CHƯƠNG 6: CÂY

Mục tiêu của chương

- Định nghĩa và các khái niệm về cây.

- Cách cài đặt cây: cài đặt bằng mảng hoặc danh sách liên kết.

- Cách phép duyệt cây: duyệt thứ tự trước, thứ tự giữa, và thứ tự sau. Ngoài ra, còn giới thiệu một số loại cây đặc biệt, có nhiều ứng dụng trong thực tiễn đó là cây nhị phân, cây nhị phân tìm kiếm và cây nhị phân tìm kiếm cân bằng AVL.

- Vận dụng lý thuyết giải các bài tập duyệt cây.

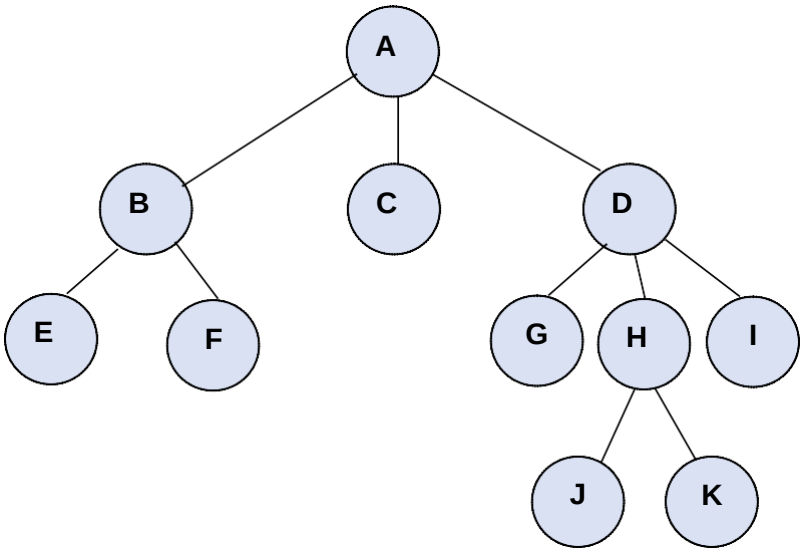
Nội dung của chương

Nghiên cứu các bước thực hiện các thuật toán trên các loại cây.

6.1. Các khái niệm cơ bản

Định nghĩa cây: là một tập hợp các phần tử (gọi là nút của cây), trong đó có một nút đặc biệt gọi là gốc. Các nút còn lại được chia thành các tập rời nhau T_1, T_2 , ..., T_n theo quan hệ phân cấp trong T, cũng là một cây. Mỗi nút ở cấp i sẽ quản lý một số nút ở cấp i+1, đây gọi là quan hệ cha con.

Ví dụ 6.1: Minh họa hình ảnh của một cây



Hình 6.1: Hình ảnh cây

Trong đó:

- Nút A gọi là nút gốc của cây.
- Các nút B, C, D là gốc của các cây con của A.
- Nút A là cha của các nút B, C, D và B, C, D là các nút con của A.

Một số khái niệm cơ bản

▪ *Bậc của một nút:*

Là số cây con của nút đó.

Ví dụ cây trong hình 6.1 thì bậc của A là 3, bậc của H là 2, bậc của D là 3.

▪ *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.

Ví dụ bậc của cây là 3 (hình 6.1).

▪ *Nút gốc:*

Là nút không có nút cha.

Ví dụ A là nút gốc của cây trong hình 6.1.

▪ *Nút lá:*

Là nút có bậc bằng 0.

Ví dụ E, C, K, I,… là các nút lá (hình 6.1).

▪ *Nút nhánh:*

Là nút có bậc khác 0 và không phải là gốc .

Ví dụ B, D, H…là các nút nhánh (hình 6.1).

▪ *Mức của một nút:*

Mức (gốc (T)) = 0.

Gọi T_1, T_2, T_3, ... , Tn là các cây con của T_0

Mức (T_1) = mức (T_2) = ... = mức (Tn) = mức (T_0) + 1.

Ví dụ nút A có mức là 0, D có mức là 1, G có mức là 2, J có mức là 3 (hình 6.1).

▪ *Độ dài đường đi từ gốc đến nút x:*

Bằng số nút trên đường đi đó trừ đi 1.

Ví dụ đường đi từ A đến G là 2, đường đi từ A đến K là 3 (hình 6.1).

▪ *Chiều cao của một cây:*

Là số mức lớn nhất của nút có trên cây đó cộng thêm 1.

Ví dụ cây trên có chiều cao là 4 (hình 6.1).

Một số ví dụ về cấu trúc cây như: sơ đồ tổ chức của một công ty, mục lục của một quyển sách, cấu trúc một cây thư mục, …

6.2. Cây nhị phân

6.2.1. Định nghĩa và các tính chất

Định nghĩa:

- Cây nhị phân là một dạng quan trọng của cấu trúc cây. Nó có đặc điểm là mọi nút trên cây chỉ có tối đa 2 nút con.

Đối với các cây con của một nút người ta phân biệt cây con trái và cây con phải. Như vậy, cây nhị phân là một cây có thứ tự.

- Một số dạng đặc biệt của cây nhị phân:

+ Cây nhị phân lệch trái.

+ Cây nhị phân lệch phải.

+ Cây ziczăc

- Cây nhị phân hoàn chỉnh là cây mà số nút trên mọi mức, trừ mức cuối cùng đều đạt tối đa.

- Cây nhị phân đầy đủ là cây mà số nút đạt tối đa trên mọi mức. Nó là trường hợp đặc biệt của cây nhị phân hoàn chỉnh.

Trong các cây nhị phân có cùng số lượng nút thì cây nhị phân suy biến có chiều cao lớn nhất, còn cây nhị phân hoàn chỉnh có chiều cao nhỏ nhất.

Bổ đề:

1. Số các nút tối đa ở mức i của cây nhị phân là 2^{i-1} ($i \geq 1$).

2. Số các nút tối đa của cây nhị phân có chiều cao h là $2^h - 1$ ($h \geq 1$)

Chứng minh:

1. Sẽ được chứng minh bằng quy nạp

Bước cơ sở: Với i = 1, cây nhị phân có tối đa $1 = 2^0$ nút.

Vậy mệnh đề đúng với i = 1

Bước quy nạp: Giả sử kết quả đúng với mức i, nghĩa là ở mức này cây nhị phân có tối đa 2^{i-1} nút, ta chứng minh mệnh đề đúng với mức i + 1.

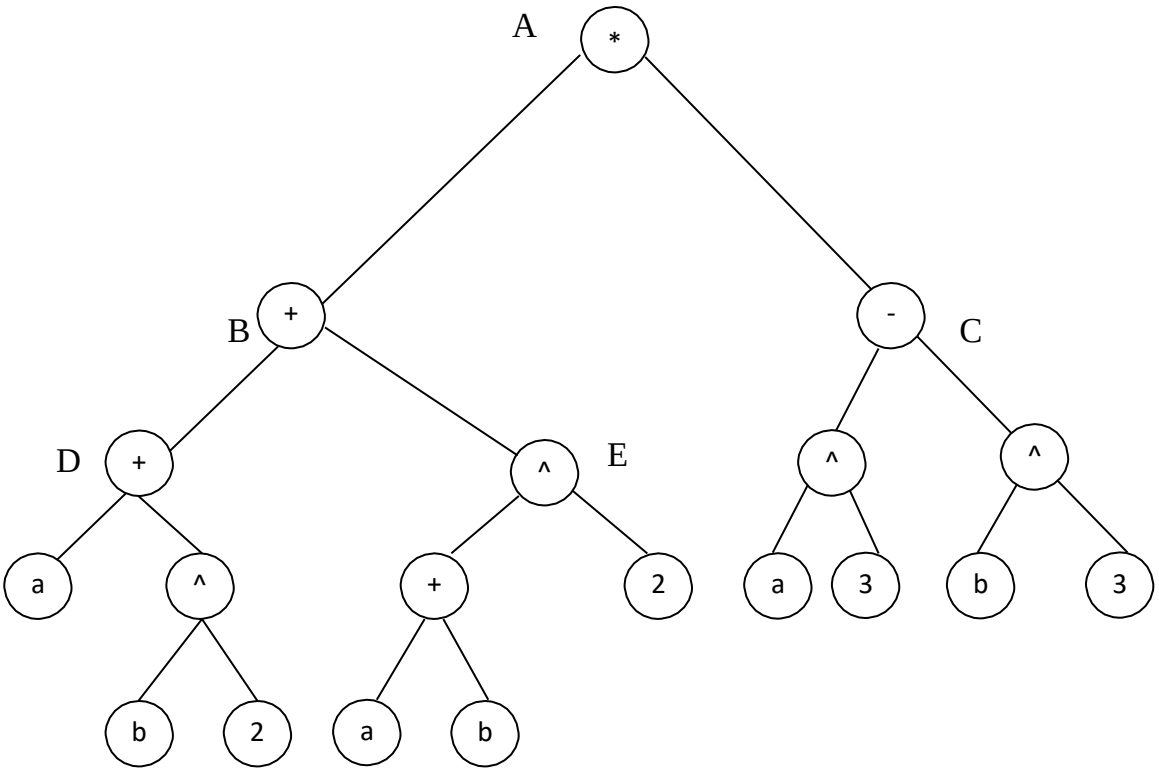
Theo định nghĩa cây nhị phân thì tại mỗi nút có tối đa hai cây con nên mỗi nút ở mức i có tối đa hai con. Do đó theo giả thiết quy nạp ta suy ra tại mức i+ 1 ta có:

$$2^{i-1} \times 2 = 2^i \text{ nút.}$$

2. Ta đã biết rằng chiều cao của cây là số mức lớn nhất có trên cây đó. Theo i) ta suy ra số nút tối đa có trên cây nhị phân với chiều cao h là :

$$2^0 + 2^1 + \ldots + 2^{h-1} = 2^{h-1}.$$

Cây nhị phân có thể ứng dụng trong nhiều bài toán thông dụng. Ví dụ cây nhị phân dưới đây cho ta hình ảnh của một biểu thức toán học:



Hình 6.2: Hình ảnh cây nhị phân minh họa biểu thức toán học

Các tính chất:

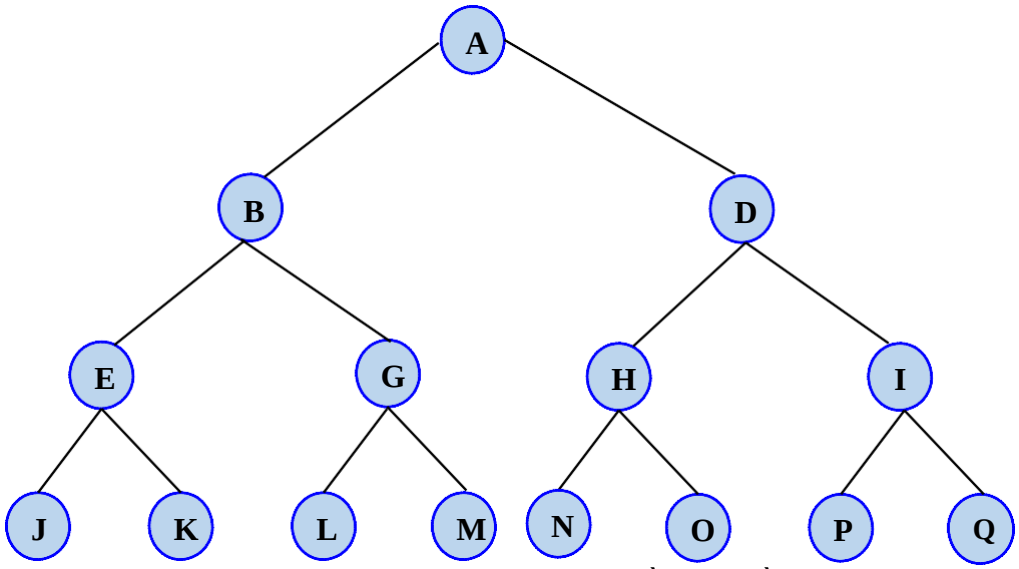
- Số nút nằm ở mức i $\leq 2^i$.
- Với h là chiều cao của cây thì

  Số nút lá $\leq 2^{h-1}$

  Thì $\log_2$(số nút trong cây) $\leq$

  h Số nút trong cây $\leq 2^h - 1$.

Ví dụ cây nhị phân có chiều cao là 4 hình 21:



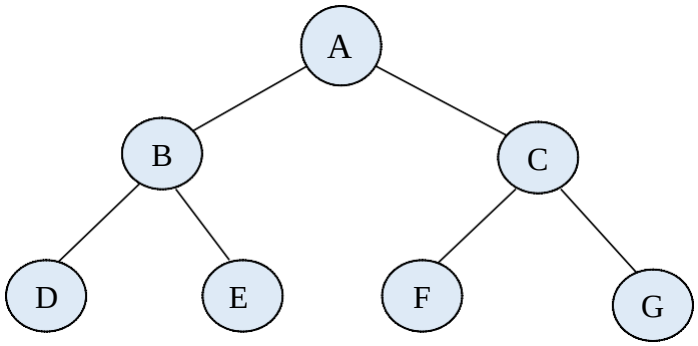
Hình 6.3: Cây nhị phân có chiều cao bằng 4

6.2.2. Các cách biểu diễn cây nhị phân

a. Lưu trữ kế tiếp

Nếu có một cây nhị phân hoàn chỉnh (ưu tiên trái) hoặc đầy đủ ta có thể đánh số thứ tự cho các nút bắt đầu từ 1 kể từ mức 1 trở đi, hết mức này đến mức khác, trên mỗi mức các nút được đánh số kể từ trái qua phải.

Ví dụ 6.2a: Cho cây



Hình 6.4: Cây nhị phân đơn giản

Khi đó ta có:

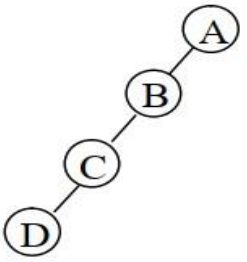
- Các nút con của nút i có chỉ số thứ tự là 2i và 2i + 1.
- Nút cha của nút j là nút có chỉ số thứ tự [j/2] (j div 2).

Như vậy với các đánh số ở trên ta có thể lưu trữ cây bằng một vectơ lưu trữ V theo nguyên tắc nút thứ i được lưu trữ ở V[i], kí hiệu: V: i → V[i]. Cách lưu trữ như vậy gọi là lưu trữ kế tiếp của cây nhị phân. Với cách lưu trữ này ta có thể biết được địa chỉ các nút con của một nút và ngược lại

Ví dụ 6.2b: Với cây đã cho ở trên hình ảnh lưu trữ kế tiếp là:

A	B	C	D	E	F	G
v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]

Nếu cây nhị phân không đầy đủ thì cách lưu trữ này không thích hợp vì sẽ gây ra lãng phí bộ nhớ do có nhiều phần tử bỏ trống (ứng với cây con rỗng). Ta hãy xét cây như hình dưới. Để lưu trữ cây này ta phải dùng mảng gồm 31 phần tử mà chỉ có 5 phần tử khác rỗng; hình ảnh lưu trữ miền nhớ của cây này như sau



Hình 6.5: Cây nhị phân đặc biệt

A	**B**	∅	**C**	∅	∅	∅	**D**	∅	∅	∅	∅	∅	∅	∅	**E**	∅	**…**

(∅ : chỉ chỗ trống)

Nếu cây nhị phân luôn biến động nghĩa là có phép bổ sung, loại bỏ các nút thường xuyên tác động thì cách lưu trữ này gặp phải một số nhược điểm như tốn thời gian khi phải thực hiện các thao tác này, độ cao của cây phụ thuộc vào kích thước của mảng...

b. Lưu trữ móc nối

Cách lưu trữ này khắc phục được các nhược điểm của cách lưu trữ trên đồng thời phản ánh được dạng tự nhiên của cây.

Trong cách lưu trữ này mỗi nút tương ứng với một phần tử nhớ có quy cách như sau:

left	info	right

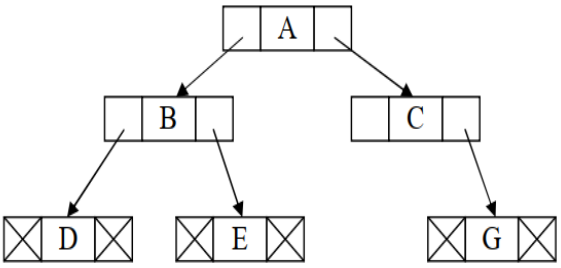
Trường info ứng với thông tin (dữ liệu) của nút

Trường left ứng với con trỏ, trỏ tới cây con trái của nút đó

Trường right ứng với con trỏ, trỏ tới cây con phải của nút

Nếu các con trỏ của các nút không trỏ đến nút con nào cả thì có giá trị Null. Ngoài ra người ta thường sử dụng thêm một biến trỏ T trỏ đến nút gốc của cây và quy ước cây rỗng khi và chỉ khi T = Null.

Ví dụ 6.2c: Hình ảnh lưu trữ móc nối của cây nhị phân ở trên là:



Hình 6.6: Hình ảnh lưu trữ móc nối của cây nhị phân

Với cách lưu trữ móc nối ta dễ dàng truy nhập từ nút cha đến nút con, tuy nhiên điều ngược lại thì khó có thể thực hiện được.

Khai báo như sau:

```
struct node
    {
        int Info;
        struct node *left,*right;
    };
typedef node *tree;
```

tree root;

Do tính chất mềm dẻo của cách biểu diễn bằng cấp phát liên kết, phương pháp này được dùng chủ yếu trong biểu diễn cây nhị phân. Khi nói về cây nhị phân, chúng ta sẽ dùng phương pháp biểu diễn này.

6.2.3. Các phép duyệt cây nhị phân

Có nhiều kiểu duyệt cây khác nhau, và chúng cũng có những ứng dụng khác nhau. Đối với cây nhị phân, do cấu trúc đệ quy của nó, việc duyệt cây tiếp cận theo kiểu đệ quy là hợp lý và đơn giản nhất. Một số kiểu duyệt thông dụng.

Có 3 kiểu duyệt chính có thể áp dụng trên cây nhị phân: duyệt theo thứ tự trước (NLR), thứ tự giữa (LNR) và thứ tự sau (LRN). Tên của 3 kiểu duyệt này được đặt dựa trên trình tự của việc thăm nút gốc so với việc thăm 2 cây con.

6.2.3.1. Duyệt theo thứ tự trước (Node-Left-Right)

Kiểu duyệt này trước tiên thăm nút gốc sau đó thăm các nút của cây con trái rồi đến cây con phải. Thủ tục duyệt có thể trình bày đơn giản như sau: (trong mục này việc xử lý các nút chỉ là in giá trị của phần tử đó)

Duyệt cây theo thứ tự trước (Node-Left-Right):

```
void NLR(Tree T ) {
        if (T!=NULL ) {
                <Thăm node>;
                NLR( T → left); //duyệt thứ tự giữa sang nhánh cây con bên trái
                NLR( T → right); /duyệt thứ tự giữa sang nhánh cây con bên tphải
        }
}
```

6.2.3.2. Duyệt theo thứ tự giữa (Left- Node-Right)

Kiểu duyệt này trước tiên thăm các nút của cây con trái sau đó thăm nút gốc rồi đến cây con phải. Thủ tục duyệt như sau:

Duyệt cây theo thứ tự giữa (Left-Node-Right):

```
void LNR(Tree T ) {
        if (T!=NULL ) {
                NLR( T → left); //duyệt thứ tự giữa sang nhánh cây con bên trái
                <Thăm node>;
                NLR( T → right); /duyệt thứ tự giữa sang nhánh cây con bên phải
        }
}
```

6.2.3.3. Duyệt theo thứ tự sau (Left-Right-Node)

Kiểu duyệt này trước tiên thăm các nút của cây con trái sau đó thăm đến cây con

phải rồi cuối cùng mới thăm nút gốc. Thủ tục duyệt như sau:

Duyệt cây theo thứ tự sau (Left-Right-Node):

```
void LRN (Tree T ) {
    if (T!=NULL ) {
        LRN ( T → left); //duyệt thứ tự giữa sang nhánh cây con bên trái
        LRN ( T → right); /duyệt thứ tự giữa sang nhánh cây con bên phải
        <Thăm node>;
    }
}
```

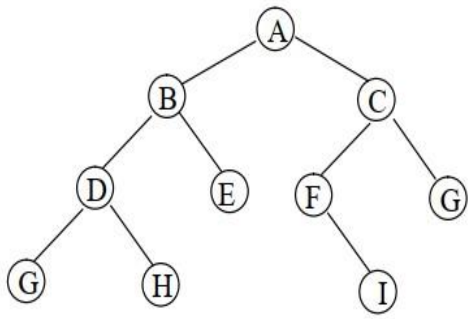
Các phép duyệt cây hình 24 là:

Theo thứ tự trước: A, B, D, E, C, F, G

Theo thứ tự giữa: D, B, E, A, F, C, G

Theo thứ tự sau: D, E, B, F, G, C, A

Ví dụ 6.3: Cho cây nhị phân sau:



Hình 6.7: Cây nhị phân có 10 nút

Các phép duyệt cây là:

Duyệt theo thứ tự trước:

A B D G H E C F I G

Duyệt theo thứ giữa:

G D H B E A F I C G

Duyệt theo thứ tự sau:

G H D E B I F G C A

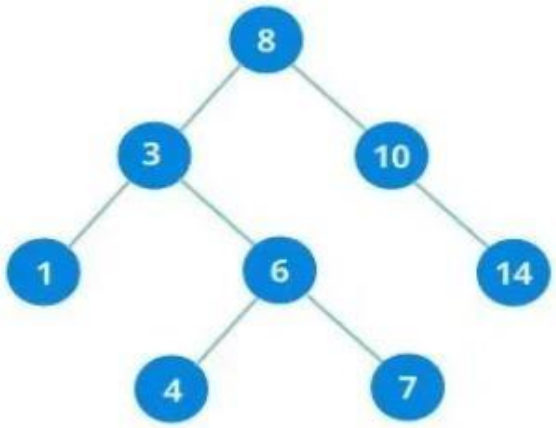
6.3. Cây nhị phân tìm kiếm

Cây tìm kiếm nhị phân (Binary Search Tree) là một cây nhị phân và có thêm các ràng buộc sau đây:

1. Giá trị của tất cả các Node ở cây con bên trái phải <= giá trị của Node gốc.
2. Giá trị của tất cả các Node ở cây con bên phải phải > giá trị của Node gốc.

3. Tất cả các cây con(bao gồm bên trái và phải) cũng đều phải đảm bảo 2 tính chất trên.

Ví dụ 6.3: Cây nhị phân tìm kiếm



Hình 6.8: Cây nhị tìm kiếm

6.3.1. Biểu diễn cây nhị phân (sử dụng danh sách liên kết)

```
typedef struct node {
        Item Infor; //Thông tin của node
        struct nde *left;//Con trỏ node bên trái
        struct nde *right;//Con trỏ node bên phải
} *Tree;
```

Khởi tạo cây nhị phân tìm kiếm:

```
void Init(Tree *T){
*T=NULL; //Đưa cây T về trạng thái rỗng
```

Cấp phát miền nhớ cho một node:

```
Tree GetNode(){
        Tree p;//Khai báo một node kiểu Tree
        p=new node; //Cấp phát miền nhớ cho node
        return (p); //Trả lại node được cấp phát
}
```

Giải phóng một node p cho cây T:

```
void FreeNode(Tree  p){
delete (p); //giải phóng node p
}
```

Kiểm tra tính rỗng của cây T:

```
int isEmpty(Tree  *T){
        if(*T==NULL) //nếu T rỗng
```

```
            return 1; //trả lại giá trị đúng
        return 0; //trả lại giá trị sai
}
```

Tạo một node cho cây T:

```
Tree MakeNode( Item  x){ //x là giá trị node cần bổ sung vào cây
        Tree p; //Khai báo một node kiểu Tree
        p = GetNode();//Cấp phát miền nhớ cho p
        p→infor = x;  //Thiết lập thành phần thông tin cho node
        p→left = NULL; //Tạo liên kết trái cho node
        p→right = NULL;  //Tạo liên kết phải cho node
        return (p); //Trả lại node được tạo ra
}
```

Tìm node p có nội dung là x trên cây://

```
void Search(Tree T, Item x){
        Tree Search(Tree T,  int  x) {
        NODEPTR p;
        if( T→infor==x)// Nếu node gốc là node cần tìm
                return T; //Trả lại node gốc
        if(T==NULL) //Nếu T rỗng
                return NULL; //Trả lại giá trị NULL
        p = Search(T→left, x); //Tìm ở nhánh cây con trái
        if ( p ==NULL) // Nếu không thấy p ở nhánh cây con bên trái
                p = Search(T→right,x); // Tìm p ở nhánh cây con phải
        return ;
}
```

Tạo node gốc cho cây:

```
Tree Make-Root(Tree T, int x){
        if(p==NULL){ // Nếu cây rỗng
                MakeNode (T, x);//Tạo node x cho cây làm gốc
        }
        return(T);
}
```

Tạo node lá bên trái node có nội dung là x:

```
void Add-Left(Tree T, Item x, Item y ) {
```

```
        Tree p, q;
        p = Search (T, x); //Tìm node có nội dung là x trên cây
        if (p==NULL ) { //Nếu node có nội dung x không có trên cây
                <Thông báo node cha x không có thực>;
                return; //Không thể thêm được node lá trái
        }
        //Nếu p đã có nhánh cây con bên trái
        else if ( (p → left) !=NULL )
                <Thông báo node cha x có nhánh cây con trái>;
                return; // Không thêm được node lá trái
        else {
                q = MakeNode(y); // Tạo node lá trái của p là q.
                p → left =q; //Node lá bên trái của p là q
        }
}
```

Tạo node lá bên phải node có nội dung là x:

```
void Add-Right(Tree T, Item x, Item y ) {
        Tree p, q;
        p = Search (T, x); //Tìm node có nội dung là x trên cây
                if (p==NULL ) { //Nếu node có nội dung x không có trên cây
                        <Thông báo node cha x không có thực>;
                        return; //Không thể thêm node lá phải
                }
                //Nếu p đã có nhánh cây con bên phải
                else if ( (p → right !=NULL )
                        <Thông báo node cha x có nhánh cây con phải>;
                        return; // Không thêm được node lá phải
                else {
                        q = MakeNode(y); // Tạo node lá phải của p là q.
                        p → right =q; //Node lá bên phải của p là q
                }
}
```

Loại bỏ node lá bên trái node có nội dung là x:

```
void Remove-Left(Tree T, int  x ) {
```

```
        Tree p, q;
        p = Search (T, x); //Tìm node có nội dung là x trên cây
        if (p==NULL ) { //Nếu node có nội dung x không có trên cây
                <Thông báo node cha x không có thực>;
                return; //Không thể loại bỏ được node lá trái
        }
        //Nếu p có nhánh cây con bên trái
        else if ( (p → left) →right !=NULL || (p → left)-l>left !=NULL ))
                <Thông báo node cha x có nhánh cây con trái>;
                return; // Không thêm loại bỏ được cây con trái
        else {
                q = p → left; // Node lá trái là q.
                p → left =NULL; //Ngắt liên kết cho node p
                delete(q); //Loại bỏ node lá trái q
        }
}
```

Loại bỏ node lá bên phải node có nội dung là x:

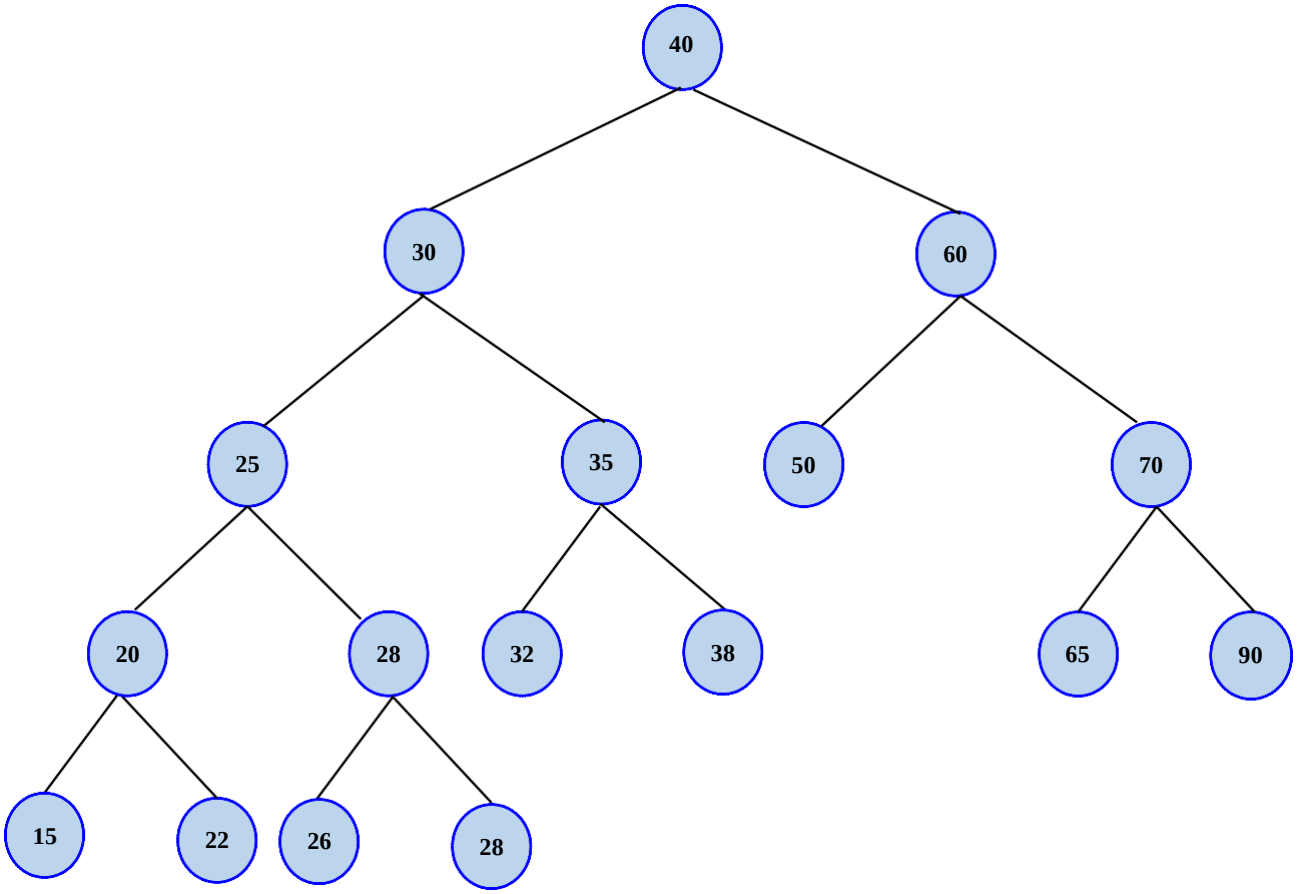
```
void Remove-Left(Tree T, int  x ) {
        Tree p, q;
        p = Search (T, x); //Tìm node có nội dung là x trên cây
                if (p==NULL ) { //Nếu node có nội dung x không có trên cây
                <Thông báo node cha x không có thực>;
                return; //Không thể loại bỏ được node lá phải
        }
        //Nếu p có nhánh cây con bên phải
        else if ( (p → right) →right !=NULL || (p → right) →left !=NULL ))
                <Thông báo node cha x có nhánh cây con phải>;
                return; // Không thêm loại bỏ được cây con trái
        else {
                q = p →left; // Node lá trái là q.
                p →left =NULL; //Ngắt liên kết cho node p
                delete(q); //Loại bỏ node lá phải q
        }
}
```

6.4. Cây nhị phân tìm kiếm cân bằng AVL

AVL viết tắt của tên các nhà phát minh Adelson, Velski và Landis.

Định nghĩa là cây nhị phân tìm kiếm tự cân bằng. Cây tìm kiếm cân bằng có tính chất độ cao của cây con bên trái và độ cao cây con bên phải chên lệch nhau không quá 1.

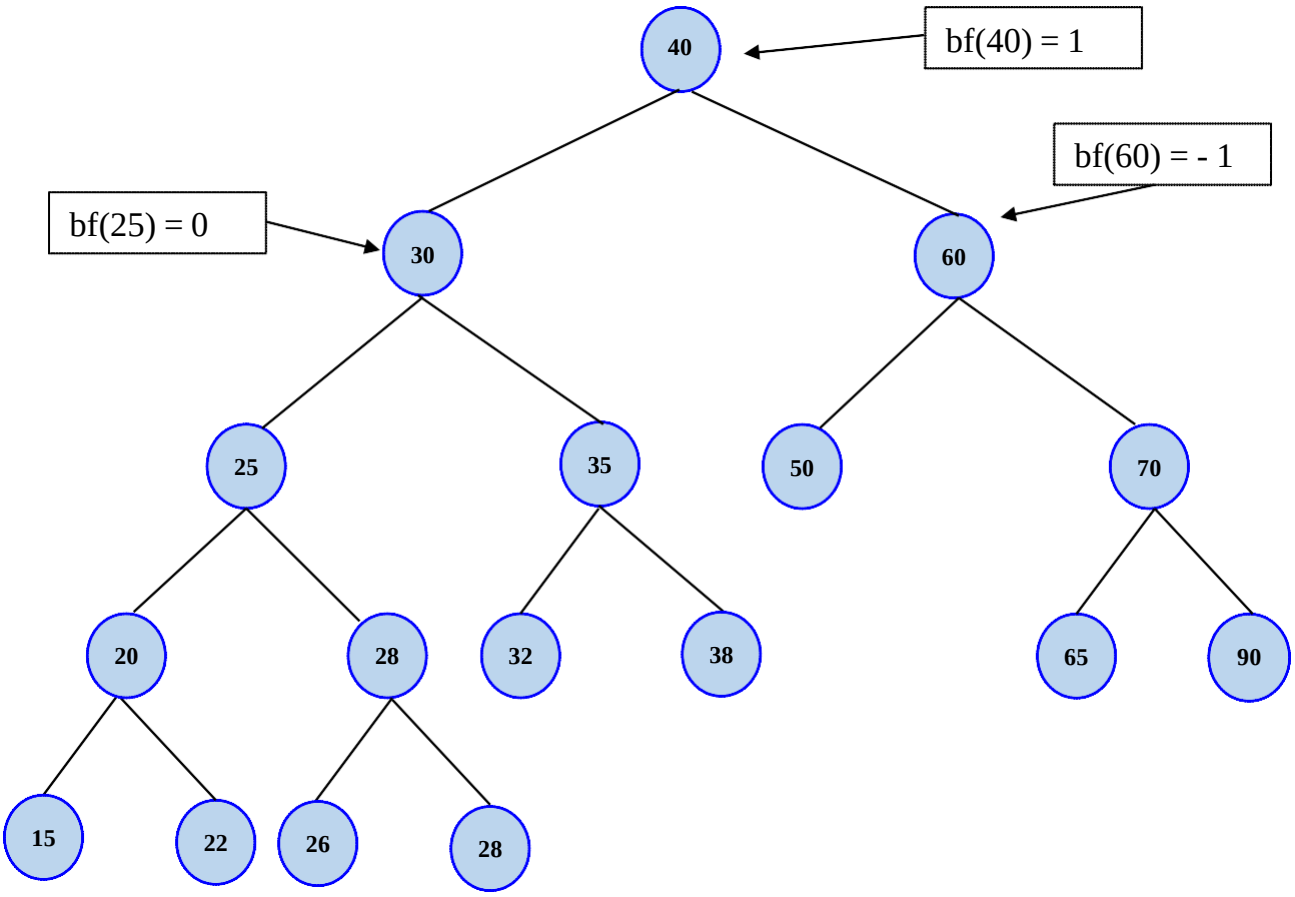
Cây AVL: Cho phép ta tìm kiếm, bổ sung, loại bỏ node nhanh hơn cây nhị phân thông thường vì độ cao của cây luôn là O(log (n)).



Hình 6.9a: Cây nhị tìm kiếm cân bằng AVL

Cây nhị phân tìm kiếm cân bằng (AVL): gọi lh(p), rh(p) là chiều sâu của cây con phải và chiều sâu của cây con trái. Khi đó, bf(p) = lh(p) – rh(p) là chỉ số cân bằng của node p và cây tìm kiếm cân bằng xảy ra trong các trường hợp sau:

1. $lh(p) = rh(p) \Longleftrightarrow bf(p) = 0$: node p cân bằng tuyệt đối.
2. $lh(p) = rh(p) + 1 \Longleftrightarrow bf(p) = 1$: node p lệch về phía trái.
3. $lh(p) = rh(p) - 1 \Longleftrightarrow bf(p) = -1$: node p lệch về phía phải.



Hình 6.9b: Cây nhị tìm kiếm cân bằng AVL minh họa ba trường hợp

Biểu diễn cây tìm kiếm cân bằng: Giống như biểu diễn cây thông thường

```
struct avl_node { //Định nghĩa node của cây
        int data; //Thành phần thông tin
        struct avl_node *left; //Thành phần con trỏ sang cây con trái
        struct avl_node *right; //Thành phần con trỏ sang cây con phải
} *root; //đây là gốc của cây avl
```

6.4.1. Thêm node vào cây AVL

Bài toán cơ bản trên cây AVL được quan tâm nhiều hơn cả là thêm node vào cây tìm kiếm để được một cây tìm kiếm cân bằng. Loại node trên cây tìm kiếm cũng nhận được một cây tìm kiếm cân bằng. Phương pháp thêm node được tiến hành như sau:

Thuật toán thêm node w vào cây AVL:

Bước 1. Thực hiện thêm node w vào cây tìm kiếm giống như cây thông thường.

Bước 2. Xuất phát từ node w, duyệt lên trên để tìm node mất cân bằng đầu tiên. Gọi z là node mất cân bằng đầu tiên, y là con của z và x là cháu của z tính từ đường đi từ w đến z.

Bước 3. Cân bằng lại cây bằng các phép quay thích hợp tại cây con gốc z. Có 4 khả năng có thể xảy ra như sau:

a) Node y là node con trái của z và x là node con trái của y (left-left-case). Trường hợp này ta thực hiện phép xoay phải (right rotation).

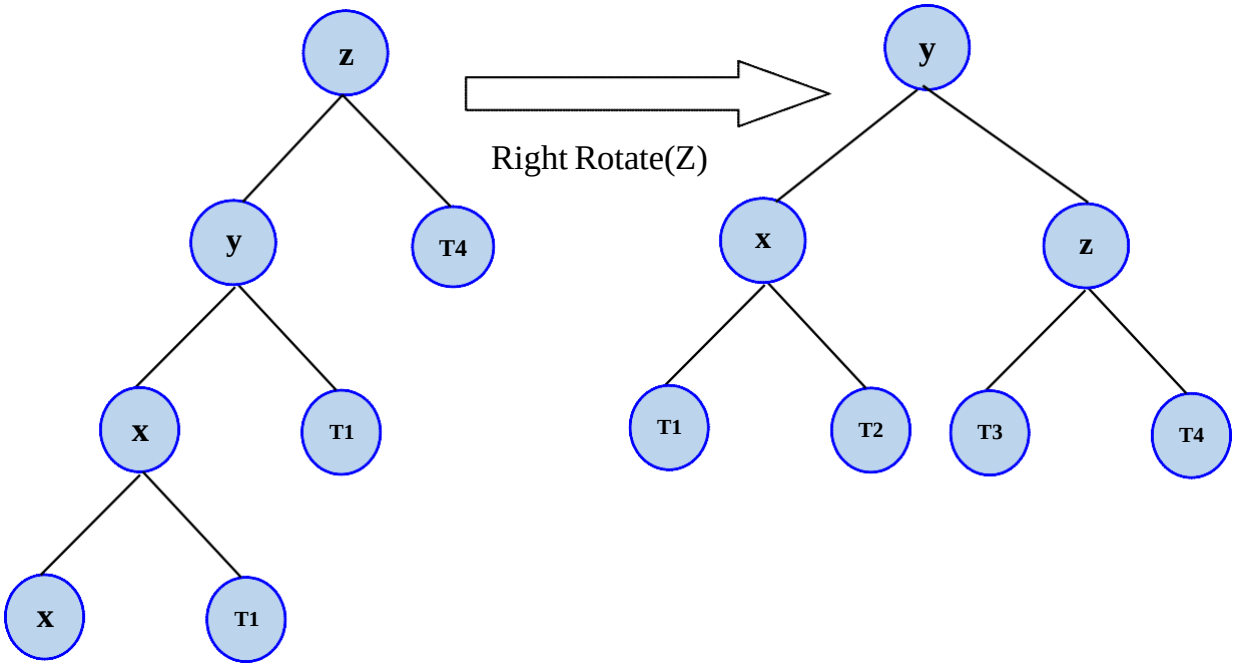
b) Node y là node con trái của z và x là node con phải của y (left-right-case). Trường hợp này ta thực hiện phép xoay trái sau đó xoay phải.

c) Node y là node con phải của z và x là node con phải của y (right-right-case). Trường hợp này ta thực hiện phép xoay trái (left rotation).

d) Node y là node con phải của z và x là node con trái của y (right-left-case). Trường hợp này ta thực hiện phép xoay trái sau đó xoay phải.

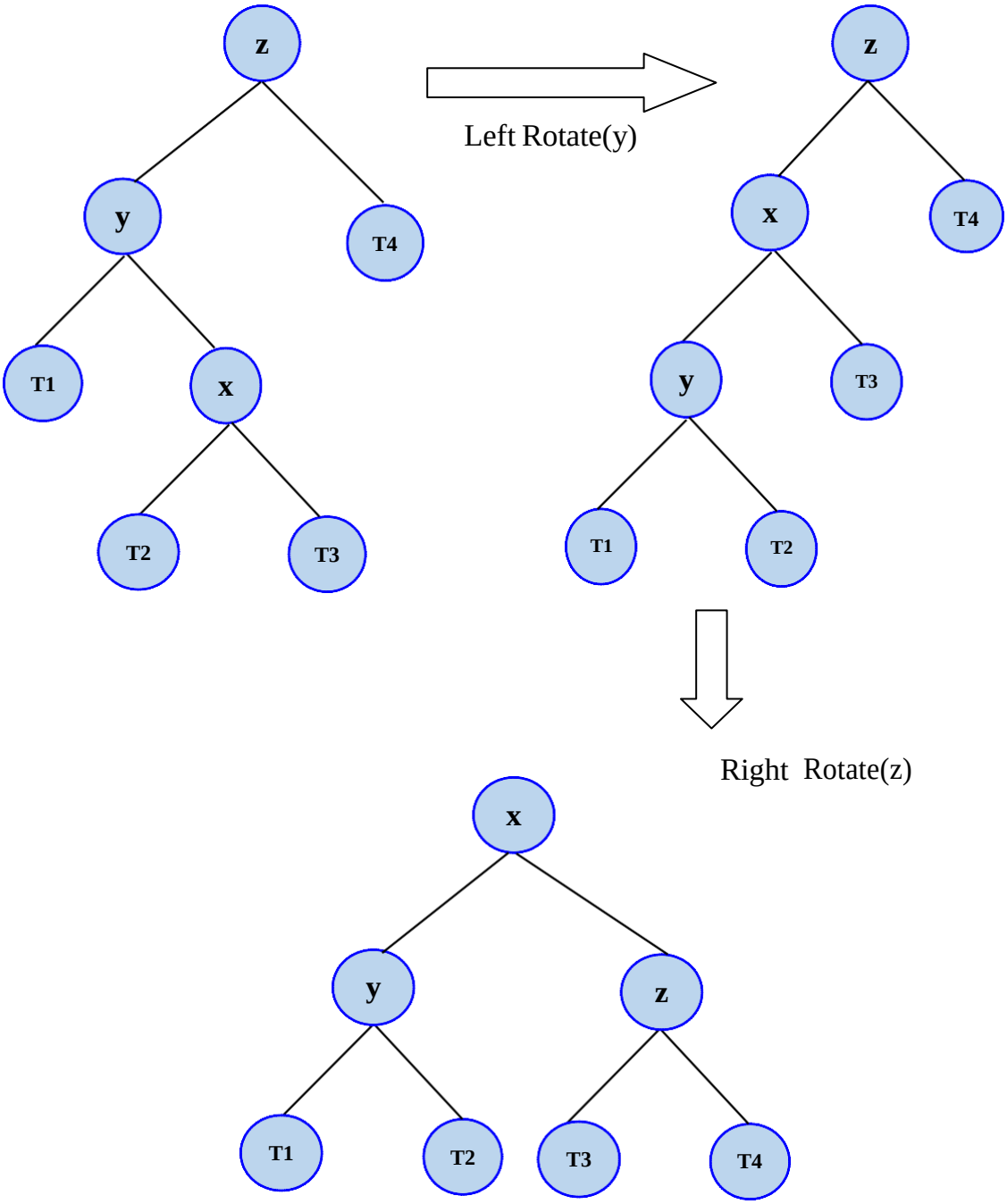
Giải sử T1, T2, T3, T4 là các cây con gốc z, khi đó các phép cân bằng lại (re-balance) được mô trả như sau:

a) Trường hợp left-left-case:



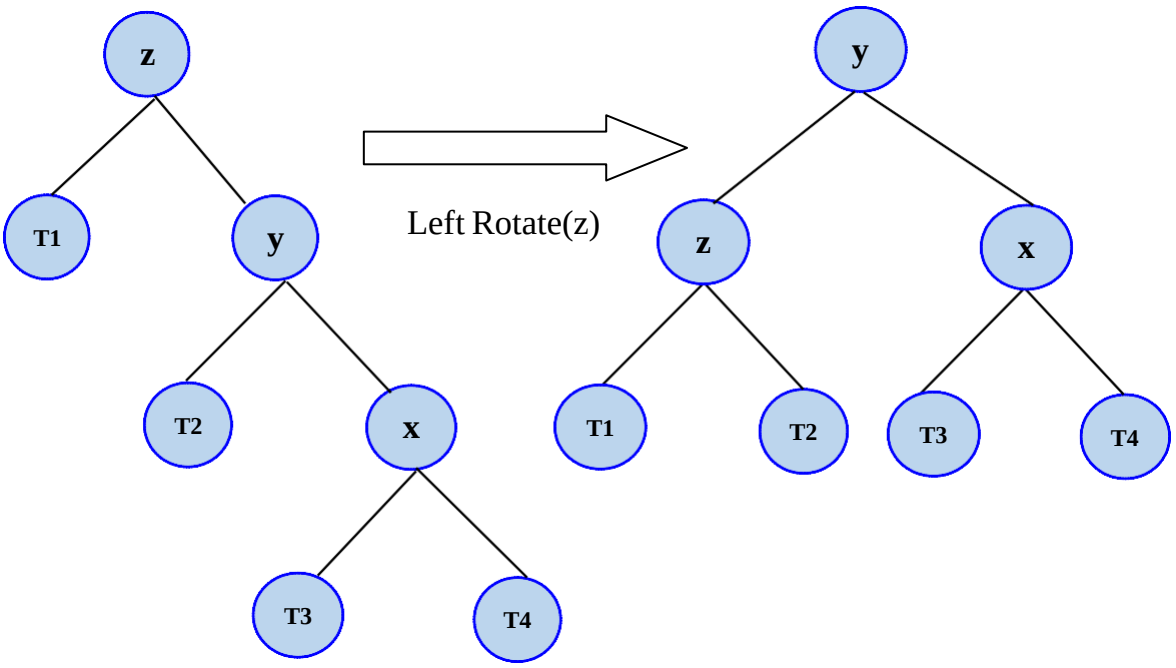
Hình 6.10a: Cây cân bằng trường hợp 1

b) Trường hợp left-right-case:



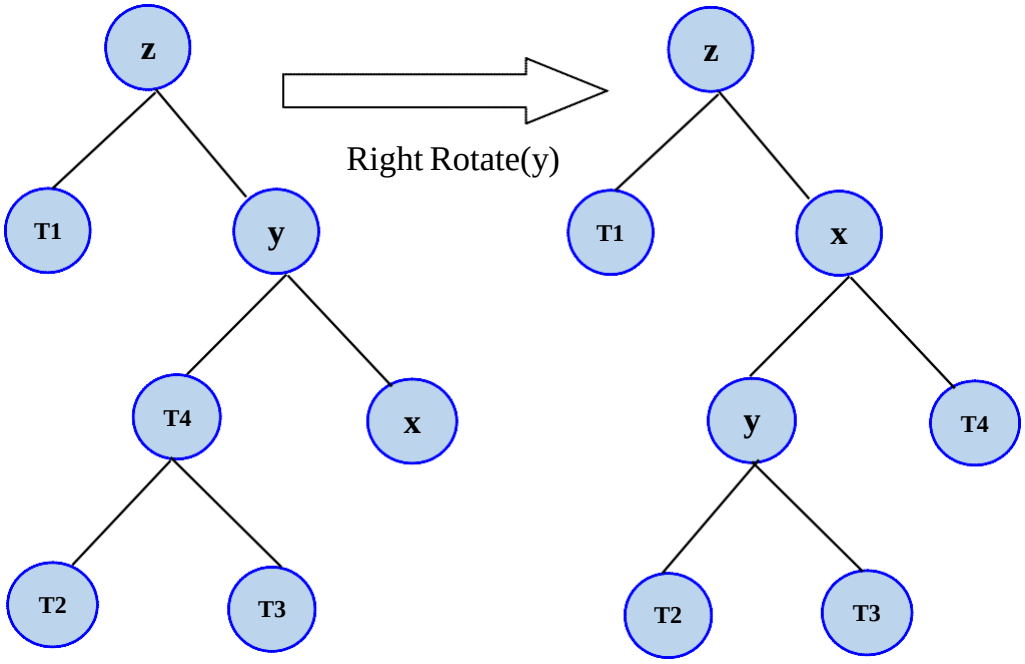
Hình 6.10b: Cây cân bằng trường hợp 2

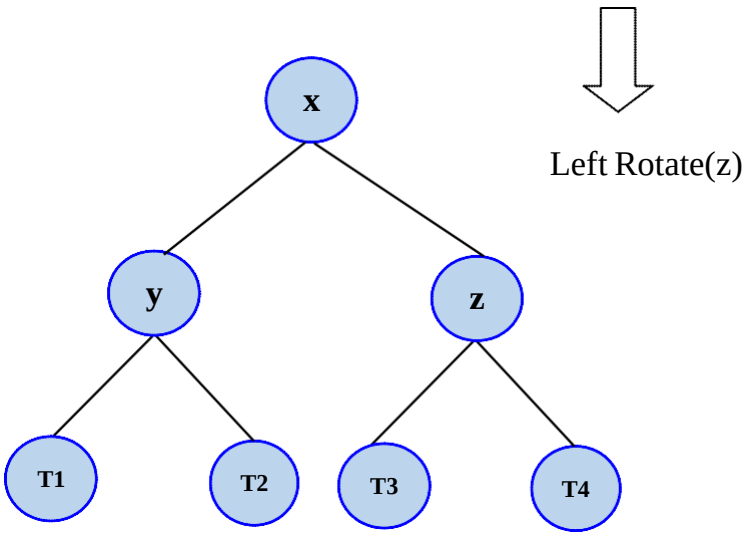
c) Trường hợp right-right-case



Hình 6.10c: Cây cân bằng trường hợp 3

d) Trường hợp right-left-case:





Hình 6.10d: Cây cân bằng trường hợp 4

6.4.2. Loại node trên cây AVL

Để chắc chắn sau khi loại node trên cây tìm kiếm cũng nhận được một cây tìm kiếm cân bằng ta tiến hành như sau:

Thuật toán loại node w trên cây AVL:

Bước 1. Thực hiện loại node w vào cây tìm kiếm giống như cây tìm kiếm thông thường.

Bước 2. Xuất phát từ node w, duyệt lên trên để tìm node mất cân bằng đầu tiên. Gọi z là node mất cân bằng đầu tiên, y node cao hơn của z và x là cao hơn y.

Bước 3. Cân bằng lại cây bằng các phép quay thích hợp tại cây con gốc z. Có 4 khả năng có thể xảy ra như sau:

a) Node y là node con trái của z và x là node con trái của y (left-left-case). Trường hợp này ta thực hiện phép xoay phải (right rotation).

b) Node y là node con trái của z và x là node con phải của y (left-right-case). Trường hợp này ta thực hiện phép xoay trái sau đó xoay phải.

c) Node y là node con phải của z và x là node con phải của y (right-right-case). Trường hợp này ta thực hiện phép xoay trái (left rotation).

d) Node y là node con phải của z và x là node con trái của y (right-left-case). Trường hợp này ta thực hiện phép xoay trái sau đó xoay phải.

6.4.3. Một số thao tác trên cây AVL

Khai báo node cây AVL:

```
struct node {
        int key;//thành phần dữ liệu
        struct node *left; //thành con trỏ đến cây con trái
```

```
    struct node *right; //thành con trỏ đến cây con phải
    int height; //chỉ số cân bằng của cây
};
```

Phép quay phải (rightRotation):

```
struct node *rightRotate(struct node *y) {
    struct node *x = y->left; //x trỏ đến node bên trái của y
    struct node *T2 = x->right; //T2 là cây con phải của x->right
    //Thực hiện quay phải tại y x-
    >right = y; y->left = T2;
    //Cập nhật lại chiều cao
    y->height = max(height(y->left), height(y-
    >right))+1;
    x->height = max(height(x->left), height(x->right))+1;
    return x;
}
```

Phép quay trái (leftRotation):

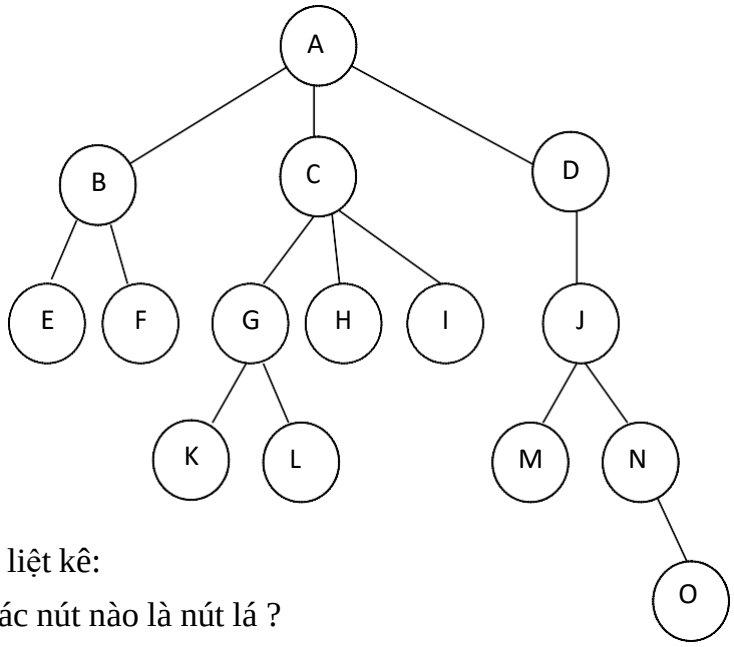
```
struct node *leftRotate(struct node *x) {
    struct node *y = x->right;
    struct node *T2 = y->left;
    // thực hiện quay trái y-
    >left = x;
    x->right = T2;
    // cập nhật độ cao
    x->height = max(height(x->left), height(x->right))+1; y-
    >height = max(height(y->left), height(y->right))+1;
    // trả lại gốc mới root
    return y;
}
```

Bài tập vận dụng

1. Tạo cây nhị phân tìm kiếm với các số nhập vào theo thứ tự:

   8 3 5 2 20 11 30 9 18 4

2. Cho cây như hình sau



Hãy liệt kê:

a. Các nút nào là nút lá ?

b. Các nút nào là nút nhánh ?

c. Cha của nút G là nút nào ?

d. Con của nút C là các nút nào ?

e. Các nút nào là anh em của nút B ?

f. Mức của D, của L là bao nhiêu ?

g. Bậc của B, bậc của D là bao nhiêu ?

h. Bậc của cây này là bao nhiêu ?

i. Chiều cao của cây này là bao nhiêu ?

j. Độ dài từ A đến F, từ A dến G là bao nhiêu ?

k. Có bao nhiêu đường đi từ gốc A có độ dài 3 trên cây này?

3. Cho cây nhị phân tìm kiếm T gồm 12 số nguyên với phép duyệt LRN cho kết quả như sau: 1, 3, 2, 6, 7, 5, 4, 10, 9, 12, 11, 8

a. Hãy vẽ cây nhị phân tìm kiếm T.

b. Duyệt cây T theo thứ tự NLR.

4. Xây dựng thủ tục thêm một nút lá vào cây nhị phân, chèn một nút vào cây nhị phân.

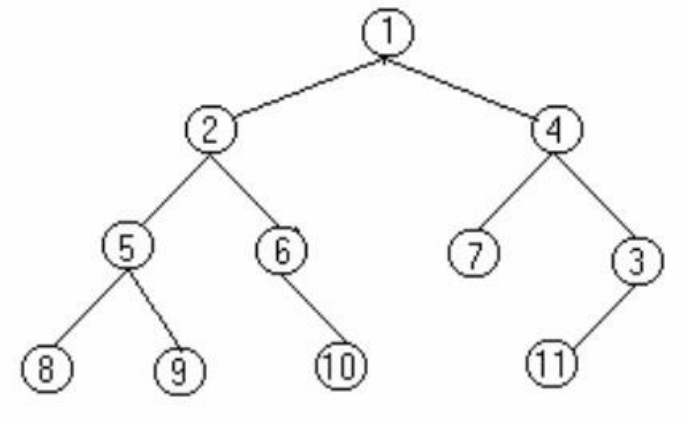
5. Xây dựng cây nhị phân biểu diễn biểu thức ((a+b)*4) – (a-b)/3) và tìm dạng hậu tố của biểu thức này.

6. Xây dựng cây nhị phân biểu diễn biểu thức (x+4)2 * (x2-4x+2) – 7 và tìm dạng hậu tố của biểu thức này.

7. Xây dựng cây nhị phân biểu diễn biểu thức (x+4)2 * (x2-4x+2) – 7 và tìm dạng

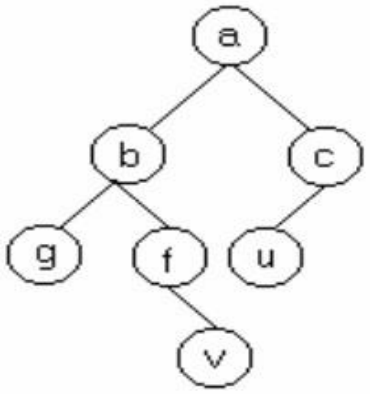
tiền tố của biểu thức này.

8. Cho cây nhị phân như hình vẽ



Hãy viết dãy các nút được thăm khi duyệt cây theo thứ tự trước, thứ tự giữa và thứ tự sau

9. Cho cây nhị phân sau:



Hãy viết dãy các nút được thăm khi duyệt cây theo thứ tự trước, thứ tự giữa và thứ tự sau.

10. Viết chương trình thực hiện các thao tác cơ bản trên cây AVL: chèn một nút, xóa một nút, tạo cây AVL, hủy cây AVL.

11. Minh họa quá trình hình thành cây cân bằng AVL với các giá trị lần lượt là:

1, 9, 2, 15, 12, 8, 4, 11, 7, 19, 18, 3, 15, 6, 21, 13, 10.

12. Tìm cây nhị phân thỏa đồng thời hai điều kiện kết xuất sau:

- Theo thứ tự đầu NLR của nó là dãy ký tự sau:

A, B, C, D, E, Z, U, T, Y

- Theo thứ tự giữa LNR của nó là dãy ký tự sau:

D, C, E, B, A, U, Z, T, Y

13. Biểu diễn biểu thứ số học dưới đây trên cây nhị phân, từ đó rút ra dạng biểu thức hậu tố của chúng: (a + b) * (c - d).