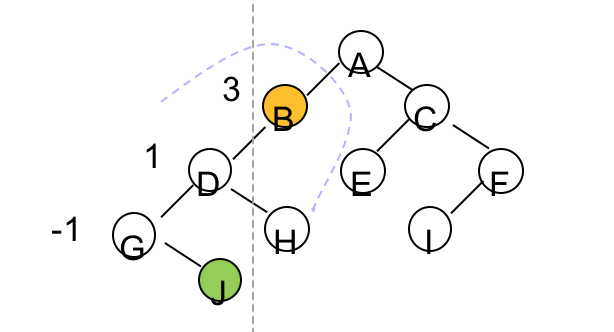
二叉查找树可以在插入节点的同时进行自我平衡的操作的树叫做自平衡二叉查找树

1、 一棵树的平衡系数是其左子树的高度减去其右子树的高度，当一棵树的平衡系数为-1、 0、1的时候，这棵树是处于平衡状态的，反之则是不平衡的。

为了编写程序方便，当出现平衡系数的绝对值大于等于2的情况的时候，就要将这棵非 平衡树转换成平衡树的状态。

1. 离插入节点最近的平衡系数的绝对值大于等于二的节点作为根节点所形成的树称为最 小非平衡子树。



例如这棵二叉树，当插入节点“J”的时候，以B为根节点所形成的子树称为最小非平衡子树，因为其左子树高度3与右子树高度0相差3，已经大于等于2.为了让树处于一个平衡树的状态，我们需要将子树B进行一定的旋转操作。

旋转分为四种情况：

1.从右侧开始的左旋转（非平衡状态均发生在右子树侧：RR）

2.从左侧开始的右旋转（非平衡状态均发生在左子树侧：LL）

3.先进行右旋转再进行左旋转（非平衡状态从上到下先发生在左子树，再发生到整棵树：LR）

4.先进行左旋转，再进行右旋转（非平衡状态从上到下先发生在右子树，再发生到整棵树：RL）

如何通过一个输入序列生成一个AVL树

1. 数据结构

struct node{

int data;

int height;

node\* leftchild;

node\* rightchild;

};

树节点

1. 算法
   1. 计算当前节点在树中的高度

int height(node\* t){

if(t!=NULL){

return t->height;

}

return 0;

}

b) 当树不平衡时进行旋转

左单旋转

node\* singleleft(node\* t){

node\* tem=t->leftchild;

t->leftchild=tem->rightchild;

tem->rightchild=t;

t->height = max(height(t->rightchild),height(t->leftchild))+1;

return tem;

}

右单旋转

node\* singleright(node\* t){

node\* tem=t->rightchild;

t->rightchild=tem->leftchild;

tem->leftchild=t;

t->height = max(height(t->rightchild),height(t->leftchild))+1;

return tem;

}

右左旋转

node\* doublelr(node\* t){

t->leftchild = singleright(t->leftchild);

node\* tem = singleleft(t);

return tem;

}

左右旋转

node\* doublerl(node\* t){

t->rightchild = singleleft(t->rightchild);

node\* tem = singleright(t);

return tem;

}

c) 插入函数

node\* insert\_in\_avl(int data,node\* t){ //data：要插入到树中的数据

if(t==NULL){ //t：树的根节点的指针

t = (node\*)malloc(sizeof(node));

t->data=data;

t->height=0;

t->leftchild=NULL;

t->rightchild=NULL;

}else if(data < t->data){

t->leftchild = insert\_in\_avl(data,t->leftchild);

if(height(t->leftchild)-height(t->rightchild) == 2){

if(data < t->leftchild->data){

t = singleleft(t);

}

else{

t = doublelr(t);

}

}

}else if(data > t->data){

t->rightchild = insert\_in\_avl(data,t->rightchild);

if(height(t->rightchild)-height(t->leftchild) == 2){

if(data > t->rightchild->data){

t = singleright(t);

}

else{

t = doublerl(t);

}

}

}

t->height = max(height(t->leftchild),height(t->rightchild))+1;

return t;

}