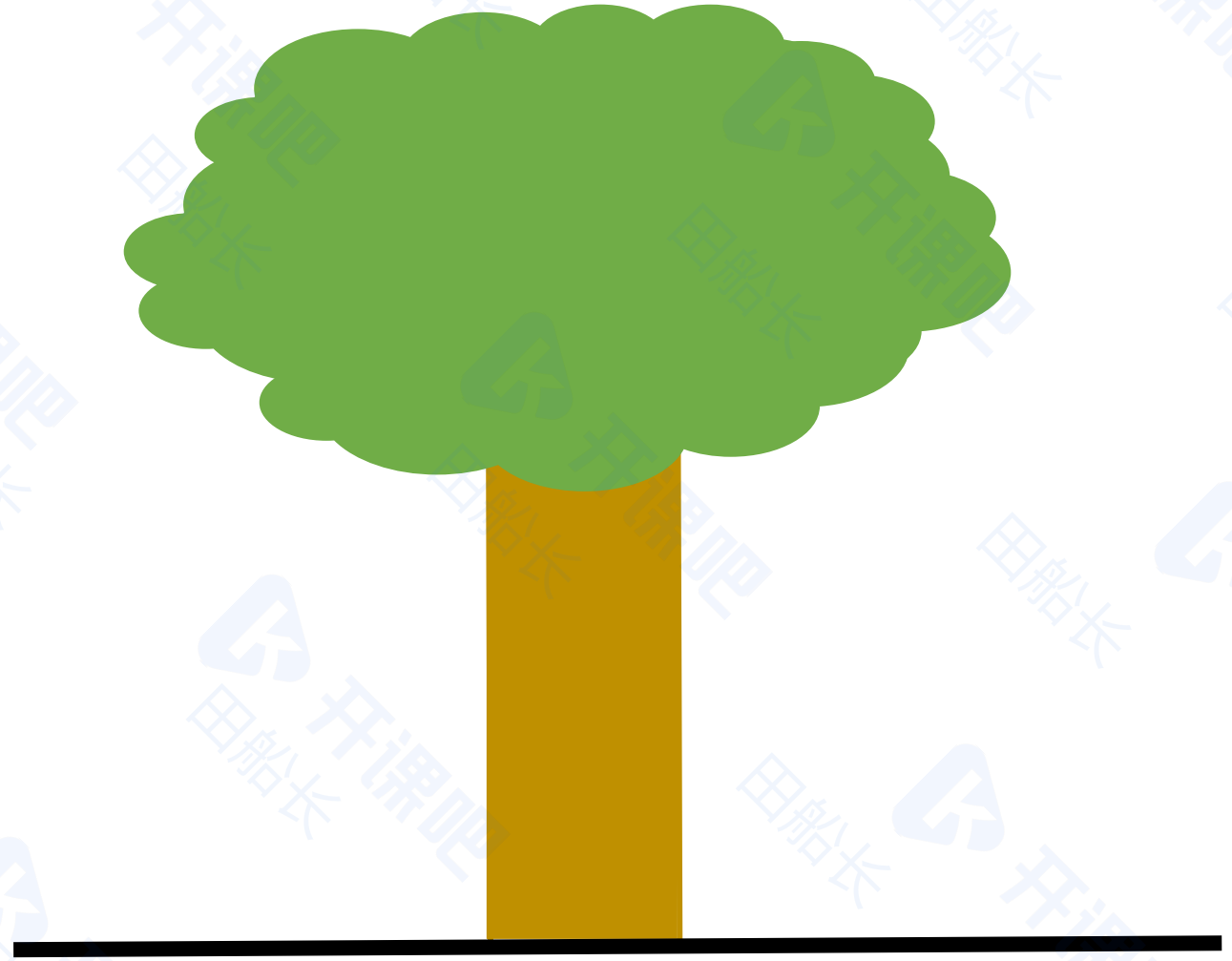
开课吧

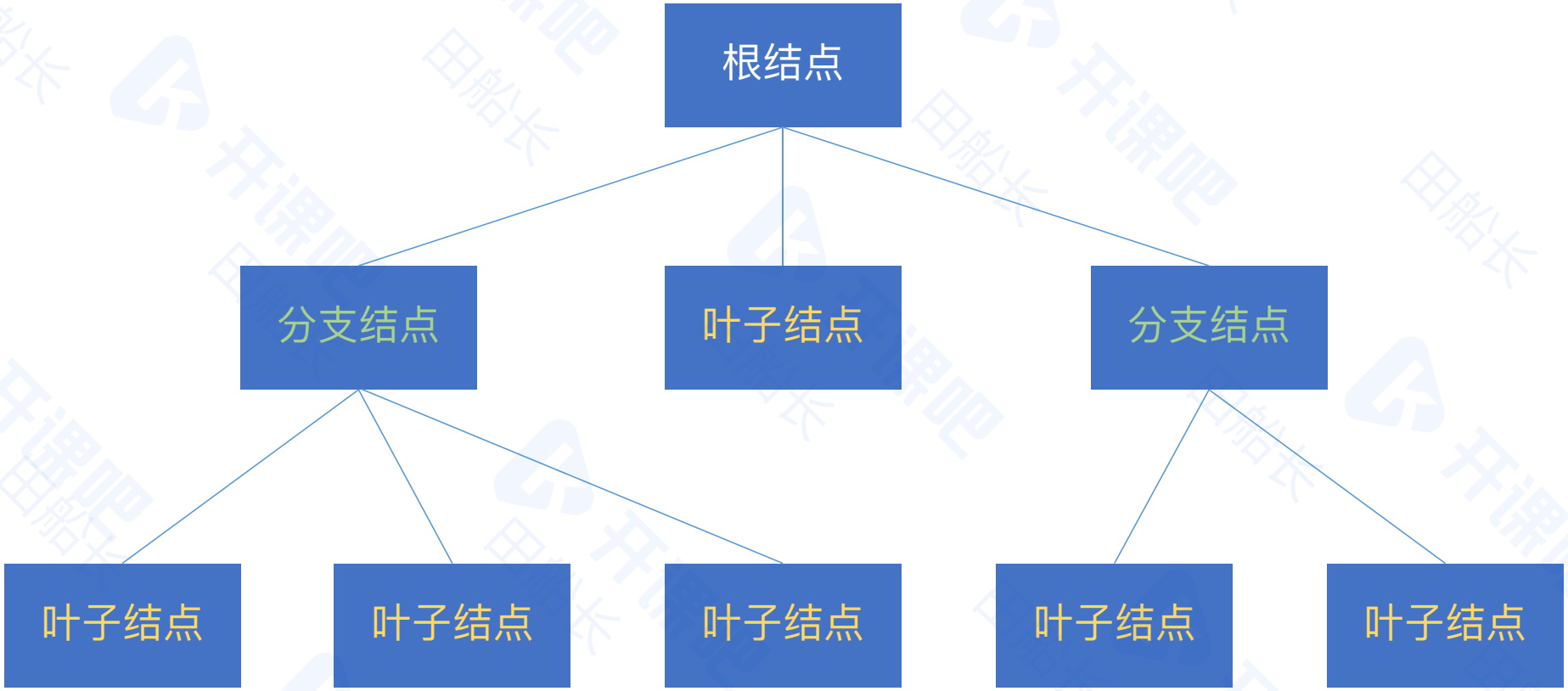
# 树形结构

田船长

# 现实中的树



# 数据结构中的树



# 树中结点的结构定义

```
struct Node { //树结点的定义
    EleType data; //树结点的数据域
    Node *child[]; //子结点指针集合
};
```

对于子结点指针，有可能用数组实现
也有可能用链表实现
树还可以使用数组实现
只需记录每个结点的父结点即可

# 树的基本术语

1．父结点、子结点、兄弟结点，祖先、子孙

2．结点的度，树的度

3．叶子结点（终端结点）与分支结点（非终端结点）

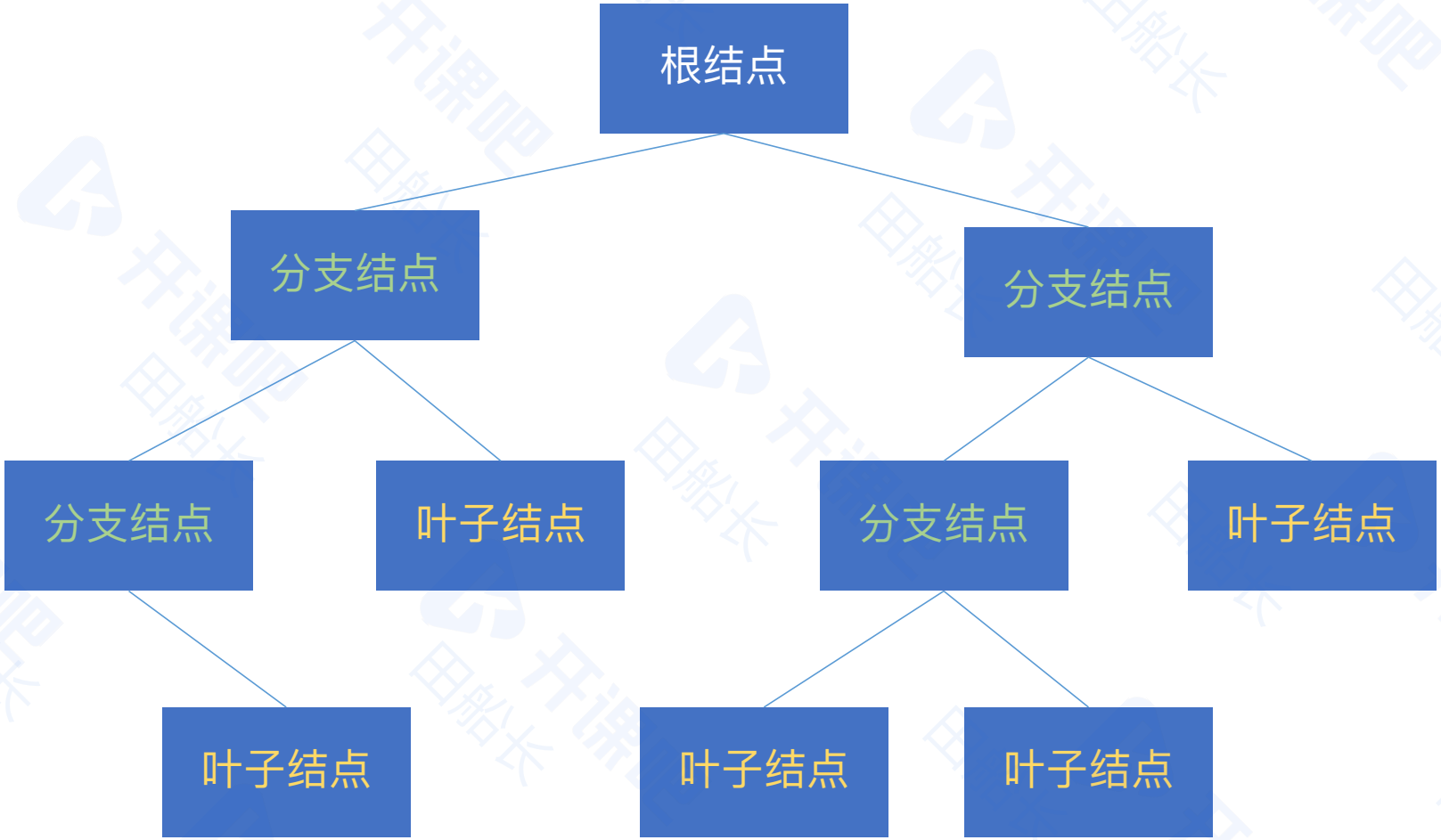
4．结点的深度，高度和层次

5．路径与路径长度

# 什么是森林

树的集合称为森林

# 二叉树



# 二叉树中结点的结构定义

```
struct Node { //二叉树结点的定义
    EleType data; //二叉树结点的数据域
    Node *left_child, *right_child; //左子结点与右子结点指针
};
```

左子树与右子树也经常写作left（lchild）和right（rchild）
同理二叉树也可以使用数组来存储
在后面的内容中会讲到

# 二叉树的基本术语

1．左孩子（左子树），右孩子（右子树）

2．每层的结点个数

3．满二叉树，完全二叉树

4．使用一维数组存储二叉树

5．广义表

# 二叉树的遍历

1．二叉树的先序遍历

2．二叉树的中序遍历

3．二叉树的后序遍历

一般使用递归实现

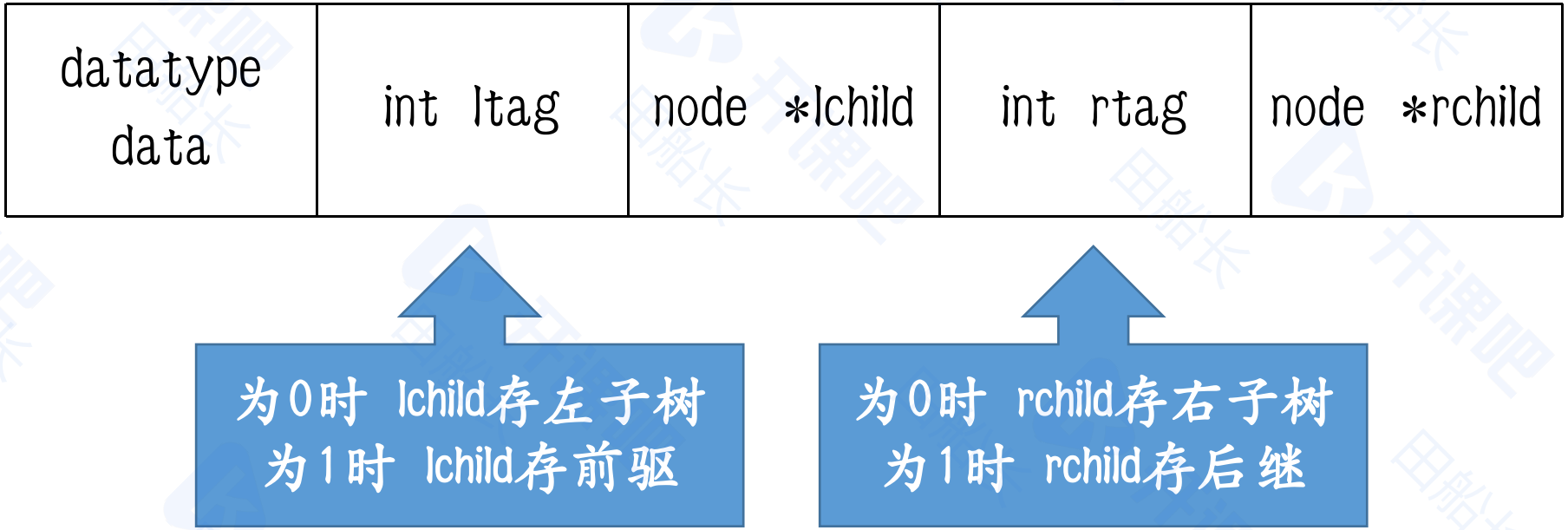
4．二叉树的层序遍历

一般使用队列实现

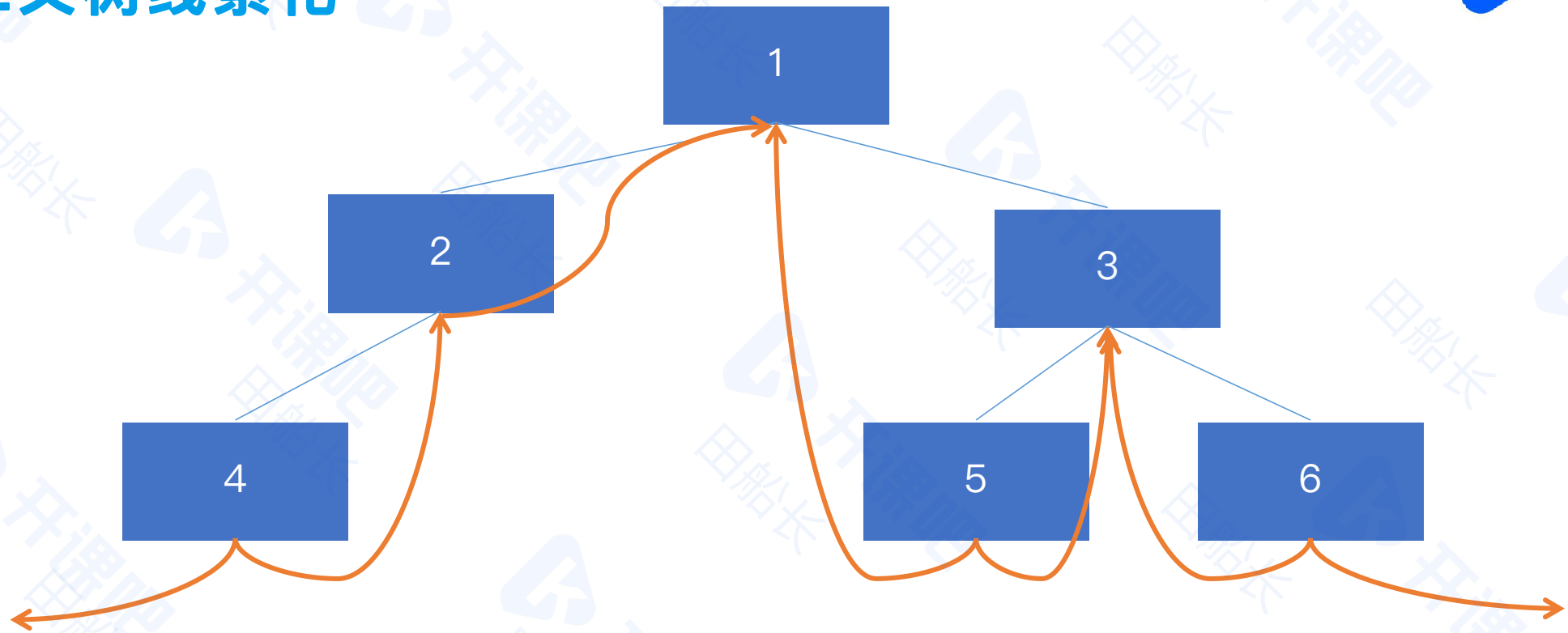
5．根据遍历还原树

# 二叉树线索化

借用树中结点内的空指针，保存前驱及后继的信息
线索化二叉树结点与普通二叉树结点在结构上略有不同



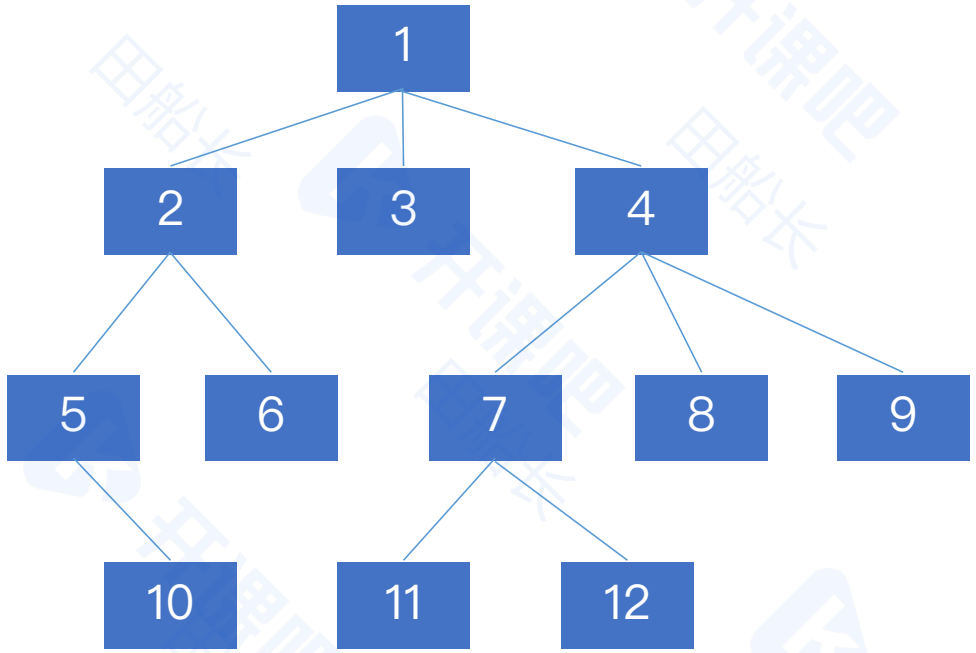
# 二叉树线索化

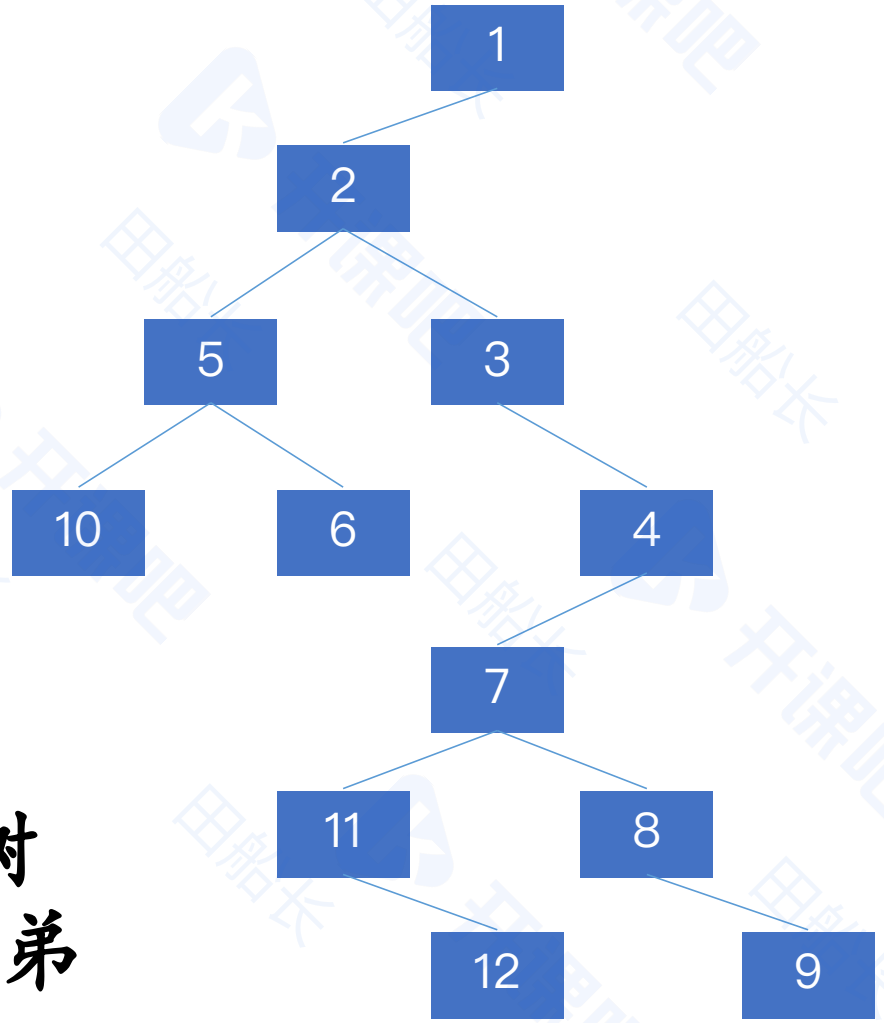



图为二叉树中序遍历线索化后的树

线索化二叉树的关键：左前驱，右后继

# 树与二叉树的转换




一颗树和与之对应的二叉树

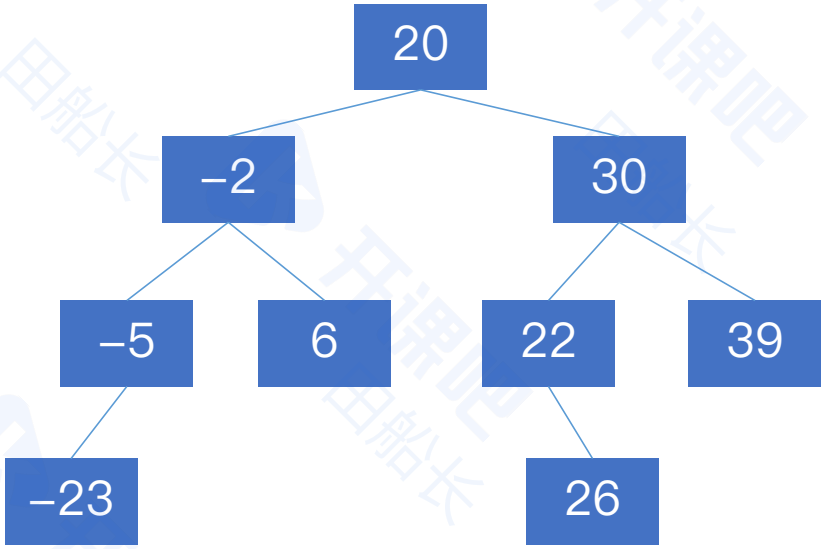
转换的关键：左孩子、右兄弟

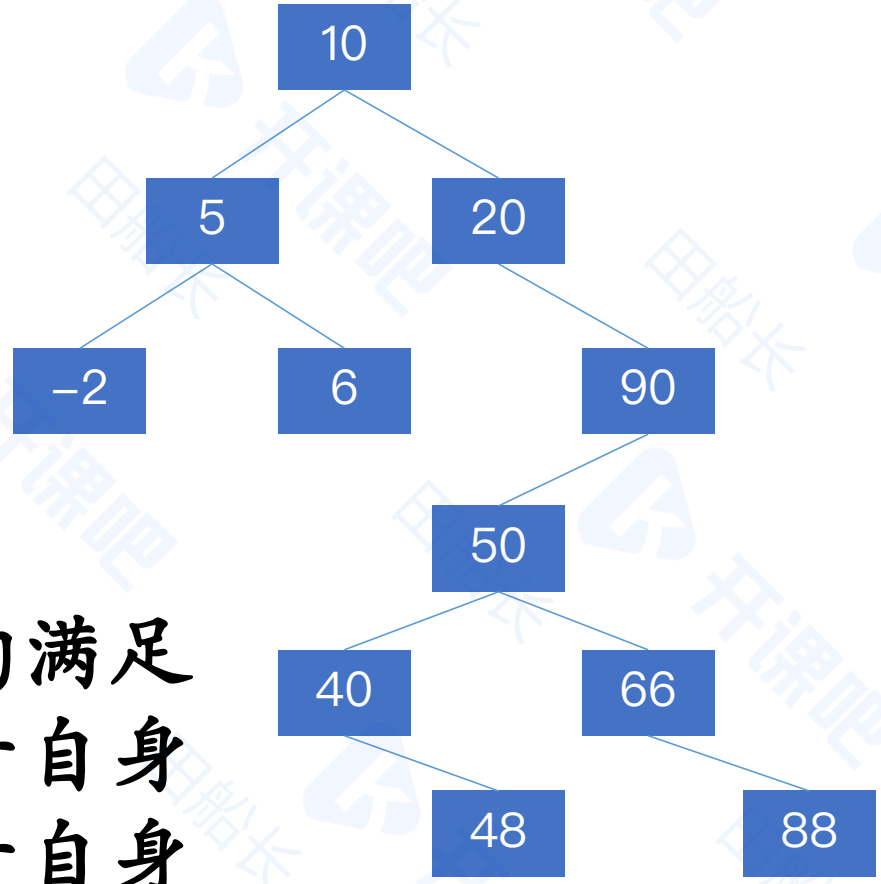
# 树与森林的遍历

树有先（根）序遍历与后（根）序遍历
树的先序遍历与对应二叉树的先序遍历相同
树的后序遍历与对应二叉树的中序遍历相同

森林有先序遍历与中序遍历
森林的先序遍历为内部树的先序遍历相连
森林的中序遍历为内部树的后序遍历相连
（森林中的树，树根互为兄弟）

# 二叉排序树（二叉查找树、二叉搜索树、BST）

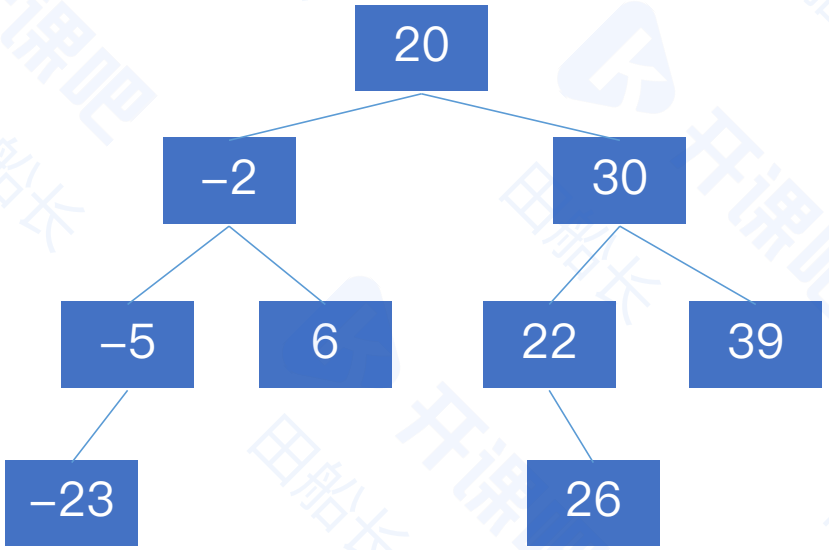



如果一棵二叉树中的任意结点均满足
它左子树上所有的结点值都小于自身
它右子树上所有的结点值都大于自身
　　那么这棵树即为二叉排序树

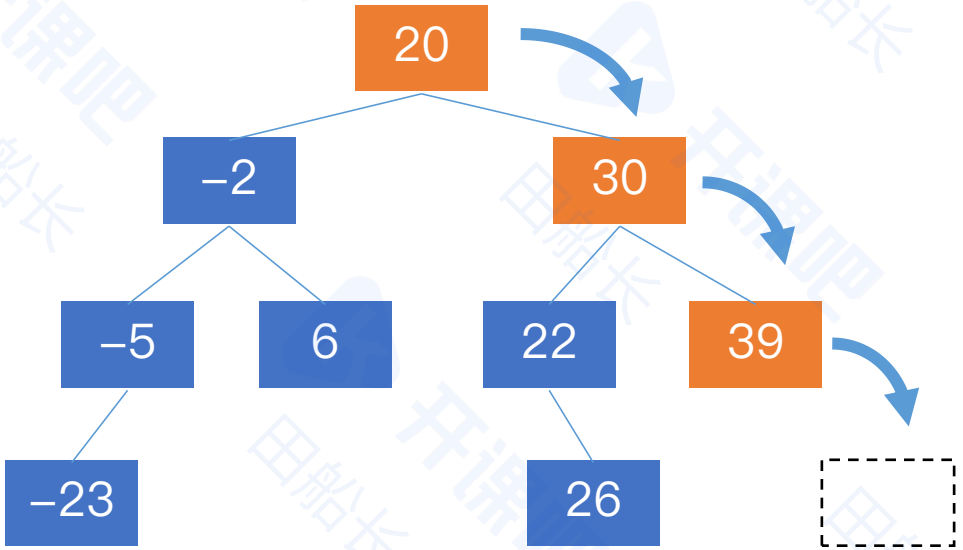
# 二叉排序树（二叉查找树、二叉搜索树、BST）

插入元素42



# 二叉排序树（二叉查找树、二叉搜索树、BST）

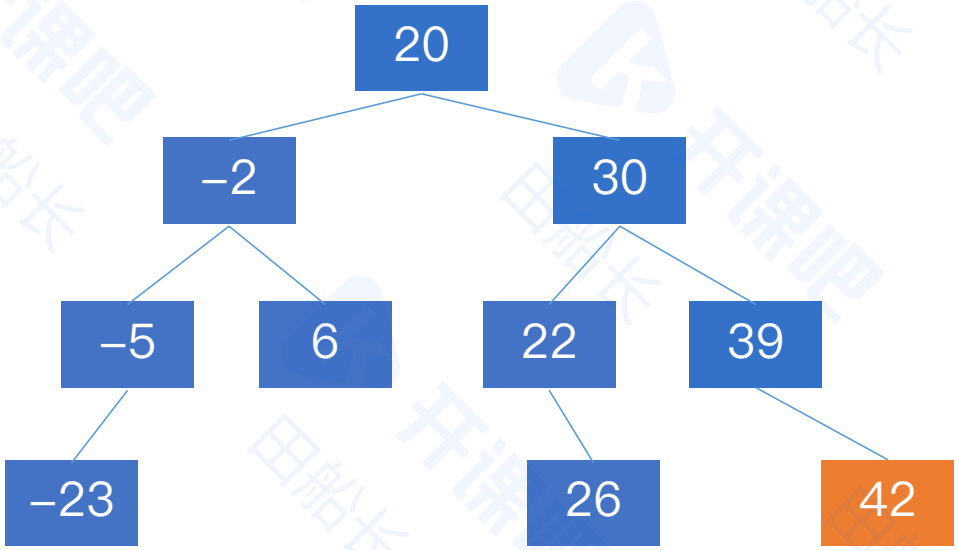
插入元素42




从树根出发，进行查找，找到对应插入位置

# 二叉排序树（二叉查找树、二叉搜索树、BST）

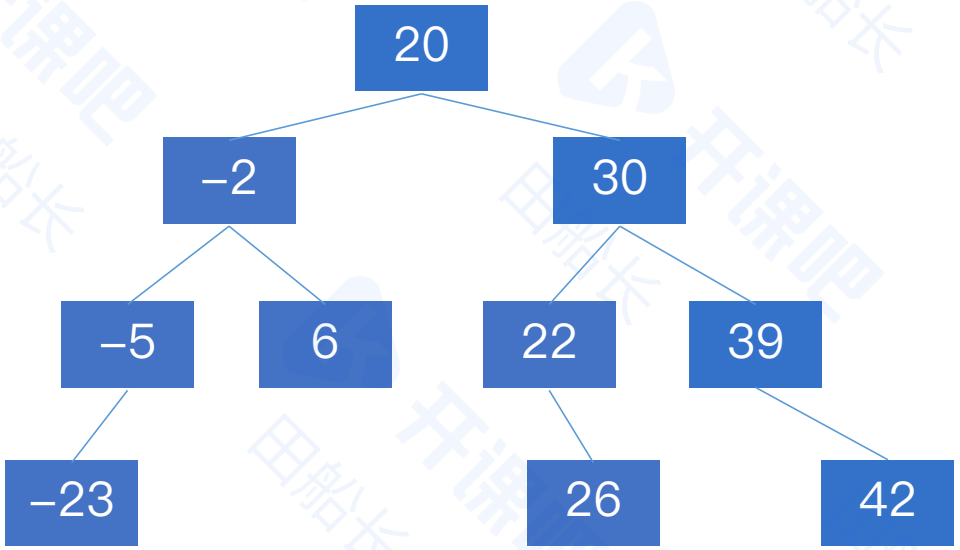
插入元素42




在对应位置进行插入操作

# 二叉排序树（二叉查找树、二叉搜索树、BST）

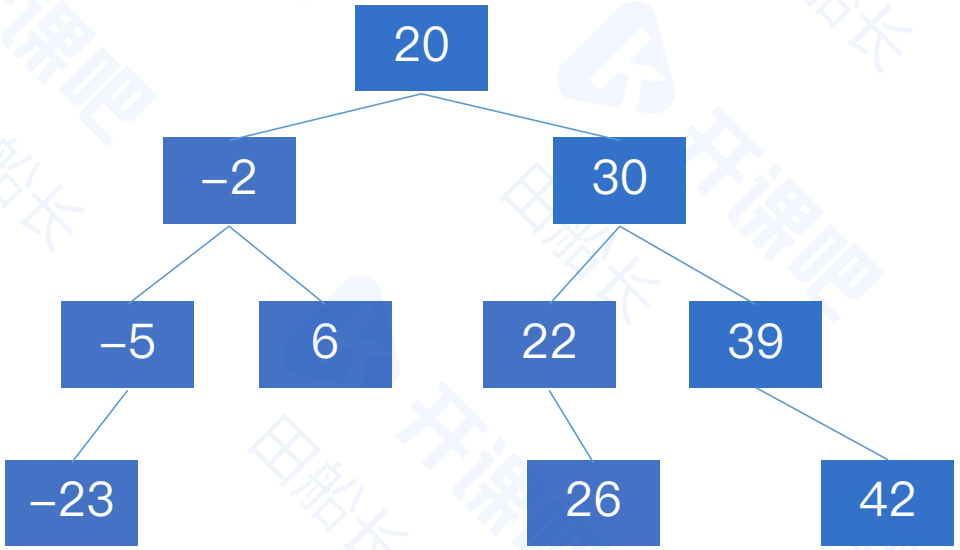
插入元素42




插入完毕

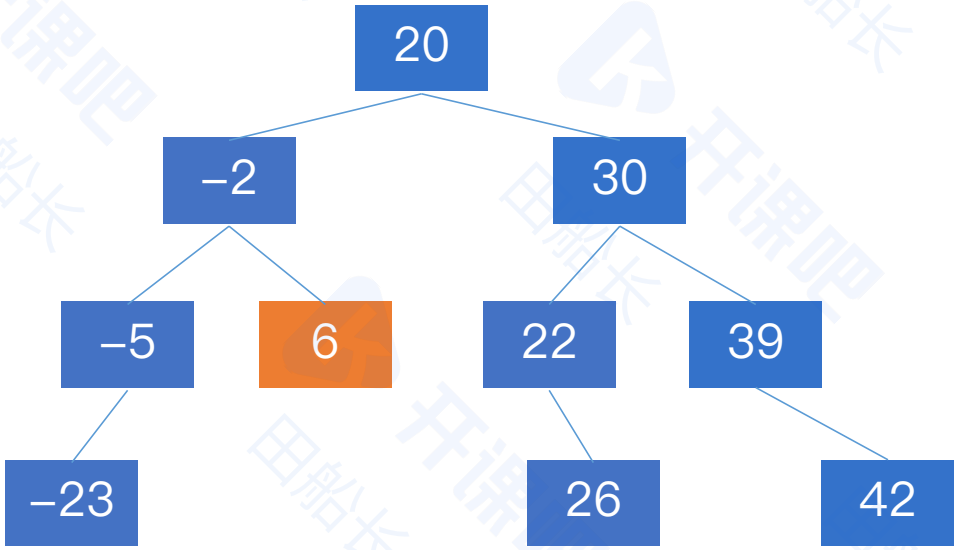
# 二叉排序树（二叉查找树、二叉搜索树、BST）

删除元素6



# 二叉排序树（二叉查找树、二叉搜索树、BST）

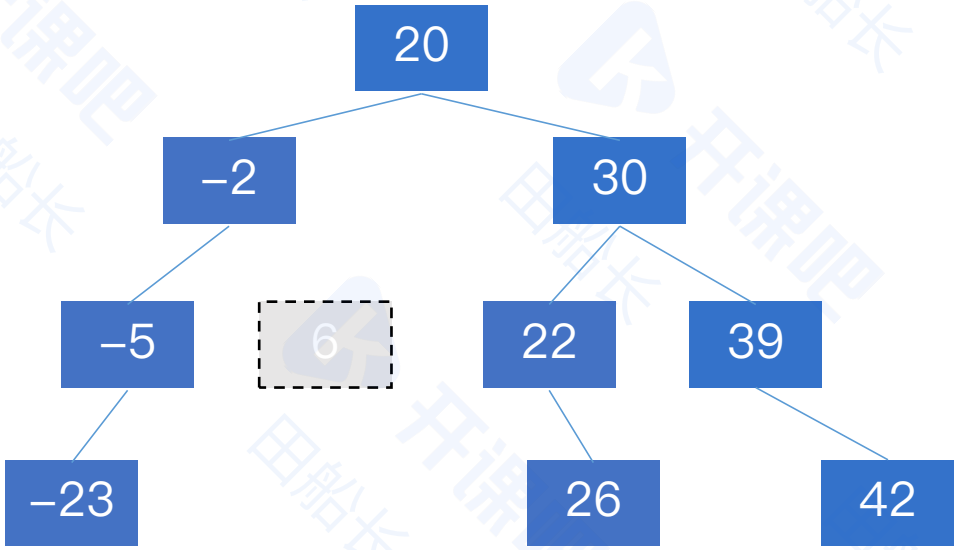
删除元素6




从树根出发，进行查找，找到对应元素

# 二叉排序树（二叉查找树、二叉搜索树、BST）

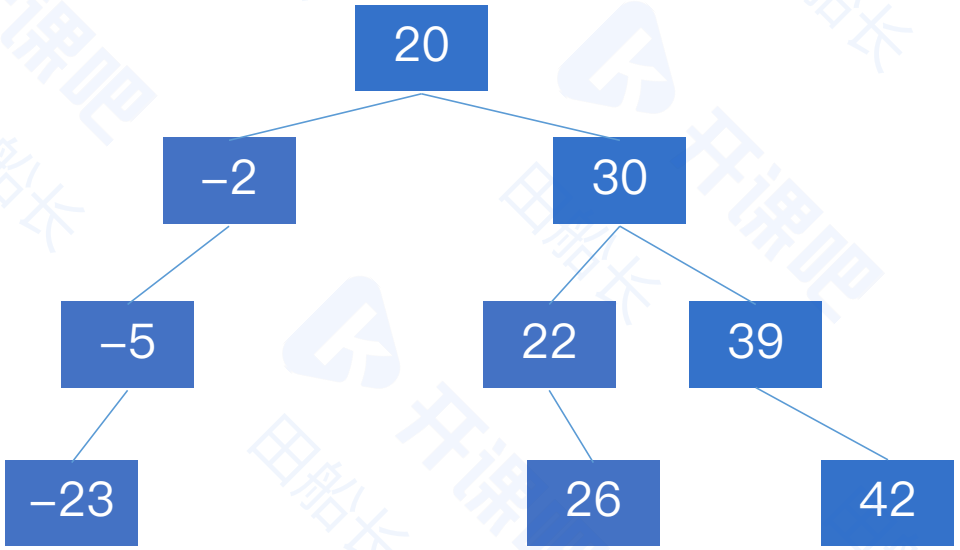
删除元素6




是叶子结点，直接删除

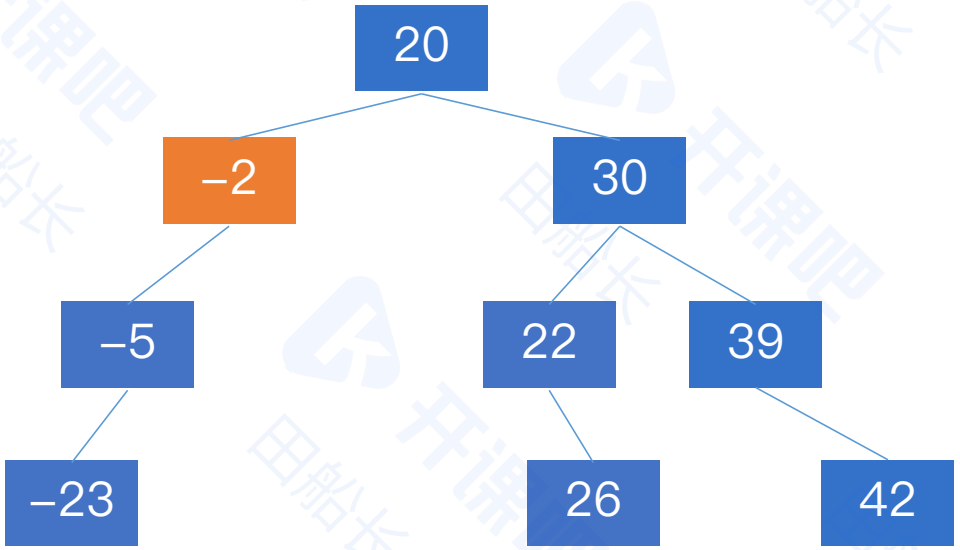
# 二叉排序树（二叉查找树、二叉搜索树、BST）

删除元素-2



# 二叉排序树（二叉查找树、二叉搜索树、BST）

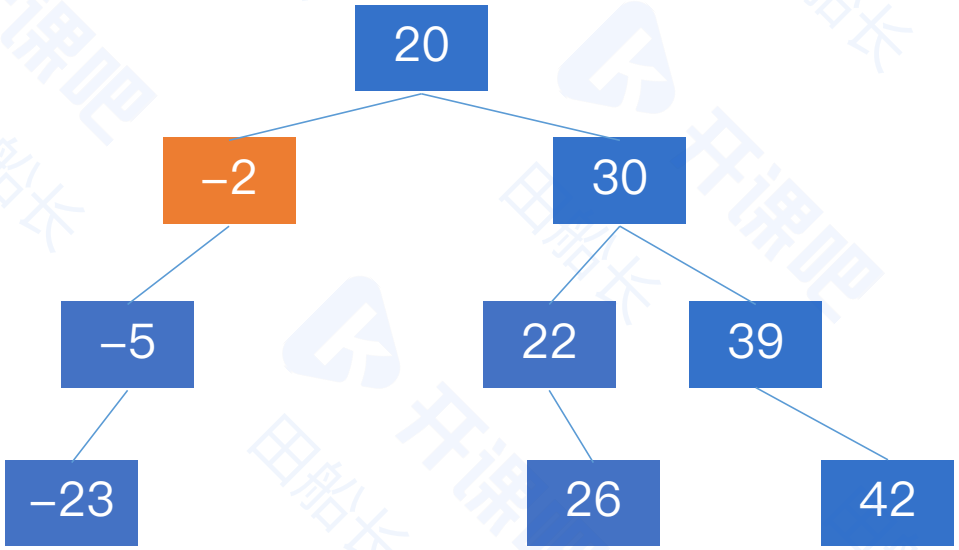
删除元素-2




从树根出发，进行查找，找到对应元素

# 二叉排序树（二叉查找树、二叉搜索树、BST）

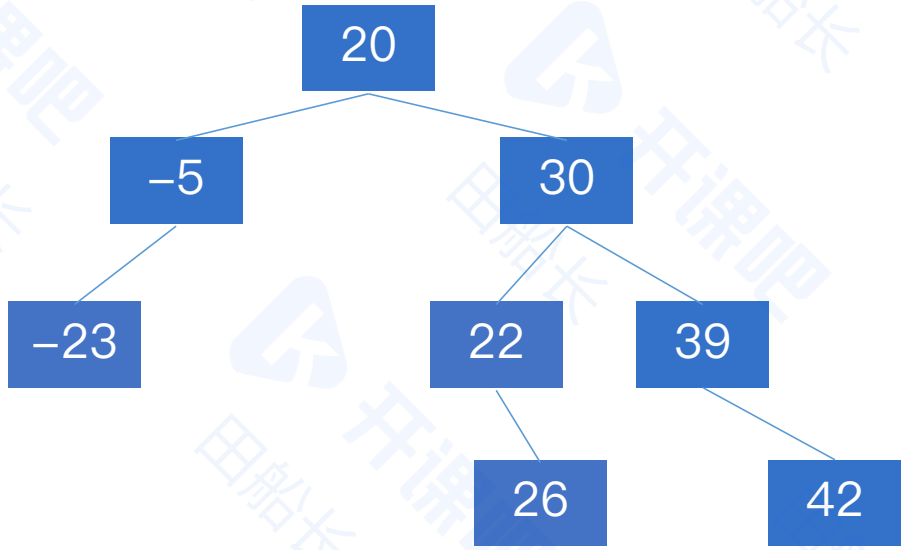
删除元素-2




若该元素只有左子树或只有右子树
则用其左子树或右子树将其替代进行删除

# 二叉排序树（二叉查找树、二叉搜索树、BST）

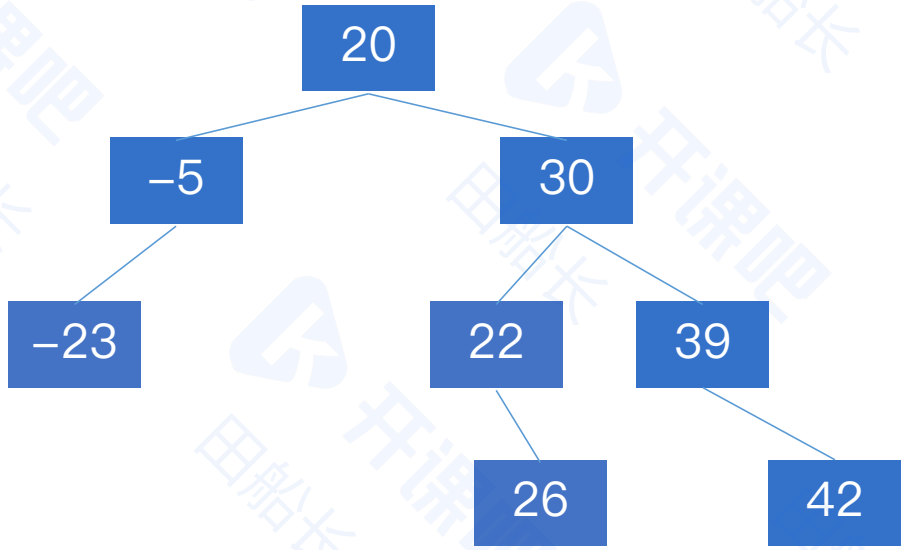
删除元素-2




若该元素只有左子树或只有右子树
则用其左子树或右子树将其替代进行删除

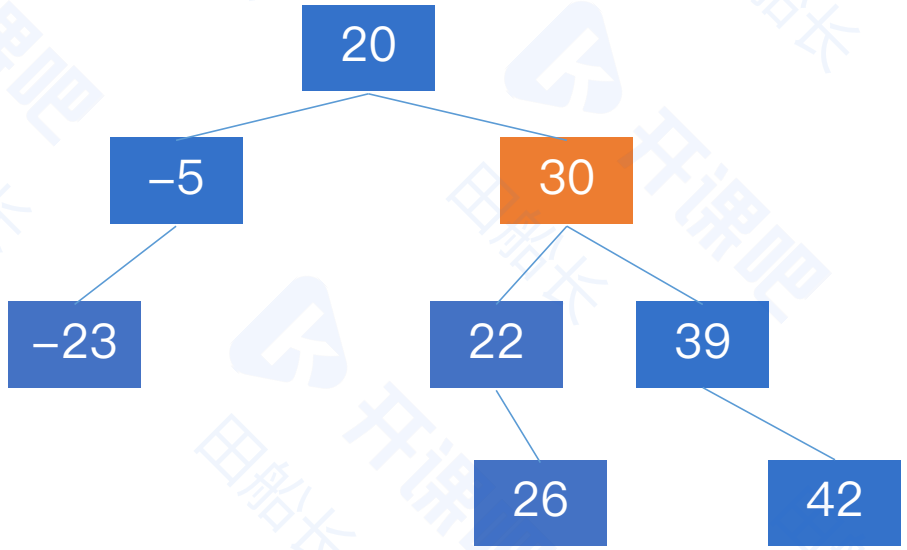
# 二叉排序树（二叉查找树、二叉搜索树、BST）

删除元素30



# 二叉排序树（二叉查找树、二叉搜索树、BST）

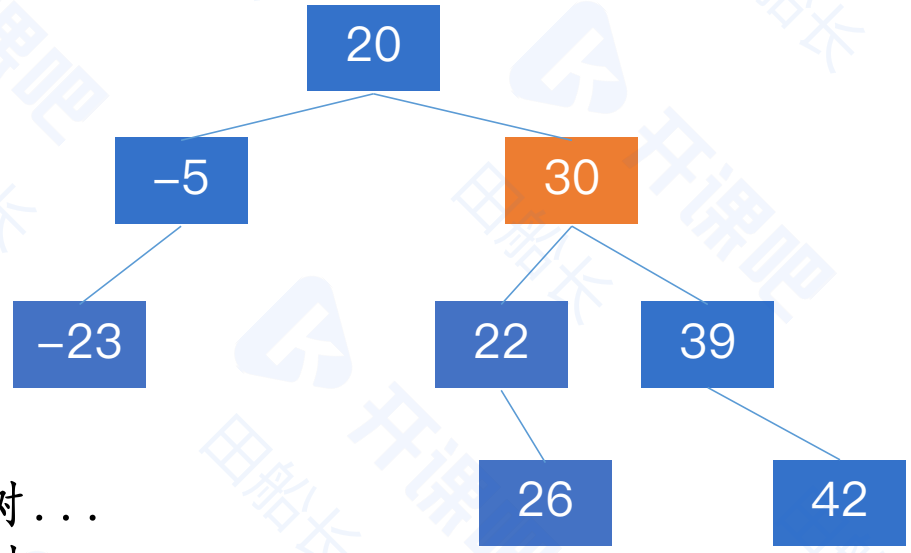
删除元素30




从树根出发，进行查找，找到对应元素

# 二叉排序树（二叉查找树、二叉搜索树、BST）

删除元素30

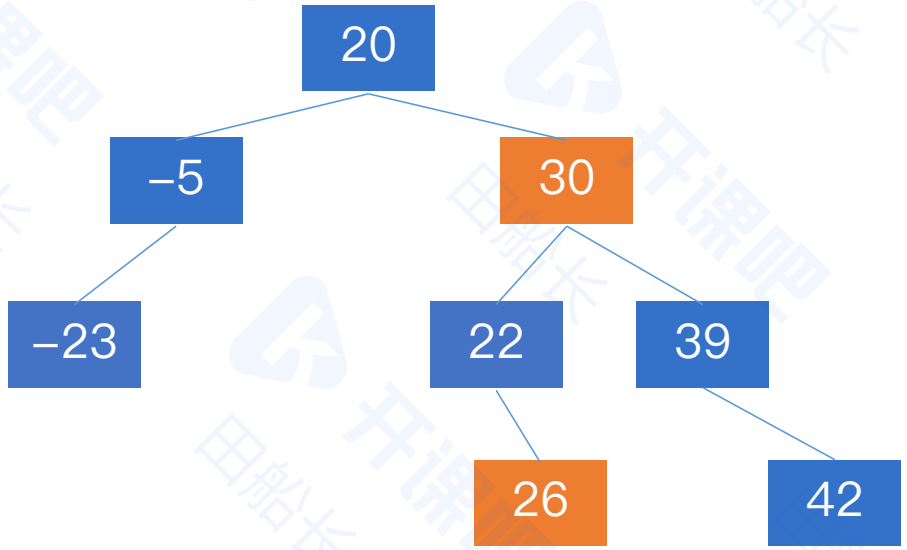



前驱：左子树的右子树的右子树...
后继：右子树的左子树的左子树...
走到没有结点可以走为止

该元素有左子树也有右子树，不能直接删除
此时先找到它的前驱（或后继），与该元素交换
接下来继续对交换后的结点进行删除

# 二叉排序树（二叉查找树、二叉搜索树、BST）

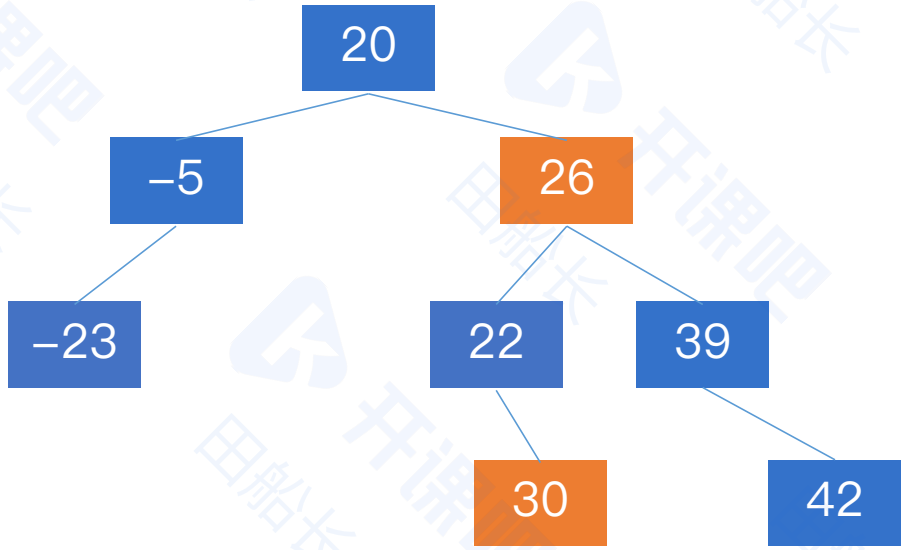
删除元素30




用前驱进行交换，26为前驱

# 二叉排序树（二叉查找树、二叉搜索树、BST）

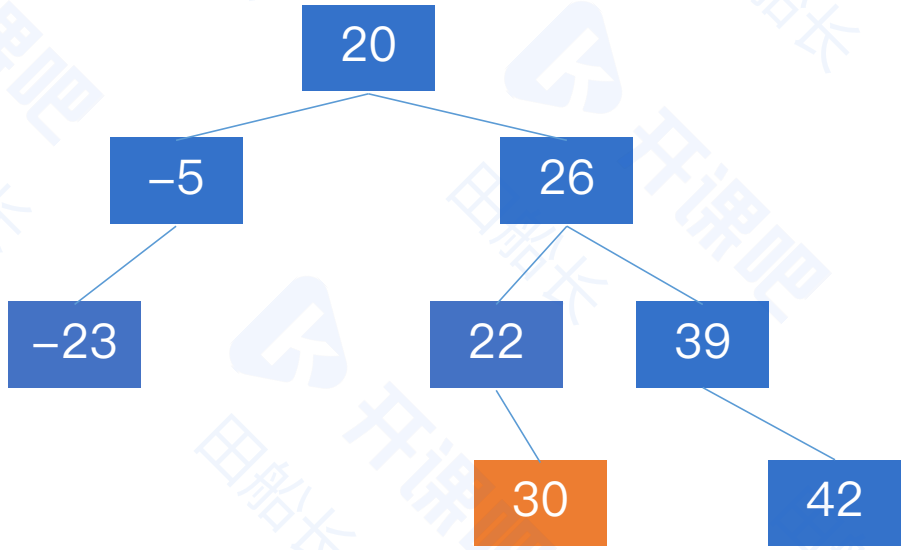
删除元素30




26为前驱，进行交换

# 二叉排序树（二叉查找树、二叉搜索树、BST）

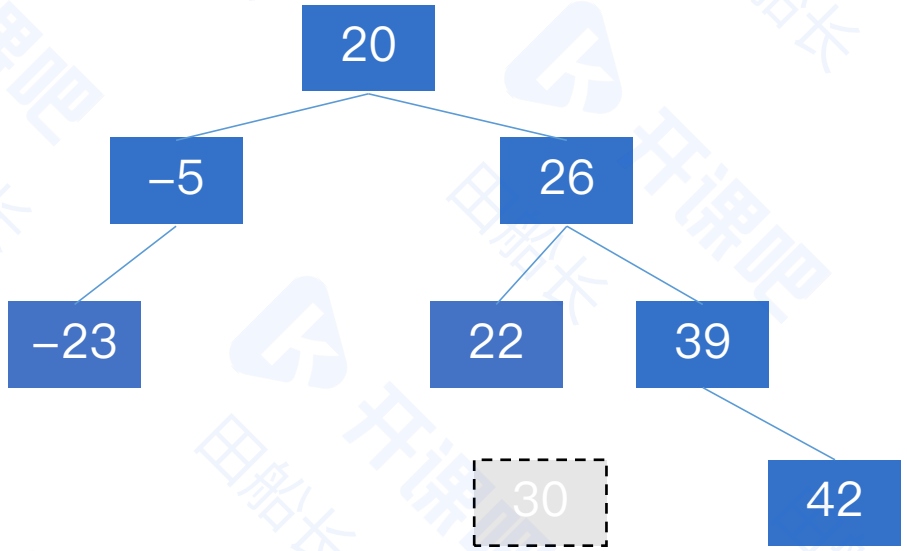
删除元素30




继续删除30

# 二叉排序树（二叉查找树、二叉搜索树、BST）

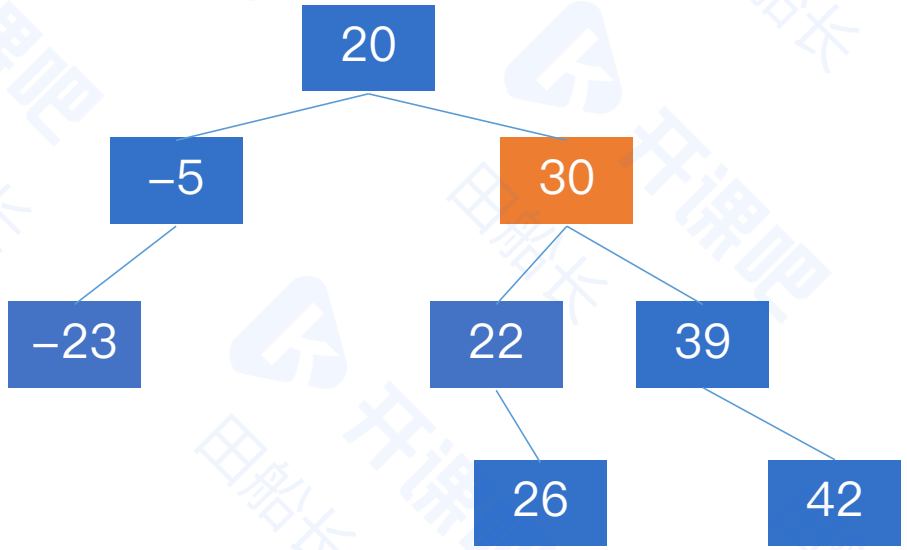
删除元素30




此时其为叶子结点，直接删除即可，删除完毕

# 二叉排序树（二叉查找树、二叉搜索树、BST）

删除元素30

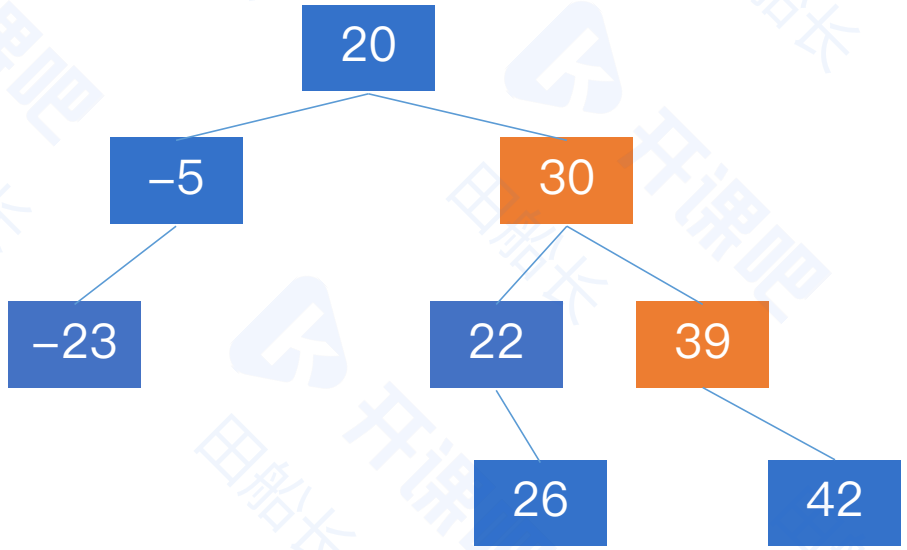



将树恢复原状

上一次用前驱进行交换，这一次用后继进行交换

# 二叉排序树（二叉查找树、二叉搜索树、BST）

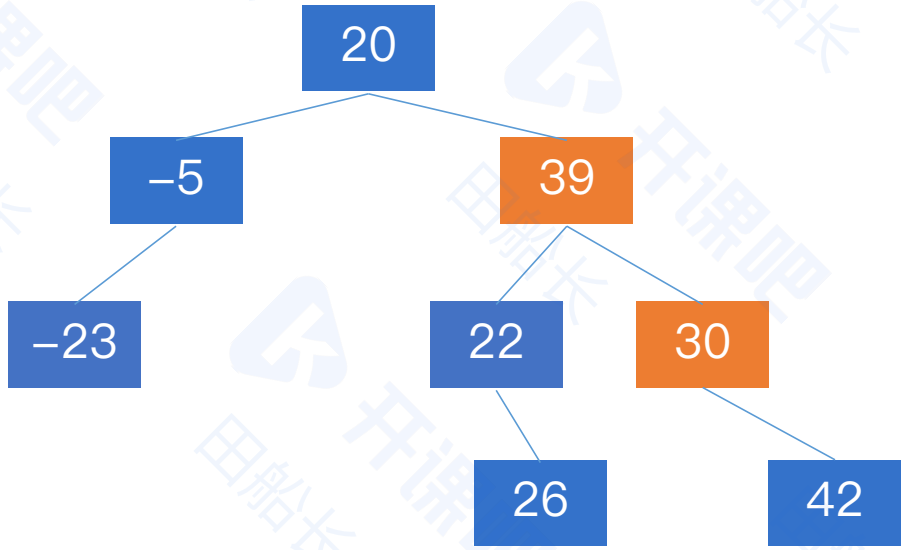
删除元素30




39为后继

# 二叉排序树（二叉查找树、二叉搜索树、BST）

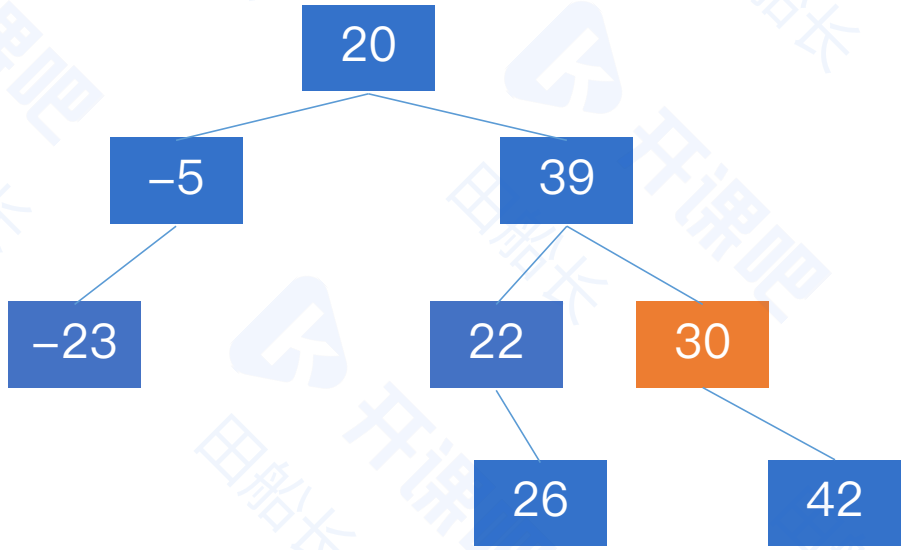
删除元素30




39为后继，进行交换

# 二叉排序树（二叉查找树、二叉搜索树、BST）

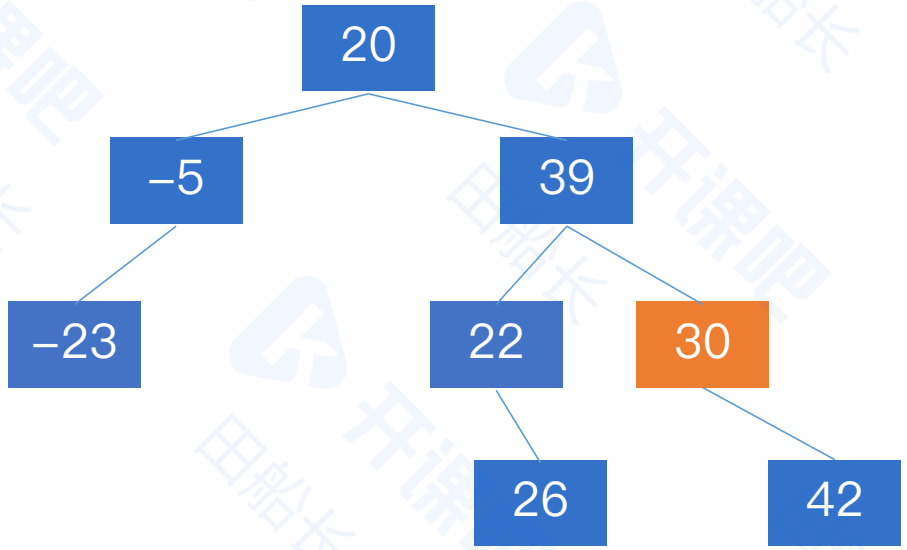
删除元素30




继续删除30

# 二叉排序树（二叉查找树、二叉搜索树、BST）

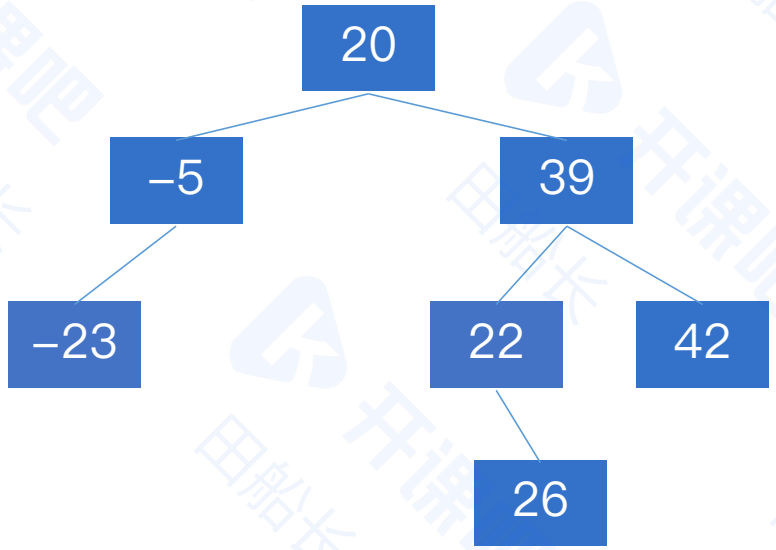
删除元素30




该元素只有右子树
用其右子树将其替代进行删除

# 二叉排序树（二叉查找树、二叉搜索树、BST）

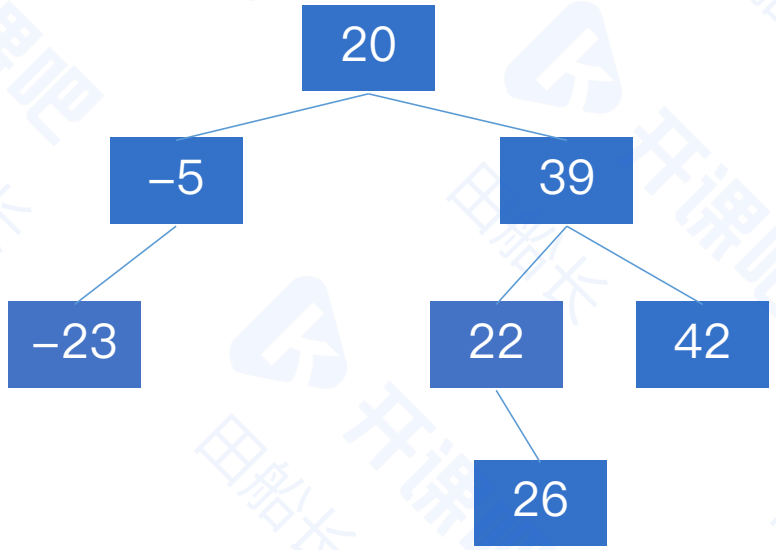
删除元素30




该元素只有右子树
用其右子树将其替代进行删除

# 二叉排序树（二叉查找树、二叉搜索树、BST）

删除元素30



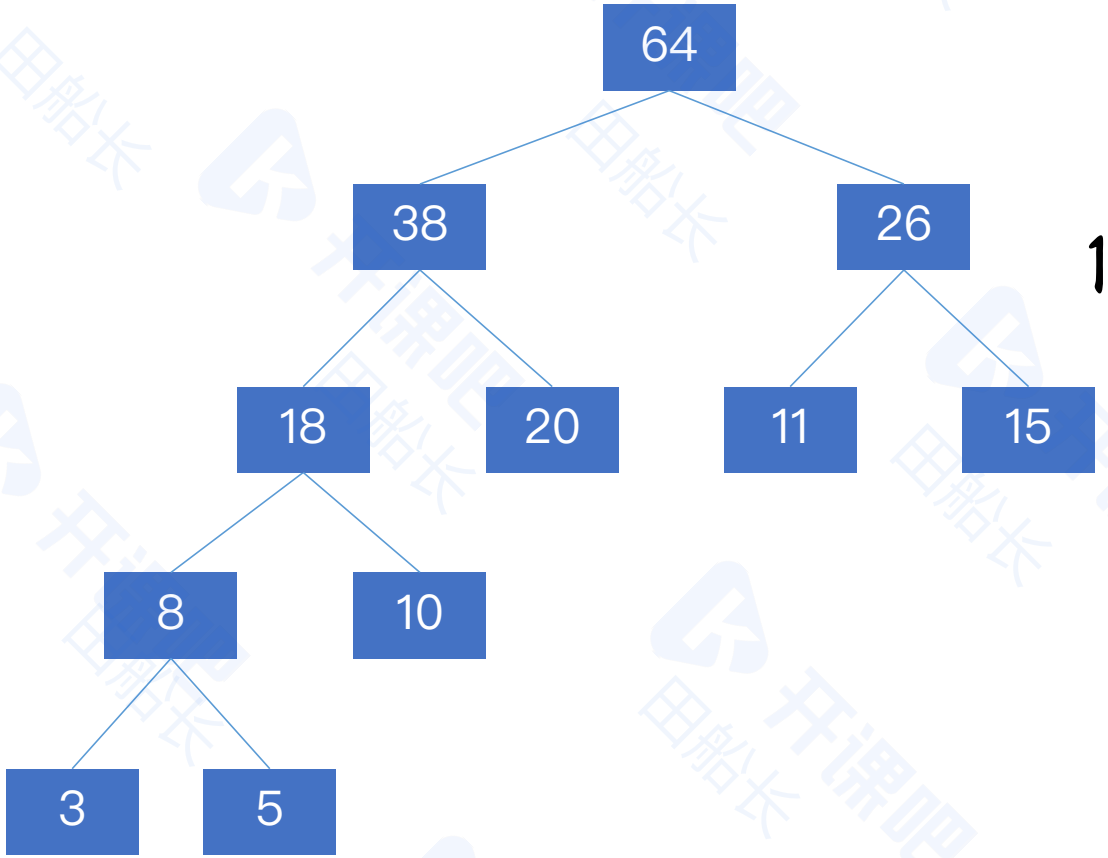

删除完毕

# 哈夫曼树


给定N个权值作为N个叶子结点，构造一棵二叉树
带权路径长度最小的二叉树称为哈夫曼树

构造方法为，每次选出两个权值最小的结点进行合并
将合并后的点放回后，重复上述过程
直到最终剩余一个结点作为树根

# 哈夫曼树




带权路径长度为：

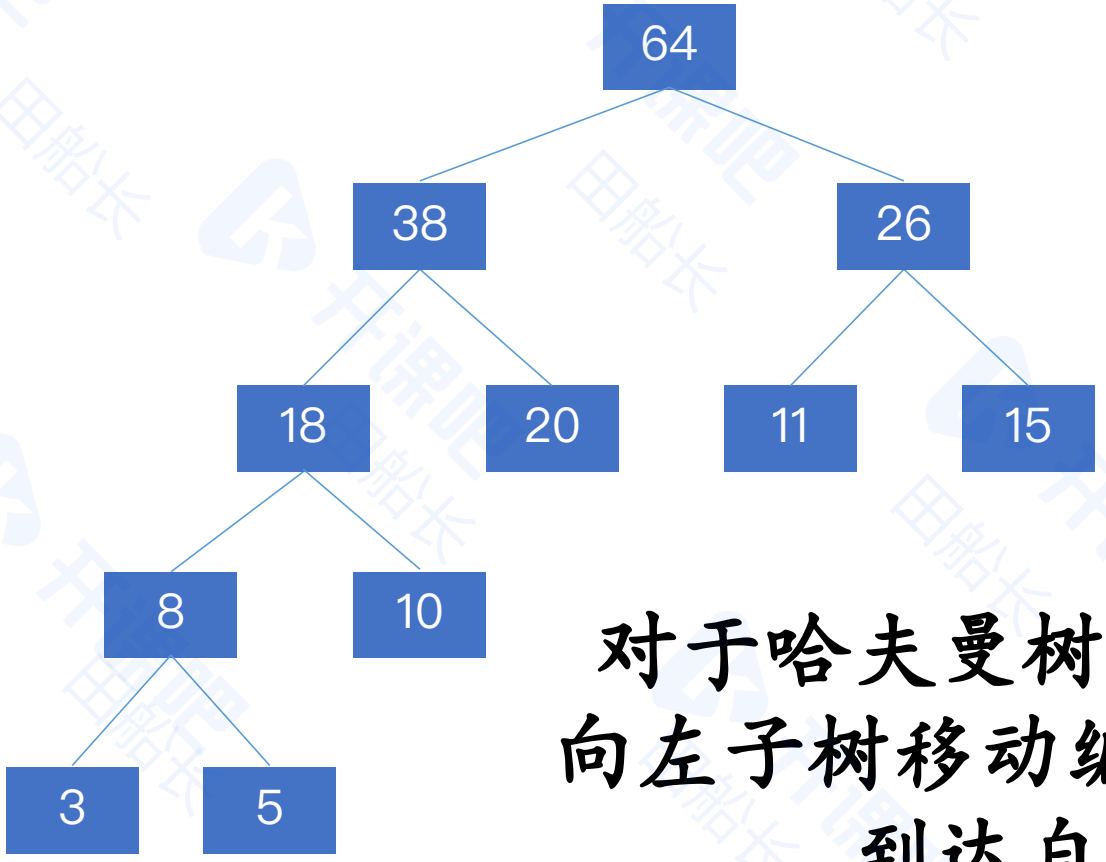
15*2+11*2+20*2+10*3+5*4+3*4

或

64+38+26+18+8

上页结点所对应的一棵哈夫曼树

# 哈夫曼编码




15：11
11：10
20：01
10：001
5 ：0001
3 ：0000

对于哈夫曼树中的叶子结点，从根节点出发
向左子树移动编码为0，向右子树移动编码为1
到达自身的所有路径连接起来
即为该叶子结点的编码