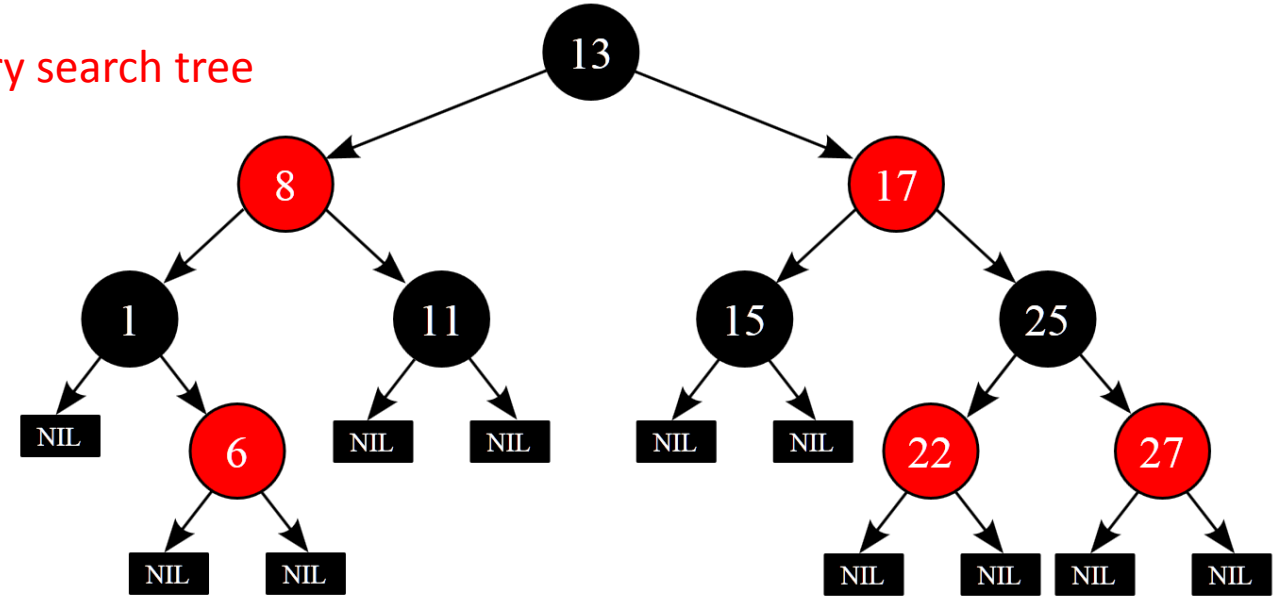
Red-Black tree for sorting(红黑树)

a type of binary search tree



Self-balancing tree algorithm: AVL and Red-Black tree; The key operation: rotating the node.

5性质

1）每个结点要么是红的，要么是黑的。

2）根结点是黑的。

3）每个叶结点，即空结点（NIL）是黑的。

4）如果一个结点是红的，那么它的俩个儿子都是黑的。

5）对每个结点，从该结点到其子孙结点的所有路径上包含相同数目的黑结点。 (此处必须对黑点数目进行约束，why？)

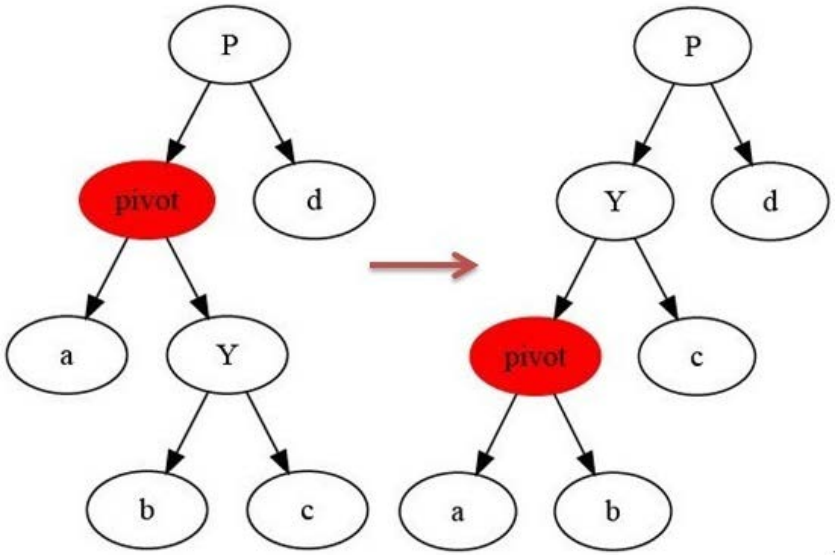
从根出发到任意叶子的路径中，最长的不会超过最短的两倍长

RBtree接近平衡

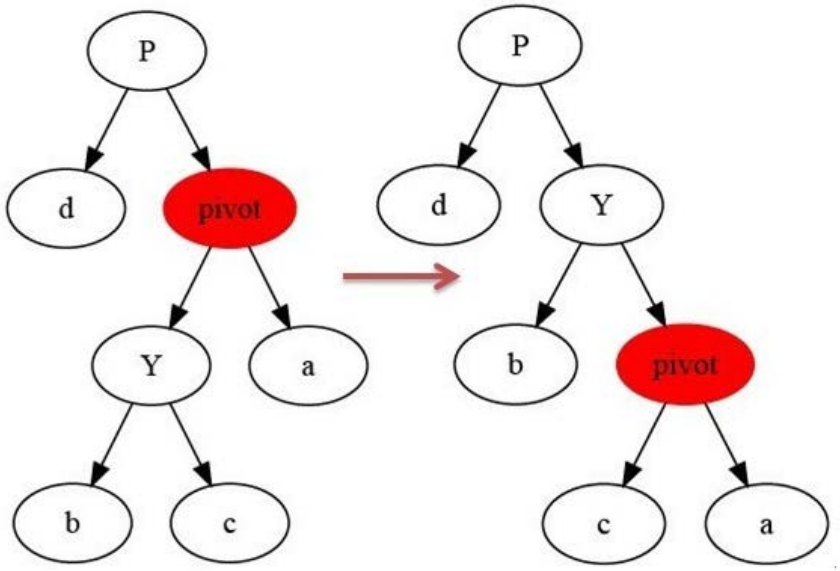
AVLtree更严格平衡 → 插入删除较慢，搜索非常快，适合做字典(建立一次就无需修改的结构)

rotation

左旋



右旋



仅考虑插入红点,以维护p5

insertion

- In RB tree, [in place of an existing black leaf],we insert red interior node, which must have 2 black leaves.

1）每个结点要么是红的，要么是黑的。

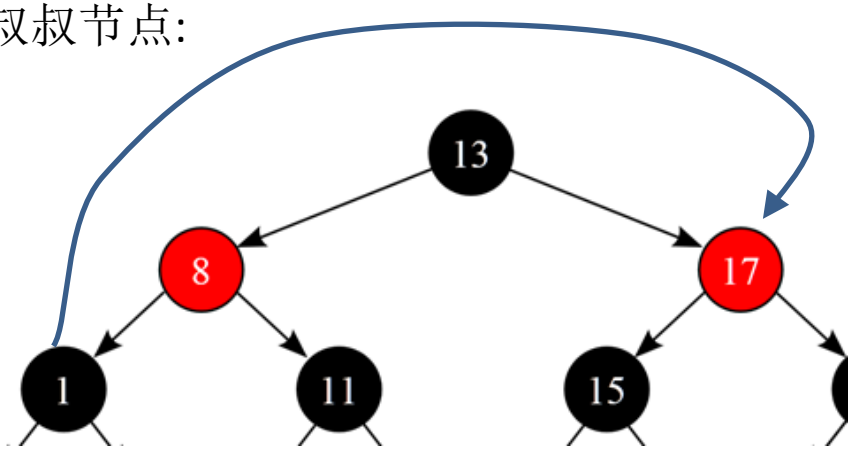
2）根结点是黑的。

3）每个叶结点，即空结点（NIL）是黑的。

4）如果一个结点是红的，那么它的俩个儿子都是黑的。

5）对每个结点，从该结点到其子孙结点的所有路径上包含相同数目的黑结点。

叔叔节点:



三种情况

- Case 1根节点
- Case 2黑爸
- 红爸☹:

假设插入节点在根的左侧，
右侧可镜像对称

--(红叔)

Case 3

--(黑叔)

Case 4, 插入的是左节点

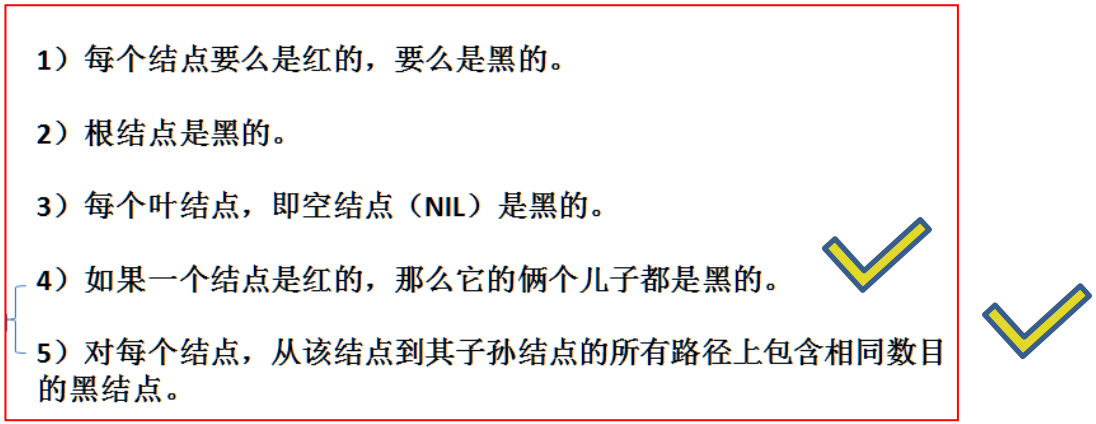
Case 5, 插入的是右节点

根节点

- 不违反任何一条性质，
- 仅需染黑即可

黑爸

- Still valid
- Property4 not invalidated.
- Property5 holds.



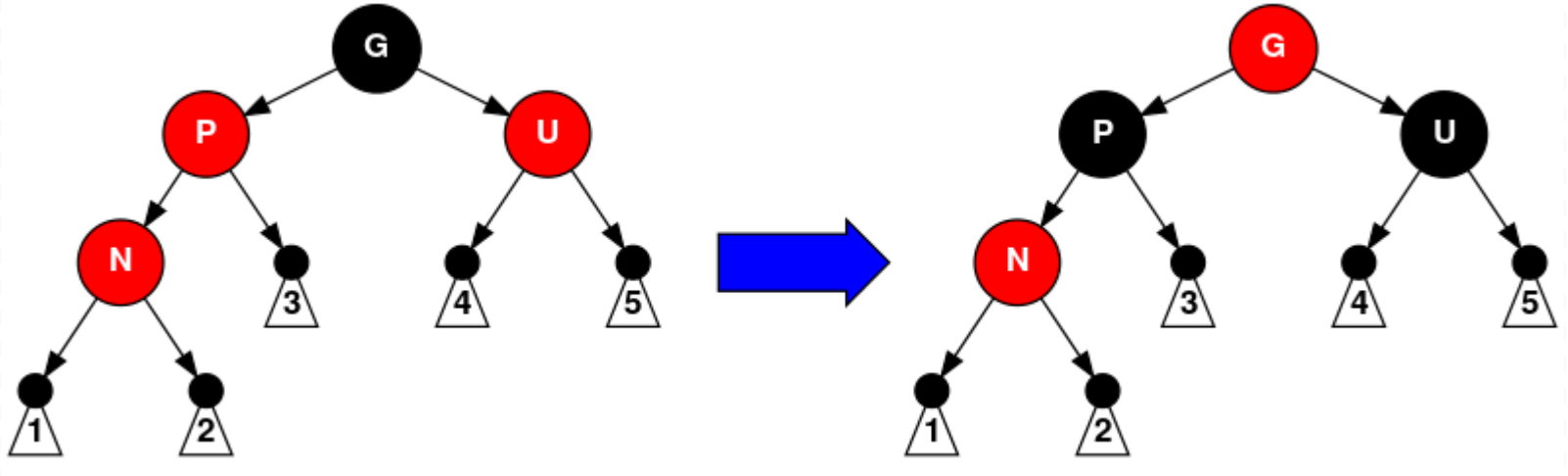
红爸，红叔

- Prop4 is violated!

策略:

爸叔换色，祖父也换色.

若发现P2,P4被违反将当前节点设为祖父，递归执行该策略.



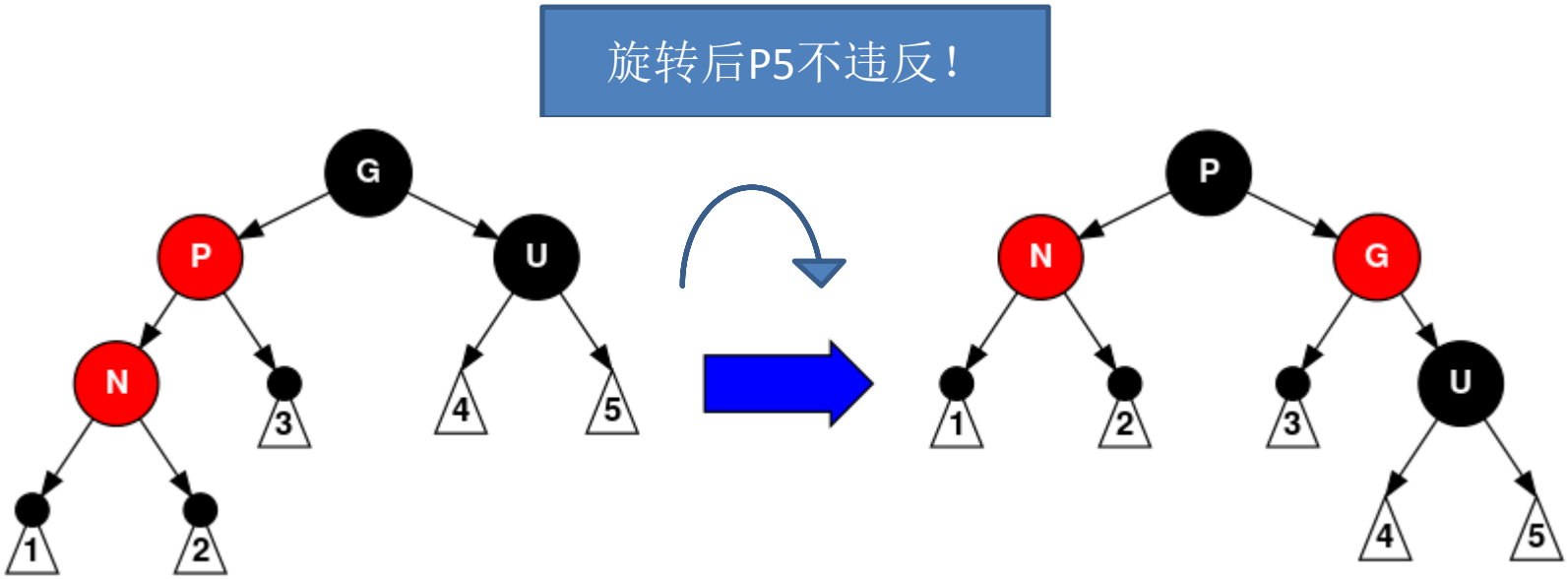
红爸,黑叔,插入在左

- Prop4 is violated!

策略: Need to do right rotation.

爸叔换色，祖父也换色.

若发现P2,P4被违反将当前节点设为祖父，递归执行该策略.



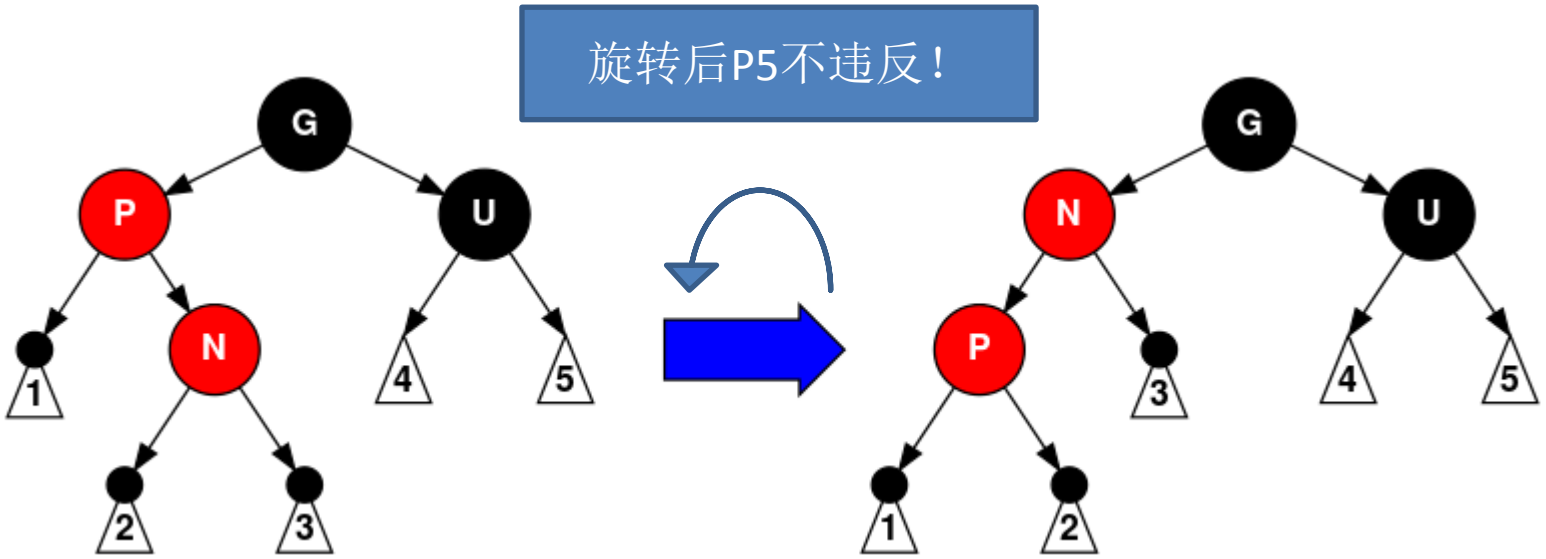
红爸,黑叔,插入在右

- Prop4 is violated!

策略: Need to do left rotation.

无需换色!

若发现P2,P4被违反将当前节点设为祖父，递归执行该策略.



红爸黑叔