

08-B3

二叉搜索树

算法及实现：删除

邓俊辉

deng@tsinghua.edu.cn

主算法

❖ `template <typename T> bool BST<T>::remove(const T & e) {`

`BinNodePosi<T> & x = search(e); //定位目标节点`

`if (!x) return false; //确认目标存在 (此时_hot为x的父亲)`

`removeAt(x, _hot); //分两大类情况实施删除 , 更新全树规模`

`_size--; //更新全树规模`

`updateHeightAbove(_hot); //更新_hot及其历代祖先的高度`

`return true;`

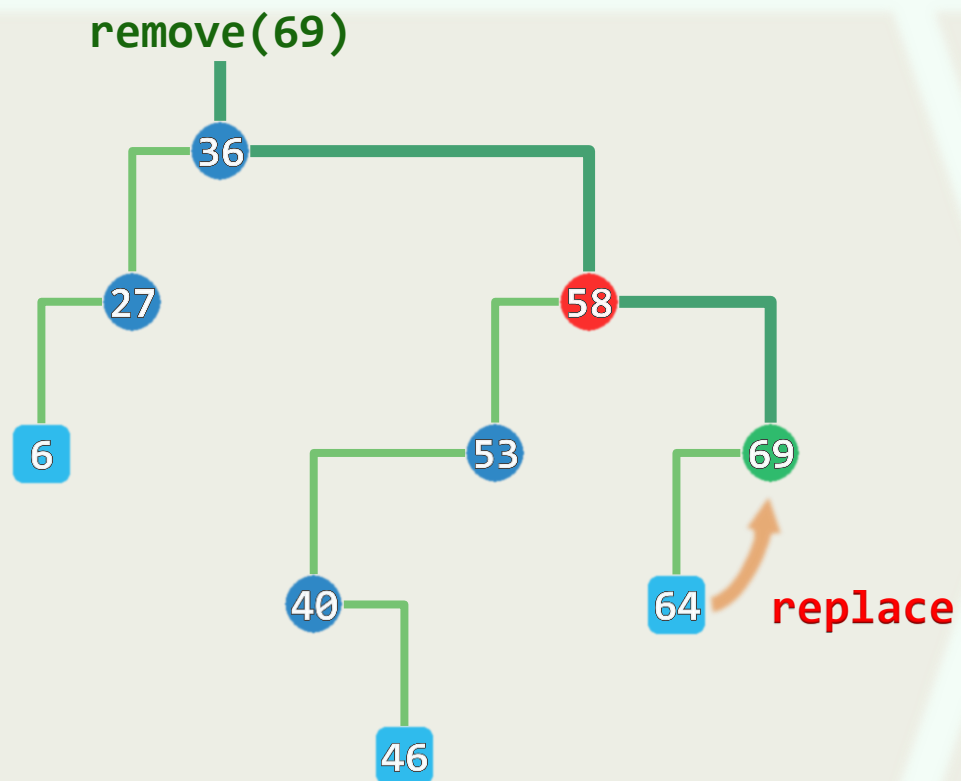
`} //删除成功与否 , 由返回值指示`

❖ 累计 $O(h)$ 时间 : `search()`、`updateHeightAbove()` ; 还有`removeAt()`中可能调用的`succ()`

单分支：实例

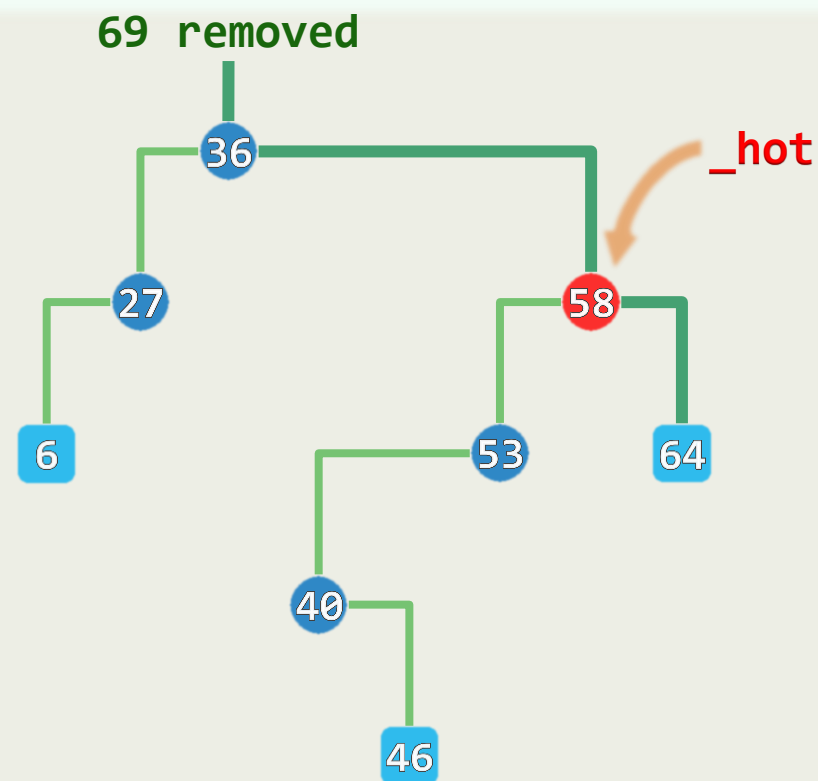
❖ 若 *x (69) 的某一子树为空，则可

将其替换为另一子树 (64) //可能亦为空



❖ 验证：如此操作之后，二叉搜索树的

拓扑结构依然完整；顺序性依然满足



单分支：实现

```
❖ template <typename T> static BinNodePosi<T>
removeAt( BinNodePosi<T> & x, BinNodePosi<T> & hot ) {
    BinNodePosi<T> w = x; //实际被摘除的节点，初值同x
    BinNodePosi<T> succ = NULL; //实际被删除节点的接替者
    if      ( ! HasLChild( *x ) ) succ = x = x->rChild; //左子树为空
    else if ( ! HasRChild( *x ) ) succ = x = x->lChild; //右子树为空
    else { /* ...左、右子树并存的情况，略微复杂些... */ }

    hot = w->parent; //记录实际被删除节点的父亲
    if ( succ ) succ->parent = hot; //将被删除节点的接替者与hot相联
    release( w->data ); release( w ); return succ; //释放被摘除节点，返回接替者
} //此类情况仅需 $O(1)$ 时间
```

双分支：实例

❖ 若： $*x$ (36) 左、右孩子并存

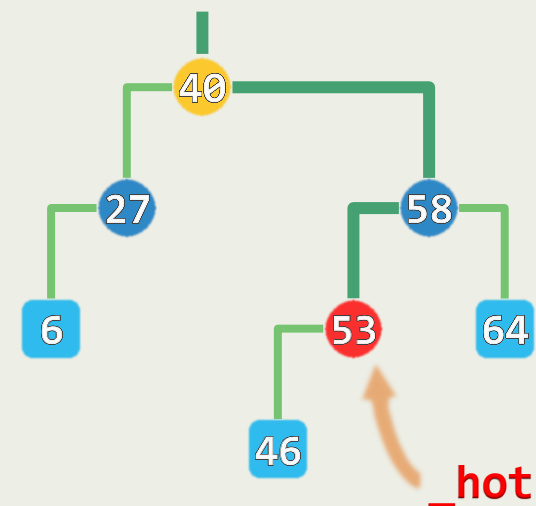
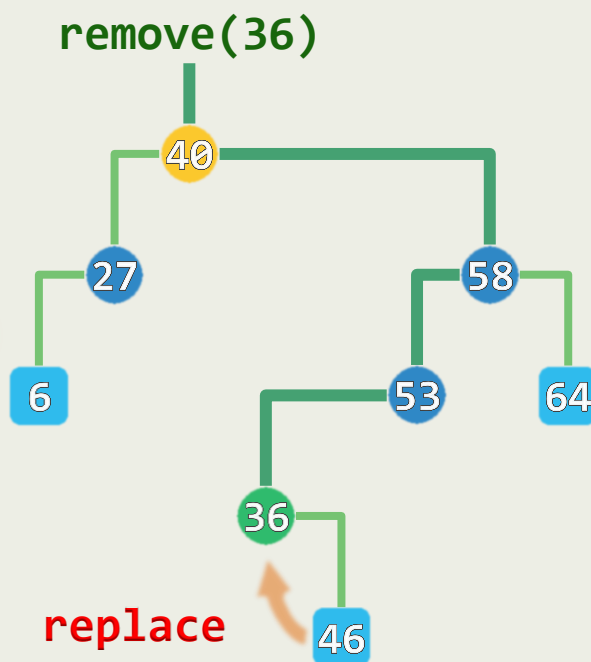
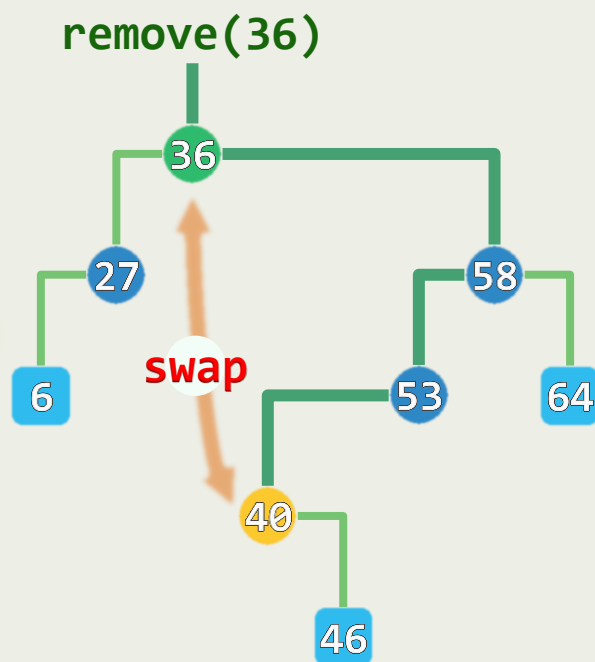
❖ 于是问题转化为删除 w ，可按前一情况处理

则：调用`BinNode::succ()`找到 x 的直接后继

❖ 尽管顺序性在中途曾一度不合

(必无左孩子)；交换 $*x$ (36) 与 $*w$ (40)

但最终必将重新恢复



双分支：实现

❖ `template <typename T> static BinNodePosi<T>`

`removeAt(BinNodePosi<T> & x, BinNodePosi<T> & hot) {`

`/* */`

`else { //若x的左、右子树并存，则`

`w = w->succ(); swap(x->data, w->data); //令*x与其后继*w互换数据`

`BinNodePosi<T> u = w->parent; //原问题即转化为，摘除非二度的节点w`

`(u == x ? u->rc : u->lc) = succ = w->rc; //兼顾特殊情况：u可能就是x`

`}`

`/* */`

`} //时间主要消耗于succ()，正比于x的高度——更精确地，search()与succ()总共不过 $O(h)$`