Cây nhị phân tìm kiếm cân bằng

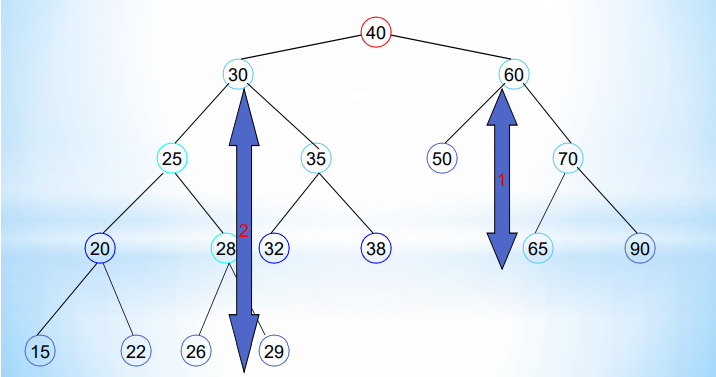
Tổng quan:

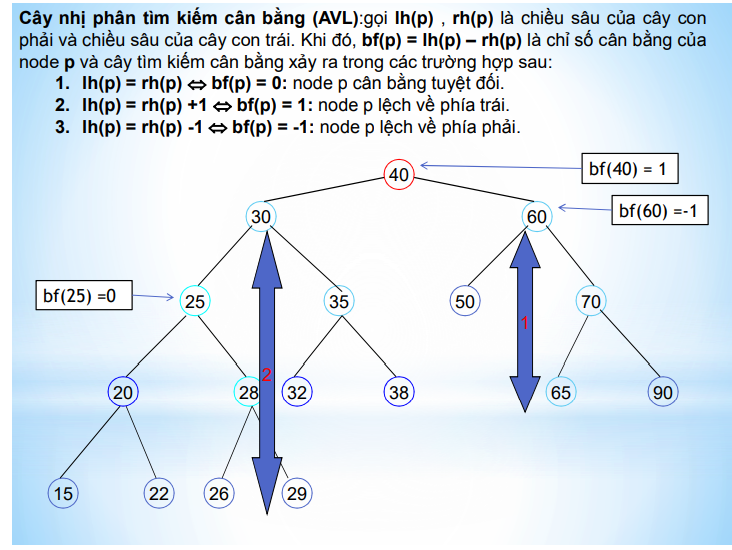
- Tìm kiếm trên cây BST không cải thiện được nhiều so với cây nhị phân thông

thường. Độ phức tạp thuật toán tìm kiếm node trên cây BTS trong trường hợp tốt nhất là log(N), trong trường hợp xấu nhất là O(N), với N là số node trên cây. Trường hợp xấu nhất xảy ra khi cây nhị phân tìm kiếm là cây lệch trái hoặc cây lệnh phải. Trường hợp tốt nhất xảy ra khi cây nhị phân tìm kiếm có độ cao là Log2(N). Giảm chiều sâu (độ cao) của các cây nhị phân tìm kiếm để cải thiện thời gian tìm kiếm node trên cây là mục tiêu chính của cây nhị phân tìm kiếm cân bằng.

* Định nghĩa:

- Cây nhị phân tìm kiếm cân bằng trước hết nó là cây nhị phân tìm kiếm, kèm theo điều kiện: độ cao của cây con bên trái và độ cao cây con bên phải luôn lệch nhau không quá 1 (áp dụng với mọi node trong cây)





* Biểu diễn:

- Biểu diễn cây nhị phân tìm kiếm cân bằng cũng giống như biểu diễn cây nhị phân

tìm kiếm thông thường. Tuy nhiên, để thuận tiện trong việc tính toán chỉ số cân bằng của các node trên cây ta đưa thêm thông tin độ cao mỗi node. Cây AVL được biểu diễn như sau:

struct node { //định nghĩa cấu trúc node

int data; //thành phần dữ liệu của node

struct node \*left; //thành phần con trỏ cây con trái

struct node \*right; //thành phần con trỏ cây con phải

int height; //độ cao của node

} \*AVL\_Tree; //định nghĩa cây AVL

* Các thao tác:

Khác với cây nhị phân tìm kiếm thông thường, khi thêm node vào cây nhị phân

tìm kiếm cân bằng ta vẫn nhận được một cây nhị phân tìm kiếm cân bằng. Khi loại bỏ node khỏi cây nhị phân tìm kiếm cân bằng. Hai thao tác thêm node và loại bỏ node luôn duy trì độ cao của cây luôn là log2(N), với N là số node trên cây. Để bảo toàn được hai phép thêm và loại bỏ node, cây AVL trang bị hai phép xoay cây bên trái và xoay cây bên phải để giảm độ cao cây con phải và giảm độ cao cây con trái.

Lớp các thao tác cụ thể trên cây AVL khác với lớp các thao tác trên cây nhị phân tìm kiếm các thao tác: xoay trái cây AVL, xoay phải cây nhị phân AVL, thêm node vào cây AVL, loại bỏ node trên cây AVL.

Lớp các thao tác trên cây AVL được thể hiện như dưới đây:

int max(int a, int b); // tìm số lớn nhất trong hai số

node\* newNode(int value); //tạo node rời rạc có giá trị value

int height(node \*N); //lấy độ cao node N

node\* search(node\* root, int value);//tìm node có giá trị value

node \* rightRotate(node \*y); //phép quay phải tại node y

node \* leftRotate(node \*x);//quay trái tại node x

int getBalance(node \*N);//lấy chỉ số cân bằng của node

node\* insert(node\* node, int data);//thêm node vào cây AVL

node\* deleteNode(node\* root, int key);//loại node trên cây AVL

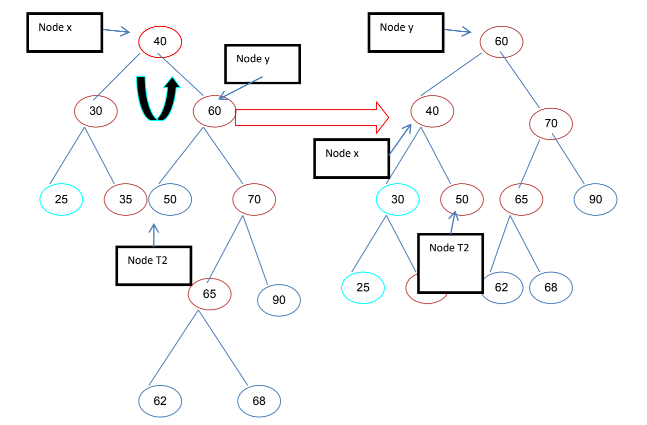
void preOrder(node \*root){//duyệt theo thứ tự trước

void AVL\_Tree ::inOrder(node \*root); //duyệt theo thứ tự giữa

void AVL\_Tree ::postOrder(node \*root); //duyệt theo thứ tự sau

node \* AVL\_Tree ::minValueNode(node\* root);//tìm node có giá trị nhỏ nhất

void AVL\_Tree ::Function(void); //hàm điều khiển các thao tác

* Thao tác xoay trái cây AVL: khi ta thêm hoặc loại bỏ node trên cây AVL có thể làm cây mất cân bằng. Phép xoay trái trên cây AVL nhằm hạ độ cao cây con phải. Ví dụ ta cần xoay trái tại node 40

node \* AVL\_Tree ::leftRotate(node \*x){//quay trái tại node x

node \*y = x->right; //y là node con phải của x

node \*T2 = y->left; //T2 là node con trái của y

// phép quay được thực hiện như dưới đây

y->left = x; //node trái của y bây giờ là x

x->right = T2; //node phải của x bây giờ là T2

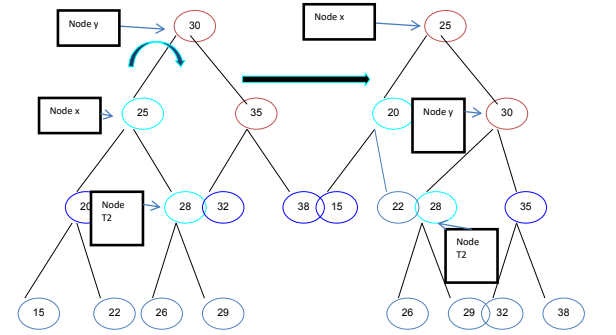
// cập nhật lại độ cao

x->height = max(height(x->left), height(x->right))+1;

y->height = max(height(y->left), height(y->right))+1;

return y;//node gốc mới là y

* Thao tác xoay phải cây AVL: khi ta thêm hoặc loại bỏ node trên cây AVL có thể làm cây mất cân bằng. Phép xoay phải trên cây AVL nhằm hạ độ cao cây con trái. Ví dụ ta cần xoay phải tại node 40



node \* AVL\_Tree ::rightRotate(node \*y){ //phép quay phải tại node y

node \*x = y->left; //x là node con trái của y

node \*T2 = x->right; //T2 là node con phải của x

//phép quay được thực hiện như dưới đây

x->right = y; //node con phải của x bây giờ là y

y->left = T2; //node con trái của y bây giờ là T2

// cập nhật lại độ cao

y->height = max(height(y->left), height(y->right))+1;

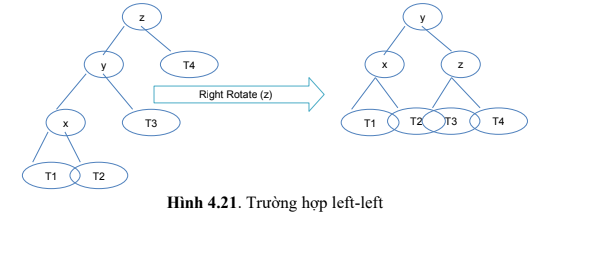
x->height = max(height(x->left), height(x->right))+1;

return x; //trả lại gốc mới là x

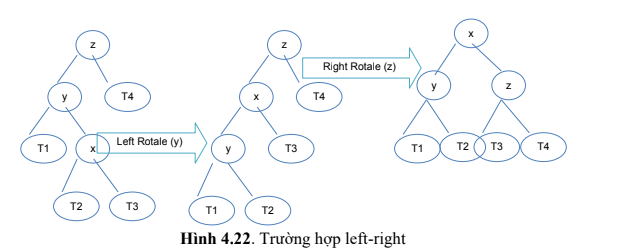
* Thêm node vào cây AVL.
* Hai bài toán cơ bản trên cây AVL được quan tâm nhiều hơn cả là thêm node vào cây tìm kiếm để được một cây tìm kiếm cân bằng. Loại node trên cây tìm kiếm cũng nhận được một cây tìm kiếm cân bằng. Phương pháp thêm node được tiến hành như sau:
* Thuật toán thêm node w vào cây 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:
* 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 soay phải (right rotation).
* 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 soay trái sau đó soay phải.
* 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 soay trái(left rotation).
* 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 soay trái sau đó soay 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:

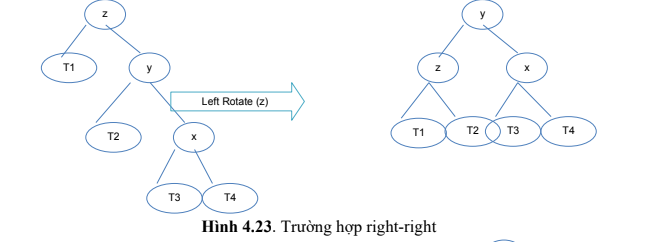
1. Trường hợp left-left-case:



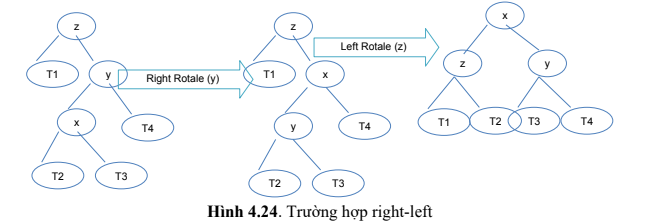
1. Trường hợp left-right-case:



1. Trường hợp right-right-case:



1. Trường hợp right-left



node\* AVL\_Tree ::insert(node\* node, int data){//thêm node vào cây AVL

//Bước 1. Thực hiện thêm node giống như cây tìm kiếm

if (node == NULL)

return(newNode(data));

if (data < node->data)

node->left = insert(node->left, data);

else

node->right = insert(node->right, data);

// Bước 2. Cập nhật độ cao của cây

node->height = max(height(node->left), height(node->right)) + 1;

//Bước 3. Lấy chỉ số cân bằng của cây

int balance = getBalance(node);

// 4 trường hợp làm cây mất cân bằng được xem xét

//Trường hợp a: Left-Left Case

if (balance > 1 && data < node->left->data) //ta quay phải

return rightRotate(node);

// Trường hợp b: Right Right Case

if (balance < -1 && data > node->right->data)//ta quay trái

return leftRotate(node);

// Trường hợp c: Left Right Case

if (balance > 1 && data > node->left->data){

node->left = leftRotate(node->left);//quay trái trước

return rightRotate(node); // quay phải sau

}

// Trường hợp d: Right Left Case

if (balance < -1 && data < node->right->data){

node->right = rightRotate(node->right);//quay phải trước

return leftRotate(node);//quay trái sau

}

return node;//trả lại node

}

Thao tác loại bỏ node trên cây AVL: thuật toán loại node w trên cây AVL được thực hiện như sau:

Bước 1. Thực hiện loại bỏ 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 .

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 soay phải tại node z (right rotation). Ví

dụ của phép xoay left-left trên cây trong Hình 4.21.

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 soay trái tại node y sau đó soay phải tại

node z. Ví dụ của phép xoay left-right trên cây trong Hình 4.22.

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 soay trái tại node z (left rotation).

Ví dụ của phép xoay right-right trên cây trong Hình 4.23.

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 soay trái trái tại node y sau đó soay phải

tại node z. Ví dụ của phép xoay right-left trên cây trong Hình 4.23.

node\* AVL\_Tree ::deleteNode(node\* root, int key){//loại node trên cây AVL

//Bước 1: Thực hiện loại node giống như cây tìm kiếm

if (root == NULL)//nếu cây rỗng

return root;

if ( key < root->data )//nếu điếu này xảy ra

root->left = deleteNode(root->left, key);//tìm node sang bên trái

else if( key > root->data )//nếu điều này xảy ra

root->right = deleteNode(root->right, key); //tìm node sang bên phải

else { //nếu tìm thấy đúng node giá trị key

// nếu node là lá hoặc có một cây con

if( (root->left == NULL) || (root->right == NULL) ){

node \*temp = root->left ? root->left : root->right;

if(temp == NULL){ //lấy temp = NULL

temp = root;

root = NULL;

}

else // nếu node chỉ có một cây con

\*root = \*temp; //thay thế root bằng temp

free(temp);//giải phóng temp

}

else { //trường hợp node có hai cây con

//lấy node trái nhất của cây con phải

node\* temp = minValueNode(root->right);

root->data = temp->data;//thay nội dung node hiện tại

// sau đó loại bỏ node temp

root->right = deleteNode(root->right, temp->data);

}

}

//Bước2. Tìm node làm cây mất bằng cây:

if (root == NULL)//nếu cây rỗng

return root; //không cần cân bằng lại

// Cập nhật độ cao node

root->height = max(height(root->left), height(root->right)) + 1;

// Bước 3. Cân bằng cây

int balance = getBalance(root);

// 4 trường hợp làm cây mất cân bằng được xem xét:

//Trường hợp a. Cây mất cân bằng trái (Left Left Case)

if (balance > 1 && getBalance(root->left) >= 0)

return rightRotate(root); //ta quay phải

// Trường hợp b. Left Right Case

if (balance > 1 && getBalance(root->left) < 0) {

root->left = leftRotate(root->left);//ta quay trái trước

return rightRotate(root); //rồi quay phải sau

}

// Trường hợp c. Right Right Case

if (balance < -1 && getBalance(root->right) <= 0)

return leftRotate(root); //ta quay trái

// Trường hợp d. Right Left Case

if (balance < -1 && getBalance(root->right) > 0) {

root->right = rightRotate(root->right);//quay phải trước

return leftRotate(root); //quay trái sau

}

return root;

}