

## Chương 3 CÂY NHỊ PHÂN TÌM KIẾM

1

### Mở đầu

**Kiến thức** cần thiết khi tìm hiểu về CÂY NHỊ PHÂN TÌM KIẾM:



- Các CTDL cơ bản, các phương pháp cơ bản về xếp thứ tự và tìm kiếm trên LIST.
- Kiểu dữ liệu cơ bản, dữ liệu lưu trữ trong máy tính.
- Các kiến thức về cơ sở lập trình & kỹ thuật lập trình.

**Kỹ năng** cần có:

- Có thể sử dụng Visual Studio 2010
- Có thể lập trình C++

2

### Mục tiêu dạy học

-  Cung cấp các khái niệm về cây, kiến thức cấu trúc cây nhị phân, các thuật toán trên cây nhị phân tìm kiếm.
-  Rèn luyện và nâng cao kỹ năng lập trình, ứng dụng cấu trúc dữ liệu cây nhị phân và các thuật toán trên cây nhị phân tìm kiếm giải quyết các bài toán ứng dụng.

3

### Nội dung chính

#### 3.1 Các khái niệm

- Cây
- Cây nhị phân

#### 3.2 Cây nhị phân tìm kiếm

#### 3.3 Tổng kết chương

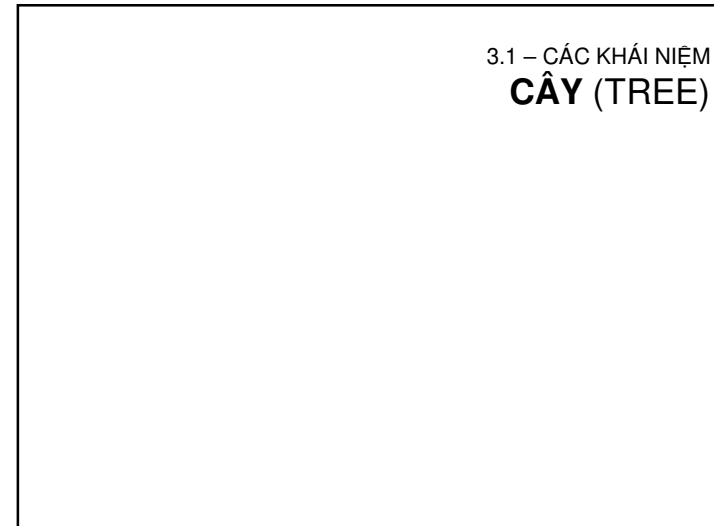
#### 3.4 Bài tập chương 3

Tài liệu tham khảo

4



5

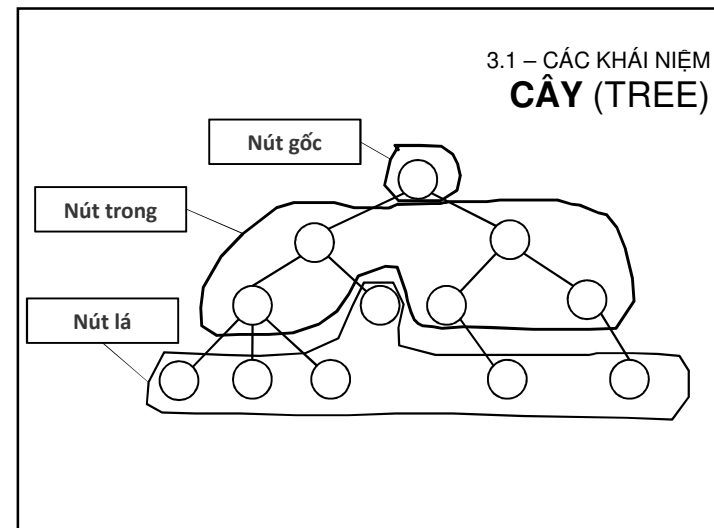


6

### 3.1 – CÁC KHÁI NIỆM CÂY (TREE)

- Cây** là một tập hợp các phần tử hay còn gọi là các node (nút) có quan hệ cha – con, phần tử cha sẽ lưu trữ (quản lý) địa chỉ bộ nhớ của phần tử con.
- Node gốc (root) là node duy nhất trong cây không có nút cha. Biến root sẽ lưu địa chỉ của node gốc.
- Mỗi node trong cây (trừ node gốc), có *duy nhất* một node cha. Một node cha có thể có nhiều con.
- Node là node không có phần tử con

7



8

### 3.1 – CÁC KHÁI NIỆM **CÂY (TREE)**

#### **BẬC CỦA CÂY**

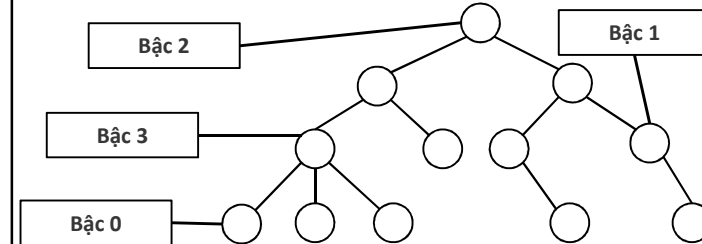
- 📖 Bậc của node là số node con của node đó. Node lá có bậc 0
- 📖 Bậc của cây là bậc cao nhất của node trong cây. Một cây có bậc là n được gọi là cây bậc n.

9

### 3.1 – CÁC KHÁI NIỆM **CÂY (TREE)**

#### **BẬC CỦA CÂY**

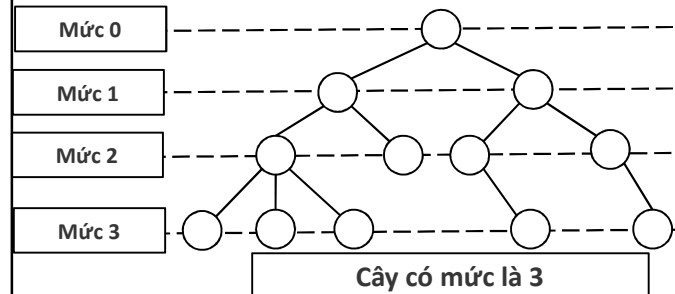
Cây có bậc 3



10

### 3.1 – CÁC KHÁI NIỆM **CÂY (TREE)**

#### **MỨC TRONG CÂY**



11

### 3.1 – CÁC KHÁI NIỆM **CÂY NHỊ PHÂN (BINARY TREE)**

#### **ĐỊNH NGHĨA CÂY NHỊ PHÂN**

- 📖 Cây nhị phân là một cây, trong đó mỗi phần tử trong cây chỉ có tối đa 2 phần tử con (phần tử con bên trái, phần tử con bên phải)

12

3.1 – CÁC KHÁI NIỆM

## CÂY NHỊ PHÂN (BINARY TREE)

### MINH HỌA CÂY NHỊ PHÂN

Mỗi phần tử trong cây nhị phân chứa 3 thành phần: thành phần info để lưu trữ giá trị phần tử, thành phần left để lưu địa chỉ của phần tử con bên trái, thành phần right để lưu trữ địa chỉ của phần tử con bên phải

13

3.1 – CÁC KHÁI NIỆM

## CÂY NHỊ PHÂN (BINARY TREE)

### CẤU TRÚC MỘT NÚT TRÊN CÂY NHỊ PHÂN

📖 Mỗi phần tử trong cây nhị phân chứa **3 thành phần**:

- info: phần tử chứa thông tin / giá trị của nút (node)
- left: phần lưu trữ địa chỉ của nút bên trái (hay cây con trái)
- right: phần lưu trữ địa chỉ của nút bên phải (hay cây con phải)

|      |      |       |
|------|------|-------|
| left | info | right |
|------|------|-------|

14

3.1 – CÁC KHÁI NIỆM

## CÂY NHỊ PHÂN (BINARY TREE)

### KHAI BÁO CẤU TRÚC CÂY

```
// cấu trúc 1 node
struct Node
{
    int info; // lưu trữ giá trị của phần tử, giá trị khóa của node
    Node *left; // left lưu địa chỉ phần tử bên trái (cây con trái)
    Node *right; // right lưu trữ địa chỉ phần tử bên phải (cây con phải)
};
Node *root;
```

|      |      |       |
|------|------|-------|
| left | info | right |
|------|------|-------|

15

3.1 – CÁC KHÁI NIỆM

## CÂY NHỊ PHÂN (BINARY TREE)

### KHỞI TẠO CÂY RỖNG

```
// cấu trúc 1 node
void tree_empty()
{
    root = NULL;
}
// root là biến toàn cục;
```

16



## 3.2 CÂY NHỊ PHÂN TÌM KIẾM

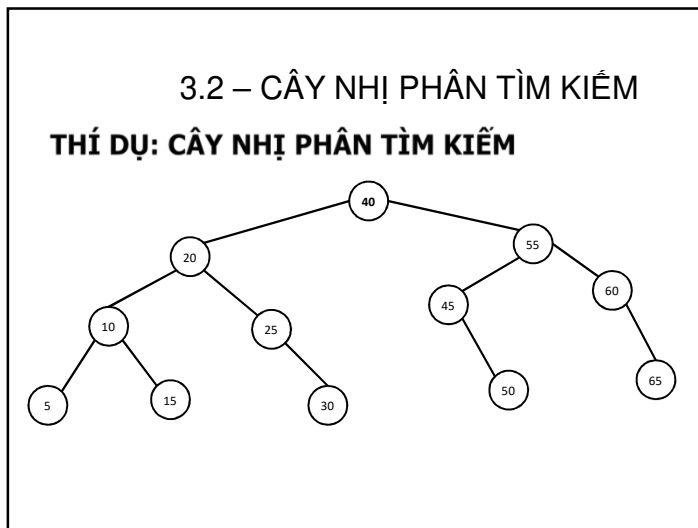
17

## 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

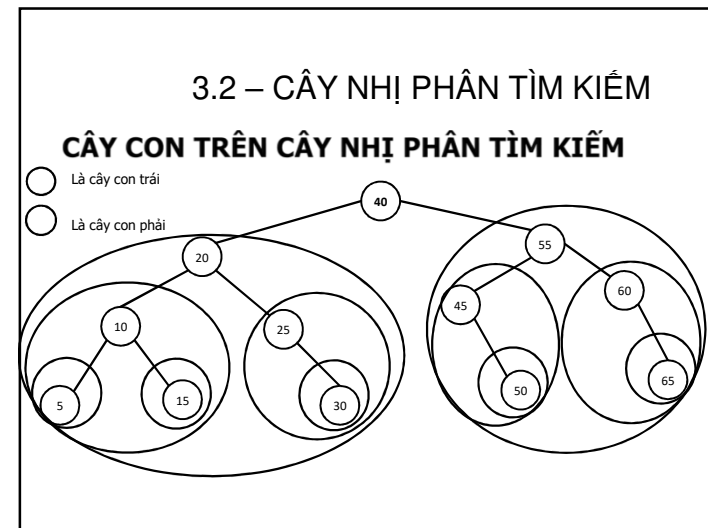
### CÂY NHỊ PHÂN TÌM KIẾM

📖 *Cây nhị phân tìm kiếm là cây nhị phân mà giá trị (khóa) của phần tử bên trái của một node có giá trị nhỏ hơn giá trị (khóa) của node, giá trị (khóa) của các phần tử bên phải của một node thì lớn hơn giá trị (khóa) của node đó*

18



19



20

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

#### CÁC THAO TÁC TRÊN CÂY NHỊ PHÂN TÌM KIẾM

- 📖 Tìm một node trên cây nhị phân tìm kiếm
- 📖 Thêm một node mới vào cây
- 📖 Duyệt cây nhị phân tìm kiếm
- 📖 Xóa một node trên cây

21

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

#### TÌM KIẾM TRÊN CÂY NHỊ PHÂN TK

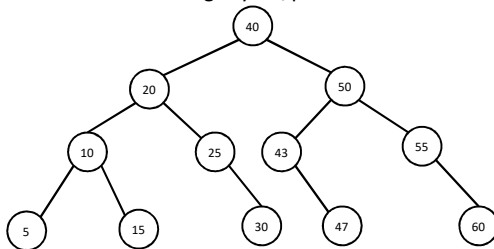
Do tính chất của cây nhị phân tìm kiếm, một node trong cây sẽ có giá trị lớn hơn các node bên trái của nó và có giá trị nhỏ hơn giá trị của các node bên phải của nó. Do đó, để tìm một node gốc với  $x$ . Nếu giá trị node gốc bằng  $x$ , tìm thấy và kết thúc. Nếu giá trị node gốc lớn hơn  $x$ , thì so sánh  $x$  với node con bên trái của node gốc (nếu có). Nếu giá trị node gốc nhỏ hơn  $x$ , so sánh  $x$  với node con bên phải node gốc (nếu có). Thực hiện tuần tự như trên cho đến khi tìm thấy  $x$ , hoặc đi đến NULL.

22

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

#### THÍ DỤ 1:

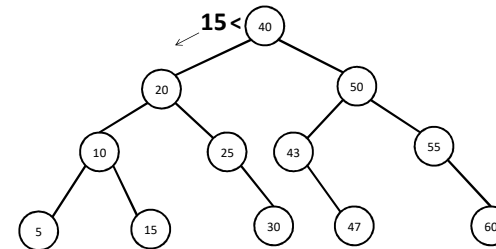
Ta tìm  $x = 15$  trong cây nhị phân tìm kiếm sau



23

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

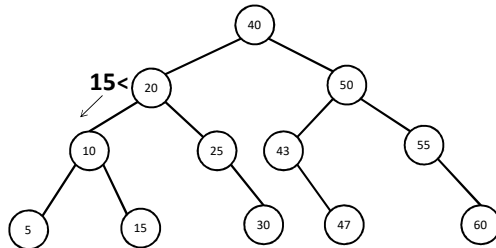
Bắt đầu so sánh  $x = 15$  với giá trị nút gốc (là 40). Giá trị  $15 < 40$ , xét node con bên trái của node gốc



24

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

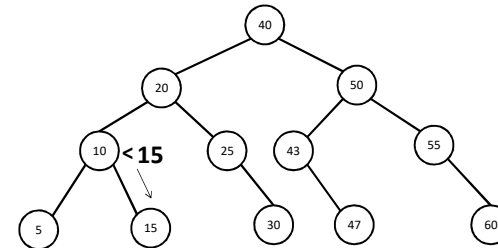
So sánh  $x = 15$  với giá trị node có giá trị 20. Giá trị  $15 < 20$ , xét node con bên trái của node có giá trị 20



25

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

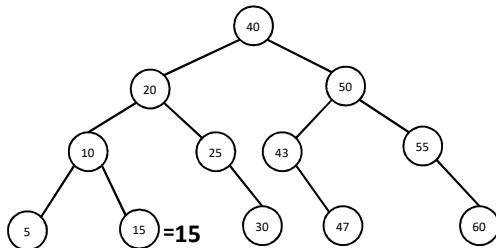
So sánh  $x = 15$  với giá trị node có giá trị 10. Giá trị  $15 > 10$ , xét node con bên phải của node có giá trị 10



26

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

So sánh  $x = 15$  với giá trị node có giá trị 15. Giá trị  $x$  bằng giá trị node đang xét, ta kết luận tìm thấy và kết thúc

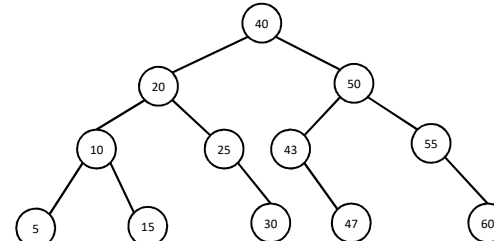


27

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

#### THÍ DỤ 2:

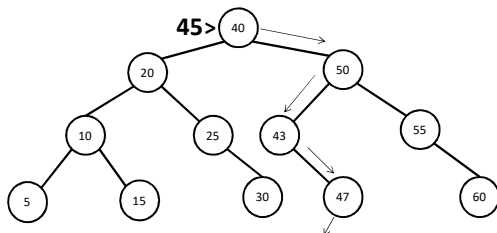
Ta tìm  $x = 45$  trong cây nhị phân tìm kiếm sau:



28

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

Bắt đầu so sánh  $X = 45$  với giá trị nút gốc (là 40). Giá trị  $45 > 40$ , xét node con bên phải của node gốc (có giá trị 50).  $X = 45 < 50$ , xét node con bên trái node 50 (có giá trị 43).  $X = 45 > 43$ , xét node con bên phải node 43 (có giá trị 47).  $X = 45 < 47$ , xét node con bên trái node 47 (có giá trị NULL). Kết luận không tìm thấy và thuật toán kết thúc.



29

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

#### THUẬT TOÁN TÌM KIẾM

**Bước 1:** Gán địa chỉ node gốc cho biến p (bắt đầu node gốc)

```
Node *p = root;
```

**Bước 2:** Tại node có địa chỉ do p quản lý, so sánh giá trị của node và giá trị phần tử x:

```
if (p->info == x)
    return p;
else if (p->info > x)
    p = p->left;
else
    p = p->right;
```

Nếu giá trị node = x, tìm thấy một node có giá trị x. Thuật toán kết thúc

Nếu giá trị node > x, chuyển sang xét node con bên trái node đang xét.

Nếu giá trị node < x, chuyển sang xét node con bên phải node đang xét.

**Bước 3:** Nếu giá trị node đang xét khác NULL, lặp lại **bước 2**. Ngược lại kết thúc. Không tìm thấy.

30

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

#### CHƯƠNG TRÌNH

```
Node *search(Node *p, int x)
{
    while (p != NULL)
    {
        if (p->info == x)
            return p;
        else
        {
            if (p->info > x)
                p = p->left;
            else // p->info < x
                p = p->right;
        }
    }
    return NULL;
}
```

```
Node *p = root;

if (p->info == x)
    return p;
else if (p->info > x)
    p = p->left;
else
    p = p->right;
```

31

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

#### CHƯƠNG TRÌNH ĐỆ QUY

```
Node *search(Node *p, int x)
```

**Gọi thực thi:** search(root, x);  
// chú ý: x được nhập trước khi thủ tục search(root, x) được gọi

```
{
    if (p != NULL)
    {
        if (p->info == x)
            return p;
        else
        {
            if (x > p->info)
                return search(p->right, x);
            else
                return search(p->left, x);
        }
    }
    return NULL;
}
```

32



### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

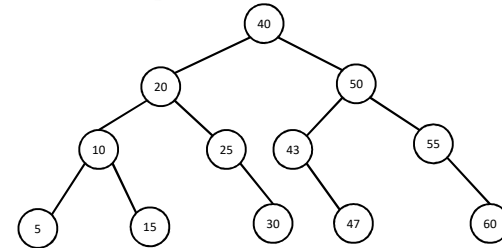
#### THÊM MỘT NÚT MỚI VÀO CÂY

- Để thêm *một node mới* (có giá trị x) vào cây nhị phân tìm kiếm, đầu tiên tìm kiếm node có giá trị x trong cây (giá trị node là khóa của node trong cây).
- Nếu tìm thấy đã có một node có giá trị x (giá trị khóa) trong cây, không thêm và kết thúc thuật toán.
- Nếu không tìm thấy node có giá trị x trong cây, thêm node mới có giá trị x vào cây. Node mới thêm vào tại vị trí NULL trong quá trình kết thúc tìm kiếm ở trên. Node mới thêm vào là nút lá.

33

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

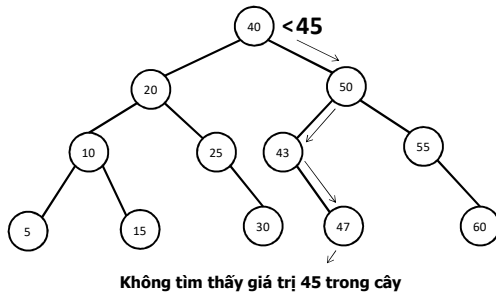
#### THÍ DỤ 1: Thêm một nút có giá trị 45 vào cây



34

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

**Bước 1:** Tìm nút có giá trị 45 trong cây.

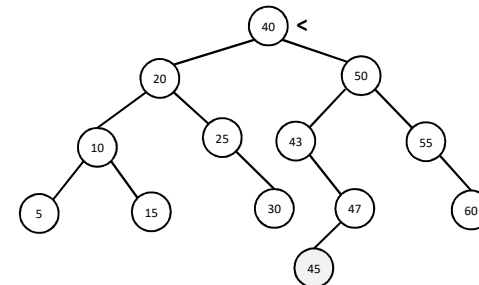


35

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

**Bước 2:** Vị trí xét để dừng thuật toán tìm kiếm là vị trí có giá trị NULL bên trái node 47. Như vậy node mới thêm vào (có giá trị 45) là node con bên trái của node có giá trị 47. Node mới thêm vào sẽ là *node lá*.

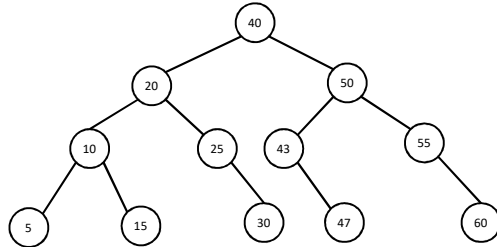
**Kết quả thêm node mới có giá tr 45 như sau**



36

## 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

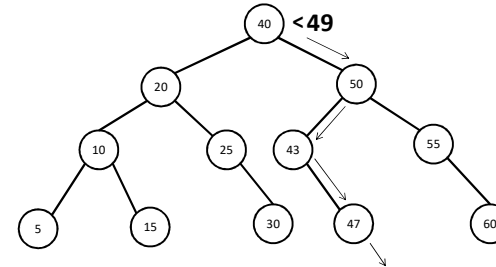
**THÍ DỤ 2:** Thêm một nút có giá trị 49 vào cây



37

## 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

**Bước 1:** Tìm nút có giá trị 49 trong cây.



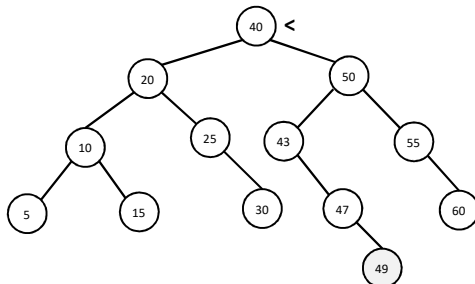
Không tìm thấy giá trị 49 trong cây

38

## 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

**Bước 2:** Vị trí xét để dừng thuật toán tìm kiếm là vị trí có giá trị NULL bên phải node 47. Như vậy node mới thêm vào (có giá trị 49) là node con bên phải của node có giá trị 47. Node mới thêm vào sẽ là *node lá*.

**Kết quả thêm node mới có giá tr 49 như sau**



39

## 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

### THUẬT TOÁN THÊM MỘT NODE MỚI CÓ GIÁ TRỊ X VÀO CÂY

**Bước 1:** Tìm giá trị x trong cây:

Bước 1.a: Gán địa chỉ node gốc cho biến p (bắt đầu từ node gốc)

Bước 1.b: Tại node có địa chỉ p, so sánh giá trị node và giá trị phân tử x:

+ Nếu giá trị node = x, tìm thấy một node có giá trị là x. Thuật toán kết thúc.

+ Nếu giá trị node > x, chuyển xét node con bên trái node đang xét.

+ Nếu giá trị node < x, chuyển xét node con bên phải node đang xét.

Bước 1.c: Nếu giá trị node đang xét khác NULL thì lặp lại *bước*

1.b. Ngược lại nếu node đang xét bằng có địa chỉ NULL thì qua *bước 2*.

**Bước 2:** thêm nút mới vào vị trí đang xét;

Node \*p = root;

```
if (p->info == x)
    return ;
else if (p -> info > x)
    p=p -> left;
else
    p=p -> right;
```

```
node *p = new node;
p -> info = x
p -> left = NULL;
p -> right = NULL;
```

40

## 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

### CHƯƠNG TRÌNH

```
void InsertNode(node *&p, int x)
{
    if (p == NULL)
    {
        p = new Node;
        p->info = x;
        p->left = NULL;
        p->right = NULL;
    }
    else
    {
        if (p->info == x)
            return; // đã có node có giá trị x
        else if (p->info > x)
            return InsertNode(p->left, x);
        else
            return InsertNode(p->right, x);
    }
}
```

Gọi thực thi: InsertNode(root, x);  
X được nhập trước khi gọi thủ tục  
InsertNode(root, x);

41

## 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

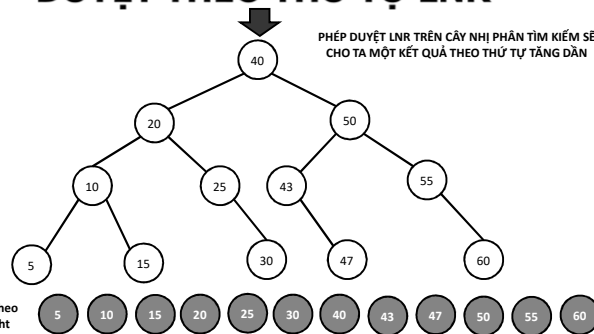
### DUYỆT CÂY NHỊ PHÂN TÌM KIẾM

- 📖 Duyệt theo thứ tự LNR (LEFT-NODE-RIGHT)
- 📖 Duyệt theo thứ tự LRN (LEFT-RIGHT-NODE)
- 📖 Duyệt theo thứ tự NLR (NODE-LEFT-RIGHT)

42

## 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

### DUYỆT THEO THỨ TỰ LNR



43

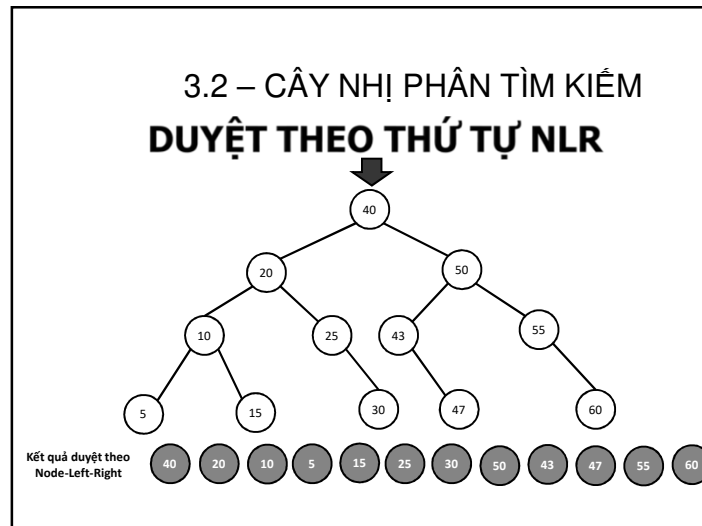
## 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

### CHƯƠNG TRÌNH

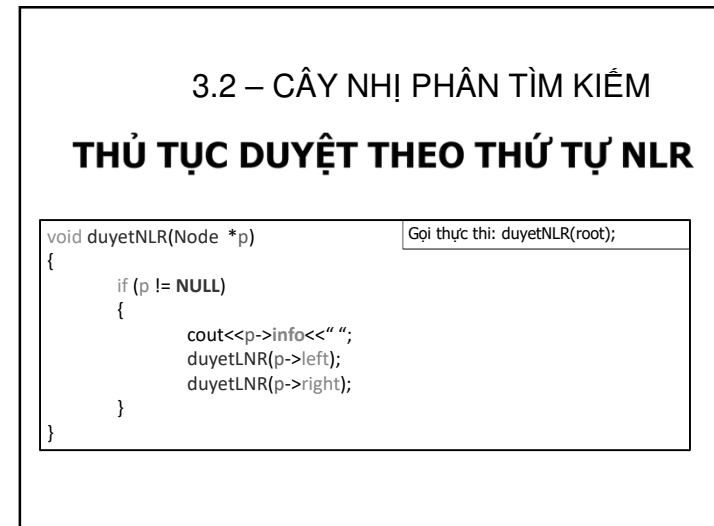
```
void duyetLNR(Node *p)
{
    if (p != NULL)
    {
        duyetLNR(p->left);
        cout<<p->info<<" ";
        duyetLNR(p->right);
    }
}
```

Gọi thực thi: duyetLNR(root);

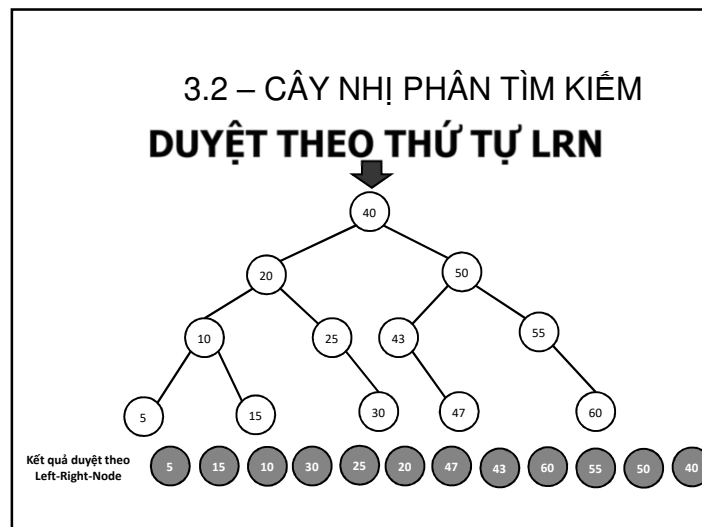
44



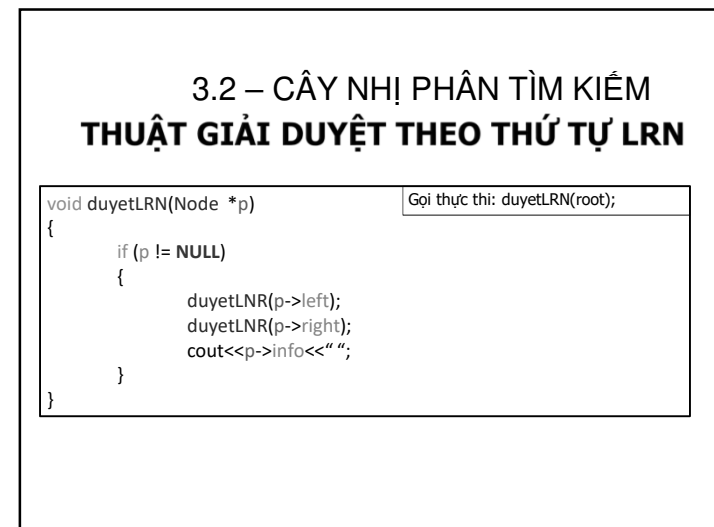
45



46



47



48

## 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

### XÓA MỘT NÚT TRÊN CÂY

49

## 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

Để xóa một node có giá trị là x trong cây nhị phân tìm kiếm, thực hiện tuần tự các bước sau:

**Bước 1:** Tìm node có giá trị x trong cây.

Nếu không tìm thấy, kết thúc.

Ngược lại nếu tìm thấy một node có giá trị là x trong cây thì chuyển sang bước 2

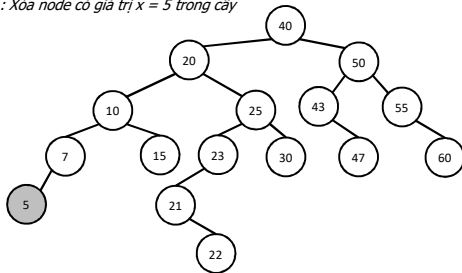
**Bước 2:** Node có giá trị x tìm thấy cần xóa, có 03 trường hợp:

50

## 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

**TRƯỜNG HỢP 1:** Node cần xóa là node lá. Việc xóa một node lá trong cây dễ dàng, bằng cách: Liên kết từ node cha node cần xóa (liên kết left hoặc right) đến node cần xóa, được sửa lại thành NULL.

*Thí dụ 1: Xóa node có giá trị x = 5 trong cây*

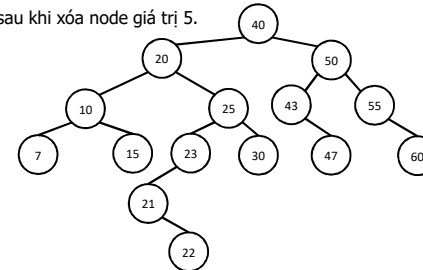


51

## 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

Node 5 là node lá. Node 5 là node *con bên trái* của node 7. Để xóa node có giá trị 5, thực hiện tuần tự các bước sau:

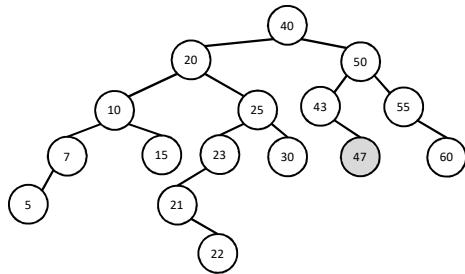
*Cập nhật liên kết left* của node chứa giá trị 7 thành NULL; xả node có giá trị 5. Kết quả của cây sau khi xóa node giá trị 5.



52

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

Thí dụ 2: Xóa node có giá trị  $x = 47$  trong cây. Node có giá trị 47 là node lá



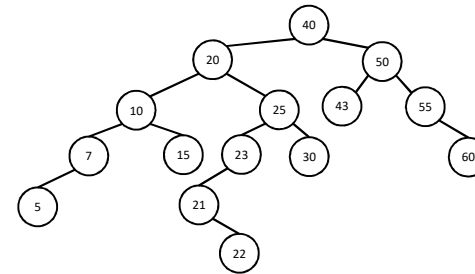
53

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

Node 47 là node lá. Node 47 là node *con bên phải* của node 43. Để xóa node có giá trị 47, thực hiện tuần tự các bước sau:

- + Điều chỉnh liên kết right của node 43 thành NULL
- + Xóa node 47

Kết quả của cây sau khi xóa node 47



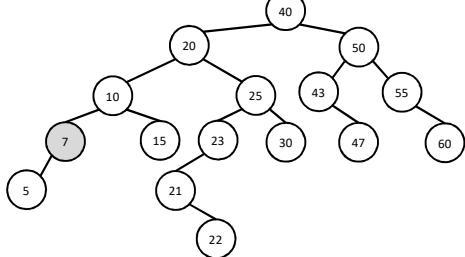
54

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

**TRƯỜNG HỢP 2:** Node cần xóa là node bậc 1. Trong trường hợp node cần xóa là bậc 1. Để xóa node bậc 1, thực hiện tuần tự các bước sau:

- + Chính lại liên kết từ node cha (liên kết left hoặc right) của node cần xóa (node bậc 1) đến node của node cần xóa.
- + Xóa node bậc 1 cần xóa

Thí dụ 3: Xóa node có giá trị  $x = 7$  trong cây sau:

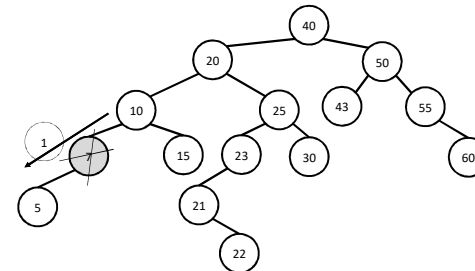


55

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

Node 7 là node bậc 1. Node 7 là node *con bên trái* của node 10, node con duy nhất của node 7 là node 5. Để xóa node có giá trị 7, thực hiện tuần tự các bước sau:

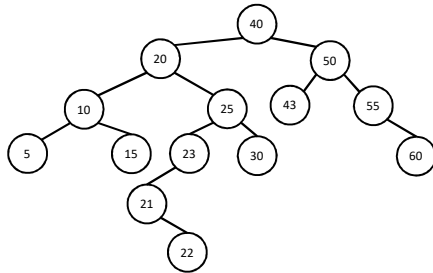
- + Điều chỉnh liên kết con bên trái của node 10 xuống node 5
- + Xóa node 7



56

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

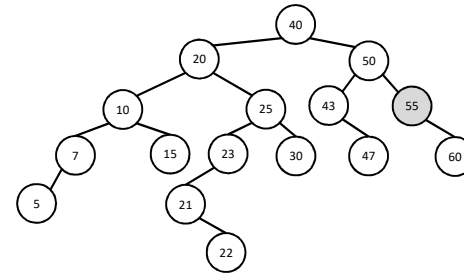
Kết quả của cây sau khi xóa node 7:



57

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

Thí dụ 3: Xóa node có giá trị  $x = 55$  trong cây sau:

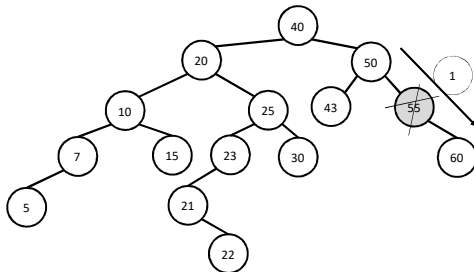


58

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

Node 55 là node bậc 1. Node 55 là node *con bên phải* của node 50, node con duy nhất của node 55 là node 60. Để xóa node 55, thực hiện tuần tự các bước sau:

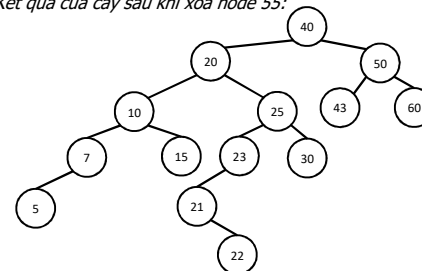
- + Điều chỉnh liên kết con bên trái của node 50 xuống node 60
- + Xóa node 55



59

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

Kết quả của cây sau khi xóa node 55:



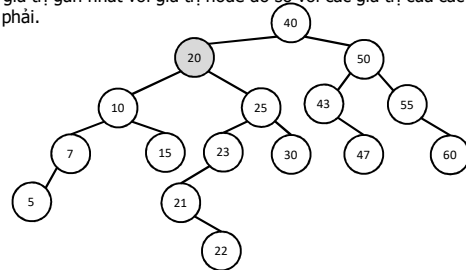
60

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

**TRƯỜNG HỢP 3:** Node cần xóa là node bậc 2 (có đầy đủ node con bên trái và bên phải). Để xóa một node bậc 2 trong cây nhị phân tìm kiếm, ta thực hiện tuần tự các bước sau:

Bước 1: Tìm node con cực trái của nhánh con bên phải của node bậc 2 cần xóa. Dùng node cực trái này làm node thế mạng cho node cần xóa. Vì node cực trái của nhánh con bên phải của một node có giá trị gần nhất với giá trị node đó so với các giá trị của các node trong nhánh con bên phải.

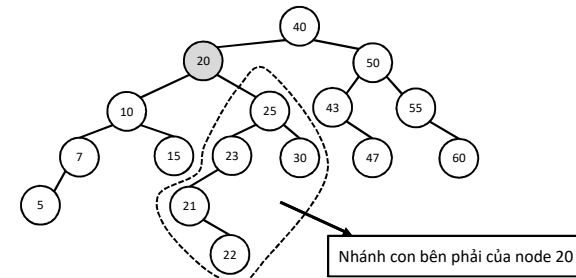
*Thí dụ 3: Xóa node có giá trị  $x = 20$  trong cây sau:*



61

### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

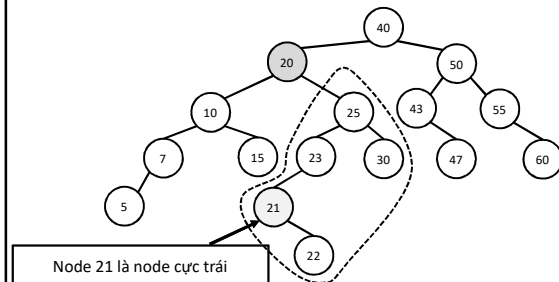
- Tìm thấy node 20 trong cây. Node 20 là bậc 2, có hai nhánh con bên trái và bên phải khác NULL.
- Để xóa node bậc 2 có giá trị là 20 trong cây:
  - + Tìm node cực trái trong nhánh con bên phải node 20



62

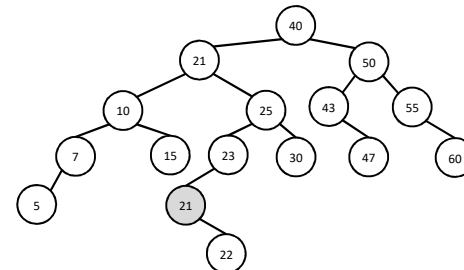
### 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

- Trong nhánh con bên phải node 20, node 21 là node cực trái



63

Bước 2: Chép giá trị node thế mạng lên node cần xóa. Trong thí dụ trên, chép giá trị 21 lên node chứa giá trị 20.

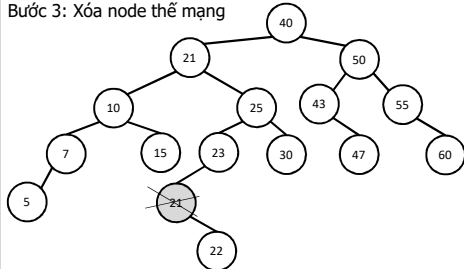


64



## 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

Bước 3: Xóa node thể mạng

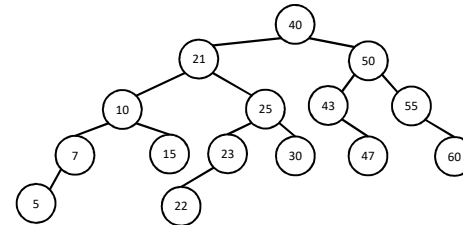


Node thể mạng là node cực trái nên liên kết trái của node này bằng NULL, do đó node thể mạng có bậc 1 (hoặc bậc 0). Việc xóa node thể mạng về trường hợp 2 (xóa node bậc 1)

65

## 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

Chỉnh liên kết trái của node cha node thể mạng trở vào node con bên phải node thể mạng.



66

## 3.2 – CÂY NHỊ PHÂN TÌM KIẾM

### CHƯƠNG TRÌNH

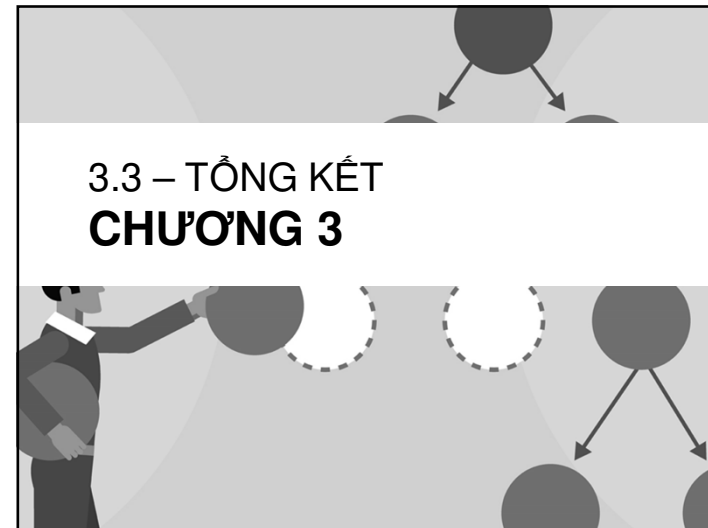
```
int Delete(Node *&T, int x)
{
    if (T == NULL) return 0;
    if (T->info == x)
    {
        Node *p = T;
        if (T->left == NULL)
            T = T->right;
        else if (T->right == NULL)
            T = T->left;
        else // có 2 con
            searchStandFor(p, q->right);
        delete p;
        return 1;
    }
    if (T->info < x) return Delete(T->right, x);
    if (T->info > x) return Delete(T->left, x);
}
```

```
void searchStandFor(Node *&p, Node *&q)
{
    if (q->left == NULL)
    {
        p->info == q->info
        p = q;
        q = q->right;
    }
    else
        searchStandFor(p, q->left);
}
```

Thủ tục gọi xóa một node trong cây:  
Delete(root, x);  
// chú ý: x phải được nhập trước khi gọi hàm

67

## 3.3 – TỔNG KẾT CHƯƠNG 3



68

### 3.3 – Tổng kết chương 4


- 📖 Các **khái niệm** như định nghĩa cây, nút gốc, nút lá, ....., chiều cao cây được trình bày.
- 📖 Các nội dung về cấu trúc **cây nhị phân** và các thuật toán trên **cây nhị phân tìm kiếm**: khởi tạo cây rỗng, thêm phần tử vào cây, tìm một phần tử trong cây, xóa một nút trong cây, các phép duyệt cây.
- 📖 Mỗi **node** (phần tử) trong cây nhị phân tìm kiếm ngoài thành phần lưu trữ giá trị node (info) còn có hai thành phần kiểu địa chỉ bộ nhớ dùng để lưu trữ địa chỉ của node con bên trái (left) và lưu địa chỉ node con bên phải (right).

69

### 3.3 – Tổng kết chương 3

- 📖 Cấu trúc cây nhị phân tìm kiếm quản lý một tập các phần tử **có số lượng khá lớn**, được cấp phát rời rạc trong bộ nhớ.
- 📖 Cây nhị phân tìm kiếm có **khả năng tìm kiếm nhanh**, do tính chất, giá trị của một node sẽ lớn hơn các giá trị bên nhánh con bên trái và nhỏ hơn các giá trị của nhánh con bên phải.
- 📖 Việc **thêm/xóa node trong cây** khá phức tạp, do phải thực hiện nhiều phép so sánh.

70



### 3.4- Bài tập rèn luyện CHƯƠNG 3



71

### 3.4 - Bài tập chương 3

#### CÂU HỎI

- 📖 **Câu 1:** Hãy trình bày các vấn đề sau: Định nghĩa và đặc điểm của cây nhị phân tìm kiếm; Các thao tác thực hiện tốt trong kiểu này; Hạn chế của kiểu CTDL này?
- 📖 **Câu 2:** Hãy so sánh cây nhị phân tìm kiếm và các CTDL cơ bản: danh sách đặc, danh sách liên kết, danh sách hạn chế.

72

### 3.4 - Bài tập chương 3

#### BÀI TẬP THỰC HÀNH

**Bài 1:** Quản lý một cây nhị phân (mỗi phần tử có kiểu int)

- 1.1. Khai báo cấu trúc cây nhị phân tìm kiếm.
- 1.2. Viết thủ tục khởi tạo cây rỗng.
- 1.3. Viết thủ tục thêm một phần tử vào cây (dùng đệ quy).
- 1.4. Viết thủ tục tìm một phần tử trong cây (dùng đệ quy).
- 1.5. Viết thủ tục xóa một nút trong cây (dùng đệ quy).
- 1.6. Viết thủ tục duyệt cây theo thứ tự NLR (dùng đệ quy)
- 1.7. Viết thủ tục duyệt cây theo thứ tự LNR (dùng đệ quy)
- 1.8. Viết thủ tục duyệt cây theo thứ tự LRN (dùng đệ quy)

73

### 3.4 - Bài tập chương 3

#### BÀI TẬP LÀM THÊM

**Bài 2:** Quản lý một cây nhị phân (bài làm thêm)

- 2.1. Khai báo cấu trúc cây nhị phân tìm kiếm.
- 2.2. Viết thủ tục khởi tạo cây rỗng.
- 2.3. Viết thủ tục thêm một phần tử vào cây (không dùng đệ quy).
- 2.4. Viết thủ tục tìm một phần tử trong cây (không dùng đệ quy).
- 2.5. Viết thủ tục xóa một nút trong cây (dùng đệ quy).
- 2.6. Viết thủ tục duyệt cây theo thứ tự NLR (dùng stack)
- 2.7. Viết thủ tục duyệt cây theo thứ tự LNR (dùng stack)
- 2.8. Viết thủ tục duyệt cây theo thứ tự LRN (dùng queue)

74

#### Hướng dẫn

- Tất cả sinh viên phải trả lời các **câu hỏi** của chương, làm **bài tập thực hành** tại phòng máy (**bài làm thêm** ở nhà, và **bài nâng cao** khuyến khích hoàn tất) và nộp bài qua LMS của trường.
- Câu hỏi chương 3 làm trên file WORD; trong bài làm ghi rõ họ tên, lớp, bài tập chương và các thông tin cần thiết.
- Khuyến khích sử dụng tiếng Anh trong bài tập.

⇒ **Ngày nộp:** trước khi học chương 6

⇒ **Cách nộp:** sử dụng **github** để nộp bài, sau đó nộp lên LMS của trường.

75

#### Tài liệu tham khảo

- **Lê Xuân Trường**, (Chương 4) *Cấu trúc dữ liệu*, NXB Trường Đại học Mở TP-HCM, 2016.
- **Dương Anh Đức**, *Giáo trình cấu trúc dữ liệu & giải thuật (Chương 4)*, 2010, ĐH KHTN TP.HCM
- **Thomas H.Cormen, Charles E.Leiserson, Ronald L. Rivest, Clifford Stein**, (Chapter 12) *Introduction to Algorithms*, Third Edition, 2009.
- **Adam Drozdek**, (Chapter 6) *Data Structures and Algorithms in C++*, Fourth Edition, CENGAGE Learning, 2013.

76

### KẾT THÚC CHƯƠNG 3



**Trường Đại học Mở TP.HCM**

*Khoa Công Nghệ Thông Tin*