# 二叉排序树

## 一.项目背景：

依次输入关键字并建立二叉排序树，实现二叉排序数的插入和查找功能。

## 二.项目功能

二叉排序树就是指将原来已有的数据根据大小构成一棵二叉树，二叉树中的所有结点数据满足一定的大小关系，所有的左子树中的结点均比根结点小，所有的右子树的结点均比根结点大。

二叉排序树查找是指按照二叉排序树中结点的关系进行查找，查找关键自首先同根结点进行比较，如果相等则查找成功；如果比根节点小，则在左子树中查找；如果比根结点大，则在右子树中进行查找。这种查找方法可以快速缩小查找范围，大大减少查找关键的比较次数，从而提高查找的效率。

项目示例：



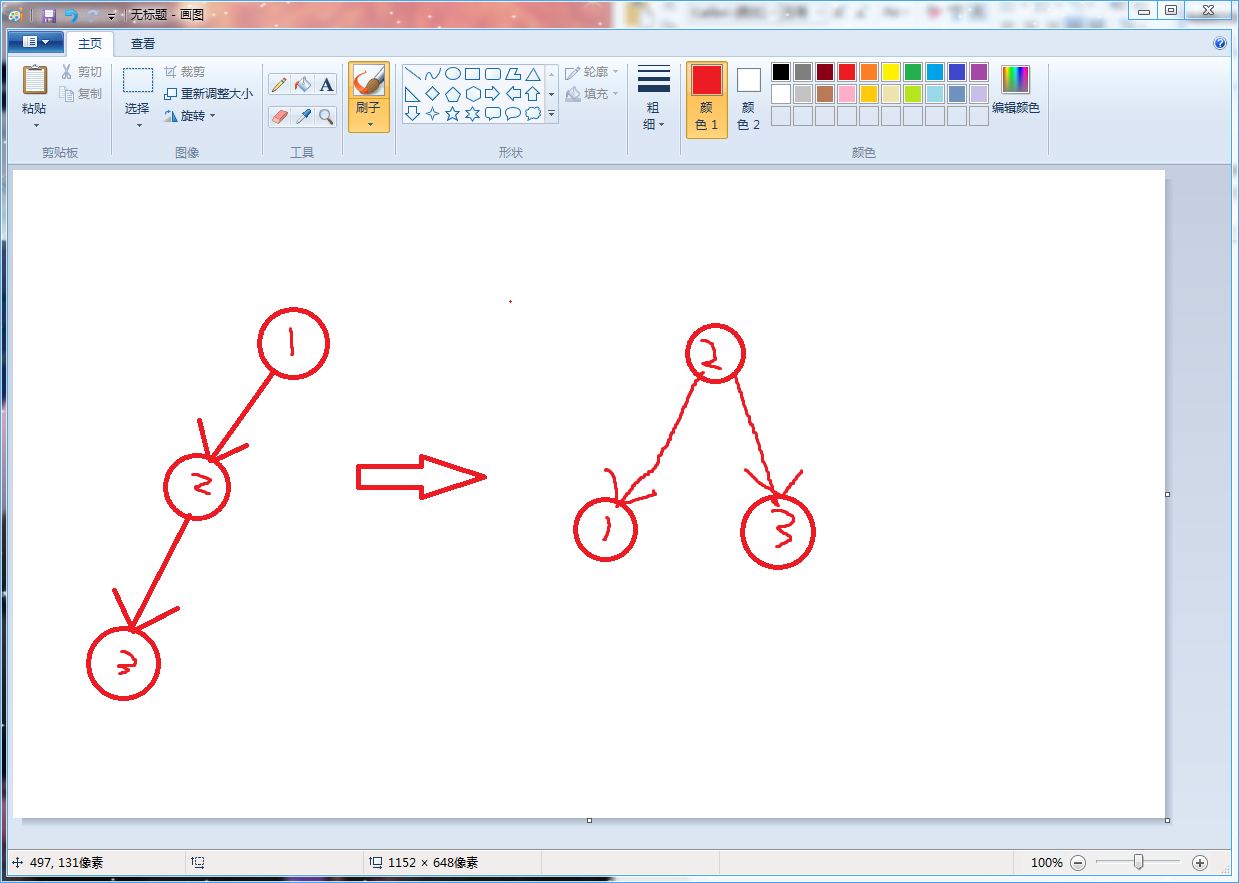
## 三.程序描述

对于本项目我认为二叉排序树不够稳定，遇到某些极端数据，查询的时间复杂度会退化为O(n)，故我实现了平衡二叉树（AVL），插入与查询的时间复杂度都为O(logn)，是一个很稳定高效的数据结构。

平衡二叉树的算法在书上，Google，百度都可以找到很多资料，故这里不再赘述。

现展示核心代码：

//单左旋



pointer \*SingleRotateWithLeft(pointer \*x)

{

pointer \*y;

y=x->left;

x->left=y->right;

y->right=x;

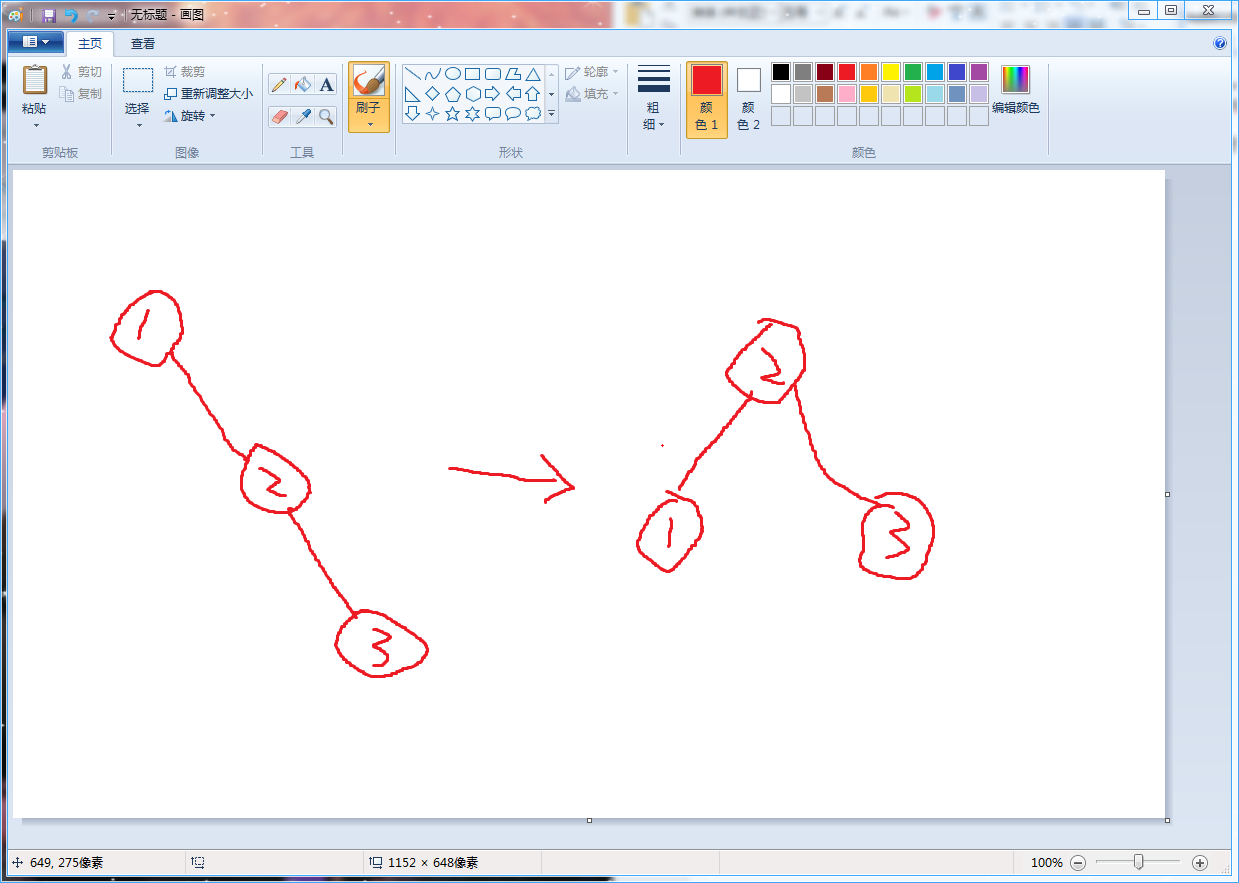
x->height=max(h(x->left),h(x->right))+1;

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

return y;

}

//单右旋



pointer \*SingleRotateWithRight(pointer \*x)

{

pointer \*y;

y=x->right;

x->right=y->left;

y->left=x;

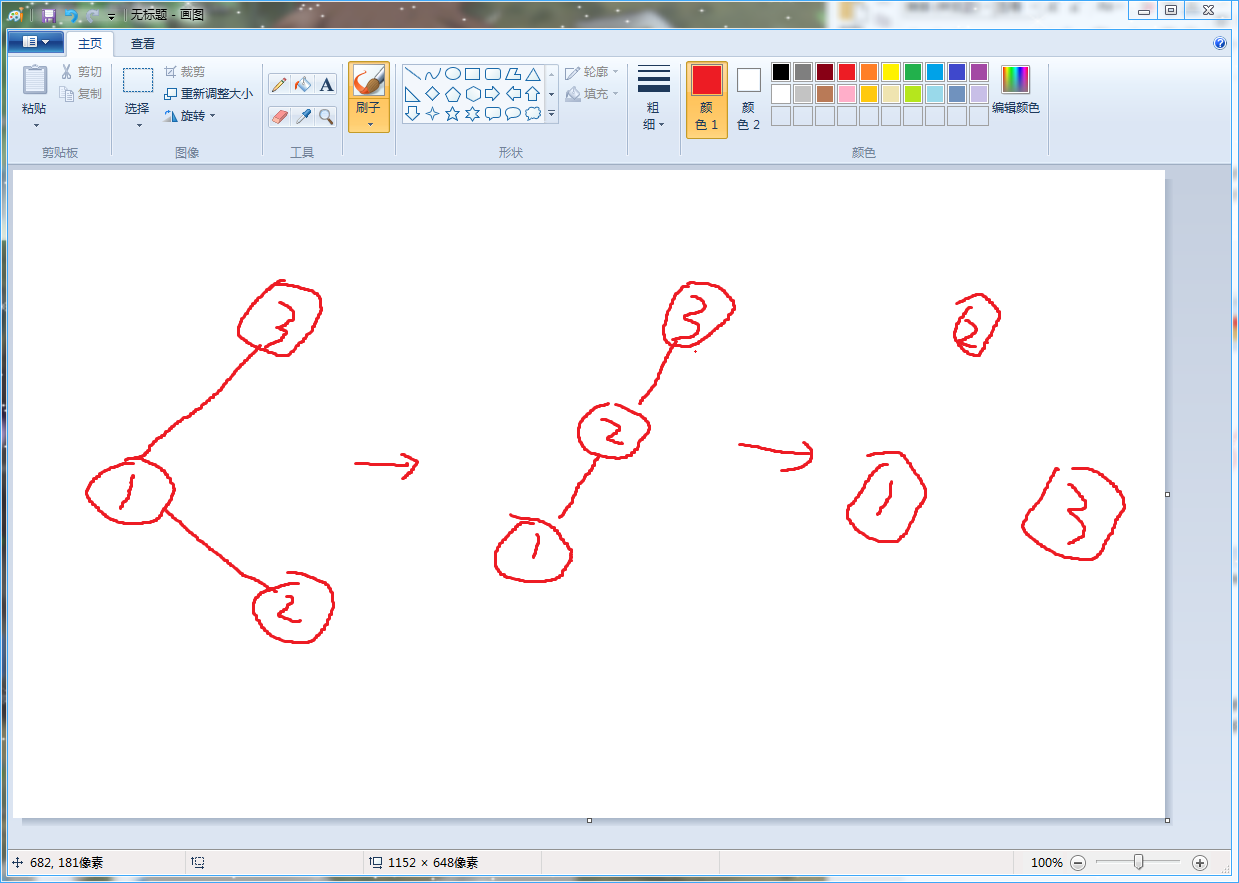
x->height=max(h(x->left),h(x->right))+1;

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

return y;

}

//双左旋



pointer \*DoubleRotateWithLeft(pointer \*x)

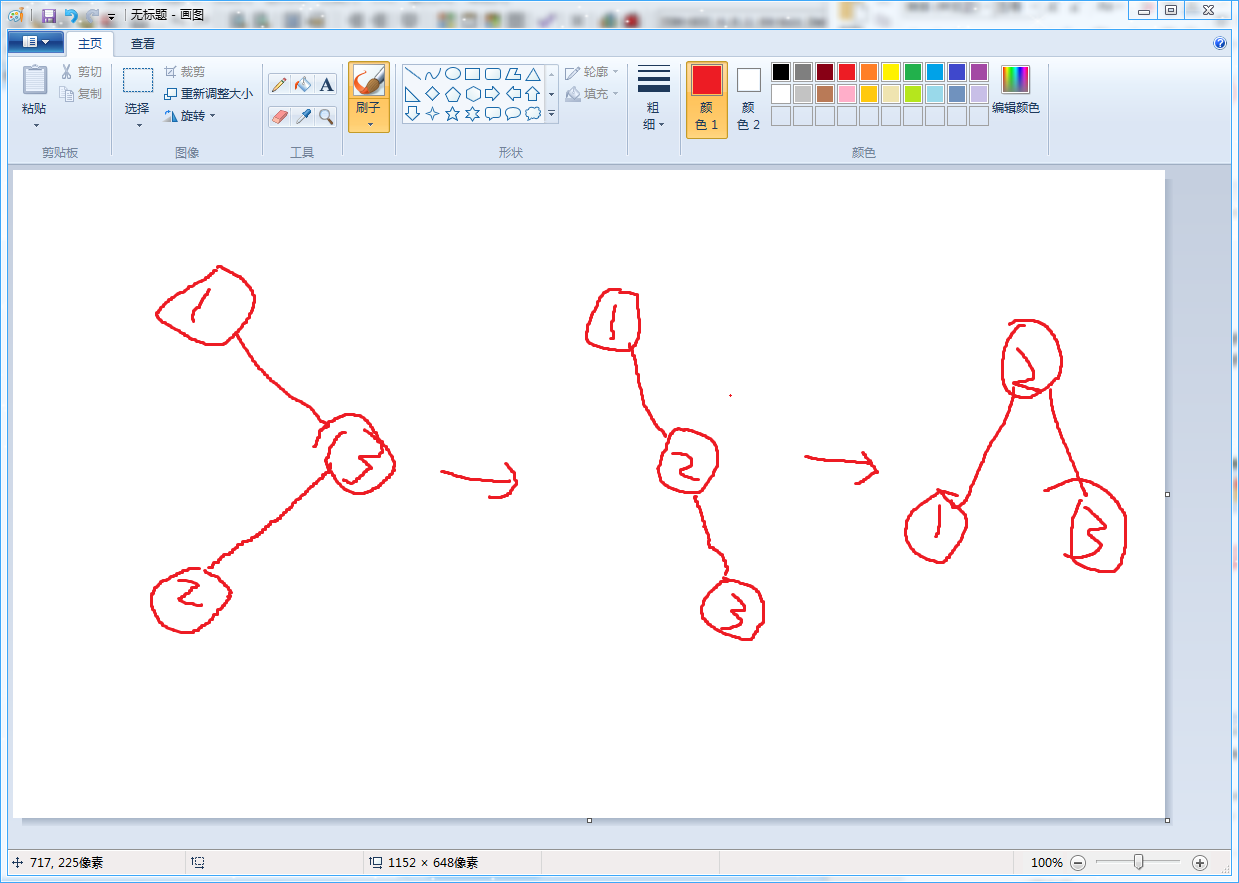
{

x->left=SingleRotateWithRight(x->left);

return SingleRotateWithLeft(x);

}

//双右旋



pointer \*DoubleRotateWithRight(pointer \*x)

{

x->right=SingleRotateWithLeft(x->right);

return SingleRotateWithRight(x);

}

实现以上4个操作就可以写出一个平衡二叉树了

具体细节请参见源代码