

二叉搜索树

AVL树：删除

08-D4

邓俊辉

deng@tsinghua.edu.cn

单旋

◆ 同时至多一个失衡节点g ,

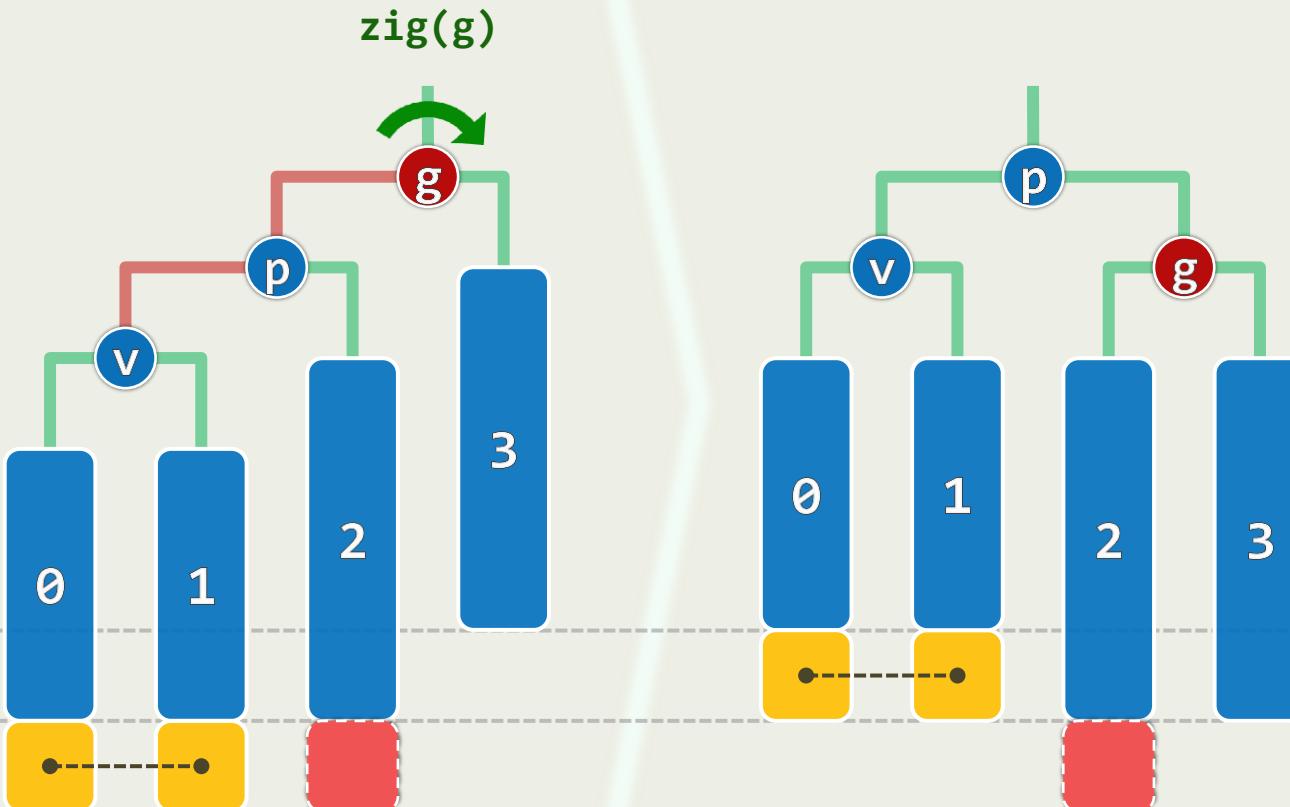
首个可能就是x的父亲_hot

◆ 复衡后子树高度未必复原

更高祖先仍可能随之失衡

◆ 失衡可能持续向上传播

最多需做 $\mathcal{O}(\log n)$ 次调整



双旋

◆ 同时至多一个失衡节点g ,

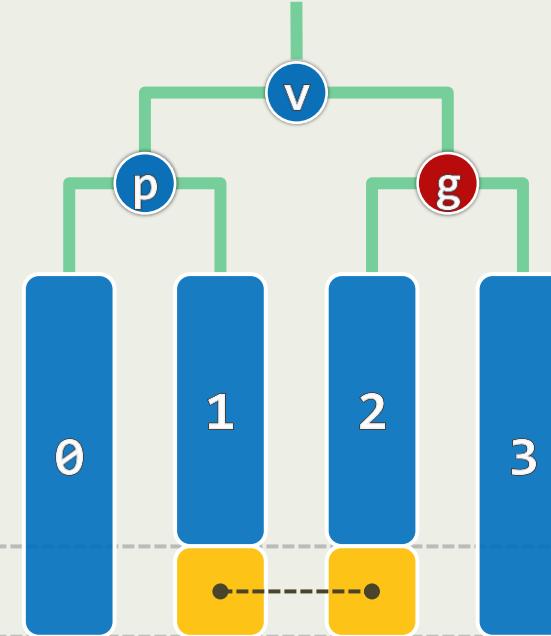
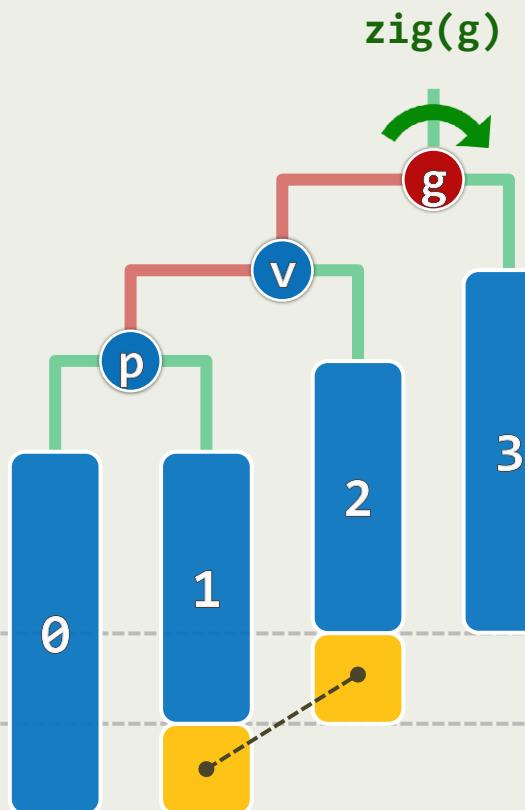
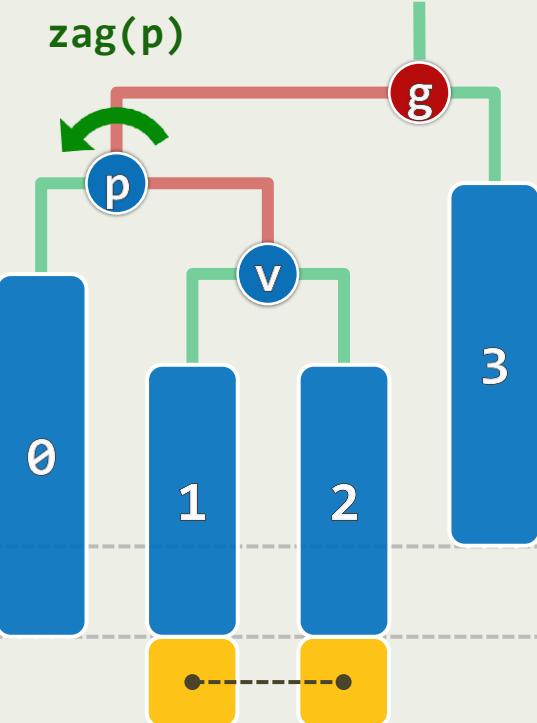
首个可能就是x的父亲_hot

◆ 复衡后子树高度未必复原

更高祖先仍可能随之失衡

◆ 失衡可能持续向上传播

最多需做 $\mathcal{O}(\log n)$ 次调整



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  
}
```