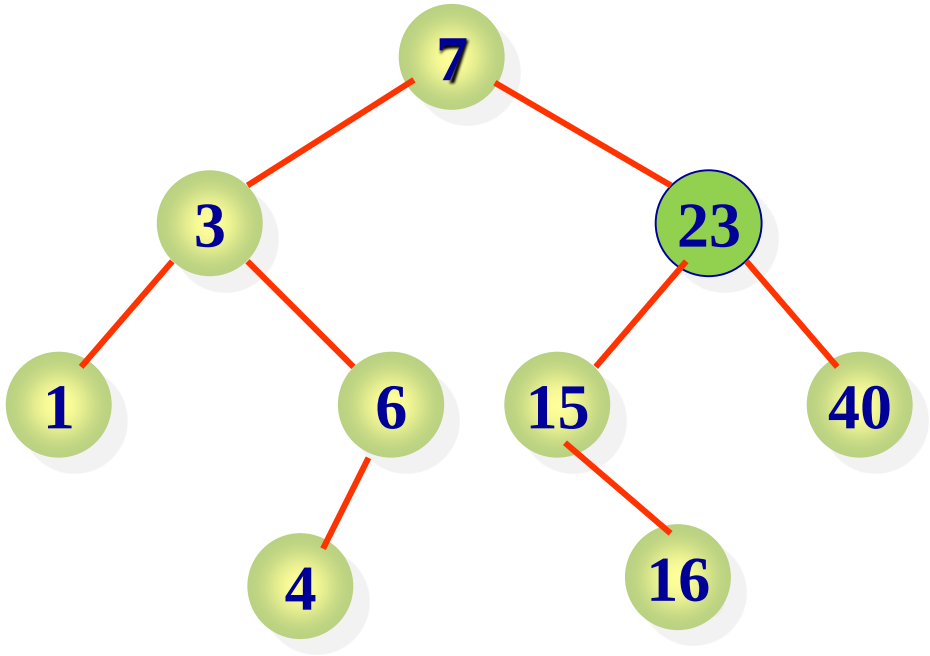


Xóa 36

Bước 3: Xóa node thay thế

Xóa node 2 cây con

Kết quả: xóa 36



Bài tập

Cho dãy số theo thứ tự nhập từ trái sang phải:

20, 15, 35, 30, 11, 13, 17, 36, 47, 16, 38, 28, 14

- Vẽ cây nhị phân tìm kiếm cho dãy số trên
- Trình bày từng bước và vẽ lại cây sau khi lần lượt xoá các nút: **11** và **35**

Bài tập

- Viết hàm xóa node có giá trị x trong cây nhị phân số nguyên

Bài tập

```
void Remove(Tree & t, int x)
{  if (t != NULL)
   {  if (x < t->key) Remove(t->pLeft, x);
      else if (x > t->key) Remove(t->pRight, x);
           else
           {   TNode * pHuy = t;
               if (t->pLeft == NULL)
                  t = t->pRight;
               else if (t->pRight == NULL)
                         t = t->pLeft;
                    else  SearchStandFor(pHuy, t->pRight);

               delete pHuy;
            }
   }
}
```

Bài tập

```
void SearchStandFor(Tree &pHuy, Tree &pTT)
{
   if(pTM->pLeft!=NULL)
      SearchStandFor(pHuy, pTT->pLeft);
   else
   {
      pHuy->key=pTT->key;
      pHuy=pTT;
      pTT=pTT->pRight;
   }
}
```

Đánh giá

- Thao tác tìm kiếm, thêm mới, xóa có độ phức tạp trung bình O(h) (h là chiều cao của cây)
- Trường hợp tốt nhất, cây có n nút sẽ có độ cao h = log2(n) → Chi phí tìm kiếm sẽ tương đương tìm kiếm nhị phân trên mảng có thứ tự
- Trường hợp xấu nhất, cây có thể bị suy biến thành 1 DSLK. Lúc đó các thao tác trên sẽ có độ phức tạp O(n)

→ Vì vậy cần có cải tiến cấu trúc của CNPTK để đạt được chi phí cho các thao tác là $\log_2(n)$