

## 10C2 红黑树：结构

#数据结构邓神

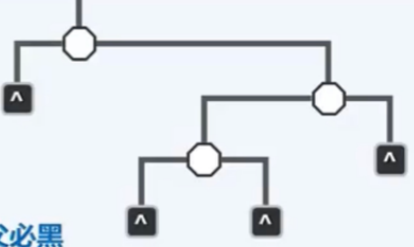
### 定义规则

#### 红与黑

- ❖ 1972, R. Bayer, "symmetric binary B-tree" ✓
- 1978, L. Guibas & R. Sedgwick, "red-black tree" ✓
- 1982, H. Olivie, "half-balanced binary search tree" ✓

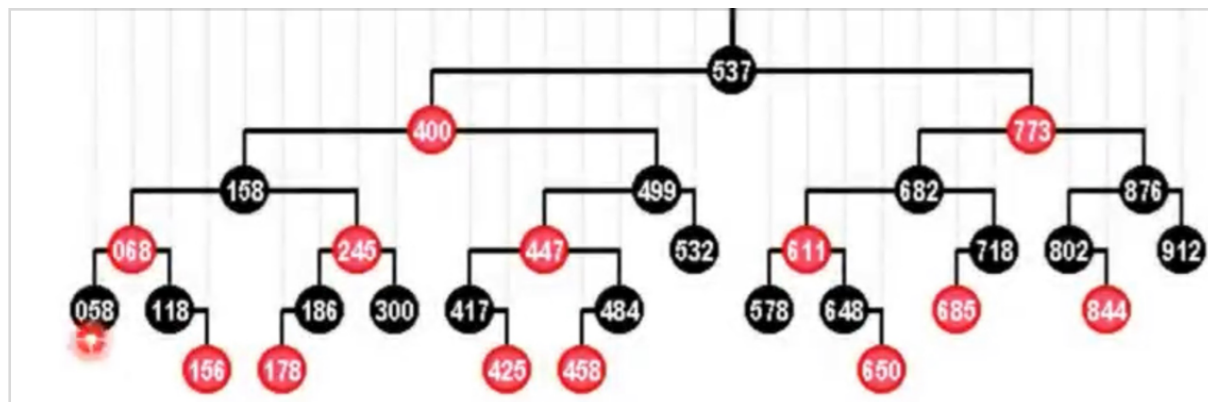
❖ 由红、黑两类节点组成的BST //亦可给边染色  
(统一增设外部节点NULL, 使之成为真二叉树)

- (1) 树根: 必为黑色
- (2) 外部节点: 均为黑色
- (3) 其余节点: 若为红, 则只能有黑孩子 //红之子、之父必黑
- (4) 外部节点到根: 途中黑节点数目相等 //黑深度



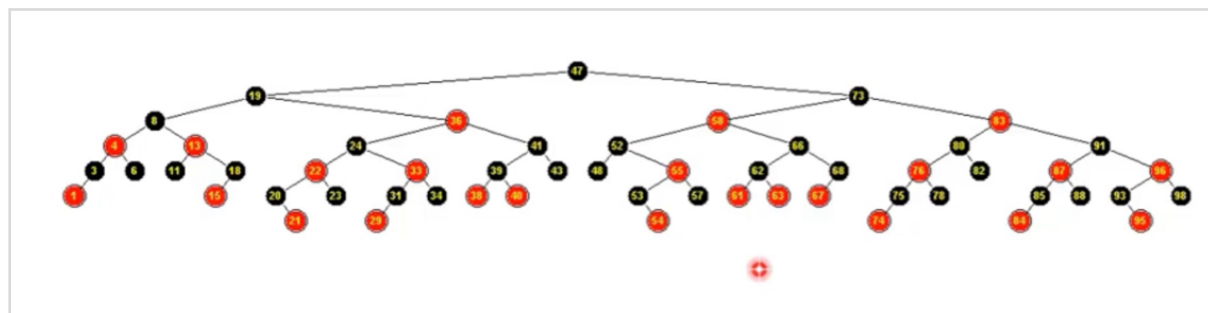
不会出现同时为红的父子两代, 红节点的孩子和父亲都是黑色的

### 实例验证

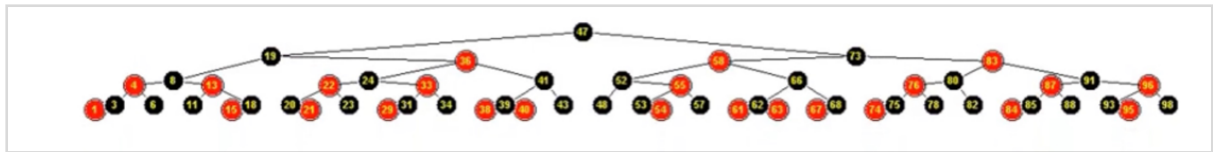


我们发现在这颗树确实符合红黑树的标准

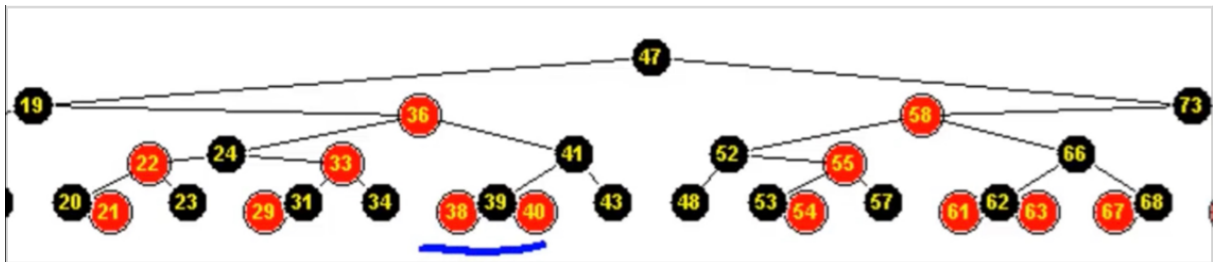
### 提升变换 lift



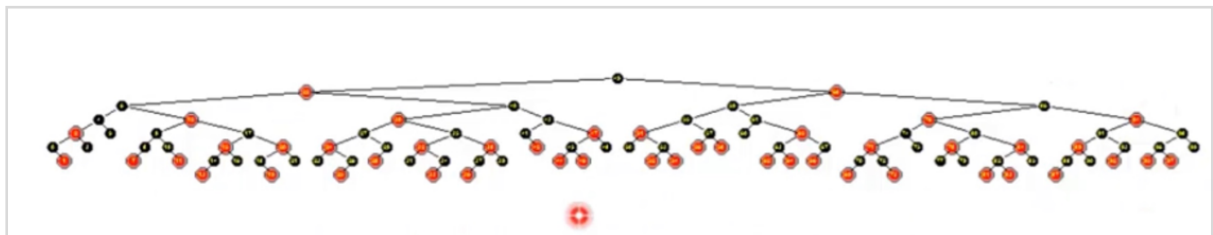
lift后



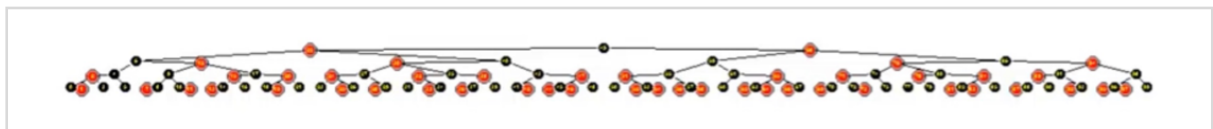
局部



末端节点



实施了底层变换后，变平了??? 变成了BBST?



RPreplay\_Final163921741...  
2021年12月11日

所有底层的节点沿着同一水平高度平齐的分布

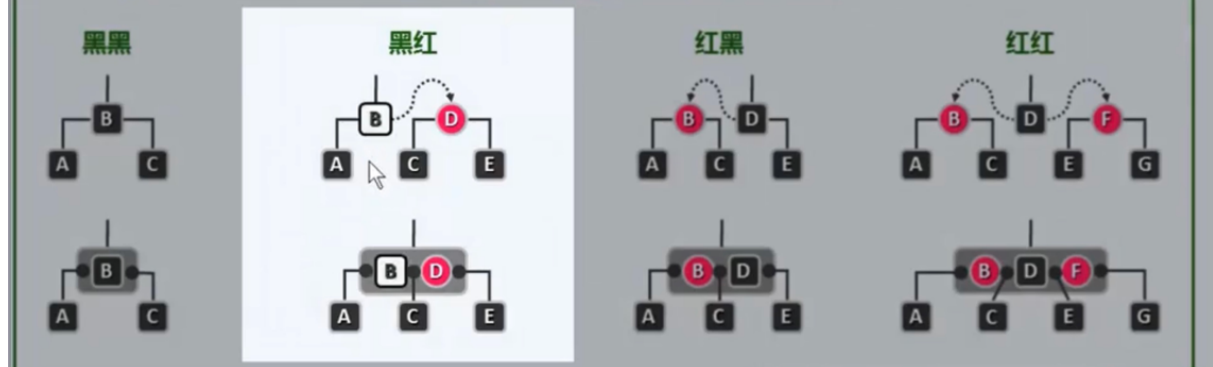
难道是巧合吗?

为什么???

红黑树 即是 B- 树 | (2,4)树 == 红黑树

## (2, 4)树 == 红黑树

- ❖ 提升各红节点，使之与其（黑）父亲等高——于是每棵红黑树，都对应于一棵(2, 4)-树
- ❖ 将黑节点与其红孩子视作（关键码合并为）超级节点...
- ❖ 无非四种组合，分别对应于4阶B-树的一类内部节点 //反过来呢？

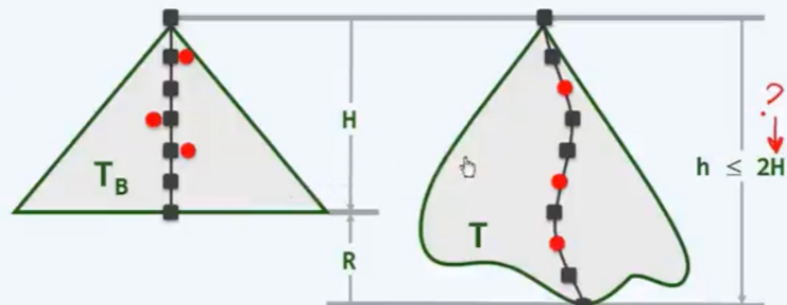


平衡性

## 红黑树 ∈ BBST

- ❖ 由等价性，既然B-树是平衡的，红黑树自然也是 //更严谨地...
- ❖ 定理：包含n个内部节点的红黑树T，高度  $h = O(\log n)$  //n+1个外部节点  
 $\Rightarrow \log_2(n+1) \leq h \leq 2 * \log_2(n+1)$

- ❖ 若：T高度为h，黑高度为H  
 则： $h = R + H \leq 2H$
- ❖ 若T所对应的B-树为 $T_B$   
 则H即是 $T_B$ 的高度



- ❖  $T_B$ 的每个节点，包含且仅包含T的一个黑节点

- ❖ 于是， $H \leq \log_{[4/2]} \frac{n+1}{2} + 1 \leq \log_2(n+1)$

接口定义

## RedBlack

```
❖ template <typename T> class RedBlack : public BST<T> { //红黑树
public:
    ✓ //BST::search()等其余接口可直接沿用
    ✓ BinNodePosi(T) insert( const T & e ); //插入 (重写)
    ✓ bool remove( const T & e ); //删除 (重写)
protected:
    → void solveDoubleRed( BinNodePosi(T) x ); //双红修正
    → void solveDoubleBlack( BinNodePosi(T) x ); //双黑修正
    ✓ int updateHeight( BinNodePosi(T) x ); //更新节点x的高度
};
```

↓

```
❖ template <typename T> int RedBlack<T>::updateHeight( BinNodePosi(T) x ) {
    x->height = max( stature( x->lc ), stature( x->rc ) );
    if ( IsBlack( x ) ) x->height++; return x->height; //只计黑节点
}
```

```
template <typename T> class RedBlack : public BST<T> {
public:
    BinNodePosi(T) insert(const T & e);
    bool remove(const T & e);
protected:
    void solveDoubleRed(BinNodePosi(T) x);
    void solveDoubleBlack(BinNodePosi(T) x);
    int updateHeight(BinNodePosi(T) x); // 黑高度
};

template <typename T> int RedBlack<T>::updateHeight(BinNodePosi(T) x){
    x->height = max(stature(x->lc),stature(x->rc));
    if (IsBlack(x)) x->height++;
    return x->height;// 只统计黑节点
}
```