

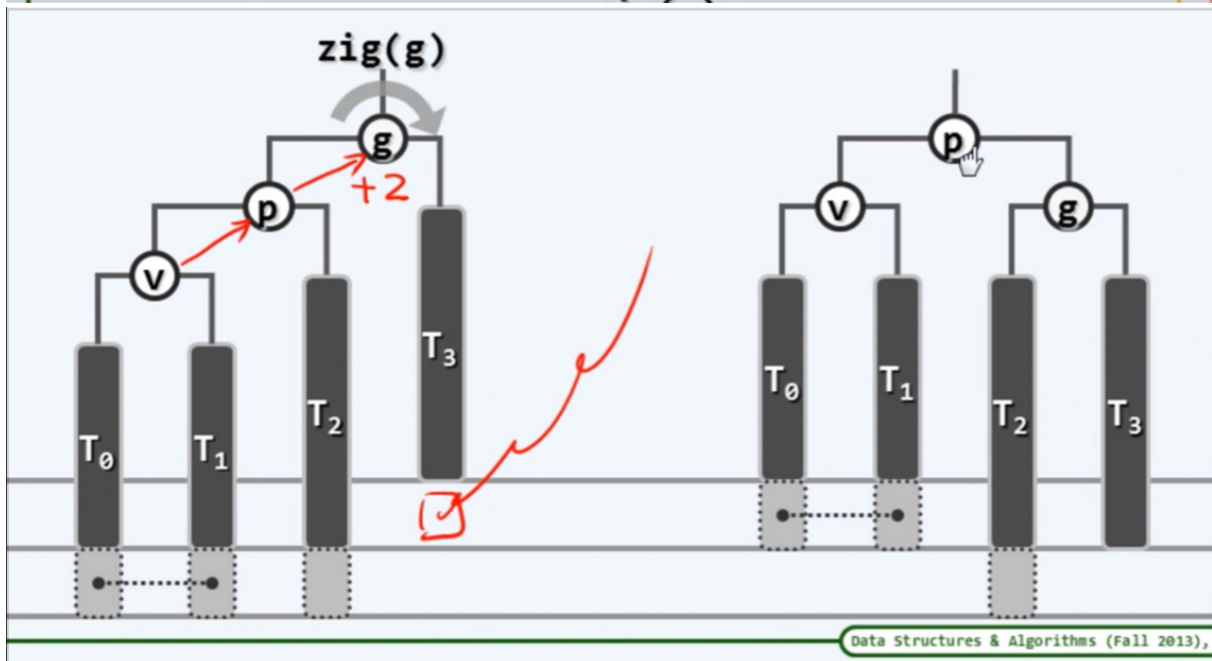
08D4 删除

#数据结构邓神

单旋

删除：单旋

- ❖ 同时至多一个失衡节点 g ，首个可能就是 x 的父亲_hot
- ❖ g 经单旋调整后复衡，子树高度未必复原；更高祖先仍可能失衡
- ❖ 因有失衡传播现象，可能需做 $O(\log n)$ 次调整

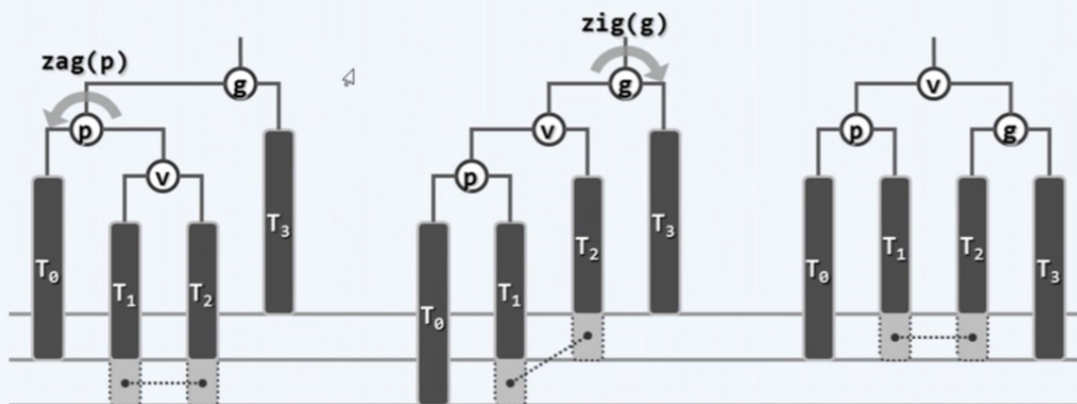


双旋转

Zigzag

删除：双旋

❖ 同时至多一个失衡节点 g ，首个可能就是 x 的父亲 $_{hot}$



旋转后高度减少了1

实现

删除：实现

```
❖ template <typename T> bool AVL<T>::remove( const T & e ) {
    BinNodePosi(T) & x = search( e ); if ( !x ) return false; //若目标的确存在
    removeAt( x, _hot ); _size--; //则在按BST规则删除之后，_hot及祖先均有可能失衡
    // 以下，从_hot出发逐层向上，依次检查各代祖先g
    for ( BinNodePosi(T) g = _hot; g; g = g->parent ) {
        if ( ! AvlBalanced( *g ) ) //一旦发现g失衡，则通过调整恢复平衡
            g = FromParentTo( *g ) = rotateAt( tallerChild( tallerChild( g ) ) );
        updateHeight( g ); //并更新其高度
    } //可能需做过 $\Omega(\log n)$ 次调整；无论是否做过调整，全树高度均可能下降
    return true; //删除成功
}
```

```
template <typename T> bool AVL<T>::remove(const T & e){
    BinNodePosi<T> & x = search(e);
    if (!x) {
        return false;
    }
    removeAt(x,_hot);
    _size--;
    for (BinNodePosi<T> g = _hot; g ; g = g->parent) {
```

```
    if(!AvLBalanced(*g)) {  
        g = FromParentTo(*g) = rotateAt(tallerChild(tallerChild(g)));  
    }  
    updateHeight(g);  
}  
return true;  
}
```