```cpp
//日期: 2018/ 时间:
#include <stdio.h>
#include <stdlib.h>
#include <algorithm>
using namespace std;

struct node{
    int v,height;   //节点权值，height为当前子树的高度
    node *lchild,*rchild;
};

//生成一个新节点，v为节点的权值
node* newNode(int v){
    node* Node = new node;
    Node->v = v;
    Node -> height = 1;
    Node->lchild = Node->rchild = NULL;
    return Node;
}

//获取以root为根节点的子树的当前height
int getHeight(node* root){
    if(root == NULL)    return 0;   //空节点高度为0
    return root->height;
}

int getBalanceFactor(node* root){   //计算节点root的平衡因子
    //左子树的高度减去右子树的高度
    return getHeight(root->lchild) - getHeight(root->rchild);
}

//更新节点root的height
void updateHeight(node* &root){
    //max(左孩子的height,右孩子的height)+1
    root->height = max(getHeight(root->lchild),getHeight(root->rchild)) + 1;
}

//查找操作 与 二叉树查找操作一模一样
void search(node* root,int x){
    if(root == NULL){       //空树，查找失败
        printf("search failed\n");
        return;
    }
    if(root->v == x){
        printf("%d\n",root->v);
        return;
    }else if(x < root->v){
        search(root->lchild,x);
    }else{
        search(root->rchild,x);
    }
}

//左旋
void L(node* &root){
    node* temp = root->rchild;      //root指向节点A,temp指向节点B
```

```
57      //1、让B的左子树成为A的右子树；2、让A成为B的左子树；3、将根节点设为B
58      root->rchild = temp->lchild;
59      temp->lchild = root;
60
61      updateHeight(root);      //更新节点A的高度
62      updataHeight(temp);      //更新节点B的高度
63
64      root = temp;
65  }
66
67  //右旋
68  void R(node* &root){
69      node* temp = root->lchild;      //root指向节点A,temp指向节点B
70      root->lchild = temp->rchild;
71      temp->rchild = root;
72
73      updateHeight(root);
74      updateHeight(temp);
75
76      root = temp;
77  }
78
79  //仅仅是在二叉树插入操作上加上平衡操作
80  void insert(node* &root,int v){
81      if(root == NULL){       //达到空节点
82          root = newNode(v);
83          return;
84      }
85      if(v < root->v){        //v比根节点的权值小
86          insert(root->lchild,v); //往左子树插入
87          updateHeight(root);     //更新树高
88          if(getBalanceFactor(root) == 2){
89              if(getBalanceFactor(root->lchild) == 1){    //LL型
90                  R(root);
91              }else if(getBalanceFactor(root->lchild) == -1){ //LR型
92                  L(root->lchild);
93                  R(root);
94              }
95          }
96      }else{
97          insert(root->rchild,v);
98          updateHeight(root);
99          if(getBalanceFactor(root) == -2){
100             if(getBalanceFactor(root->rchild) == -1){   //RR型
101                 L(root);
102             }else if(getBalanceFactor(root->rchild) == 1){  //RL型
103                 R(root->rchild);
104                 L(root);
105             }
106         }
107     }
108 }
109
110 //AVL树的建立
111 node* Create(int data[],int n){
112     node* root = NULL;
```

```
113     for(int i=0;i<n;i++){
114         insert(root,data[i]);
115     }
116     return root;
117 }
118
119 int main(){
120
121
122     return 0;
123 }
124
125
```