CÂY CÂN BẰNG

1.CÂY NHỊ PHÂN CÂN BẰNG HOÀN TOÀN

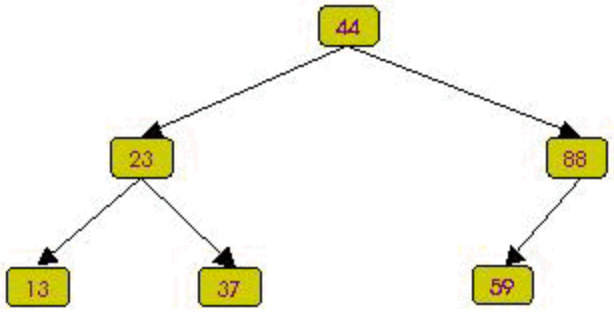
1.1. Định nghĩa

Cây cân bằng hoàn toàn là cây nhị phân tìm kiếm mà tại mỗi nút của nó, số nút của cây con trái chênh lệch không quá một so với số nút của cây con phải.

1.2. Đánh giá

Một cây rất khó đạt được trạng thái cân bằng hoàn toàn và cũng rất dễ mất cân bằng vì khi thêm hay hủy các nút trên cây có thể làm cây mất cân bằng, chi phí cân bằng lại cây cao vì phải thao tác trên toàn bộ cây.

Đối với cây cân bằng hoàn toàn, trong trường hợp xấu nhất ta chỉ phải tìm qua $\log_2^N$ phần tử (N là số nút trên cây).
Sau đây là ví dụ một cây cân bằng hoàn toàn (CCBHT):



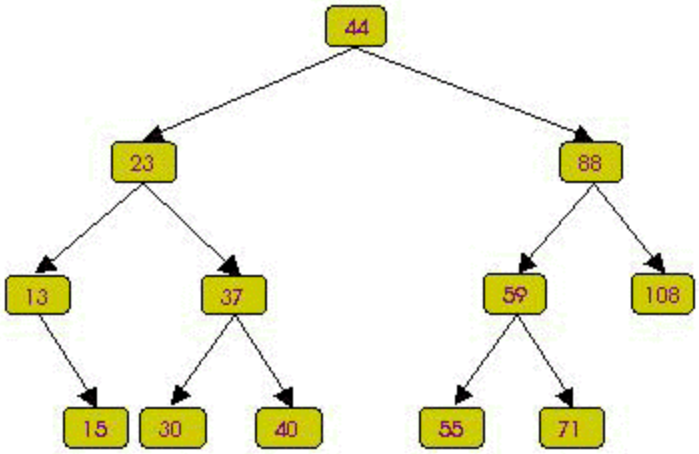
CCBHT có N nút có chiều cao $h \approx \log_2 N$. Đây chính là lý do cho phép bảo đảm khả năng tìm kiếm nhanh trên CTDL này. Do CCBHT là một cấu trúc kém ổn định nên trong thực tế không thể sử dụng. Nhưng ưu điểm của nó lại rất quan trọng. Vì vậy, cần đưa ra một CTDL khác có đặc tính giống CCBHT nhưng ổn định hơn.

2. CÂY NHỊ PHÂN CÂN BẰNG (AVL Tree)

2.1. Định nghĩa:

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.

Dưới đây là ví dụ cây nhị phân cân bằng :



Dễ dàng thấy CCBHT là cây cân bằng. Điều ngược lại có thể không đúng không đúng.

2.2. Lịch sử cây cân bằng (AVL Tree)

AVL là tên viết tắt của các tác giả người Nga đã đưa ra định nghĩa của cây cân bằng Adelson-Velskii và Landis (1962). Vì lý do này, người ta gọi cây nhị phân cân băng là cây AVL. Từ cây AVL, người ta đã phát triển thêm nhiều loại CTDL hữu dụng khác như cây đỏ-đen (Red-Black Tree), B-Tree, …

2.3. Chiều cao của cây AVL

Một vấn đề quan trọng, như đã đề cập đến ở phần trước, là ta phải khẳng định cây AVL có N nút phải có chiều cao khoảng log2(n).

Để đánh giá chính xác về chiều cao của cây AVL, ta xét bài toán: cây AVL có chiều cao h sẽ phải có tối thiểu bao nhiêu nút ?

Gọi N(h) là số nút tối thiểu của cây AVL có chiều cao h.

Ta có N(0) = 0, N(1) = 1 và N(2) = 2.

Cây AVL có chiều cao h sẽ có 1 cây con AVL chiều cao h-1 và 1 cây con AVL chiều cao h-2. Như vậy:

$N(h) = 1 + N(h-1) + N(h-2)$ (1)

Ta lại có: $N(h-1) > N(h-2)$
Nên từ (1) suy ra:

$N(h) > 2N(h-2)$

$N(h) > 2^2N(h-4)$

…

$N(h) > 2^iN(h-2i)$

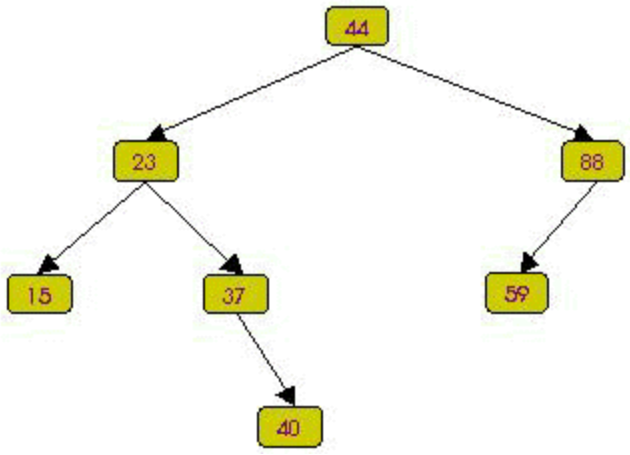
$i = h/2$

$N(h) > 2^{h/2}$

$h < 2\log2(N(h))$

Như vậy, cây AVL có chiều cao O(log2(n)).

Ví dụ: cây AVL tối thiểu có chiều cao h=4



2.4. Cấu trúc dữ liệu cho cây AVL

Chỉ số cân bằng của một nút: Chỉ số cân bằng của một nút là hiệu của chiều cao cây con phải và cây con trái của nó.

Đối với một cây cân bằng, chỉ số cân bằng (CSCB) của mỗi nút chỉ có thể nhận một trong ba giá trị sau đây:

CSCB(p) = 0 <=> Độ cao cây trái (p) = Độ cao cây phải (p)
CSCB(p) = 1 <=> Độ cao cây trái (p) < Độ cao cây phải (p)
CSCB(p) =-1 <=> Độ cao cây trái (p) > Độ cao cây phải (p)

Xét nút P, ta dùng các ký hiệu sau:

P->balFactor = CSCB(P);
Độ cao cây trái P ký hiệu là h_{left}
Độ cao cây phải P ký hiệu là h_{right}

Để khảo sát cây cân bằng, ta cần lưu thêm thông tin về chỉ số cân bằng tại mỗi nút. Lúc đó, cây cân bằng có thể được khai báo như sau:

typedef struct tagAVLNode

```
{
        char balFactor; //Chỉ số cân bằng
        Data key;
        struct tagAVLNode* pLeft;
        struct tagAVLNode* pRight;
}
AVLNode;

typedef AVLNode *AVLTree;
```

Để tiện cho việc trình bày, ta định nghĩa một số hăng số sau:

```
#define LH -1 //Cây con trái cao hơn
#define EH -0 //Hai cây con bằng nhau
#define RH  1 //Cây con phải cao hơn
```

2.5. Đánh giá cây AVL

Cây cân bằng là CTDL ổn định hơn CCBHT vì khi thêm, hủy làm cây thay đổi chiều cao các trường hợp mất cân bằng mới có khả năng xảy ra.

Cây AVL với chiều cao được khống chế sẽ cho phép thực thi các thao tác tìm, thêm, hủy với chi phí O (log2(n)) và bảo đảm không suy biến thành O(n).

3. CÁC THAO TÁC CƠ BẢN TRÊN CÂY AVL

Ta nhận thấy trường hợp thêm hay hủy một phần tử trên cây có thể làm cây tăng hay giảm chiều cao, khi đó phải cân bằng lại cây.

Việc cân bằng lại một cây sẽ phải thực hiện sao cho chỉ ảnh hưởng tối thiểu đến cây nhằm giảm thiểu chi phí cân bằng. Như đã nói ở trên, cây cân

bằng cho phép việc cân bằng lại chỉ xảy ra trong giới hạn cục bộ nên chúng ta có thể thực hiện được mục tiêu vừa nêu.

Như vậy, ngoài các thao tác bình thường như trên CNPTK, các thao tác đặc trưng của cây AVL gồm:

Thêm một phần tử vào cây AVL.

Hủy một phần tử trên cây AVL.

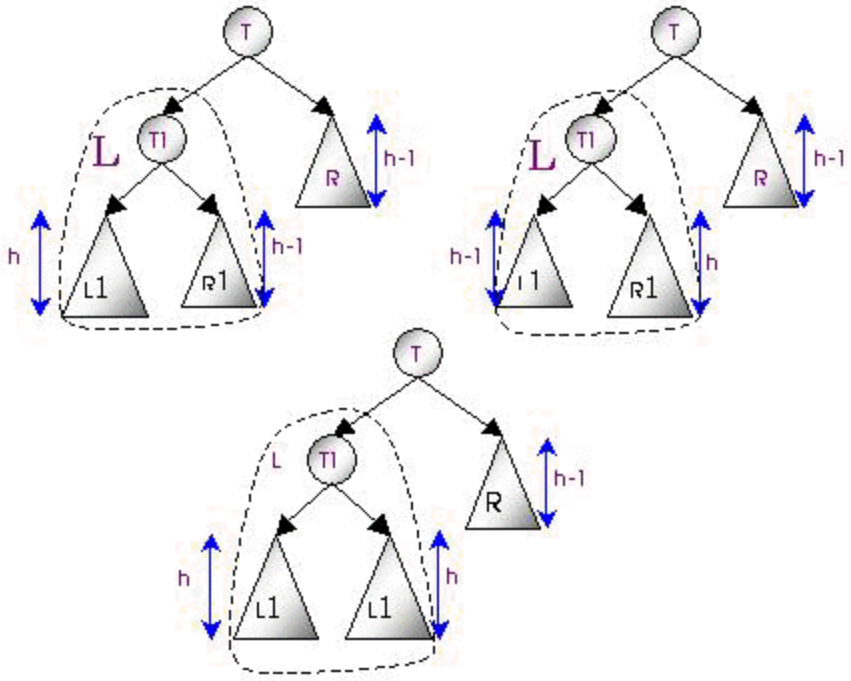
Cân bằng lại một cây vừa bị mất cân bằng.

3.1. CÁC TRƯỜNG HỢP MẤT CÂN BẰNG

Ta sẽ không khảo sát tính cân bằng của 1 cây nhị phân bất kỳ mà chỉ quan tâm đến các khả năng mất cân bằng xảy ra khi thêm hoặc hủy một nút trên cây AVL.

Như vậy, khi mất cân bằng, độ lệch chiều cao giữa 2 cây con sẽ là 2. Ta có 6 khả năng sau:

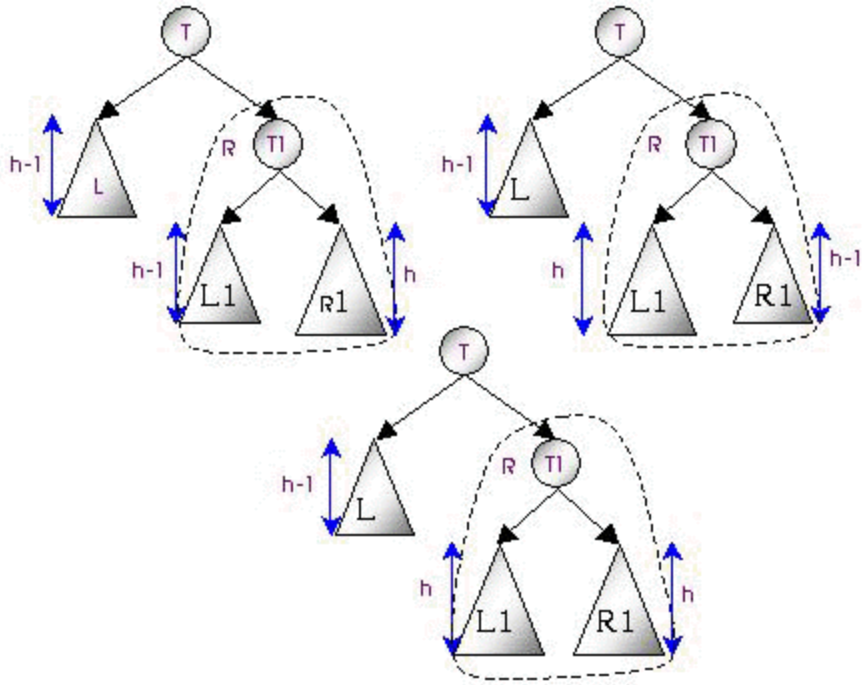
Trường hợp 1: cây T lệch về bên trái (có 3 khả năng)



Trường hợp 2: cây T lệch về bên phải

Ta có các khả năng sau:

Ta có thể thấy rằng các trường hợp lệch về bên phải hoàn toàn đối xứng với các trường hợp lệch về bên trái. Vì vậy ta chỉ cần khảo sát trường

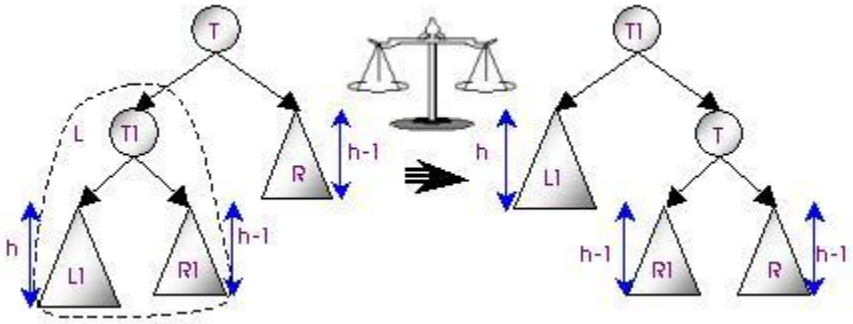


hợp lệch về bên trái.

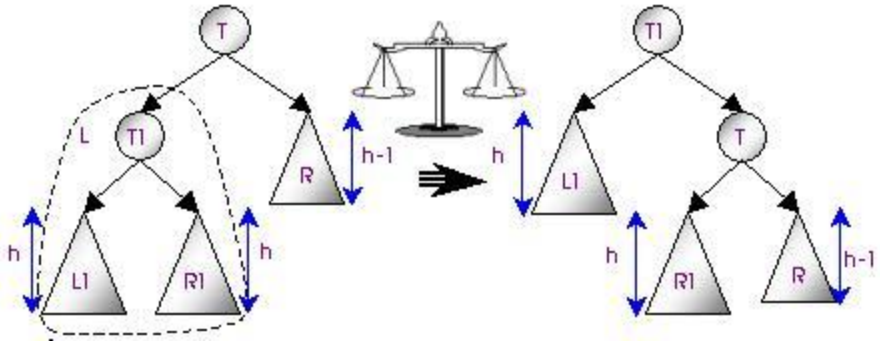
Trong 3 trường hợp lệch về bên trái, trường hợp T1 lệch phải là phức tạp nhất. Các trường hợp còn lại giải quyết rất đơn giản.

Sau đây, ta sẽ khảo sát và giải quyết từng trường hợp nêu trên

T/h 1.1: cây T1 lệch về bên trái. Ta thực hiện phép quay đơn Left-Left



T/h 1.2: cây T1 không lệch. Ta thực hiện phép quay đơn Left-Left



Ttoán quay đơn Left-Left:

B1: T là gốc; T1 = T->pLeft;
T->pLeft = T1->pRight;
T1->pRight = T;

B2:// đặt lại chỉ số cân bằng

Nếu T1->balFactor = LH thì:
T->balFactor = EH;
T1->balFactor = EH;
break;

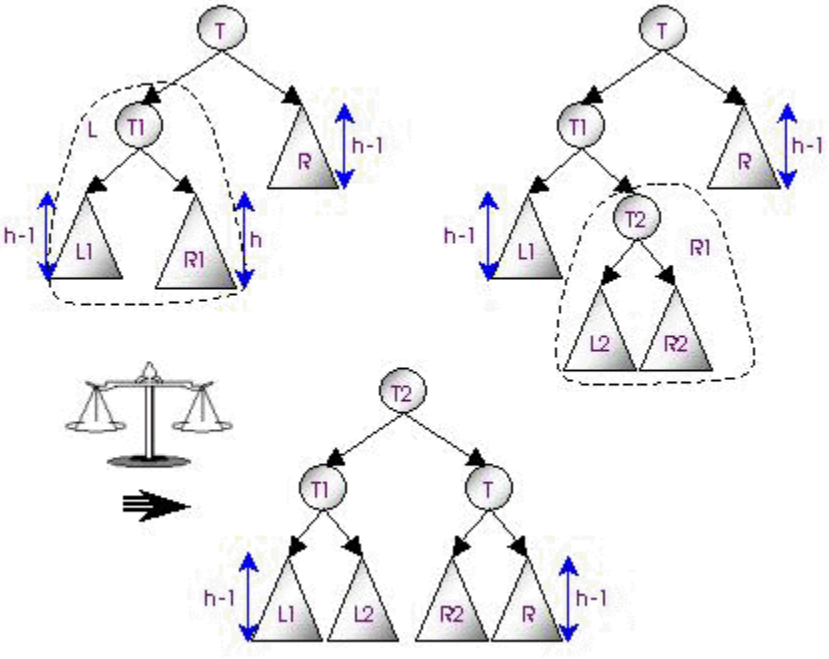
Nếu T1->balFactor = EH thì:
T->balFactor = LH;
T1->balFactor = RH;
break;

B3:// T trỏ đến gốc mới
T = T1;

T/h 1.3: cây T1 lệch về bên phải. Ta thực hiện phép quay kép Left-Right

Do T1 lệch về bên phải ta không thể áp dụng phép quay đơn đã áp dụng trong 2 trường hợp trên vì khi đó cây T sẽ từ trạng thái mất cân bằng do lệch trái thành mất cân bằng do lệch phải.

Hình vẽ dưới đây minh họa phép quay kép áp dụng cho trường hợp này:



Ttoán quay kép Left - Right

B1: gốc T;

T1 = T->pLeft; T2 = T1->pRight; T->pLeft = T2->pRight;

T2->pRight = T;T1->pRight = T2->pLeft;T2->pLeft = T1;

B2: //đặt lại chỉ số cân bằng

Nếu T2->balFactor = LH thì:

T->balFactor = RH; T1->balFactor = EH; break;

Nếu T2->balFactor = EH thì:

T->balFactor = EH; T1->balFactor = EH; break;

Nếu T2->balFactor = RH thì:

T->balFactor = EH; T1->balFactor = LH; break;

B3:

T2->balFactor = EH;

T = T2;

Lưu ý rằng, trước khi cân bằng cây T có chiều cao h+2 trong cả 3 trường hợp 1.1, 1.2 và 1.3.

Sau khi cân bằng, trong 2 trường hợp 1.1 và 1.3 cây có chiều cao h+1; còn ở trường hợp 1.2 cây vẫn có chiều cao h+2. Và trường hợp này cũng là trường hợp duy nhất sau khi cân bằng nút T cũ có chỉ số cân bằng khác 0.

Thao tác cân bằng lại trong tất cả các trường hợp đều có độ phức tạp O(1).

Với những xem xét trên, xét tương tự cho trường hợp cây T lệch về bên phải, ta có thể xây dựng 2 hàm quay đơn và 2 hàm quay kép sau:

3.2.THÊM MỘT PHẦN TỬ TRÊN CÂY AVL:

Việc thêm một phần tử vào cây AVL diễn ra tương tự như trên CNPTK. Tuy nhiên, sau khi thêm xong, nếu chiều cao của cây thay đổi, từ vị trí thêm vào, ta phải tìm ngược lên gốc để kiểm tra các nút bị mất cân bằng không. Nếu có, ta phải cân bằng lại ở nút này.

TToán: Giả sử cần thêm vào một nút mang thông tin X.

1. Tìm kiếm vị trí thích hợp để thêm nút X (đưa ra thông báo nếu đã có nút X rồi)

2. Thêm nút X vào cây

3. Cân bằng lại cây.

3.3. HỦY MỘT PHẦN TỬ TRÊN CÂY AVL:

Cũng giống như thao tác thêm một nút, việc hủy một phần tử X ra khỏi cây AVL thực hiện giống như trên CNPTK. Chỉ sau khi hủy, nếu tính cân bằng của cây bị vi phạm ta sẽ thực hiện việc cân bằng lại.
Tuy nhiên việc cân bằng lại trong thao tác hủy sẽ phức tạp hơn.