二叉搜索树

AVL树：失衡与复衡

06-02

邓俊辉

deng@tsinghua.edu.cn

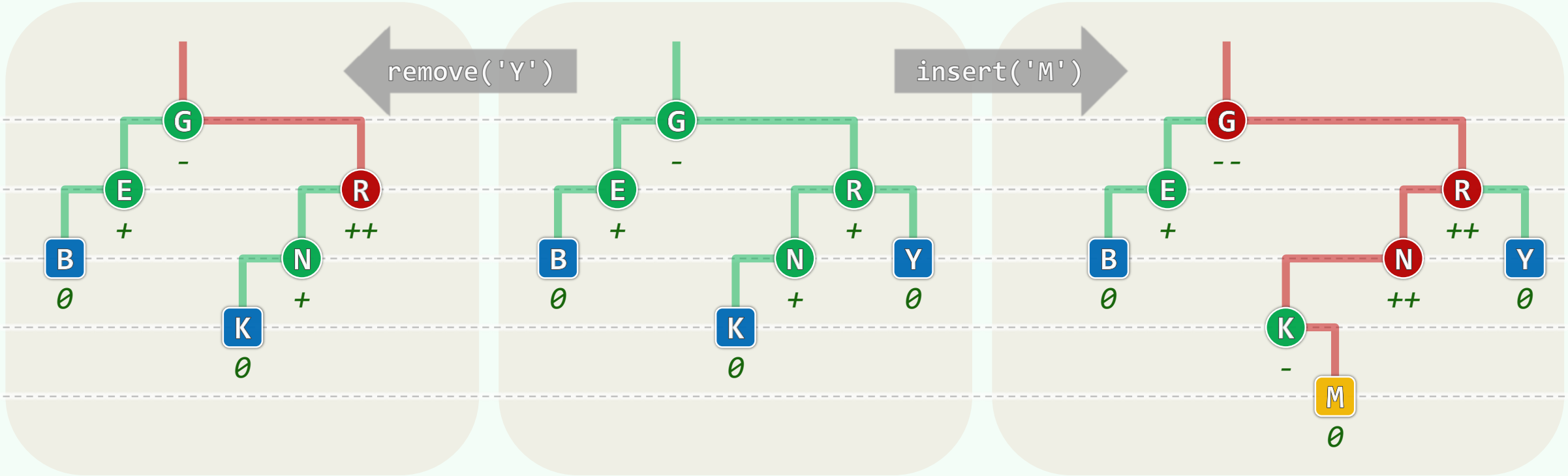
不取于相，如如不动

接口

```
#define Balanced(x)  ( stature( (x)->lc ) == stature( (x)->rc ) ) //理想平衡

#define BalFac(x)  ( stature( (x)->lc ) - stature( (x)->rc ) ) //平衡因子

#define AvlBalanced(x)  ( ( -2 < BalFac(x) ) && ( BalFac(x) < 2 ) ) //AVL平衡条件


template <typename T> class AVL : public BST<T> { //由BST派生

public: //BST::search()等接口，可直接沿用

   BinNodePosi<T> insert( const T & ); //插入（重写）

   bool remove( const T & ); //删除（重写）

};
```

失衡

❖ 按BST规则动态操作之后，AVL平衡性可能破坏 //当然，只涉及到祖先

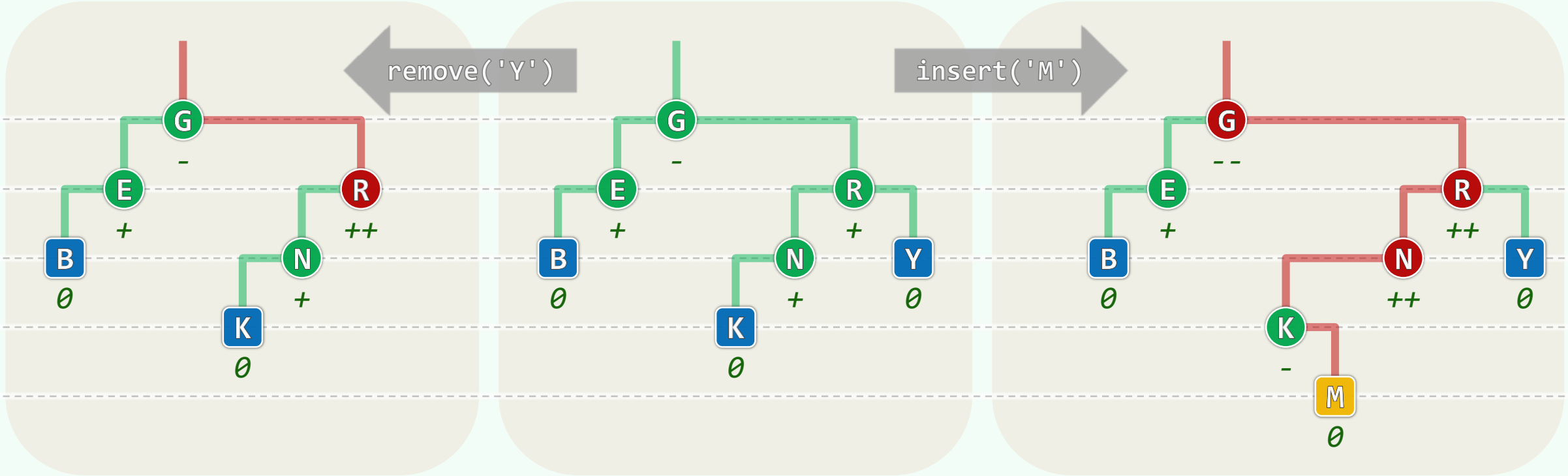


❖ 插入：从祖父开始，每个祖先都有可能失衡，且可能同时失衡！ //复杂?

❖ 删除：从父亲开始，每个祖先都有可能失衡，但至多一个！为什么? //简单?

重平衡

❖ 如何恢复平衡？蛮力不足取，须借助**等价变换**！



❖ 局部性：所有的旋转都在**局部**进行 //每次只需$O(1)$时间

快速性：在每一**深度**只需检查并旋转至多一次 //共$O(\log n)$次