CHƯƠNG III CẤU TRÚC CÂY (TREES)

TỔNG QUAN

1. Mục tiêu

Sau khi học xong chương này, sinh viên phải:

- Nắm vững khái niệm về cây
- Cài đặt được cây (trees) và thực hiện các phép toán trên cây.

2. Kiến thức cơ bản cần thiết

Để học tốt chương này, sinh viên phải nắm vững kỹ năng lập trình căn bản như:

- Kiểu mẩu tin (record) , kiểu mảng (array) và kiểu con trỏ (pointer)
- Các cấu trúc điều khiển, lệnh vòng lặp.
- Lập trình theo từng modul (chương trình con) và cách gọi chương trình con đó.
- Lập trình đệ qui và gọi đệ qui.
- Kiểu dữ liệu trừu tượng danh sách

3. Tài liệu tham khảo

[1] **Aho, A. V. , J. E. Hopcroft, J. D. Ullman**. "*Data Structure and Algorihtms*", Addison–Wesley; 1983

[2] **Đỗ Xuân Lôi** . "*Cấu trúc dữ liệu và giải thuật*". Nhà xuất bản khoa học và kỹ thuật. Hà nội, 1995. (chương 6- Trang 122; chương 10 trang 274)

[3] **N. Wirth** "C*ấu trúc dữ liệu + giải thuật= Chương trình*", 1983.

[4] **Nguyễn Trung Trực**, "*Cấu trúc dữ liệu*". BK tp HCM, 1990. (chương 3 trang 112; chương 5 trang 299)

[5] **Lê Minh Trung** ; "*Lập trình nâng cao bằng Pascal với các cấu trúc dữ liệu* "; 1997 (chương 9, 12)

4. Nội dung cốt lõi

Trong chương này chúng ta sẽ nghiên cứu các vấn đề sau:

- Các thuật ngữ cơ bản.

- Kiểu dữ liệu trừu tượng Cây
- Cài đặt cây
- Cây nhị phân
- Cây tìm kiếm nhị phân

I. CÁC THUẬT NGỮ CƠ BẢN TRÊN CÂY

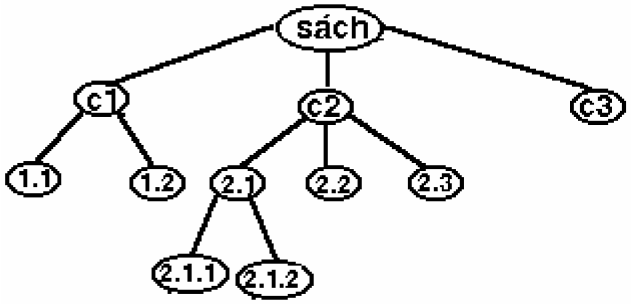
Cây là một tập hợp các phần tử gọi là nút (nodes) trong đó có một nút được phân biệt gọi là nút gốc (root). Trên tập hợp các nút này có một quan hệ, gọi là mối quan hệ *cha - con* (parenthood), để xác định hệ thống cấu trúc trên các nút. Mỗi nút, trừ nút gốc, có duy nhất một nút cha. Một nút có thể có nhiều nút con hoặc không có nút con nào. Mỗi nút biểu diễn một phần tử trong tập hợp đang xét và nó có thể có một kiểu nào đó bất kỳ, thường ta biểu diễn nút bằng một kí tự, một chuỗi hoặc một số ghi trong vòng tròn. Mối *quan hệ cha con* được biểu diễn theo qui ước *nút cha ở dòng trên nút con ở dòng dưới và được nối bởi một đoạn thẳng*. Một cách hình thức ta có thể định nghĩa cây một cách đệ qui như sau:

1. Định nghĩa

- Một nút đơn độc là một cây. Nút này cũng chính là nút gốc của cây.

- Giả sử ta có n là một nút đơn độc và k cây T1,.., Tk với các nút gốc tương ứng là n1,.., nk thì có thể xây dựng một cây mới bằng cách cho nút n là cha của các nút n1,.., nk. Cây mới này có nút gốc là nút n và các cây T1,.., Tk được gọi là các cây con. Tập rỗng cũng được coi là một cây và gọi là cây rỗng kí hiệu .

Ví dụ: xét mục lục của một quyển sách. Mục lục này có thể xem là một cây



Hình III.1 - Cây mục lục một quyển sách

Nút gốc là sách, nó có ba cây con có gốc là C1, C2, C3. Cây con thứ 3 có gốc C3 là một nút đơn độc trong khi đó hai cây con kia (gốc C1 và C2) có các nút con.

Nếu n^1,.., n^k là một chuỗi các nút trên cây sao cho n^i là nút cha của nút n^{i+1}, với i=1..k-1, thì chuỗi này gọi là một *đường đi trên cây* (hay ngắn gọn là đường đi) từ n^1 đến n^k. *Độ dài*

đường đi được định nghĩa bằng số nút trên đường đi trừ 1. Như vậy độ dài đường đi từ một nút đến chính nó bằng không.

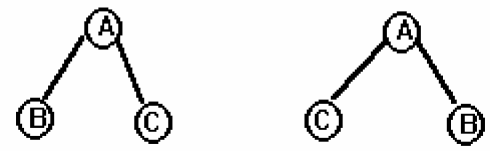
Nếu có đường đi từ nút a đến nút b thì ta nói a là *tiền bối* (ancestor) của b, còn b gọi là *hậu duệ* (descendant) của nút a. Rõ ràng *một nút vừa là tiền bối vừa là hậu duệ của chính nó*. Tiền bối hoặc hậu duệ của một nút khác với chính nó gọi là tiền bối hoặc hậu duệ thực sự. Trên cây *nút gốc* không có tiền bối thực sự. Một nút không có hậu duệ thực sự gọi là *nút lá* (leaf). Nút không phải là lá ta còn gọi là *nút trung gian* (interior). Cây con của một cây là một nút cùng với tất cả các hậu duệ của nó.

Chiều cao của một nút là độ dài đường đi lớn nhất từ nút đó tới lá. *Chiều cao của cây* là chiều cao của nút gốc. *Độ sâu của một nút* là độ dài đường đi từ nút gốc đến nút đó. Các nút có cùng một độ sâu i ta gọi là các nút có cùng một mức i. Theo định nghĩa này thì nút gốc ở mức 0, các nút con của nút gốc ở mức 1.

Ví dụ: đối với cây trong hình III.1 ta có nút C2 có chiều cao 2. Cây có chiều cao 3. nút C3 có chiều cao 0. Nút 2.1 có độ sâu 2. Các nút C1,C2,C3 cùng mức 1.

2. Thứ tự các nút trong cây

Nếu ta phân biệt thứ tự các nút con của cùng một nút thì cây gọi là cây có thứ tự, thứ tự qui ước từ trái sang phải. Như vậy, nếu kể thứ tự thì hai cây sau là hai cây khác nhau:

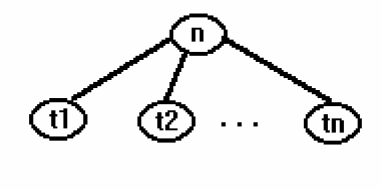


Hình III.2: Hai cây có thứ tự khác nhau

Trong trường hợp ta không phân biệt rõ ràng thứ tự các nút thì ta gọi là cây không có thứ tự. Các nút con cùng một nút cha gọi là các nút anh em ruột (siblings). Quan hệ "trái sang phải" của các anh em ruột có thể mở rộng cho hai nút bất kỳ theo qui tắc: nếu a, b là hai anh em ruột và a bên trái b thì các hậu duệ của a là "bên trái" mọi hậu duệ của b.

3. Các thứ tự duyệt cây quan trọng

Duyệt cây là một qui tắc cho phép đi qua lần lượt tất cả các nút của cây mỗi nút đúng một lần, danh sách liệt kê các nút (tên nút hoặc giá trị chứa bên trong nút) theo thứ tự đi qua gọi là danh sách duyệt cây. Có ba cách duyệt cây quan trọng: *Duyệt tiền tự* (preorder), *duyệt trung tự* (inorder), *duyệt hậu tự* (posorder). Có thể định nghĩa các phép duyệt cây tổng quát (xem hình III.3) một cách đệ qui như sau:



Hình III.3

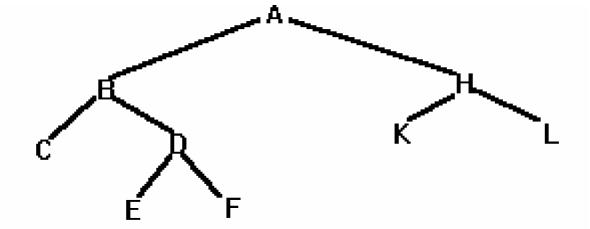
➢ Cây rỗng thì danh sách duyệt cây là rỗng và nó được coi là biểu thức duyệt tiền tự, trung tự, hậu tự của cây.

➢ Cây chỉ có một nút thì danh sách duyệt cây gồm chỉ một nút đó và nó được coi là biểu thức duyệt tiền tự, trung tự, hậu tự của cây.

➢ Ngược lại: giả sử cây T có nút gốc là n và có các cây con là T1,..,Tn thì:

- ✧ Biểu thức duyệt tiền tự của cây T là liệt kê nút n kế tiếp là biểu thức duyệt tiền tự của các cây T1, T2, .., Tn theo thứ tự đó.
- ✧ Biểu thức duyệt trung tự của cây T là biểu thức duyệt trung tự của cây T1 kế tiếp là nút n rồi đến biểu thức duyệt trung tự của các cây T2,.., Tn theo thứ tự đó.
- ✧ Biểu thức duyệt hậu tự của cây T là biểu thức duyệt hậu tự của các cây T1, T2,.., Tn theo thứ tự đó rồi đến nút n.

Ví dụ cho cây như trong hình III.4



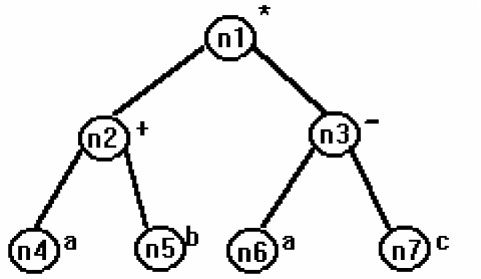
Hình III.4 Cây nhị phân

Biểu thức duyệt tiền tự: A B C D E F H K L
trung tự: C B E D F A K H L
hậu tự: C E F D B K L H A

4. Cây có nhãn và cây biểu thức

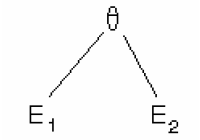
Ta thường lưu trữ kết hợp một nhãn (label) hoặc còn gọi là một giá trị (value) với một nút của cây. Như vậy nhãn của một nút không phải là tên nút mà là giá trị được lưu giữ tại nút đó. Nhãn của một nút đôi khi còn được gọi là khóa của nút, tuy nhiên hai khái niệm này là không đồng nhất. Nhãn là giá trị hay nội dung lưu trữ tại nút, còn khoá của nút có thể chỉ là một phần của nội dung lưu trữ này. Chẳng hạn, mỗi nút cây chứa một record về thông tin của sinh viên (mã SV, họ tên, ngày sinh, địa chỉ,...) thì khoá có thể là mã SV hoặc họ tên hoặc ngày sinh tuỳ theo giá trị nào ta đang quan tâm đến trong giải thuật.

Ví dụ: Cây biểu diễn biểu thức (a+b)*(a-c) như trong hình III.5.



Hình III.5: Cây biểu diễn biểu thức (a+b)*(a-c)

- Ở đây n_1, n_2,.., n_7 là các tên nút và *,+,-,a,b,c là các nhãn.
- Qui tắc biểu diễn một biểu thức toán học trên cây như sau:
- Mỗi nút lá có nhãn biểu diễn cho một toán hạng.
- Mỗi nút trung gian biểu diễn một toán tử.



Hình III.6: Cây biểu diễn biểu thức E1θ E2

- Giả sử nút n biểu diễn cho một toán tử hai ngôi θ (chẳng hạn + hoặc *), nút con bên trái biểu diễn cho biểu thức E1, nút con bên phải biểu diễn cho biểu thức E2 thì nút n biểu diễn biểu thức E1θ E2, xem hình III.6. Nếu θ là phép toán một ngôi thì nút chứa phép toán θ chỉ có một nút con, nút con này biểu diễn cho toán hạng của θ.
- Khi chúng ta duyệt một cây biểu diễn một biểu thức toán học và liệt kê nhãn của các nút theo thứ tự duyệt thì ta có:

  ❋ *Biểu thức dạng tiền tố* (prefix) tương ứng với phép *duyệt tiền tự* của cây.

  ❋ *Biểu thức dạng trung tố* (infix) tương ứng với phép *duyệt trung tự* của cây.

  ❋ *Biểu thức dạng hậu tố* (posfix) tương ứng với phép *duyệt hậu tự* của cây.

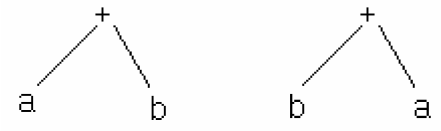
Ví dụ: đối với cây trong hình III.5 ta có:

- Biểu thức tiền tố: *+ab-ac
- Biểu thức trung tố: a+b*a-c

- Biểu thức hậu tố: ab+ac-*

Chú ý rằng

- Các biểu thức này không có dấu ngoặc.
- Các phép toán trong biểu thức toán học có thể có tính giao hoán nhưng khi ta biểu diễn biểu thức trên cây thì phải tuân thủ theo biểu thức đã cho. Ví dụ biểu thức a+b, với a,b là hai số nguyên thì rõ ràng a+b=b+a nhưng hai cây biểu diễn cho hai biểu thức này là khác nhau (vì cây có thứ tự).



Hình III.7 - Cây cho biểu thức a+b và b+a.

- Chỉ có cây ở phía bên trái của hình III.7 mới đúng là cây biểu diễn cho biểu thức a+b theo qui tắc trên.
- Nếu ta gặp một dãy các phép toán có cùng độ ưu tiên thì ta sẽ kết hợp từ trái sang phải. Ví dụ a+b+c-d = ((a+b)+c)-d.

II. KIỂU DỮ LIỆU TRỪU TƯỢNG CÂY

Để hoàn tất định nghĩa kiểu dữ liệu trừu tượng cây, cũng như đối với các kiểu dữ liệu trừu tượng khác, ta phải định nghĩa các phép toán trừu tượng cơ bản trên cây, các phép toán này được xem là các phép toán "nguyên thủy" để ta thiết kế các giải thuật sau này.

Các phép toán trên cây

- Hàm **PARENT(n,T)** cho nút cha của nút n trên cây T, nếu n là nút gốc thì hàm cho giá trị NULL. Trong cài đặt cụ thể thì NULL là một giá trị nào đó do ta chọn, nó phụ thuộc vào cấu trúc dữ liệu mà ta dùng để cài đặt cây.
- Hàm **LEFTMOST_CHILD(n,T)** cho nút con trái nhất của nút n trên cây T, nếu n là lá thì hàm cho giá trị NULL.
- Hàm **RIGHT_SIBLING(n,T)** cho nút anh em ruột phải nút n trên cây T, nếu n không có anh em ruột phải thì hàm cho giá trị NULL.
- Hàm **LABEL_NODE(n,T)** cho nhãn tại nút n của cây T.
- Hàm **ROOT(T)** trả ra nút gốc của cây T. Nếu Cây T rỗng thì hàm trả về NULL.
- Hàm **CREATEi(v,T1,T2,..,Ti)**,với i=0..n, thủ tục tạo cây mới có nút gốc là n được gán nhãn v và có i cây con T1,..,Ti. Nếu n= 0 thì thủ tục tạo cây mới chỉ gồm có

1 nút đơn độc là n có nhãn v. Chẳng hạn, giả sử ta có hai cây con T1 và T2, ta muốn thiết lập cây mới với nút gốc có nhãn là v thì lời gọi thủ tục sẽ là CREATE2(v,T1,T2).

III. CÀI ĐẶT CÂY

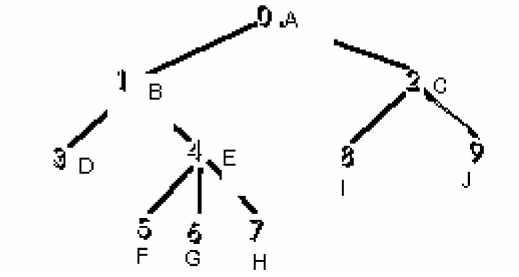
1. Cài đặt cây bằng mảng

Cho cây T có n nút, ta có thể gán tên cho các nút lần lượt là 0,1, 2, .., n-1. Sau đó ta dùng một mảng một chiều A để lưu trữ cây bằng cách cho A[i] = j với j là nút cha của nút i. Nếu i là nút gốc ta cho a[i] = -1 vì nút gốc không có cha.

Nếu cây T là cây có nhãn ta có thể dùng thêm một mảng một chiều thứ hai L chứa các nhãn của cây bằng cách cho L[i] = x với x là nhãn của nút i, hoặc khai báo mảng a là mảng của các struct có hai trường: trường Parent giữ chỉ số nút cha; trường Data giữ nhãn của nút và một trường MaxNode giữ số nút hiện tại đang có trên cây.
Với cách lưu trữ như thế, hàm PARENT(n,T) tốn chỉ một hằng thời gian trong khi các hàm đòi hỏi thông tin về các con không làm việc tốt vì phai tốn vòng lặp để dò tìm. Chẳng hạn cho một nút i tìm nút con trái nhất của nút i là không thể xác định được. Để cải thiện tình trạng này ta qui ước việc đặt tên cho các nút (đánh số thứ tự) như sau:

- Đánh số theo thứ tự tăng dần bắt đầu tại nút gốc.
- Nút cha được đánh số trước các nút con.
- Các nút con cùng một nút cha được đánh số lần lượt từ trái sang phải, chẳng hạn đánh số như cây trong hình III.8.

ví dụ:



Hình III.8 Hình ảnh một cây tổng quát

Cây trong hình III.8 được biểu diễn trong mảng như sau:

A	B	C	D	E	F	G	H	I	J	…	
-1	0	0	1	1	4	4	4	2	2	…	
0	1	2	3	4	5	6	7	8	9	…	

← Nhãn của các nút trên cây
← Cha của nút trên cây
← Chỉ số của mảng

Maxlength

MaxNode

Khai báo cấu trúc dữ liệu

```
#define MAXLENGTH ... /* chỉ số tối đa của mảng */
#define NIL -1
typedef  ... DataType;
typedef int Node;
typedef struct {
     /* Lưu trữ nhãn (dữ liệu) của nút trong cây */
     DataType Data[MAXLENGTH];
     /* Lưu trữ cha của các nút trong cây theo nguyên tắc:
Cha của nút i sẽ lưu ở vị trí i trong mảng */
     Node Parent[MAXLENGTH];
     /* Số nút thực sự trong cây */
     int MaxNode;} Tree;
 Tree T;
```

Sự lưu trữ như vậy còn gọi là sự lưu trữ kế tiếp và cách lưu trữ cây như trên, ta có thể viết được các phép toán cơ bản trên cây như sau

Khởi tạo cây rỗng:

```
void MakeNull_Tree (Tree *T){
  (*T).MaxNode=0;}
```

Kiểm tra cây rỗng

```
int EmptyTree(Tree T){
   return T.MaxNode == 0;
}
```

Xác định nút cha của nút trên cây

```
Node Parent(Node n,Tree T){
if (EmptyTree(T) || (n>T.MaxNode-1))
     return NIL;
else return T.Parent[n];
}
```

Xác định nhãn của nút trên cây

```
DataType Label_Node(Node n,Tree T){
if (!EmptyTree(T) && (n<=T.MaxNode-1))
   return T.Data[n];
}
```

Hàm xác định nút gốc trong cây

```
Node Root(Tree T){
if (!EmptyTree(T)) return 0;
else return NIL;
}
```

Hàm xác định con trái nhất của một nút

```
Node LeftMostChild(Node n,Tree T){
Node i;
int found;
if (n<0) return NIL;
i=n+1;/* Vị trí nút đầu tiên hy vọng là con của nút n */
found=0;
while ((i<=T.MaxNode-1) && !found)
  if (T.Parent[i]==n) found=1; /* Đã tìm thấy con trái nhất
của nút n */
```

```
    else i=i+1;
    if (found) return i;
    else return NIL;
  }
```

Hàm xác định anh em ruột phải của một nút

```
  Node RightSibling(Node n,Tree T){
  Node i,parent;
  int found;
  if (n<0) return NIL;
  parent=T.Parent[n];
  i=n+1;
  found=0;
  while ((i<=T.MaxNode-1) && !found)
     if (T.Parent[i]==parent) found=1;
     else i=i+1;
  if (found) return i;
  else return NIL;
  }
```

Thủ tục duyệt tiền tự

```
  void PreOrder(Node n,Tree T){
  Node i;
  printf("%c ",Label_Node(n,T));
  i=LeftMostChild(n,T);
  while (i!=NIL){
    PreOrder(i,T);
    i=RightSibling(i,T);
```

```
}
}
```

Thủ tục duyệt trung tự

```
void InOrder(Node n,Tree T){
Node i;
i=LeftMostChild(n,T);
if (i!=NIL) InOrder(i,T);
printf("%c ",Label_Node(n,T));
i=RightSibling(i,T);
while (i!=NIL){
    InOrder(i,T);
    i=RightSibling(i,T);
}
}
```

Thủ tục duyệt hậu tự

```
void PostOrder(Node n,Tree T){
Node i;
i=LeftMostChild(n,T);
while (i!=NIL){
  PostOrder(i,T);
  i=RightSibling(i,T);}
printf("%c ",Label_Node(n,T));
}
```

Ví dụ: Viết chương trình nhập dữ liệu vào cho cây từ bàn phím như tổng số nút trên cây; ứng với từng nút thì phải nhập nhãn của nút, cha của một nút. Hiển thị danh sách duyệt cây theo các phương pháp duyệt tiền tự, trung tự, hậu tự của cây vừa nhập.

Hướng giải quyết: Với những yêu cầu đặt ra như trên, chúng ta cần phải thiết kế một số chương trình con sau:
- Tạo cây rỗng MAKENULL(T)
- Nhập dữ liệu cho cây từ bàn phím READTREE(T). Trong đó có nhiều cách nhập dữ liệu vào cho một cây nhưng ở đây ta có thể thiết kế thủ tục này đơn giản như sau:

```
void ReadTree(Tree *T){
int i;
MakeNull_Tree(&*T);
//Nhập số nút của cây cho đến khi số nút nhập vào là hợp lệ
do {
   printf("Cay co bao nhieu nut?");
   scanf("%d",&(*T).MaxNode);
} while (((*T).MaxNode<1) || ((*T).MaxNode>MAXLENGTH));
printf("Nhap nhan cua nut goc ");
fflush(stdin);
scanf("%c",&(*T).Data[0]);
(*T).Parent[0]=NIL; /* nut goc khong co cha */
for (i=1;i<=(*T).MaxNode-1;i++){
   printf("Nhap cha cua nut %d ",i);
   scanf("%d",&(*T).Parent[i]);
   printf("Nhap nhan cua nut %d ",i);
   fflush(stdin);
   scanf("%c",&(*T).Data[i]);
}
}
```

- Hàm xác định con trái nhất của một nút LEFTMOST_CHILD(n,T). Hàm này được dựng trong phép duyệt cây.
- Hàm xác đinh anh em ruột phải của một nút RIGHT_SIBLING (n,T). Hàm này được dựng trong phép duyệt cây.
- Các chương trình con hiển thị danh sách duyệt cây theo các phép duyệt.

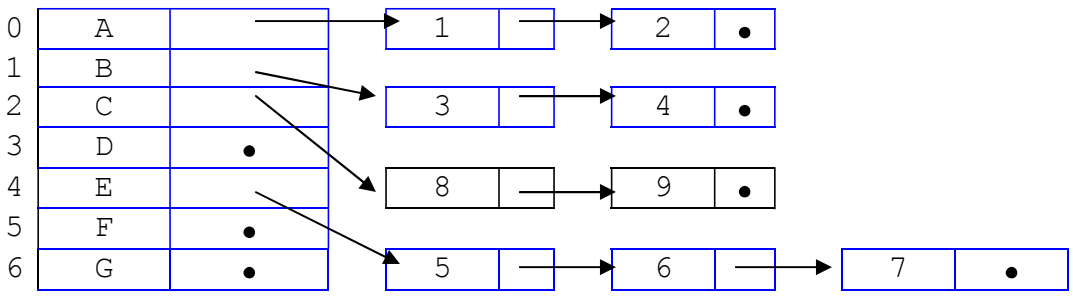
Với những chương trình con được thiết kế như trên, ta có thể tạo một chương trình chính để thực hiện theo yêu cầu đề bài như sau:

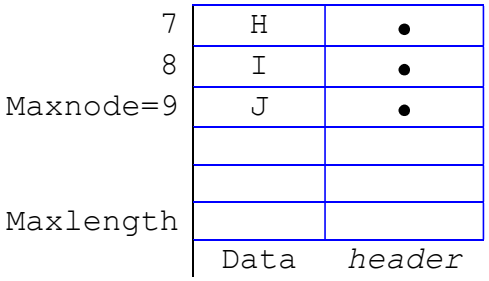
```
  void main(){
  printf("Nhap du lieu cho cay tong quat\n");
  ReadTree(&T);
  printf("Danh sach duyet tien tu cua cay vua nhap la\n");
  PreOrder(Root(T),T);
  printf("\nDanh sach duyet trung tu cua cay vua nhap la\n");
  InOrder(Root(T),T);
  printf("\nDanh sach duyet hau tu cua cay vua nhap la\n");
  PostOrder(Root(T),T);
  getch();
}
```

2. Biểu diễn cây bằng danh sách các con

Một cách biểu diễn khác cũng thường được dùng là biểu diễn cây dưới dạng mỗi nút có một danh sách các nút con. Danh sách có thể cài đặt bằng bất kỳ cách nào chúng ta đã biết, tuy nhiên vì số nút con của một nút là không biết trước nên dùng danh sách liên kết sẽ thích hợp hơn.

Ví dụ: Cây ở hình III.8 có thể lưu trữ dưới dạng như trong hình III.9





Hình III.9 Lưu trữ cây bằng danh sách các con

Có thể nhận xét rằng các hàm đòi hỏi thông tin về các con làm việc rất thuận lợi, nhưng hàm PARENT lại không làm việc tốt. Chẳng hạn tìm nút cha của nút 8 đòi hỏi ta phải duyệt tất cả các danh sách chứa các nút con.

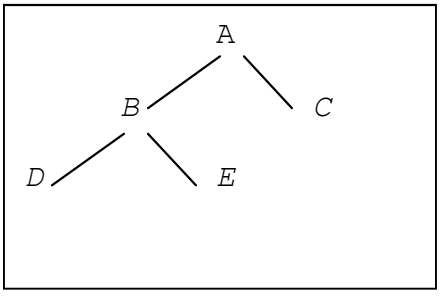
(Có thể tham khảo cách cài đặt chi tiết trong giáo trình "Thực tập Cấu trúc dữ liệu")

3. Biểu diễn theo con trái nhất và anh em ruột phải:

Các cấu trúc đã dùng để mô tả cây ở trên có một số nhược điểm, nó không trợ giúp phép tạo một cây lớn từ các cây nhỏ hơn, nghĩa là ta khó có thể cài đặt phép toán CREATEi bởi vì mỗi cây con đều có một mảng chứa các header riêng. Chẳng hạn CREATE2(v,T1,T2) chúng ta phải chép hai cây T1, T2 vào mảng thứ ba rồi thêm một nút n có nhãn v và hai nút con là gốc của T1 và T2. Vì vậy nếu chúng ta muốn thiết lập một cấu trúc dữ liệu trợ giúp tốt cho phép toán này thì tất cả các nút của các cây con phải ở trong cùng một vùng (một mảng). Ta thay thế mảng các header bằng mảng CELLSPACE chứa các struct có ba trường LABELS, LEFTMOST_CHILD, RIGHT_SIBLING. Trong đó LABELS giữ nhãn của nút, LEFTMOST_CHILD là một con nháy chỉ đến con trái nhất của nút, còn RIGHT_SIBLING là con nháy chỉ đến nút anh ruột phải. Hơn nữa mảng này giữ tất cả các nút của tất cả các cây.

Với cấu trúc này các phép toán đều thực hiện dễ dàng trừ PARENT, PARENT đòi hỏi phải duyệt toàn bộ mảng. Nếu chúng ta muốn cải tiến cấu trúc để trợ giúp phép toán này ta có thể thêm trường thứ 4 PARENT là một con nháy chỉ tới nút cha (xem hình III.11).

Để cài đặt cây theo cách này chúng ta cũng cần quản lí các ô trống theo cách tương tự như cài đặt danh sách bằng con nháy, tức là liên kết các ô trống vào một danh sách có chỉ điểm đầu là Availlable. Ở đây mỗi ô chứa 3 con nháy nên ta chỉ cần chọn 1 để trỏ đến ô kế tiếp trong danh sách, chẳng hạn ta chọn con nháy Right_sibling. Ví dụ cây trong hình III.10 có thể được cài đặt như trong hình III.11. Các ô được tô đậm là các ô trống, tức là các ô nằm trong danh sách Available.



Hình III.10 Hình ảnh cây tổng quát

	Chỉ số	Data	Leftmost_child	Right_Sibling	Parent
	1	D	null	4	3
Available →	2			8	
	3	B	1	7	5
	4	E	null	null	3
Root →	5	A	3	null	null
	6			null	
	7	C	null	null	3
	8			6	

Hình III.11

(có thể tham khảo cách cài đặt chi tiết trong giáo trình "Thực tập Cấu trúc dữ liệu")

4. Cài đặt cây bằng con trỏ

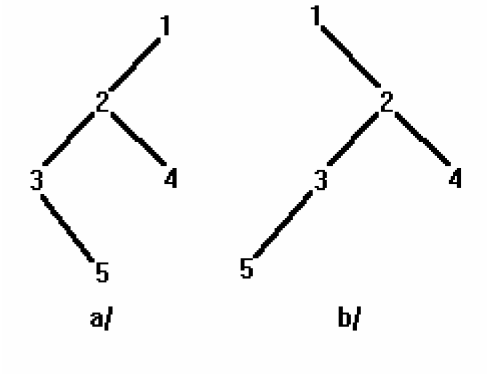
Hoàn toàn tương tự như cài đặt ở trên nhưng các con nháy Leftmost_child, Right_sibling và Parent được thay bằng các con trỏ.

? Hãy so sách các ưu khuyết điểm của các cách cài đặt cây.

IV. CÂY NHỊ PHÂN (BINARY TREES)

1. Định nghĩa

Cây nhị phân là cây rỗng hoặc là cây mà mỗi nút có tối đa hai nút con. Hơn nữa các nút con của cây được phân biệt thứ tự rõ ràng, một nút con gọi là nút con trái và một nút con gọi là nút con phải. Ta qui ước vẽ nút con trái bên trái nút cha và nút con phải bên phải nút cha, mỗi nút con được nối với nút cha của nó bởi một đoạn thẳng. Ví dụ các cây trong hình III.12.



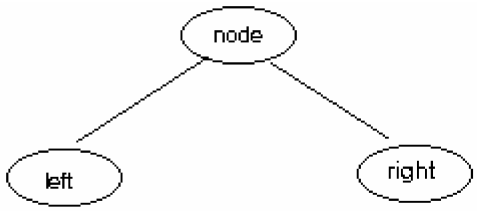
Hình III.12: Hai cây có thứ tự giống nhau nhưng là hai cây nhị phân khác nhau

Chú ý rằng, trong cây nhị phân, một nút con chỉ có thể là nút con trái hoặc nút con phải, nên có những cây có thứ tự giống nhau nhưng là hai cây nhị phân khác nhau. Ví dụ hình III.12 cho thấy hai cây có thứ tự giống nhau nhưng là hai cây nhị phân khác nhau. Nút 2 là nút con trái của cây a/ nhưng nó là con phải trong cây b/. Tương tự nút 5 là con phải trong cây a/ nhưng nó là con trái trong cây b/.

2. Duyệt cây nhị phân

Ta có thể áp dụng các phép duyệt cây tổng quát để duyệt cây nhị phân. Tuy nhiên vì cây nhị phân là cấu trúc cây đặc biệt nên các phép duyệt cây nhị phân cũng đơn giản hơn. Có ba cách duyệt cây nhị phân thường dùng (xem kết hợp với hình III.13):

- ***Duyệt tiền tự (Node-Left-Right):*** duyệt nút gốc, duyệt tiền tự con trái rồi duyệt tiền tự con phải.
- ***Duyệt trung tự (Left-Node-Right):*** duyệt trung tự con trái rồi đến nút gốc sau đó là duyệt trung tự con phải.

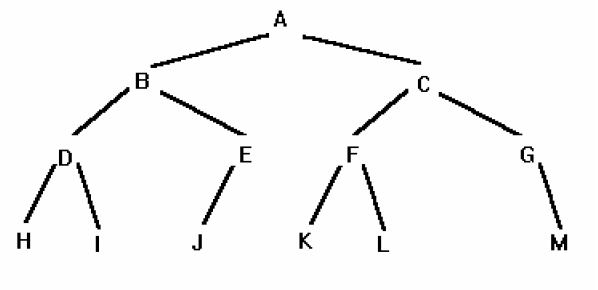


Hình III.13

- ***Duyệt hậu tự (Left-Right-Node)***: duyệt hậu tự con trái rồi duyệt hậu tự con phải sau đó là nút gốc.

Chú ý rằng danh sách duyệt tiền tự, hậu tự của cây nhị phân trùng với danh sách duyệt tiền tự, hậu tự của cây đó khi ta áp dụng phép duyệt cây tổng quát. Nhưng danh sách duyệt trung tự thì khác nhau.

Ví dụ



Hình III.14

	Các danh sách duyệt cây nhị phân	Các danh sách duyệt cây tổng quát
Tiền tự:	ABDHIEJCFKLGM	ABDHIEJCFKLGM
Trung tự:	HDIBJEAKFLC**GM**	HDIBJEAKFLC**MG**
Hậu tự:	HIDJEBKLFMGCA	HIDJEBKLFMGCA

?

1. Danh sách duyệt tiền tự và hậu tự của cây nhị phân luôn luôn giống với danh sách duyệt của cây tổng quát. (Đúng / Sai)
2. Danh sách duyệt trung tự của cây nhị phân sẽ khác với các duyệt tổng quát chỉ khi cây nhị phân bị khuyết con trái? (Đúng/ Sai)

3. Cài đặt cây nhị phân

Tương tự cây tổng quát, ta cũng có thể cài đặt cây nhị phân bằng con trỏ bằng cách thiết kế mỗi nút có hai con trỏ, một con trỏ trỏ nút con trái, một con trỏ trỏ nút con phải, trường Data sẽ chứa nhãn của nút.

```
typedef … TData;

typedef struct TNode{TData Data;

                     TNode* left,right;

                    };

typedef TNode* TTree;
```

Với cách khai báo như trên ta có thể thiết kế các phép toán cơ bản trên cây nhị phân như sau :

Tạo cây rỗng

Cây rỗng là một cây là không chứa một nút nào cả. Như vậy khi tạo cây rỗng ta chỉ cần cho cây trỏ tới giá trị NULL.

```
void MakeNullTree(TTree *T){
      (*T)=NULL;
}
```

Kiểm tra cây rỗng

```
int EmptyTree(TTree T){
   return T==NULL;
}
```

Xác định con trái của một nút

```
TTree LeftChild(TTree n){
   if (n!=NULL) return n->left;
   else return NULL;
}
```

Xác định con phải của một nút

```
TTree RightChild(TTree n){
   if (n!=NULL) return n->right;
   else return NULL;
}
```

Kiểm tra nút lá:

Nếu nút là nút lá thì nó không có bất kỳ một con nào cả nên khi đó con trái và con phải của nó cùng bằng nil

```
int IsLeaf(TTree n){
```

```
    if(n!=NULL)
      return(LeftChild(n)==NULL)&&(RightChild(n)==NULL);
    else return NULL;
}
```

Xác định số nút của cây

```
int nb_nodes(TTree T){
 if(EmptyTree(T)) return 0;
 else return 1+nb_nodes(LeftChild(T))+
               nb_nodes(RightChild(T));
}
```

Tạo cây mới từ hai cây có sẵn

```
TTree Create2(Tdata v,TTree l,TTree r){
TTree N;
N=(TNode*)malloc(sizeof(TNode));
N->Data=v;
N->left=l;
N->right=r;
return N;
}
```

Các thủ tục duyệt cây: tiền tự, trung tự, hậu tự

Thủ tục duyệt tiền tự

```
void PreOrder(TTree T){
    printf("%c ",T->Data);
   if (LeftChild(T)!=NULL) PreOrder(LeftChild(T));
```

```
    if (RightChild(T)!=NULL)PreOrder(RightChild(T));
}
```

Thủ tục duyệt trung tự

```
void InOrder(TTree T){
   if (LeftChild(T)=!NULL)InOrder(LeftChild(T));
   printf("%c ",T->data);
   if (RightChild(T)!=NULL) InOrder(RightChild(T));
}
```

Thủ tục duyệt hậu tự

```
void PosOrder(TTree T){
   if (LeftChild(T)!=NULL) PosOrder(LeftChild(T));
   if (RightChild(T)!=NULL)PosOrder(RightChild(T));
   printf("%c ",T->data);
}
```

?

Hãy biểu diễn cách gọi hàm Create2 để tạo một cây nhị phân cho trước.

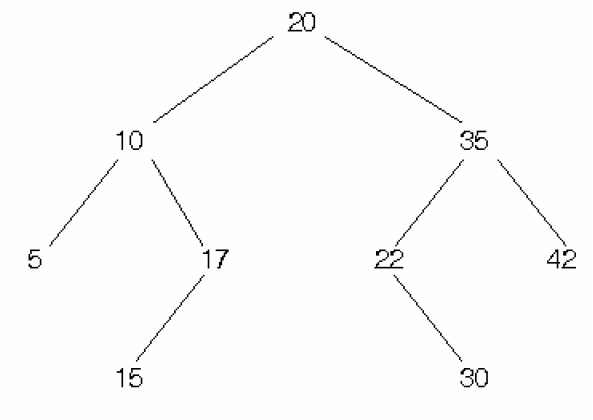
V. CÂY TÌM KIẾM NHỊ PHÂN (BINARY SEARCH TREES)

1. Định nghĩa

Cây tìm kiếm nhị phân (TKNP) là cây nhị phân mà khoá tại mỗi nút cây lớn hơn khoá của tất cả các nút thuộc cây con bên trái và nhỏ hơn khoá của tất cả các nút thuộc cây con bên phải.

Lưu ý: dữ liệu lưu trữ tại mỗi nút có thể rất phức tạp như là một record chẳng hạn, trong trường hợp này khoá của nút được tính dựa trên một trường nào đó, ta gọi là trường khoá. Trường khoá phải chứa các giá trị có thể so sánh được, tức là nó phải lấy giá trị từ một tập hợp có thứ tự.

Ví dụ: hình III.15 minh hoạ một cây TKNP có khoá là số nguyên (với quan hệ thứ tự trong tập số nguyên).



Hình III.15: Ví dụ cây tìm kiếm nhị phân

Qui ước: Cũng như tất cả các cấu trúc khác, ta coi cây rỗng là cây TKNP

Nhận xét:

- Trên cây TKNP không có hai nút cùng khoá.
- Cây con của một cây TKNP là cây TKNP.
- Khi duyệt trung tự (InOrder) cây TKNP ta được một dãy có thứ tự tăng. Chẳng hạn duyệt trung tự cây trên ta có dãy: 5, 10, 15, 17, 20, 22, 30, 35, 42.

2. Cài đặt cây tìm kiếm nhị phân

Cây TKNP, trước hết, là một cây nhị phân. Do đó ta có thể áp dụng các cách cài đặt như đã trình bày trong phần cây nhị phân. Sẽ không có sự khác biệt nào trong việc cài đặt cấu trúc dữ liệu cho cây TKNP so với cây nhị phân, nhưng tất nhiên, sẽ có sự khác biệt trong các giải thuật thao tác trên cây TKNP như tìm kiếm, thêm hoặc xoá một nút trên cây TKNP để luôn đảm bảo tính chất cuả cây TKNP.

Một cách cài đặt cây TKNP thường gặp là cài đặt bằng con trỏ. Mỗi nút của cây như là một mẩu tin (record) có ba trường: một trường chứa khoá, hai trường kia là hai con trỏ trỏ đến hai nút con (nếu nút con vắng mặt ta gán con trỏ bằng NIL)

Khai báo như sau

```
typedef <kiểu dữ liệu của khoá> KeyType;
typedef struct Node{KeyType Key;
                    Node* Left,Right;}
typedef Node* Tree;
```

Khởi tạo cây TKNP rỗng

Ta cho con trỏ quản lý nút gốc (Root) của cây bằng NIL.

```
void  MakeNullTree(Tree *Root){
   (*Root)=NULL;
 }
```

Tìm kiếm một nút có khóa cho trước trên cây TKNP

Để tìm kiếm 1 nút có khoá x trên cây TKNP, ta tiến hành từ nút gốc bằng cách so sánh khoá của nút gốc với khoá x.

- Nếu nút gốc bằng NULL thì không có khoá x trên cây.
- Nếu x bằng khoá của nút gốc thì giải thuật dừng và ta đã tìm được nút chứa khoá x.
- Nếu x lớn hơn khoá của nút gốc thì ta tiến hành (một cách đệ qui) việc tìm khoá x trên cây con bên phải.
- Nếu x nhỏ hơn khoá của nút gốc thì ta tiến hành (một cách đệ qui) việc tìm khoá x trên cây con bên trái.

Ví dụ: tìm nút có khoá 30 trong cây ở trong hình III.15

- So sánh 30 với khoá nút gốc là 20, vì 30 > 20 vậy ta tìm tiếp trên cây con bên phải, tức là cây có nút gốc có khoá là 35.
- So sánh 30 với khoá của nút gốc là 35, vì 30 < 35 vậy ta tìm tiếp trên cây con bên trái, tức là cây có nút gốc có khoá là 22.
- So sánh 30 với khoá của nút gốc là 22, vì 30 > 22 vậy ta tìm tiếp trên cây con bên phải, tức là cây có nút gốc có khoá là 30.
- So sánh 30 với khoá nút gốc là 30, 30 = 30 vậy đến đây giải thuật dừng và ta tìm được nút chứa khoá cần tìm.

Hàm dưới đây trả về kết quả là con trỏ trỏ tới nút chứa khoá x hoặc NULL nếu không tìm thấy khoá x trên cây TKNP.

```
Tree Search(KeyType x,Tree Root){
   if (Root == NULL) return NULL;  //không tìm thấy khoá x
   else if (Root->Key == x) /* tìm thấy khoá x */
        return Root;
        else if (Root->Key < x) //tìm tiếp trên cây bên phải
```

```
            return Search(x,Root->right);
        else
            {tìm tiếp trên cây bên trái}
            return Search(x,Root->left);
}
```

?

Cây tìm kiếm nhị phân được tổ chức như thế nào để quá trình tìm kiếm được hiệu quả nhất?

Nhận xét: *giải thuật này sẽ rất hiệu quả về mặt thời gian nếu cây TKNP được tổ chức tốt, nghĩa là cây tương đối "cân bằng". Về chủ đề cây cân bằng các bạn có thể tham khảo thêm trong các tài liệu tham khảo của môn này.*

Thêm một nút có khóa cho trước vào cây TKNP

Theo định nghĩa cây tìm kiếm nhị phân ta thấy trên cây tìm kiếm nhị phân không có hai nút có cùng một khoá. Do đó nếu ta muốn thêm một nút có khoá x vào cây TKNP thì trước hết ta phải tìm kiếm để xác định có nút nào chứa khoá x chưa. Nếu có thì giải thuật kết thúc (không làm gì cả!). Ngược lại, sẽ thêm một nút mới chứa khoá x này. Việc thêm một khoá vào cây TKNP là việc tìm kiếm và thêm một nút, tất nhiên, phải đảm bảo cấu trúc cây TKNP không bị phá vỡ. Giải thuật cụ thể như sau:

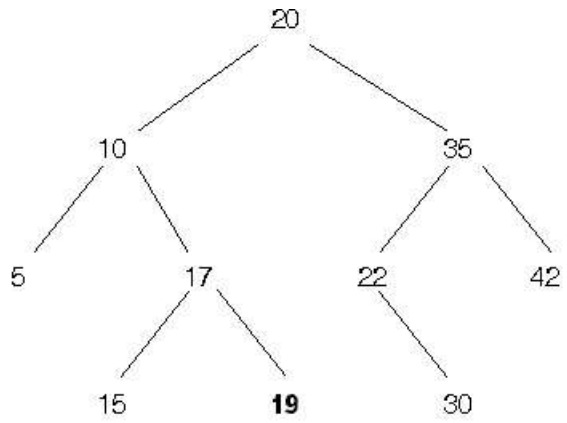
Ta tiến hành từ nút gốc bằng cách so sánh khóa cuả nút gốc với khoá x.

- Nếu nút gốc bằng NULL thì khoá x chưa có trên cây, do đó ta thêm một nút mới chứa khoá x.
- Nếu x bằng khoá của nút gốc thì giải thuật dừng, trường hợp này ta không thêm nút.
- Nếu x lớn hơn khoá của nút gốc thì ta tiến hành (một cách đệ qui) giải thuật này trên cây con bên phải.
- Nếu x nhỏ hơn khoá của nút gốc thì ta tiến hành (một cách đệ qui) giải thuật này trên cây con bên trái.

Ví dụ: thêm khoá 19 vào cây ở trong hình III.15

- So sánh 19 với khoá của nút gốc là 20, vì 19 < 20 vậy ta xét tiếp đến cây bên trái, tức là cây có nút gốc có khoá là10.

- So sánh 19 với khoá của nút gốc là 10, vì 19 > 10 vậy ta xét tiếp đến cây bên phải, tức là cây có nút gốc có khoá là 17.
- So sánh 19 với khoá của nút gốc là 17, vì 19 > 17 vậy ta xét tiếp đến cây bên phải. Nút con bên phải bằng NULL, chứng tỏ rằng khoá 19 chưa có trên cây, ta thêm nút mới chứa khoá 19 và nút mới này là con bên phải của nút có khoá là 17, xem hình III.16



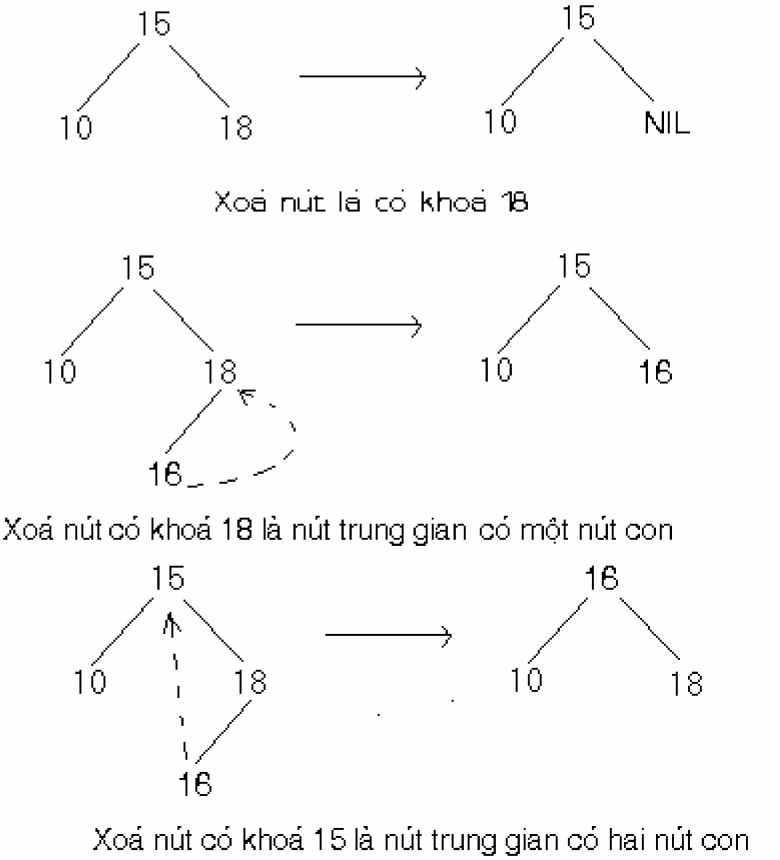
Hình III.16: Thêm khoá 19 vào cây hình III.15

Thủ tục sau đây tiến hành việc thêm một khoá vào cây TKNP.

```
void InsertNode(KeyType x,Tree *Root ){
    if (*Root == NULL){ /* thêm nút mới chứa khoá x */
        (*Root)=(Node*)malloc(sizeof(Node));
        (*Root)->Key = x;
        (*Root)->left = NULL;
        (*Root)->right = NULL;
    }
   else
    if (x < (*Root)->Key) InsertNode(x,Root->left);
     else if (x>(*Root)->Key)InsertNode(x,Root->right);
}
```

Xóa một nút có khóa cho trước ra khỏi cây TKNP

Giả sử ta muốn xoá một nút có khoá x, trước hết ta phải tìm kiếm nút chứa khoá x trên cây.



Việc xoá một nút như vậy, tất nhiên, ta phải bảo đảm cấu trúc cây TKNP không bị phá vỡ. Ta có các trường hợp như hình III.17:

Hình III.17 Ví dụ về giải thuật xóa nút trên cây

- Nếu không tìm thấy nút chứa khoá x thì giải thuật kết thúc.
- Nếu tìm gặp nút N có chứa khoá x, ta có ba trường hợp sau (xem hình III.17)

· Nếu N là lá ta thay nó bởi NULL.

· N chỉ có một nút con ta thay nó bởi nút con của nó.

· N có hai nút con ta thay nó bởi nút lớn nhất trên cây con trái của nó (nút cực phải của cây con trái) hoặc là nút bé nhất trên cây con phải của nó (nút cực trái của cây con phải). Trong giải thuật sau, ta thay x bởi khoá của nút cực trái của cây con bên phải rồi ta xoá nút cực trái này. Việc xoá nút cực trái của cây con bên phải sẽ rơi vào một trong hai trường hợp trên.

Giải thuật xoá một nút có khoá nhỏ nhất

Hàm dưới đây trả về khoá của nút cực trái, đồng thời xoá nút này.

```
KeyType DeleteMin (Tree *Root ){
   KeyType k;
   if ((*Root)->left == NULL){
        k=(*Root)->key;
        (*Root) = (*Root)->right;
        return k;
   }
   else return DeleteMin(Root->left);
}
```

Thủ tục xóa một nút có khoá cho trước trên cây TKNP

```
void DeleteNode(key X,Tree *Root){
  if ((*Root)!=NULL)
    if (x < (*Root)->Key) DeleteNode(x,Root->left)
    else if (x > (*Root)->Key)  DeleteNode(x,Root->right)
     else
      if ((*Root)->left==NULL)&&((*Root)->right==NULL)
        (*Root)=NULL;
       else
        if ((*Root)->left == NULL) (*Root) = (*Root)->right
       else
         if ((*Root)->right==NULL) (*Root) = (*Root)->left
          else (*Root)->Key = DeleteMin(Root->right);
}
```

TỔNG KẾT CHƯƠNG

Chương này giới thiệu một số khái niệm cơ bản về cây tổng quát, cây nhị phân và cây tiềm kiếm nhị phân. Bên cạnh đó, chương này cũng đề cập đến cách lưu trữ cây trong bộ nhớ như cài đặt cây bằng mảng, con trỏ, danh sách các con, con trái nhất, anh em ruột phải và cách cài đặt các phép toán cơ bản trên các dạng cây khác nhau theo từng cách cài đặt.