8.高级搜索树

（xa4）红黑树：删除

变白以为黑兮，倒上以为下

邓俊辉

deng@tsinghua.edu.cn

算法

- 首先按照BST常规算法，执行：

  r = removeAt(x, _hot)

- x由孩子r接替

  //另一孩子记作w（即黑的NULL）

- 条件1和2依然满足

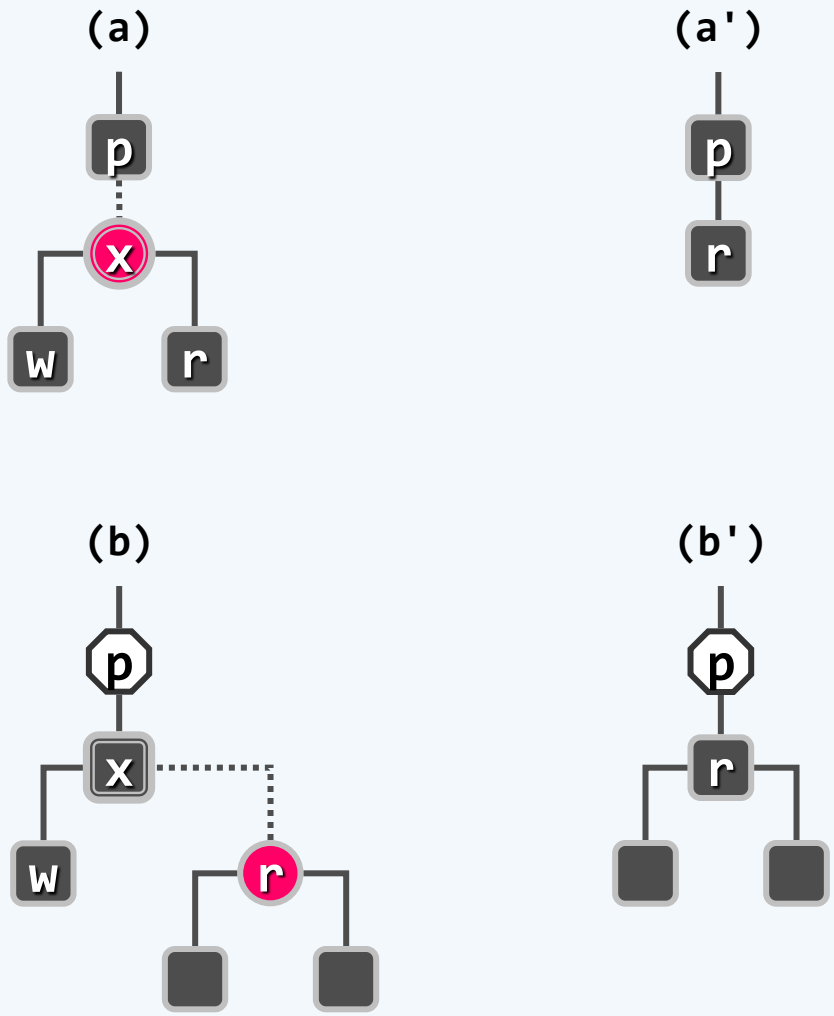
  但3和4不见得

  //在原树中，考查x与r...

- 若二者之一为红

  则3和4不难满足

  //删除遂告完成！



算法

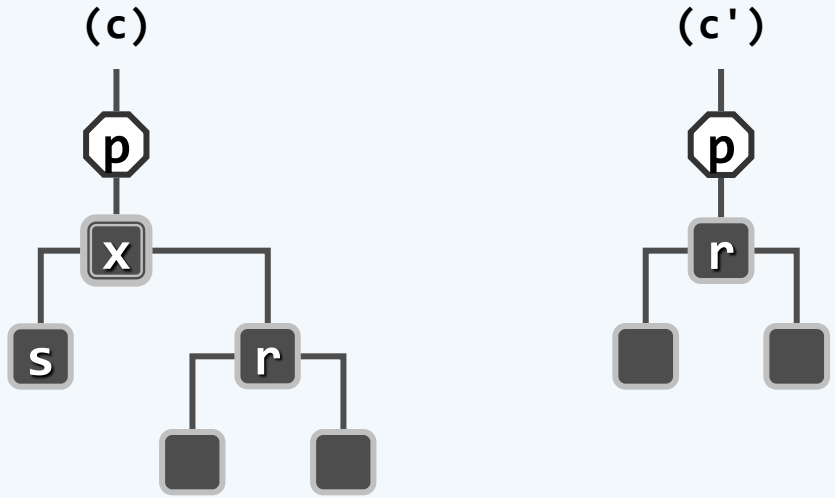
❖ 若x与r均黑double-black

则不然...

❖ 摘除x并代之以r后

全树黑深度不再统一

原B-树中x所属节点下溢



❖ 在新树中，考查

r的父亲 `p = r->parent` //亦即原树中x的父亲

r的兄弟 `s = r == p->lc ? p->rc : p->lc`

❖ 以下分四种情况处理...

实现

```
❖template <typename T> bool RedBlack<T>::remove( const T & e ) {
    BinNodePosi(T) & x = search( e ); if ( !x ) return false; //查找定位
    BinNodePosi(T) r = removeAt( x, _hot ); //删除_hot的某孩子，r指向其接替者
    if ( ! ( -- _size ) ) return true; //若删除后为空树，可直接返回
    if ( ! _hot ) { //若被删除的是根，则
       _root->color = RB_BLACK; //将其置黑，并
       updateHeight( _root ); //更新（全树）黑高度
       return true;
    } //至此，原x（现r）必非根
```

实现

```
❖ // 若父亲（及祖先）依然平衡，则无需调整
    if ( BlackHeightUpdated( * _hot ) ) return true;
  // 至此，必失衡
  // 若替代节点r为红，则只需简单地翻转其颜色
    if ( IsRed( r ) ) { r->color = RB_BLACK; r->height++; return true; }
  // 至此，r以及被其替代的x均为黑色
    solveDoubleBlack( r ); //双黑调整（入口处必有 r == NULL）
    return true;
  }
```

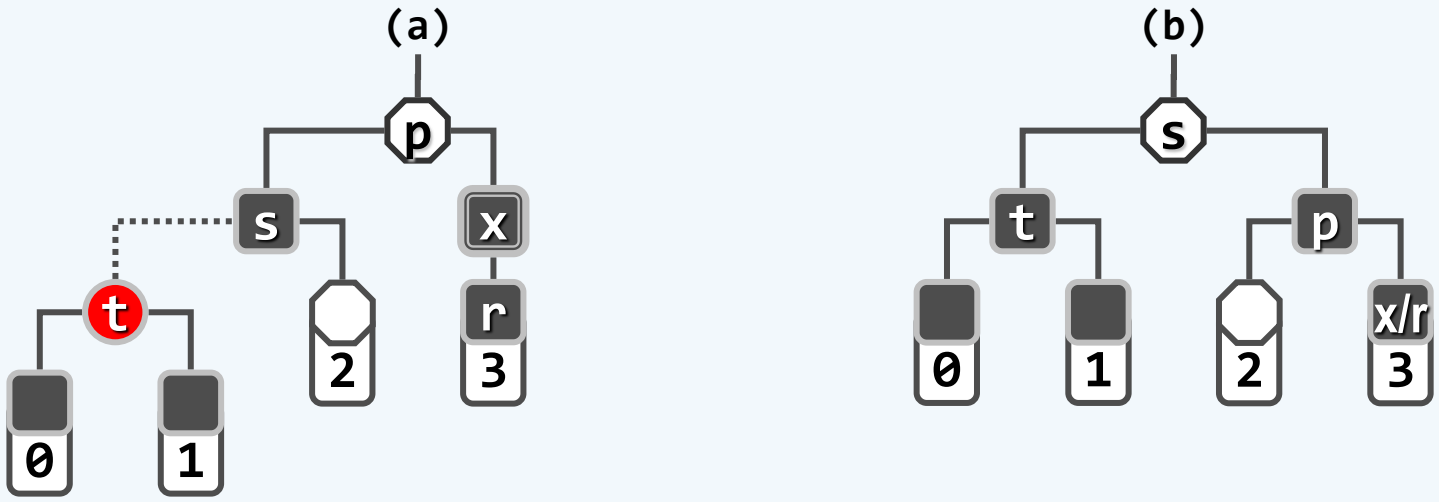
双黑修正

```
❖ template <typename T> void RedBlack<T>::solveDoubleBlack( BinNodePosi(T) r ) {
    BinNodePosi(T) p = r ? r->parent : _hot; if ( !p ) return; //r的父亲
    BinNodePosi(T) s = (r == p->lc) ? p->rc : p->lc; //r的兄弟
    if ( IsBlack( s ) ) { //兄弟s为黑
       BinNodePosi(T) t = NULL; //以下将t取作s的红孩子
       if      ( HasLChild( *s ) && IsRed( s->lc ) ) t = s->lc;
       else if ( HasRChild( *s ) && IsRed( s->rc ) ) t = s->rc;
       if ( t ) { /* ... 黑s有红孩子：BB-1 ... */ }
       else { /* ... 黑s无红孩子：BB-2R或BB-2B ... */ }
    } else { /* ... 兄弟s为红：BB-3 ... */ }
  }
```

BB-1：s为黑，且至少有一个红孩子t

❖ 3+4重构：t、s、p重命名为a、b、c

r保持黑；a和c染黑；b继承p的原色



❖ 如此，红黑树性质在全局得以恢复——删除完成！ //zig-zag等类似

❖ 在对应的B-树中，以上操作等效于...

BB-1：s为黑，且至少有一个红孩子t

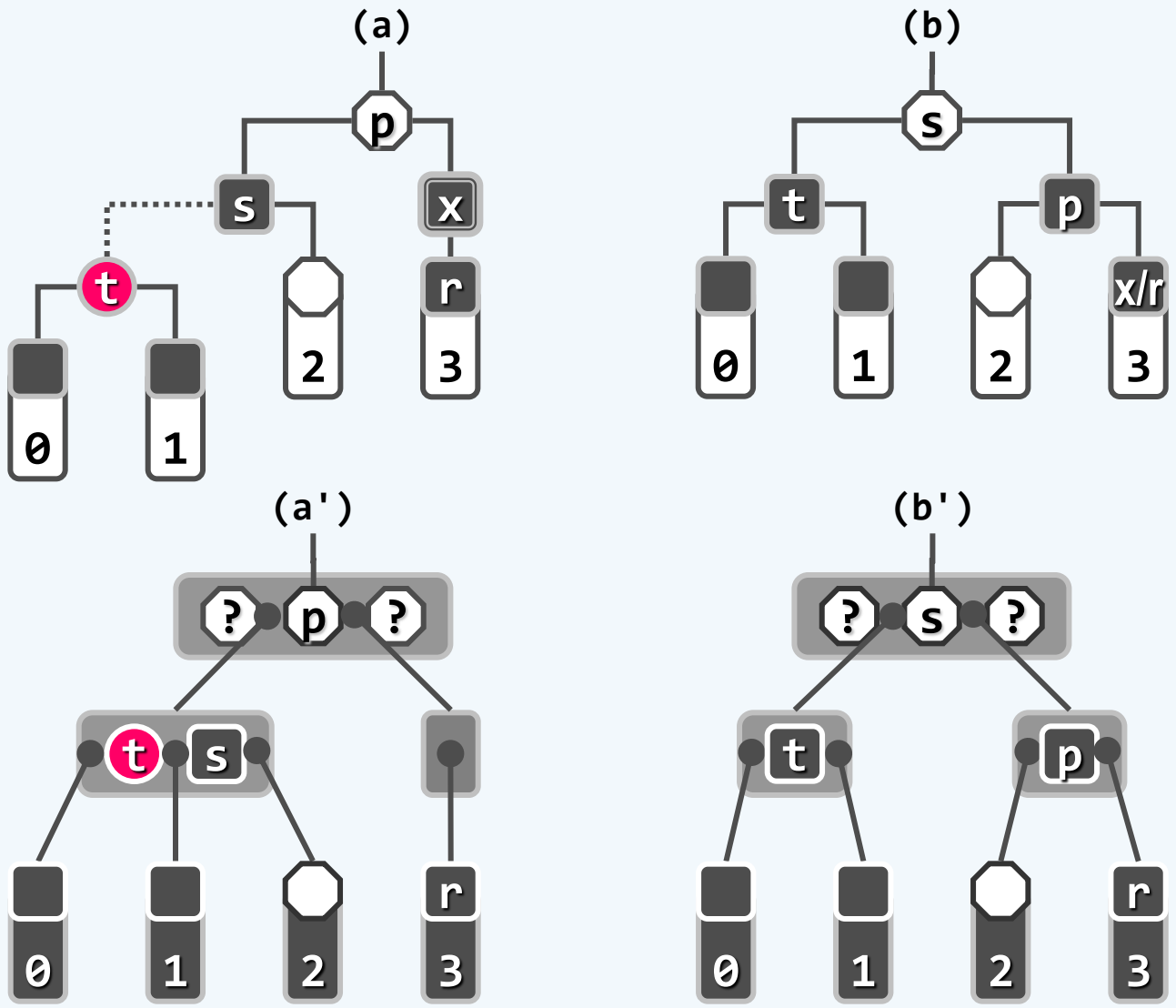
- 通过关键码的旋转

  消除超级节点的下溢

- 问号节点

  可同时存在

  颜色不定

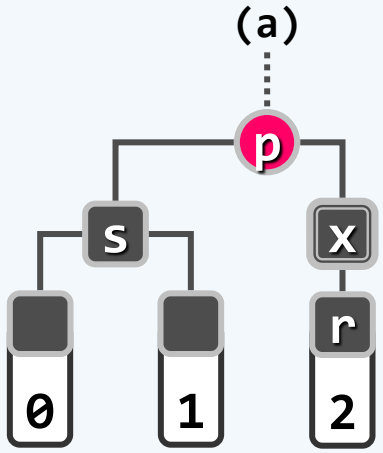


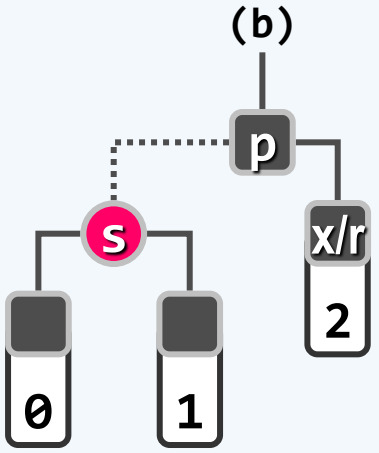
BB-1：实现

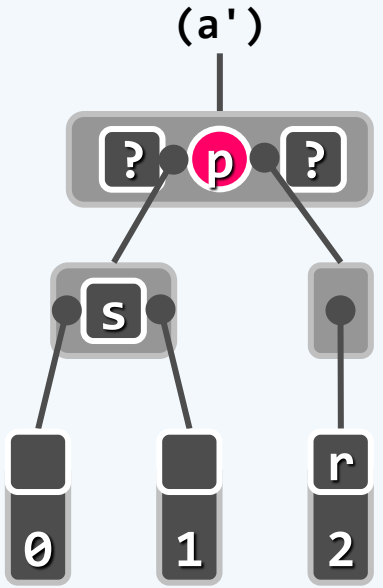
```
❖if ( IsBlack( s ) ) { //兄弟s为黑
    if ( t ) { //黑s有红孩子：BB-1
        RBColor oldColor = p->color; //备份p颜色，并对t、父亲、祖父
        BinNodePosi(T) b = FromParentTo( *p ) = rotateAt( t ); //旋转
        if ( HasLChild( *b ) ) b->lChild->color = RB_BLACK; //新子树之左子染黑
        if ( HasRChild( *b ) ) b->rChild->color = RB_BLACK; //新子树之右子染黑
        updateHeight( b->lc ); updateHeight( b->rc );
        b->color = oldColor; updateHeight( b ); //新根继承原根的颜色
    else { /* ... 黑s无红孩子：BB-2R或BB-2B ... */ }
} else { /* ... 兄弟s为红：BB-3 ... */ }
```

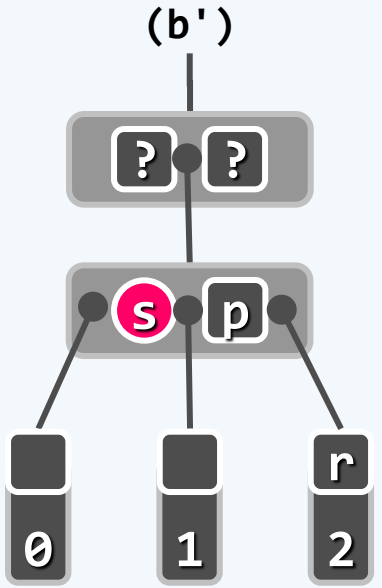
BB-2R：s为黑，且两个孩子均为黑；p为红

- r保持黑；s转红；p转黑
- 在对应的B-树中，等效于
  下溢节点与兄弟合并
- 红黑树性质在全局得以恢复
- 失去关键码p后，上层节点
  会否继而下溢？不会！
- 合并之前，在p之左或右侧
  还应有（问号）关键码
  必为黑色
  有且仅有一个



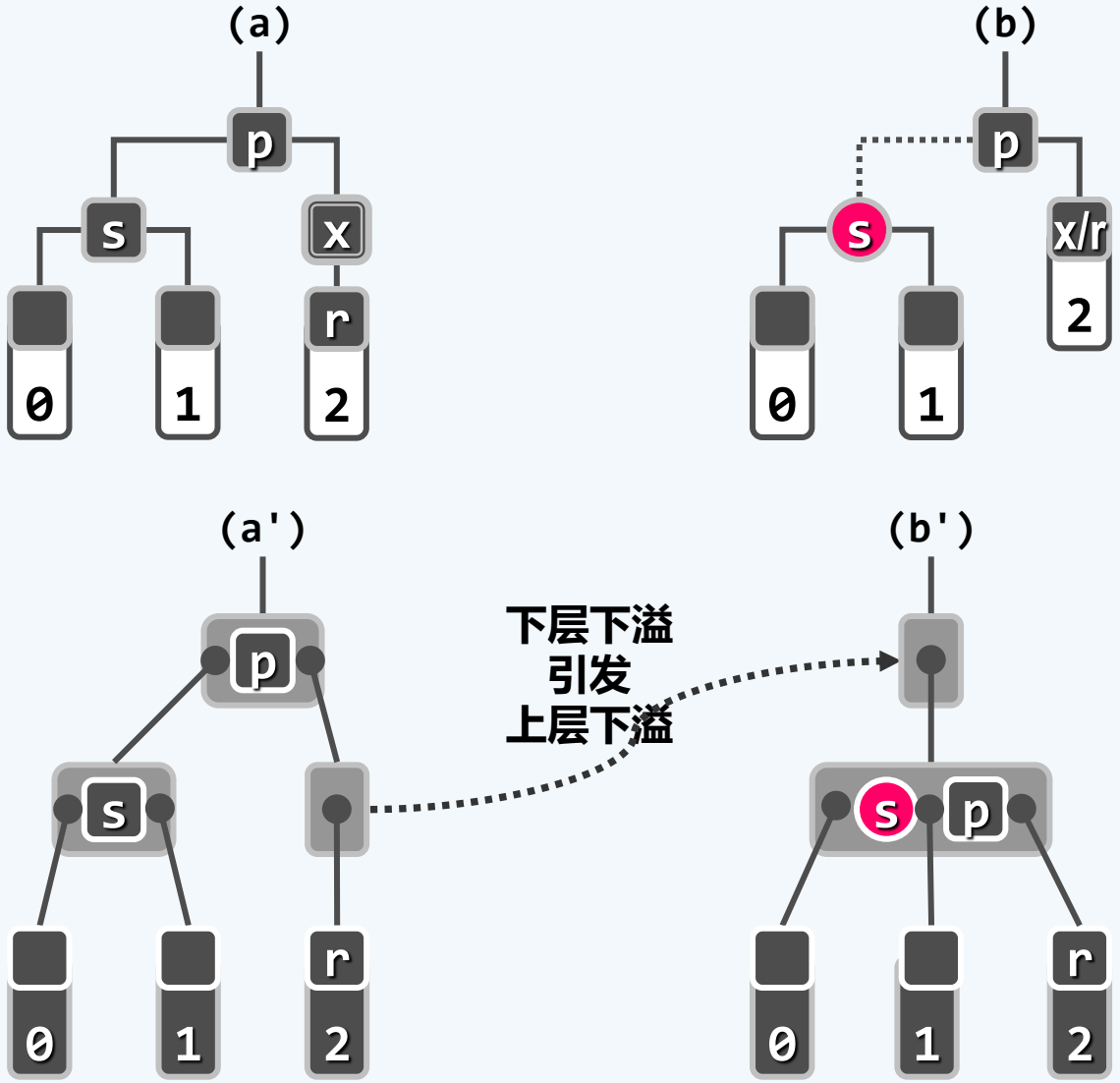






BB-2B：s为黑，且两个孩子均为黑；p为黑

- s转红；r与p保持黑
- 红黑树性质在局部得以恢复
- 在对应的B-树中，等效于
  下溢节点与兄弟合并
- 合并之前，p和s均对应于单关键码节点
- 失去关键码p后
  上层节点必然继而下溢
- 好在可继续分情况处理
  高度递增，至多O(logn)步



BB-(2R+2B)：实现

```
❖if ( IsBlack( s ) ) { //兄弟s为黑
    if ( t ) { /* ... 黑s有红孩子：BB-1 ... */ }
    else { /* 黑s无红孩子 */
       s->color = RB_RED; s->height--; //s转红
       if ( IsRed( p ) ) //BB-2R：p转黑，但黑高度不变
          { p->color = RB_BLACK; }
       else //BB-2B：p保持黑，但黑高度下降；递归修正
          { p->height--; solveDoubleBlack( p ); }
    }
  } else { /* ... 兄弟s为红：BB-3 ... */ }
```

BB-3：s为红（其孩子均为黑）

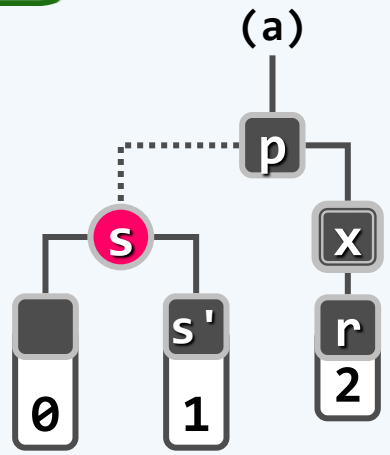
- zag(p)或zig(p)；红s转黑，黑p转红
- 黑高度依然异常，但...
- r有了一个新的黑兄弟s'

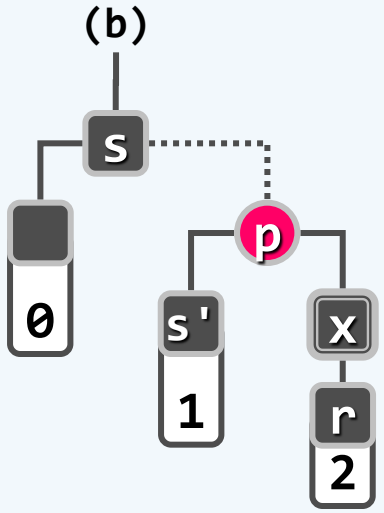
  故转化为前述情况，而且...
- 既然p已转红，接下来

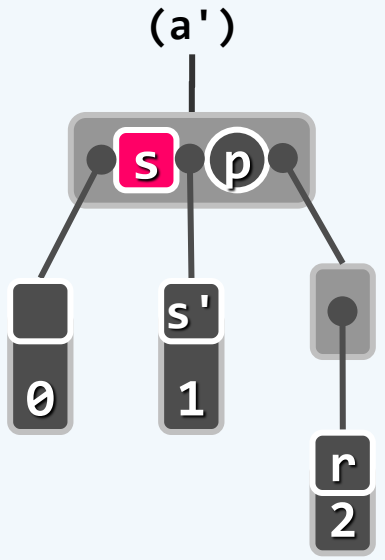
  绝不会是情况BB-2B

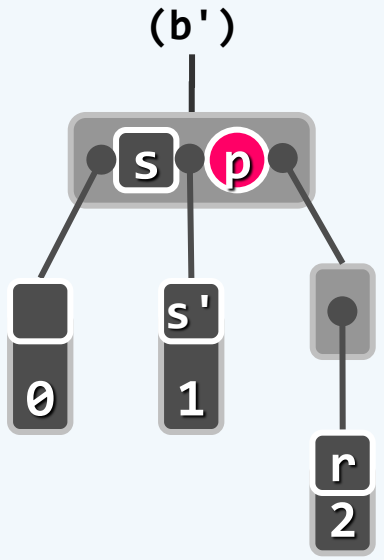
  而只能是BB-1或BB-2R
- 于是，再经一轮调整之后

  红黑树性质必然全局恢复









BB-3：实现

```
❖if ( IsBlack( s ) ) { //兄弟s为黑
    if ( t ) { /* ... 黑s有红孩子：BB-1 ... */ }
    else { /* ... 黑s无红孩子：BB-2R或BB-2B ... */ }
  } else { //兄弟s为红：BB-3
    s->color = RB_BLACK; p->color = RB_RED; //s转黑，p转红
    BinNodePosi(T) t = IsLChild( *s ) ? s->lc : s->rc; //取t与其父s同侧
    _hot = p; FromParentTo( *p ) = rotateAt( t ); //对t及其父亲、祖父做平衡调整
    solveDoubleBlack( r ); //继续修正r——此时p已转红，故后续只能是BB-1或BB-2R
  }
```

复杂度

- 红黑树的每一删除操作

  都可在O(logn)时间内完成

- 其中，至多做

  1. O(logn)次重染色
  2. 一次“3+4”重构
  3. 一次单旋

情况	旋转次数	染色次数	此后
(1) 黑s有红子t	1~2	3	调整随即完成
(2R) 黑s无红子，p红	0	2	调整随即完成
(2B) 黑s无红子，p黑	0	1	必然再次双黑 但将上升一层
(3) 红s	1	2	转为(1)或(2R)