二叉搜索树

AVL树：删除

06-D4

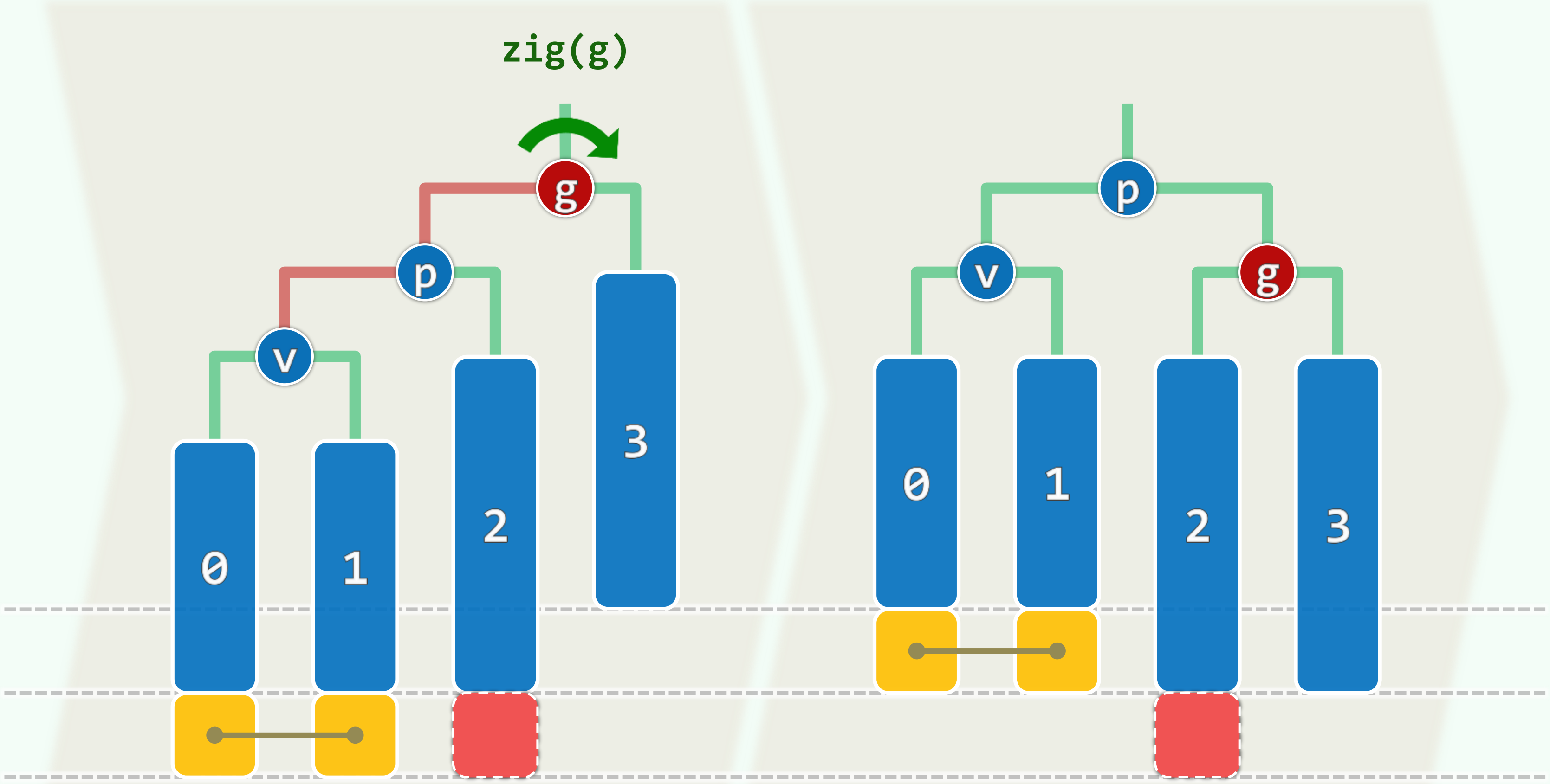
邓俊辉

deng@tsinghua.edu.cn

没有什么是一次旋转解决不了的；如果有，那就两次

单旋：黄色节点至少存在其一；红色节点可有可无

- 瞬时至多一个失衡节点g，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$O(\log n)$次调整
- 逐层上溯，便可找到g
- 确定名分：
  - p = tallerChild(g)
  - v = tallerChild(p)
- 无论p和v的方向是否一致，均可从容处理...



双旋

❖ 瞬时**至多一个**失衡节点g，

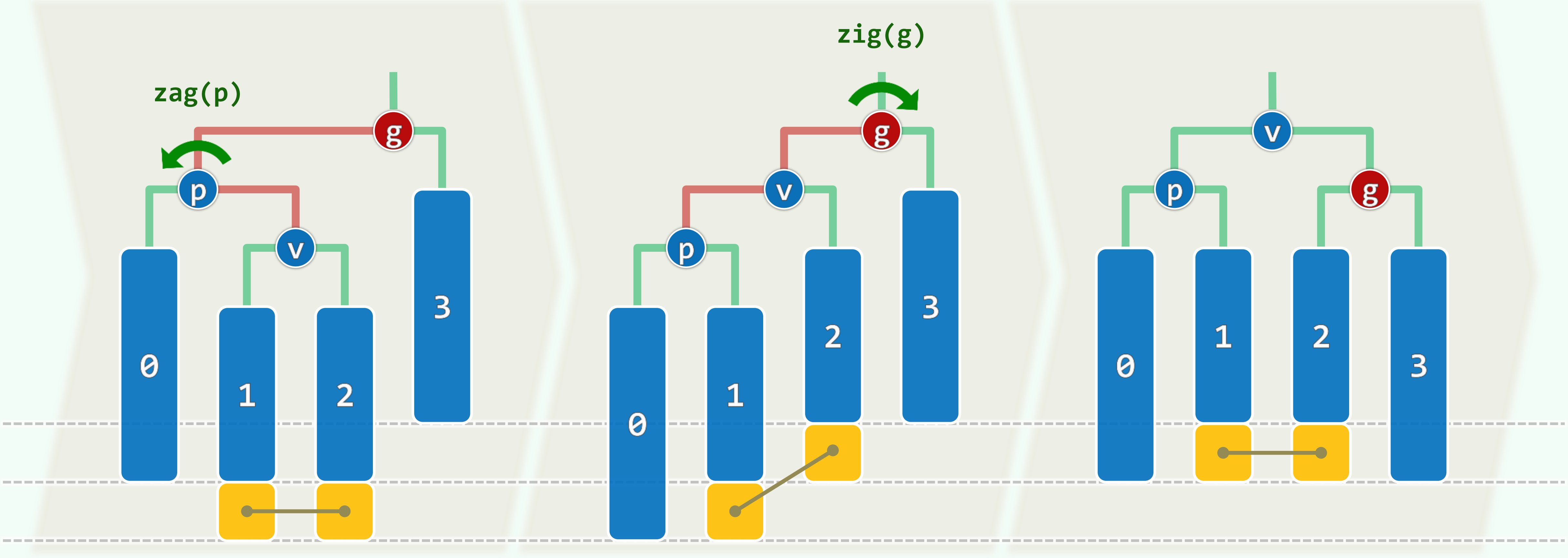
可能就是x的父亲_hot

❖ 复衡后子树高度**不能**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; //删除失败
   removeAt( x, _hot ); _size--; //则在按BST规则删除之后，_hot及祖先均有可能失衡
   for ( BinNodePosi<T> g = _hot; g; g->updateHeight(), g = g->parent ) //逐层上溯
      if ( ! AvlBalanced( g ) ) //每当发现失衡祖先g，都
         rotateAt( tallerChild( tallerChild( g ) ) ); //通过调整恢复平衡
   return true; //删除成功
} //可能需做过Ω(logn)次调整
```