





CHƯƠNG 6

CẤU TRÚC CÂY- TREE




Chương 6 – Cây

Slide 1



Mục tiêu

G6.1	Hiểu được các khái niệm cơ bản về cấu trúc Cây.	2.5	T	1.3.2
G6.2	Biểu diễn được cấu trúc lưu trữ các loại cây như: cây nhị phân, cây nhị phân tìm kiếm, cây AVL bằng hình ảnh hoặc mã giả.	3.0	T	1.3.2
G6.3	Cài đặt được các cấu trúc cây nhị phân tìm kiếm và cây AVL bằng C/C++.	3.0	T	1.3.2
G6.4	Áp dụng được cây nhị phân tìm kiếm và cây AVL để giải quyết bài toán	3.0	T, U	1.3.2





Chương 6 – Cây

Slide 2



Nội dung

- 7.1 Các khái niệm cơ bản
- 7.2 Cây nhị phân
- 7.3 Cây nhị phân tìm kiếm và AVL
- 7.4 Ứng dụng
-  Bài tập





Chương 6 – Cây

Slide 3

Các khái niệm cơ bản Cây





Chương 6 – Cây

Slide 4



7.1. Các khái niệm cơ bản Cây



☐ **Định nghĩa Cây (Tree):** một tập hợp hữu hạn các phần tử gọi là các **nút** (nodes) và tập hợp hữu hạn các **cạnh** nối các cặp nút lại với nhau mà **không** tạo thành **chu trình**. Nói cách khác, cây là 1 đồ thị không có chu trình.




```

graph TD
    A((A)) --- B((B))
    A --- C((C))
    B --- D((D))
    B --- E((E))
    C --- F((F))
  
```




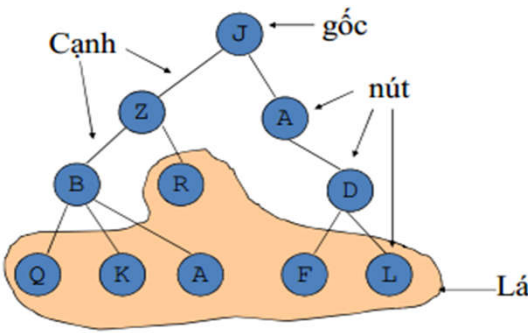
Chương 6 – Cây

Slide 5




7.1. Các khái niệm cơ bản Cây






```

graph TD
    J((J)) --- Z((Z))
    J --- A((A))
    Z --- B((B))
    Z --- R((R))
    B --- Q((Q))
    B --- K((K))
    R --- A2((A))
    A --- D((D))
    D --- F((F))
    D --- L((L))
  
```




Chương 6 – Cây

Slide 6



7.1. Các khái niệm cơ bản Cây

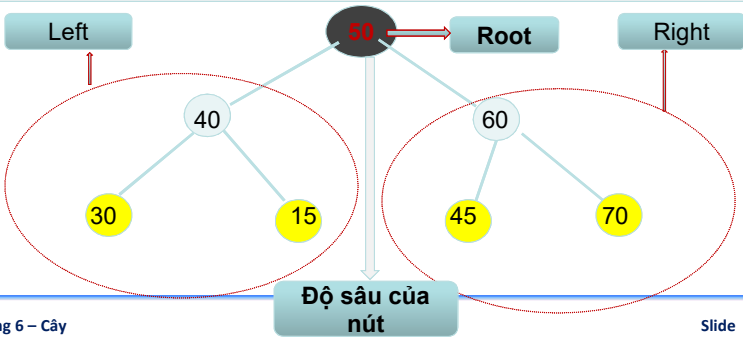



Root: Node đầu tiên định hình cây

Nút lá: Node không có node con trái, không có node con phải.

Độ sâu của cây (depth of tree): mức lớn nhất của node lá trong cây. Như vậy, độ sâu của cây bằng đúng độ dài đường đi dài nhất từ node gốc đến node lá.

Nút trung gian: Node hoặc có node con trái, hoặc có node con phải, hoặc cả hai hoặc cả hai.





Chương 6 – Cây

Slide 7



7.1. Các khái niệm cơ bản Cây



- ☐ **Bậc của một nút:** là số cây con của nút đó.
- ☐ **Bậc của một cây:** là bậc lớn nhất của các nút trong cây (số cây con tối đa của một nút thuộc cây). Cây có bậc n thì gọi là cây n-phân.
- ☐ **Mức của một nút:**
 - Mức (gốc (T)) = 0.
 - Gọi T1, T2, T3,... , Tn là các cây con của T0
 - Mức (T1) = Mức (T2) = ... = Mức (Tn) = Mức (T0) + 1.



Chương 6 – Cây

Slide 8



7.1. Các khái niệm cơ bản Cây



- ☐ **Độ dài đường đi từ gốc đến nút x:** là số nhánh cần đi qua kể từ gốc đến x.
- ☐ **Độ dài đường đi tổng của cây:**


$$P_T = \sum_{X \in T} P_X$$

trong đó P_X là độ dài đường đi từ gốc đến X .
- ☐ **Độ dài đường đi trung bình:** $PI = P_T/n$ (n là số nút trên cây T).
- ☐ **Rừng cây:** là tập hợp nhiều cây



Chương 6 – Cây

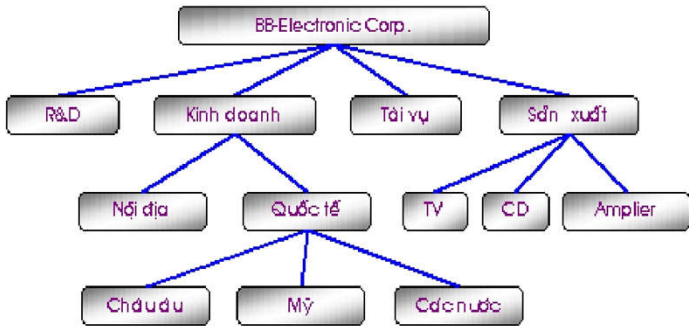
Slide 9



7.1. Các khái niệm cơ bản Cây




Ví dụ: Sơ đồ tổ chức công ty



```

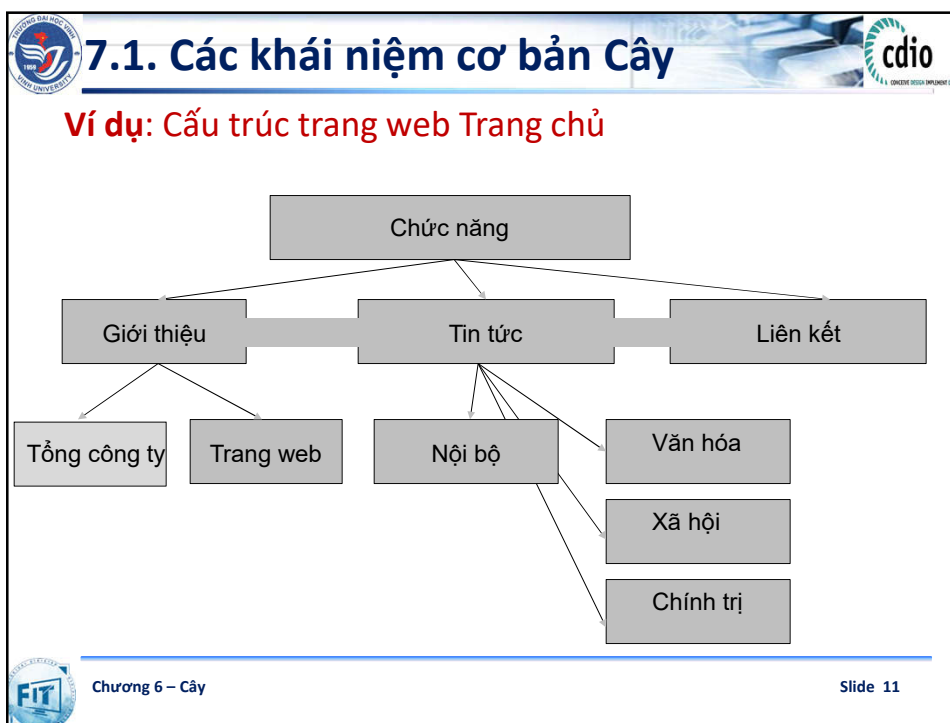
graph TD
    Root[BB-Electronic Corp.] --> RD[R&D]
    Root --> KD[Kinh doanh]
    Root --> TV[Tài vụ]
    Root --> SX[Sản xuất]
    KD --> ND[Nội địa]
    KD --> QT[Quốc tế]
    QT --> CH[Châu Âu]
    QT --> MY[Mỹ]
    QT --> CN[Các nước]
    SX --> T[TV]
    SX --> CD[CD]
    SX --> A[Amplifier]

```



Chương 6 – Cây

Slide 10



7.1. Các khái niệm cơ bản Cây

☐ **Khai báo Cây:** Mỗi nút của cây chứa:

- Phần tử
- Con trỏ tới nút con đầu tiên
- Con trỏ tới nút anh em kế tiếp


```

struct      TreeNode

{ Telem;
  TreeNode *  firstChild;
  TreeNode *  nextSibling;
}
  
```


Chương 6 – Cây

Slide 12



7.1. Các khái niệm cơ bản Cây

- ❑ **Duyệt cây:** Là cách thức duyệt qua tất cả các nút của cây sao cho mỗi nút chỉ được thăm (xử lý) đúng một lần.
- ❑ Có 2 cách duyệt chính:
 - Duyệt theo thứ tự trước
 - Duyệt theo thứ tự sau

Chương 6 – CâySlide 13



7.1. Các khái niệm cơ bản Cây

Duyệt theo thứ tự trước

1. Xuất phát từ nút gốc;
2. Thăm nút đang xét;
3. Duyệt các nút con của nút đang xét từ trái sang phải theo kiểu đệ quy.

Chương 6 – CâySlide 14


7.1. Các khái niệm cơ bản Cây

(1) (2) (3) (4)


Chương 6 – Cây Slide 15

(5) (6) (7) (8)

Chương 6 – Cây Slide 16




7.1. Các khái niệm cơ bản Cây




Duyệt theo thứ tự sau

1. Xuất phát từ nút gốc;
2. Duyệt các nút con của nút đang xét từ trái sang phải theo kiểu đệ quy;
3. Thăm nút đang xét;




Chương 6 – Cây

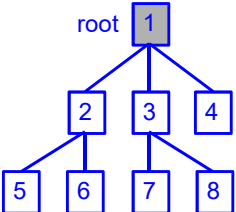
Slide 17



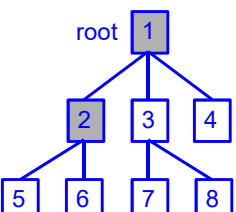
7.1. Các khái niệm cơ bản Cây



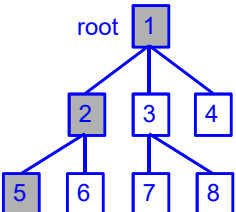
(1)



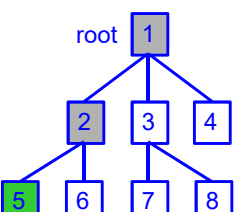
(2)




(3)



(4)

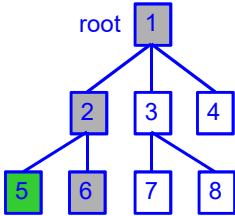
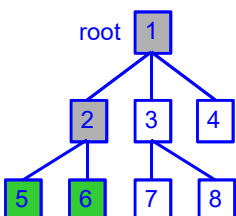


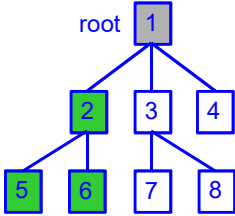
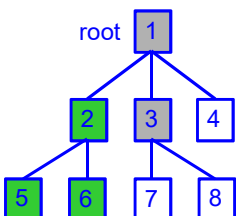


Chương 6 – Cây

Slide 18

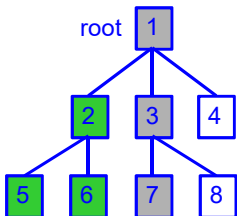
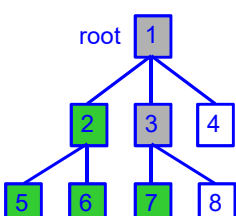
7.1. Các khái niệm cơ bản Cây

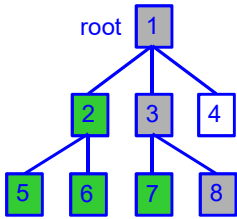
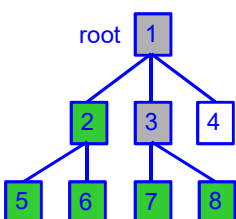
(5)  (6) 

(7)  (8) 

Chương 6 – Cây Slide 19

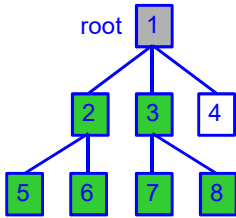
7.1. Các khái niệm cơ bản Cây

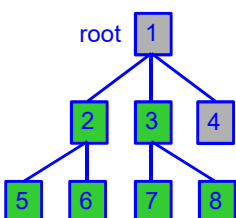
(9)  (10) 

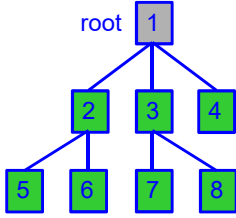
(11)  (12) 

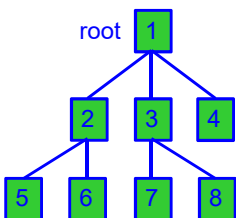
Chương 6 – Cây Slide 20

7.1. Các khái niệm cơ bản Cây

(13) 

(14) 

(15) 


(16) 

Chương 6 – Cây

Slide 21


Cây nhị phân

Binary Tree- BT




Chương 6 – Cây

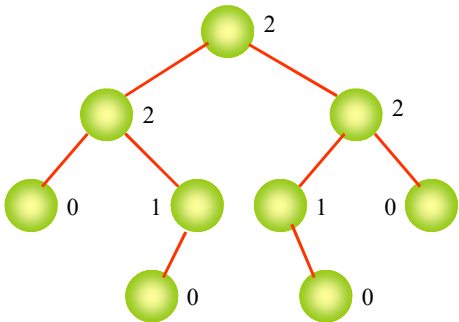
Slide 22




7.2. Cây nhị phân



❑ **Cây nhị phân** là cây mà mỗi nút có tối đa 2 cây con.




- **Bậc của một nút:** là số cây con của nút đó
- **Bậc của cây:** là bậc lớn nhất của các nút trong cây. Cây có bậc n thì gọi là cây n-phân
- **Nút gốc:** là nút không có nút cha
- **Nút lá:** là nút có bậc bằng 0
- **Nút nhánh:** là nút có bậc khác 0 và không phải là gốc




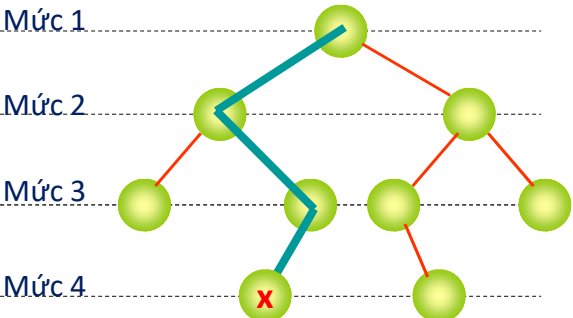
Chương 6 – Cây

Slide 23




7.2. Cây nhị phân







- **Chiều dài đường đi đến nút x:** Là số nhánh cần đi qua kể từ gốc đến x.
- **Chiều cao h của cây:** Mức lớn nhất của các nút lá.

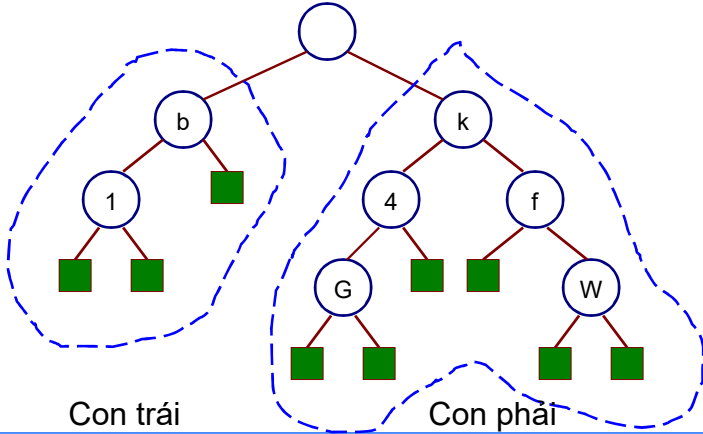
- Số nút ở mức $l \leq 2^{l-1}$
- Số nút ở mức lá $\leq 2^{h-1}$, với h là chiều cao của cây
- Chiều cao của cây $h \geq \log_2 N$, với N là số nút trong cây






7.2. Cây nhị phân






Con trái Con phải




Chương 6 – Cây

Slide 25



7.2. Cây nhị phân




☐ Khai báo cây nhị phân: Mỗi nút chứa

- Phần tử
- Con trỏ tới nút con trái (có thể bằng NULL)
- Con trỏ tới nút con phải (có thể bằng NULL)


struct BinaryNode

```
{ Telem;
  BinaryNode *   left;
  BinaryNode *   right;
}
```




Chương 6 – Cây

Slide 26




7.2. Cây nhị phân




☐ Cài đặt Cây nhị phân

- Cài đặt cây bằng mảng
- Cài đặt cây bằng danh sách
- Cài đặt cây bằng con trỏ




Chương 6 – Cây

Slide 27




7.2. Cây nhị phân




☐ Các phép duyệt cây nhị phân

- Duyệt theo thứ tự trước (Node-Left-Right)
- Duyệt theo thứ tự giữa (Left- Node-Right)
- Duyệt theo thứ tự sau (Left- Right-Node)




Chương 6 – Cây

Slide 28



7.2. Cây nhị phân



Duyệt theo thứ tự trước


Duyệt cho tới khi tất cả các nút đều được duyệt:

Bước 1: Xét nút gốc

Bước 2: Duyệt các cây con bên trái một cách đệ qui


Bước 3: Duyệt các cây con bên phải một cách đệ qui

Node-Left-Right




Chương 6 – Cây

Slide 29



7.2. Cây nhị phân




Duyệt theo thứ tự trước

```

void    NLR(TREE root)
{
    if (Root != NULL)
    {
        <Xử lý Root
        NLR(root->left);
        NLR(root->right);
    }
}

```



Chương 6 – Cây

Slide 30

Duyệt theo thứ tự trước

Kết quả: A B D H I N E J O K C F L P G M

Chương 6 – Cây

Slide 31

7.2. Cây nhị phân

Duyệt theo thứ tự giữa

Bước 1: Duyệt các cây con bên trái một cách đệ quy


Bước 2: Truy cập nút gốc

Bước 3: Duyệt các cây con bên phải một cách đệ quy


Left - Node - Right

Chương 6 – Cây

Slide 32




7.2. Cây nhị phân




Duyệt theo thứ tự giữa

```
void    LNR(TREE root)
{
    if (root != NULL)
    {
        LNR(root->left);
        LNR(root->right);
    }
}
```




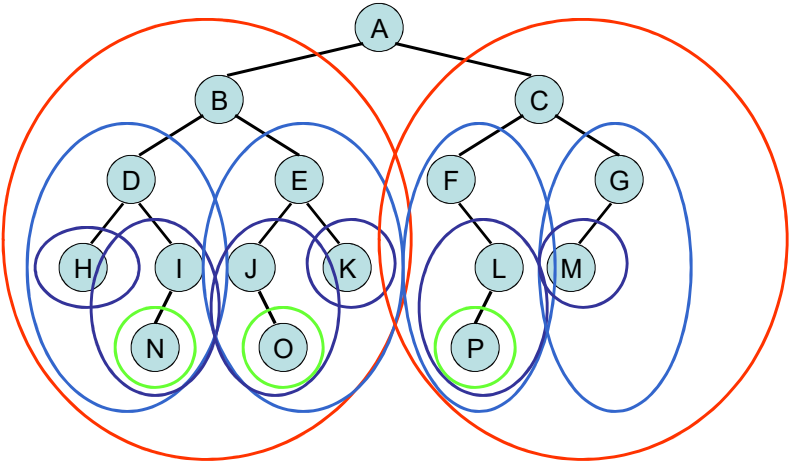
Chương 6 – Cây

Slide 33




Duyệt theo thứ tự giữa






Kết quả: H D N I B J O E K A F P L C M G




Chương 6 – Cây

Slide 34



7.2. Cây nhị phân




Duyệt theo thứ tự sau

Bước 1: Duyệt các cây con bên trái một cách đệ qui

Bước 2: Duyệt các cây con bên phải một cách đệ qui


Bước 3: Truy cập nút gốc.

Left- Right-Node




Chương 6 – Cây

Slide 35




7.2. Cây nhị phân




Duyệt theo thứ tự sau

```
void LRN(TREE root)
{
    if (root != NULL)
    {
        LRN(root->left);
        LRN(root->right);
        <Xử lý Root>;
    }
}
```




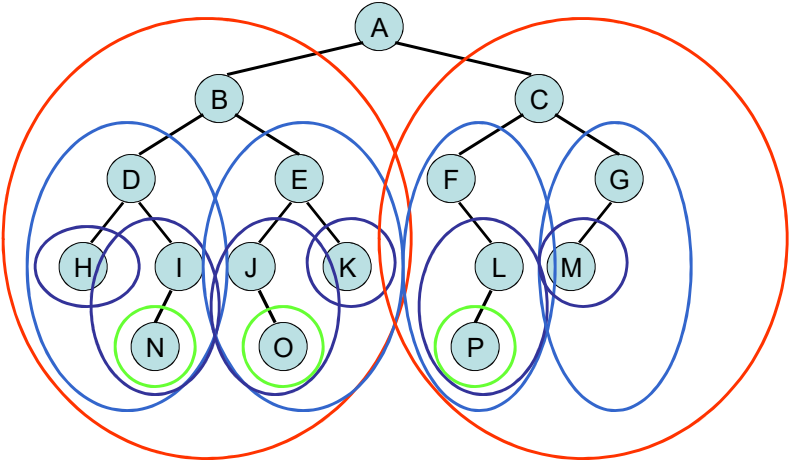
Chương 6 – Cây

Slide 36




Duyệt theo thứ tự sau






Kết quả: H N I D O J K E B P L F M G C A




Chương 6 – Cây

Slide 37




7.2. Cây nhị phân




☐ **Biểu diễn Cây nhị phân**

- Giữ lại nút con trái nhất làm nút con trái.
- Các nút con còn lại chuyển thành nút con phải.
- Như vậy, trong cây nhị phân mới, con trái thể hiện quan hệ cha con và con phải thể hiện quan hệ anh em trong cây tổng quát ban đầu.




Chương 6 – Cây

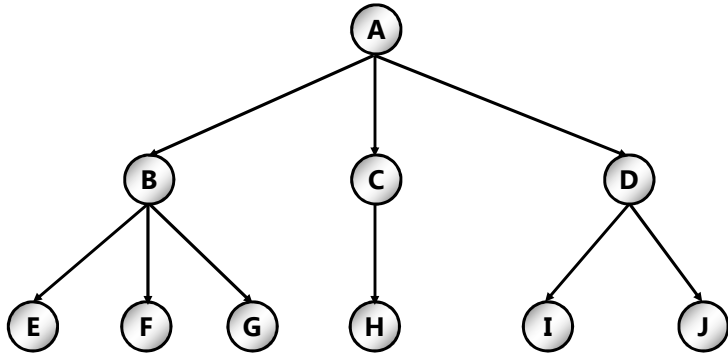
Slide 38




7.2. Cây nhị phân




□ Ví dụ: Cho Cây tổng quát






Chương 6 – Cây

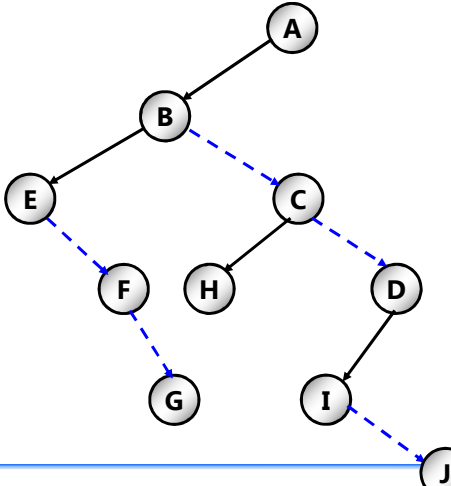
Slide 39




7.2. Cây nhị phân



□ Cây nhị phân tương ứng sẽ như sau:





Chương 6 – Cây

Slide 40



Các thao tác trên Cây nhị phân




- Khởi tạo cây
- Tạo nút mới
- Thêm nút mới vào cây nhị phân
- Tính chiều cao của Cây
- Hủy một nút trên Cây




Chương 6 – Cây

Slide 41



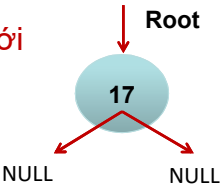
Các thao tác trên Cây nhị phân




- Khởi tạo cây
- Tạo nút mới
- Thêm một nút vào Cây
 - Thêm một nút vào bên trái
 - Thêm một nút vào bên phải

Root → NULL

Root




NULL NULL




Chương 6 – Cây

Slide 42




Các thao tác trên Cây nhị phân




Để đơn giản, ta khai báo cấu trúc dữ liệu như sau :

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




Chương 6 – Cây

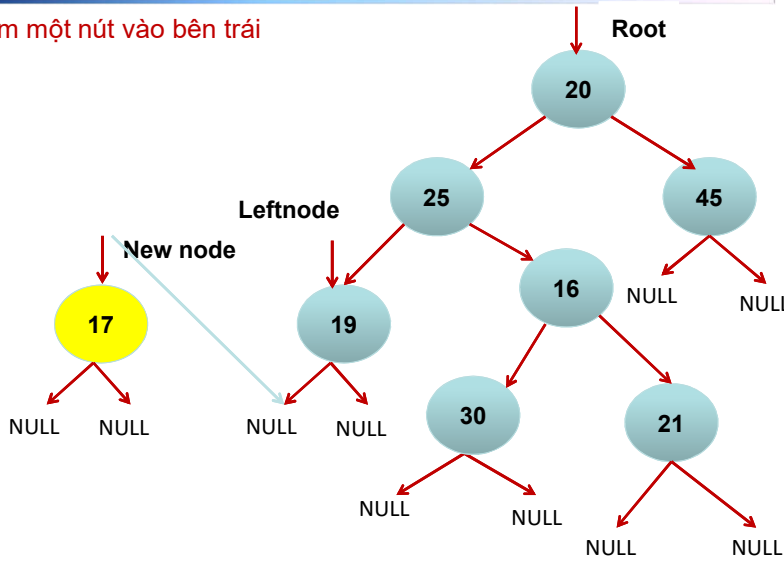
Slide 43



Các thao tác trên Cây nhị phân




- Thêm một nút vào bên trái



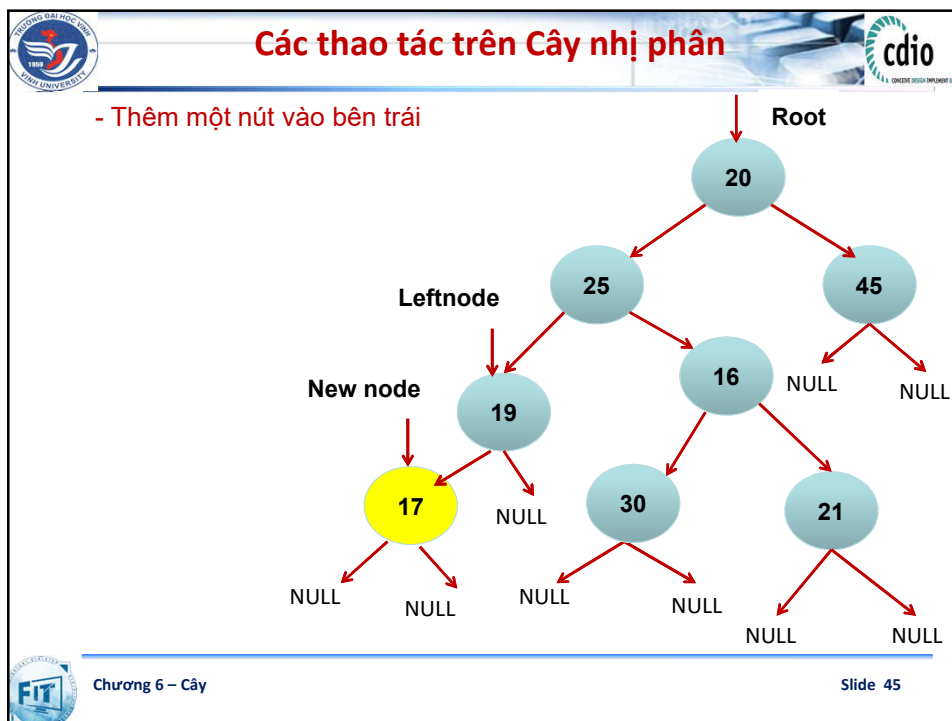
```

graph TD
    Root((20)) --> L25((25))
    Root --> R45((45))
    L25 --> L19((19))
    L25 --> R16((16))
    L19 --> L17((17))
    L19 --> R30((30))
    R16 --> L30((30))
    R16 --> R21((21))
    L17 --> NULL1[NULL]
    L17 --> NULL2[NULL]
    R30 --> NULL3[NULL]
    R30 --> NULL4[NULL]
    L30 --> NULL5[NULL]
    R30 --> NULL6[NULL]
    R21 --> NULL7[NULL]
    R21 --> NULL8[NULL]
    R45 --> NULL9[NULL]
    R45 --> NULL10[NULL]
  
```



Chương 6 – Cây

Slide 44



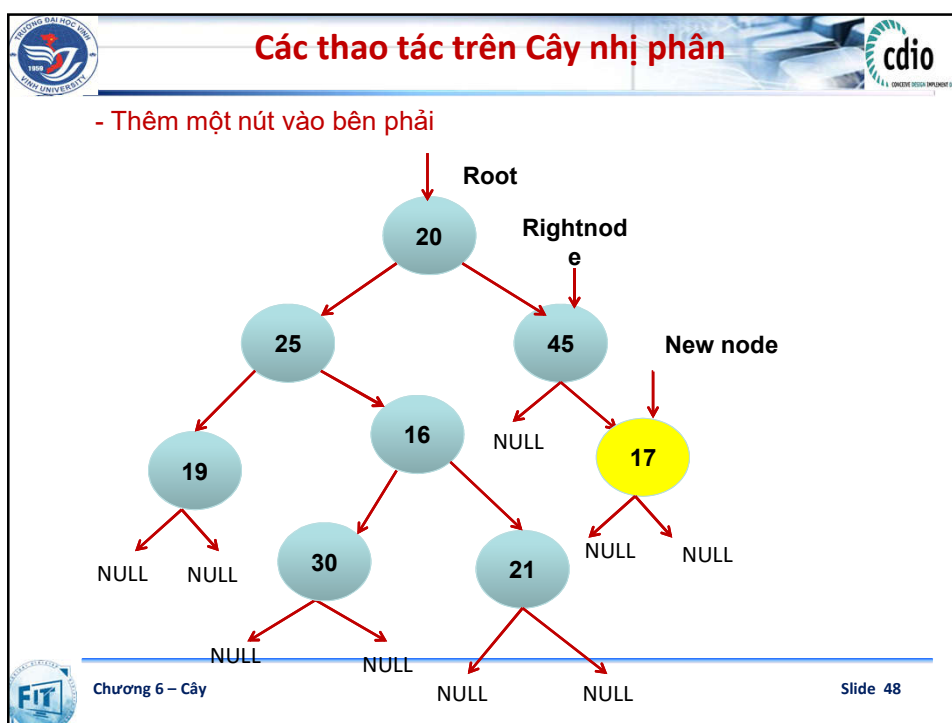
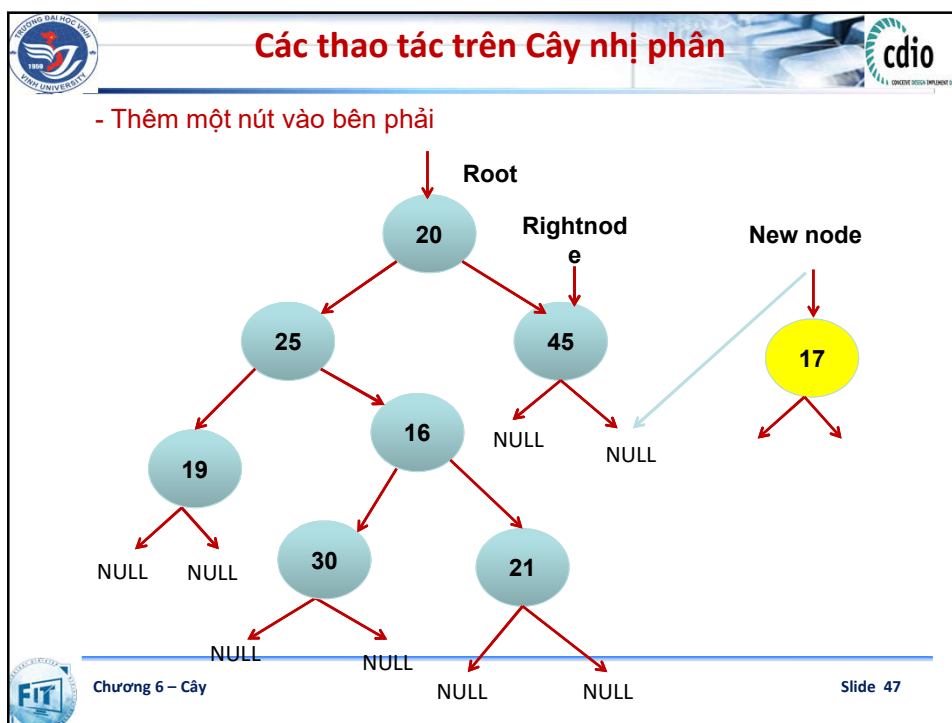
Các thao tác trên Cây nhị phân


```

void CreateTree(TREE &root)
{
    int x;
    printf("\nGia tri node :");
    x=toupper(getch());
    if(isspace(x)==0)
    {
        root=(node*)malloc(sizeof(node));
        root->data=x;
        printf("\nCon trai cua %c (ENTER NULL)",x);
        CreateTree(root->left);
        printf("\nCon phai cua %c (ENTER NULL)",x);
        CreateTree(root->right);
    }
    else root=NULL;
}
  
```


Chương 6 – Cây

Slide 46



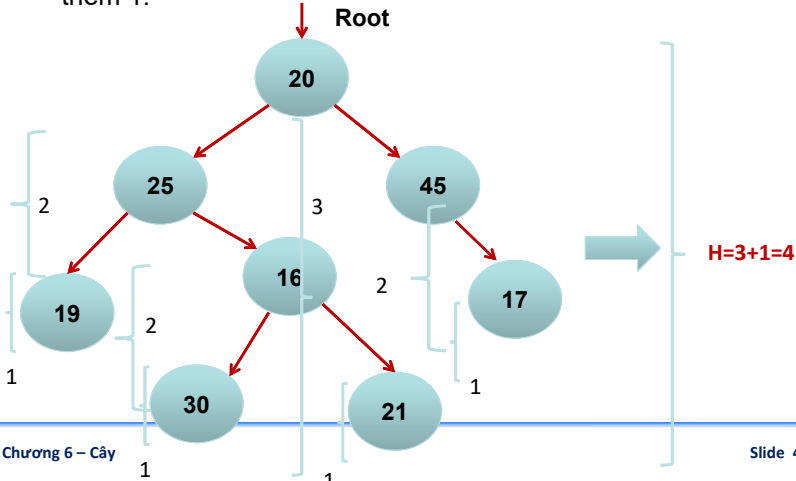


Các thao tác trên Cây nhị phân




➤ **Tính chiều cao của Cây**

✓ Chiều cao của Cây nhị phân là chiều cao lớn nhất của cây con cộng thêm 1.



$H=3+1=4$



Chương 6 – Cây

Slide 49



Cây nhị phân tìm kiếm và Cây AVL

Binary Search Tree- BST







Chương 6 – Cây

Slide 50



7.3.1. Cây nhị phân tìm kiếm





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


→ Nút có giá trị nhỏ nhất nằm ở nút trái nhất của cây

→ Nút có giá trị lớn nhất nằm ở nút phải nhất của cây




Chương 6 – Cây

Slide 51

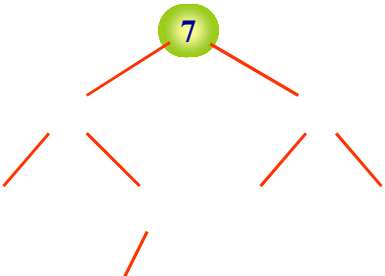


7.3.1. Cây nhị phân tìm kiếm




❖ Tạo 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




Chương 6 – Cây

Slide 52




Các thao tác trên Cây nhị phân TK



```


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

```




Chương 6 – Cây

Slide 53



Các thao tác trên Cây nhị phân TK




- Ta có thể tạo một cây nhị phân tìm kiếm bằng cách lặp lại quá trình thêm 1 phần tử vào một cây rỗng.

```


void CreateTree(tree &root)
{
    int x,n;
    printf("Nhap n = "); scanf("%d",&n);
    for(int i=1; i<=n;i++)
    {
        scanf("%d",&x);
        InsertTree(root,x);
    }
}

```




Chương 6 – Cây

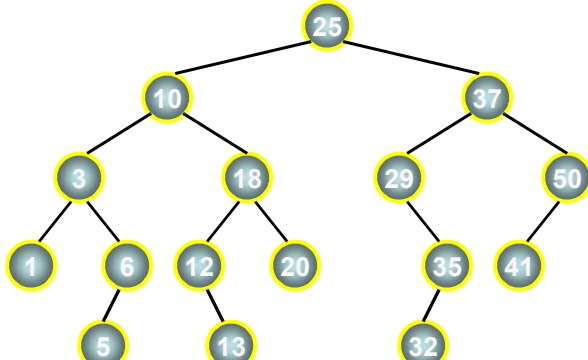
Slide 54




Các thao tác trên Cây nhị phân TK



25 37 10 18 29 50 3 1 6 5 12 20 35 13 32 41







2537101829503165122035133241

Chương 6 – Cây


Slide 55



Các thao tác trên Cây nhị phân TK




- ☐ Thao tác duyệt cây trên cây nhị phân tìm kiếm hoàn toàn giống như trên cây nhị phân.
- ☐ Lưu ý: khi duyệt theo thứ tự giữa, trình tự các nút duyệt qua sẽ cho ta một dãy các nút theo thứ tự tăng dần của khóa.




Chương 6 – Cây

Slide 56




Các thao tác trên Cây nhị phân TK



```


TNODE* searchNode(TREE root, Data X)
{
    if(root)
    {
        if(root->data == X)
            return root;
        if(root->data > X)
            return searchNode(root->left, X);
        return searchNode(root->right, X);
    }
    return NULL;
}

```




Chương 6 – Cây

Slide 57



Các thao tác trên Cây nhị phân TK



```


TNODE * searchNode(TREE root, Data x)
{
    TNODE *p = root;
    while (p != NULL)
    {
        if(x == p->data) return p;
        else
            if(x < p->data) p = p->left;
            else p = p->right;
    }
    return NULL;
}

```




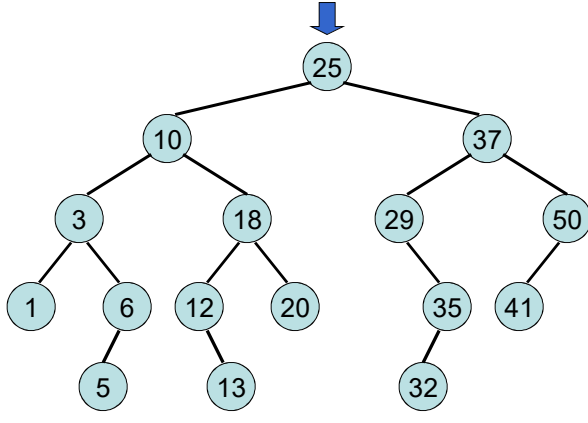
Chương 6 – Cây

Slide 58



Các thao tác trên Cây nhị phân TK






Tìm kiếm 13


Tìm thấy

Số node duyệt: 5
 Số lần so sánh: 9




Chương 6 – Cây

Slide 59




Các thao tác trên Cây nhị phân TK




Nhận xét:

- Số lần so sánh tối đa phải thực hiện để tìm phần tử X là h, với h là chiều cao của cây.
- Như vậy thao tác tìm kiếm trên CNPTK có n nút tổn chi phí trung bình khoảng $O(\log_2 n)$.




Chương 6 – Cây

Slide 60




Các thao tác trên Cây nhị phân TK



➤ Thêm một phần tử x vào cây:

- ↪ Việc thêm một phần tử X vào cây phải bảo đảm điều kiện ràng buộc của CNPTK. Ta có thể thêm vào nhiều chỗ khác nhau trên cây, nhưng nếu thêm vào một nút ngoài sẽ là tiện lợi nhất do ta có thể thực hiện quá trình tương tự thao tác tìm kiếm. Khi chấm dứt quá trình tìm kiếm cũng chính là lúc tìm được chỗ cần thêm.
- ↪ Hàm insert trả về giá trị -1, 0, 1 khi không đủ bộ nhớ, gặp nút cũ hay thành công:




Chương 6 – Cây

Slide 61




Các thao tác trên Cây nhị phân TK



```

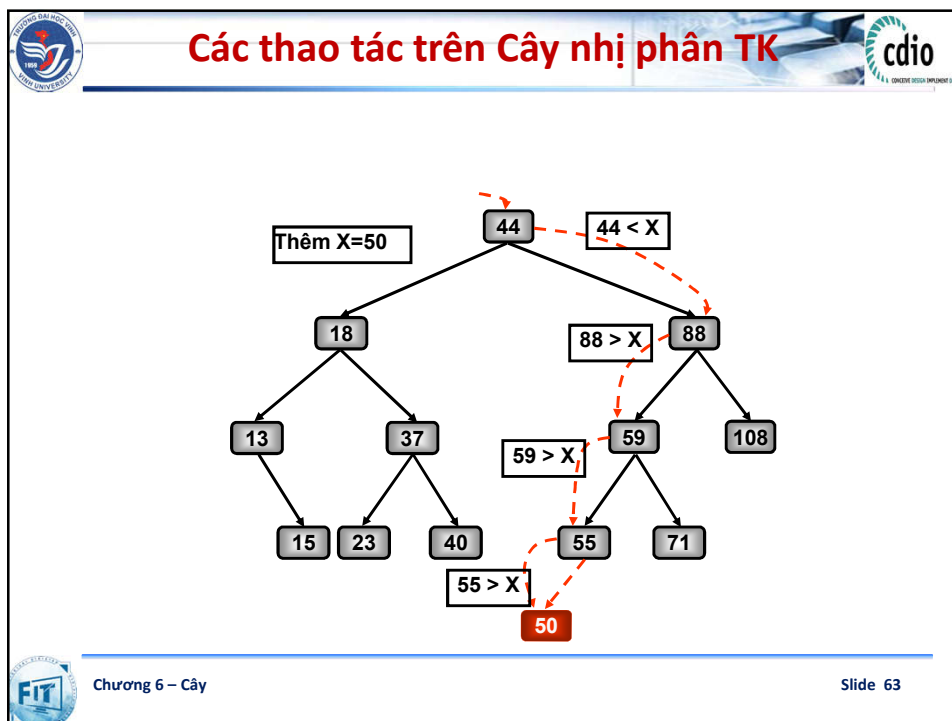
int insertNode(TREE &root, Data X)
{
    if (root) {
        if (root->data == X) return 0; // đã có
        if (root->data > X)
            return insertNode(root->left, X);
        else
            return insertNode(root->right, X);
    }
    root = new Node;
    if (root == NULL) return -1; // thiếu bộ nhớ
    root->data = X;
    root->left = root->right = NULL;
    return 1; // thêm vào thành công
}

```



Chương 6 – Cây

Slide 62



Các thao tác trên Cây nhị phân TK

- Việc hủy một phần tử X ra khỏi cây phải bảo đảm điều kiện ràng buộc của CNPTK.
- Có 3 trường hợp khi hủy nút X có thể xảy ra:
 - X là nút lá.
 - X chỉ có 1 con (trái hoặc phải).
 - X có đủ cả 2 con

Chương 6 – Cây

Slide 64



Các thao tác trên Cây nhị phân TK



Trường hợp 1: X là nút lá.




1. Xóa node này
2. Gán liên kết từ cha của nó thành rỗng




Chương 6 – Cây

Slide 65

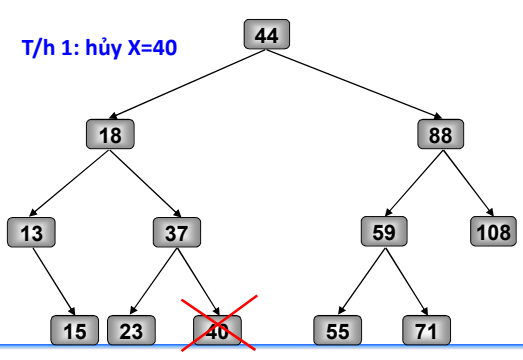



Các thao tác trên Cây nhị phân TK



➤ **Ví dụ :** chỉ đơn giản hủy X vì nó không móc nối đến phần tử nào khác.

T/h 1: hủy X=40





Chương 6 – Cây

Slide 66



Các thao tác trên Cây nhị phân TK



Trường hợp 2: X chỉ có 1 con (trái hoặc phải)




1. Gán liên kết từ cha của nó xuống con duy nhất của nó
2. Xóa node này




Chương 6 – Cây

Slide 67

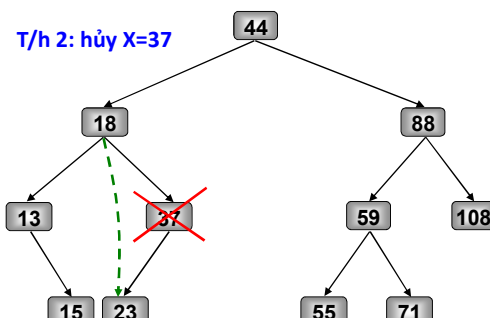



Các thao tác trên Cây nhị phân TK



Trường hợp 2: X chỉ có 1 con (trái hoặc phải)


T/h 2: hủy X=37






Chương 6 – Cây

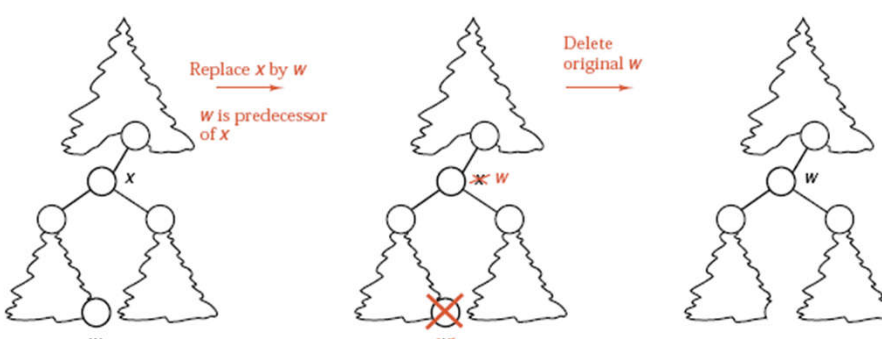
Slide 68



Các thao tác trên Cây nhị phân TK


Trường hợp 3: X có đủ 2 con






Replace x by w
w is predecessor of x
Delete original w

1. Tìm w là node trước node x trên phép duyệt cây inorder (chính là node cực phải của cây con bên trái của x)
2. Thay x bằng w
3. Xóa node w cũ (giống trường hợp 1 hoặc 2 đã xét)




Chương 6 – Cây

Slide 69




Các thao tác trên Cây nhị phân TK

Trường hợp 3: X có đủ 2 con




➤ Trường hợp cuối cùng:

- ↪ Không thể hủy trực tiếp do X có đủ 2 con
- ↪ Hủy gián tiếp:
 - Thay vì hủy X, ta sẽ tìm một phần tử thế mạng Y. Phần tử này có tối đa một con.
 - Thông tin lưu tại Y sẽ được chuyển lên lưu tại X.
 - Sau đó, nút bị hủy thật sự sẽ là Y giống như 2 trường hợp đầu.
- ↪ Vấn đề: chọn Y sao cho khi lưu Y vào vị trí của X, cây vẫn là CNPTK.




Chương 6 – Cây

Slide 70




Các thao tác trên Cây nhị phân TK




Trường hợp 3: X có đủ 2 con

- Vấn đề là phải chọn Y sao cho khi lưu Y vào vị trí của X, cây vẫn là CNPTK.
- Có 2 phần tử thỏa mãn yêu cầu:
 - ↳ Phần tử nhỏ nhất (trái nhất) trên cây con phải.
 - ↳ Phần tử lớn nhất (phải nhất) trên cây con trái.
- Việc chọn lựa phần tử nào là phần tử thế mạng hoàn toàn phụ thuộc vào ý thích của người lập trình.
- Ở đây, ta sẽ chọn phần tử phải nhất trên cây con trái làm phần tử thế mạng.




Chương 6 – Cây

Slide 71

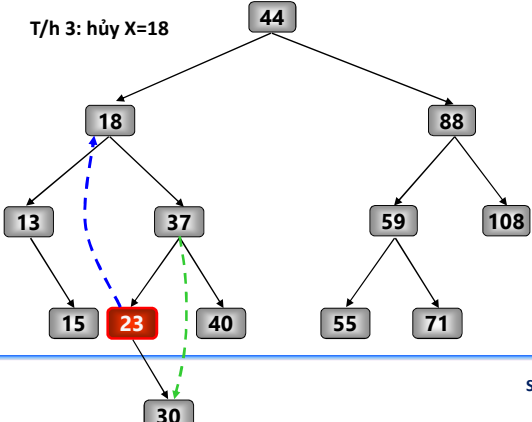



Các thao tác trên Cây nhị phân TK



- Khi hủy phần tử X=18 ra khỏi cây, phần tử 23 là phần tử thế mạng:

T/h 3: hủy X=18





Chương 6 – Cây

Slide 72



Các thao tác trên Cây nhị phân TK



➤ Hàm **delNode** trả về giá trị 1, 0 khi hủy thành công hoặc không có X trong cây:

int delNode(TREE &root, Data X)


➤ Hàm **searchStandFor** tìm phần tử thế mạng cho nút p

void searchStandFor(TREE &p, TREE &q)




Chương 6 – Cây

Slide 73




Các thao tác trên Cây nhị phân TK



```


int delNode(TREE &root, Data X)
{
    if(root== NULL) return 0;
    if(root->data > X) return delNode(root->left, X);
    if(root->data < X) return delNode(root->right, X);
    //T->Key == X
    Node* p = root;
    if(root->left == NULL)
        root = root->right;
    else
        if(root->right == NULL)
            root = root->left;
        else // T chỉ đủ 2 con
            searchStandFor(p, root->right);
    delete p;
}

```




Chương 6 – Cây

Slide 74

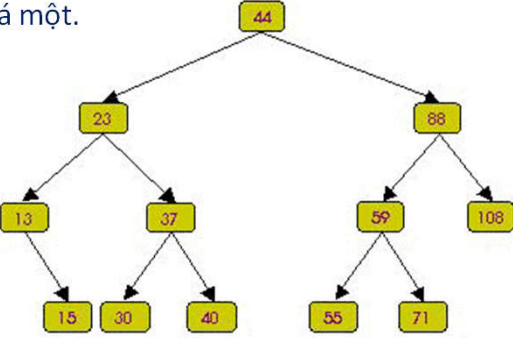



7.3.2. Cây AVL



▪ **Định nghĩa**


Cây cân bằng AVL là Cây nhị phân tìm kiếm cân bằng là cây mà tại mỗi nút của nó độ cao của cây con trái và của cây con phải chênh lệch không quá một.






Chương 6 – Cây

Slide 75



7.3.2. Cây AVL



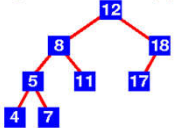
❑ **Cây AVL**

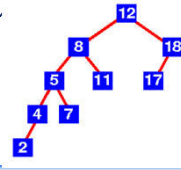
- ↳ Thuật giải cây cân bằng đầu tiên
- ↳ Khám phá bởi: Adelson-Velskii và Landis

❑ **Tính chất**


- ↳ Cây nhị phân tìm kiếm
- ↳ Độ cao con trái và con phải chênh lệch nhiều nhất là 1
- ↳ Các cây con cũng là cây AVL

AVL Tree






~~**AVL Tree**~~




Chương 6 – Cây

Slide 76




7.3.2. Cây AVL




❑ Tổ chức dữ liệu

- Chỉ số cân bằng = độ lệch giữa cây trái và cây phải của một nút
- Các giá trị hợp lệ :
 - $CSCB(p) = 0 \Leftrightarrow$ Độ cao cây trái (p) = Độ cao cây phải (p)
 - $CSCB(p) = 1 \Leftrightarrow$ Độ cao cây trái (p) < Độ cao cây phải (p)
 - $CSCB(p) = -1 \Leftrightarrow$ Độ cao cây trái (p) > Độ cao cây phải (p)




Chương 6 – Cây


Slide 77



7.3.2. Cây AVL




```
#define LH -1 //cây con trái cao hơn
#define EH 0 //cây con trái bằng cây con phải
#define RH 1 //cây con phải cao hơn
typedef struct tagAVLNode
{
    char    balFactor; //chỉ số cân bằng
    Data    key;
    struct tagAVLNode*  pLeft;
    struct tagAVLNode*  pRight;
}AVLNode;
typedef AVLNode    *AVLTree;
```




Chương 6 – Cây

Slide 78



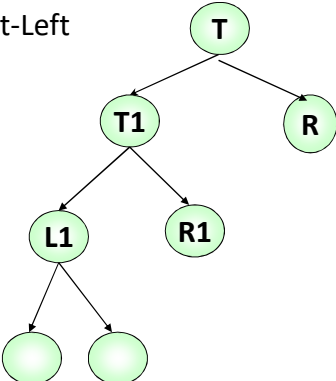
7.3.2. Cây AVL



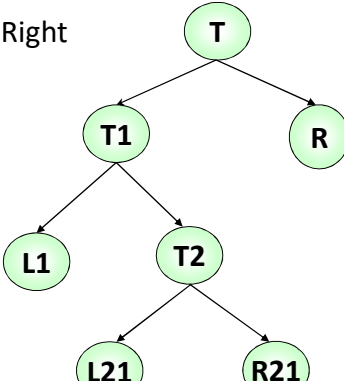
❑ Các trường hợp mất cân bằng do lệch trái


Cây mất cân bằng tại nút T

TH1: Left-Left




TH2: Left-Right






Chương 6 – Cây

Slide 79

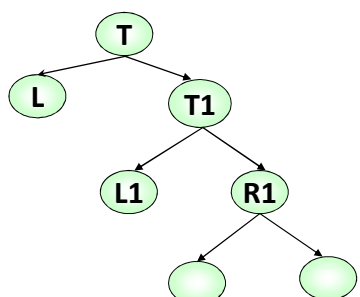


7.3.2. Cây AVL

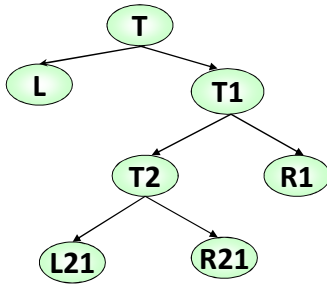



❑ Các trường hợp mất cân bằng do lệch trái

TH3: Right-Right




TH4: Right-Left






Chương 6 – Cây


Slide 80



Các thao tác trên cây cân bằng




- Khi thêm hay xoá 1 nút trên cây, có thể làm cho cây mất tính cân bằng, khi ấy ta phải tiến hành cân bằng lại.
- Cây có khả năng mất cân bằng khi thay đổi chiều cao:
 - ↪ Lệnh nhánh trái, thêm bên trái
 - ↪ Lệnh nhánh phải, thêm bên phải
 - ↪ Lệnh nhánh trái, hủy bên phải
 - ↪ Lệnh nhánh phải, hủy bên trái
- Cân bằng lại cây : tìm cách bố trí lại cây sao cho chiều cao 2 cây con cân đối:
 - ↪ Kéo nhánh cao bù cho nhánh thấp
 - ↪ Phải bảo đảm cây vẫn là Nhị phân tìm kiếm




Chương 6 – Cây

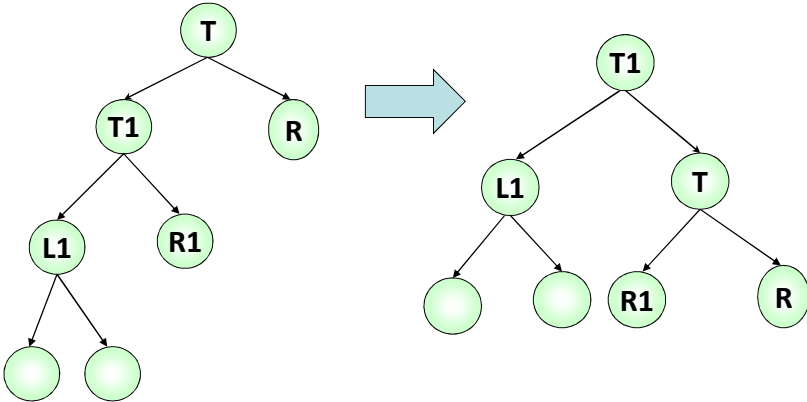
Slide 81




7.3.2. Cây AVL




❑ Cân bằng lại trường hợp 1






Chương 6 – Cây

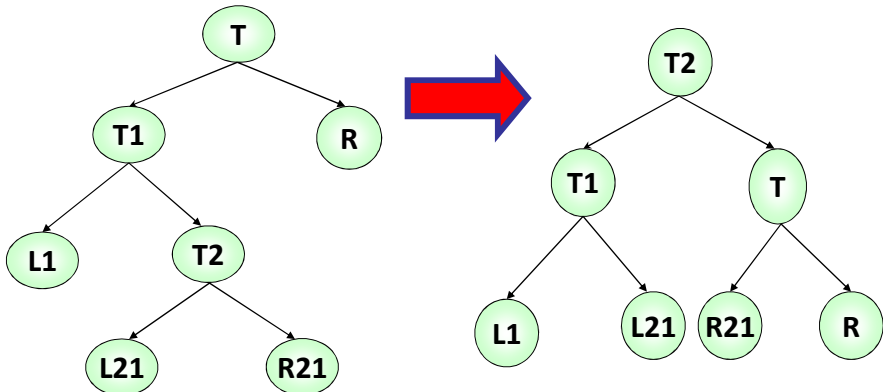
Slide 82




7.3.2. Cây AVL




❑ Cân bằng lại trường hợp 2






Chương 6 – Cây

Slide 83




7.3.2. Cây AVL





❑ Thêm 1 nút

- Thêm bình thường như trường hợp cây NPTK
- Nếu cây tăng trưởng chiều cao
 - Lăn ngược về gốc để phát hiện nút bị mất cân bằng
 - Tiến hành cân bằng lại nút đó bằng thao tác cân bằng thích hợp
- Việc cân bằng lại chỉ cần thực hiện 1 lần nơi mất cân bằng



Chương 6 – Cây


Slide 84



7.3.2. Cây AVL



❑ Hủy 1 nút

- Hủy bình thường như trường hợp cây NPTK
- Nếu cây giảm chiều cao:
 - Lăn ngược về gốc để phát hiện nút bị mất cân bằng
 - Tiến hành cân bằng lại nút đó bằng thao tác cân bằng thích hợp
 - Tiếp tục lăn ngược lên nút cha...
- Việc cân bằng lại có thể lan truyền lên tận gốc




Chương 6 – Cây

Slide 85




7.4. Ứng dụng

Tự học




Chương 6 – Cây

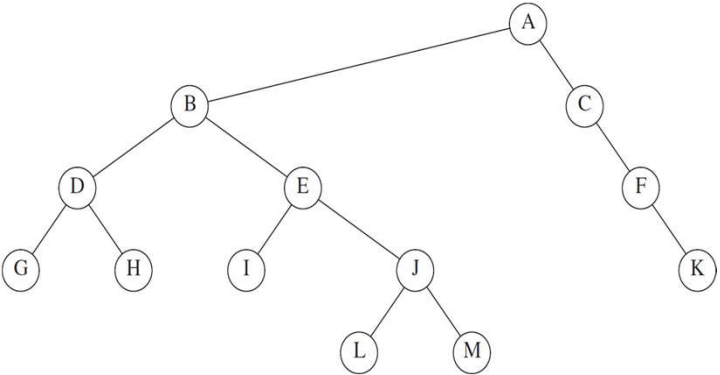
Slide 86




7.5. Bài tập




Bài tập 1: Tính chiều sâu và chiều cao của các nút C, E và H. Tính chiều sâu và chiều cao của cây.






Chương 6 – Cây

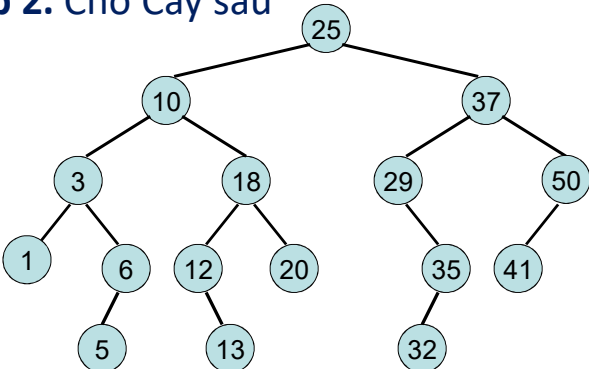
Slide 87




7.5. Bài tập



Bài tập 2. Cho Cây sau





1. Chỉ ra các thành phần của Cây
2. Tính chiều cao của Cây
3. Cây trên có phải là Cây nhị phân không? Vì sao?



Chương 6 – Cây


Slide 88





7.5. Bài tập

Bài tập 3: Cho các nút sau: 10, 27, 15, 18, 22, 11, 30, 37, 23, 21, 45

- Vẽ cây nhị phân tìm kiếm
- Ghi ra kết quả các bước Duyệt cây theo thứ tự trước, sau, giữa
- Viết hàm duyệt theo thứ tự sau

Chương 6 – Cây


Slide 89




7.5. Bài tập

Bài tập 4: Cho các nút sau: 12, 23, 17, 14, 20, 16, 31, 25, 43, 21, 48


- Vẽ cây nhị phân tìm kiếm
- Ghi ra kết quả các bước Duyệt cây theo thứ tự trước, sau, giữa

Chương 6 – Cây

Slide 90




7.5. Bài tập




Bài tập 5: Cho các nút sau: 12, 23, 17, 14, 20, 16, 31, 25, 43, 21, 48

- Vẽ cây nhị phân tìm kiếm
- Ghi ra kết quả các bước Duyệt cây theo thứ tự trước, sau, giữa




Chương 6 – Cây

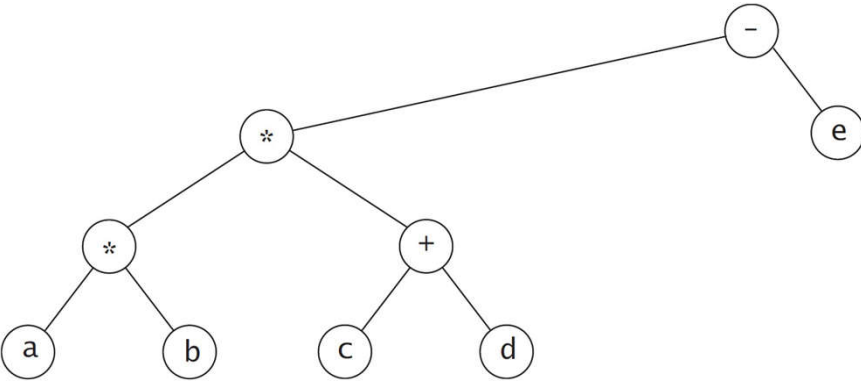
Slide 91



7.5. Bài tập




Bài tập 6: Viết các biểu thức trung tố, tiền tố và hậu tố tương ứng với cây biểu thức bên dưới.



```


graph TD
    Root((−)) --- L1_Star((*))
    Root --- e((e))
    L1_Star --- L2_Star((*))
    L1_Star --- L2_Plus((+))
    L2_Star --- a((a))
    L2_Star --- b((b))
    L2_Plus --- c((c))
    L2_Plus --- d((d))

```




Chương 6 – Cây

Slide 92




7.5. Bài tập




Bài tập 7: Viết chương trình thực hiện các thao tác sau:

- (a) Định nghĩa cấu trúc Cây
- (b) Tạo cây
- (c) Đếm số nút của T
- (d) Đếm số nút lá của T
- (e) Đếm số nút có đủ cả 2 con của T




Chương 6 – Cây

Slide 93




7.5. Bài tập



Bài tập 8:

- (a) Vẽ hình mô tả quá trình biến đổi của một cây nhị phân tìm kiếm (đang rỗng) khi chèn lần lượt các giá trị sau vào cây: 3, 1, 4, 6, 9, 2, 5, 7
- (b) Xóa nút gốc của cây thu được ở câu (a)



Chương 6 – Cây

Slide 94

Bài tập nhóm 1

- **Nghiên cứu và tìm hiểu ứng dụng của Cây AVL. Xây dựng chương trình từ điển Anh – Việt sử dụng cây AVL.**
- **Yêu cầu:**
 1. Viết báo cáo mô tả bài toán theo giải thuật lựa chọn.
 2. Cài đặt chương trình minh họa bằng Matlab.
 3. Slide báo cáo

Chương 6 – Cây

Slide 95