

10B5 删除

#数据结构邓神

算法框架

算法

```
❖ template <typename T>
bool BTree<T>::remove( const T & e ) {
    BTreeNodePosi(T) v = search( e );
    if ( ! v ) return false; //确认e存在
    Rank r = v->key.search(e); //确定e在v中的秩
    if ( v->child[0] ) { //若v非叶子, 则
        BTreeNodePosi(T) u = v->child[r + 1]; //在右子树中一直向左, 即可
        while ( u->child[0] ) u = u->child[0]; //找到e的后继 (必属于某叶节点)
        v->key[r] = u->key[0]; v = u; r = 0; //并与之交换位置
    } //至此, v必然位于最底层, 且其中第r个关键码就是待删除者
    v->key.remove(r); v->child.remove(r + 1); _size--;
    solveUnderflow( v ); return true; //如有必要, 需做旋转或合并
}
```

Data Structures (Spring 2014), Tsinghua University

这边就是在找比e大的第一个元素，然后交换删除就可以了，可以按照中序遍历理解一下

```
// delete
template <typename T> bool BTree<T>::remove(const T & e){
    BTreeNodePosi(T) v = search(e);
    if (!v){
        return false;
    }
    Rank r = _hot->key.search(e);
    if (v->child[0]){
        BTreeNodePosi(T) u = v->child[r+1];
        while (u->child[0]){
            u = u->child[0];
        }
        v->key[r] = u->key[0];
        v = u;
        r = 0;
    }
    v->key.remove(r);
    v->child.remove(r+1);
```

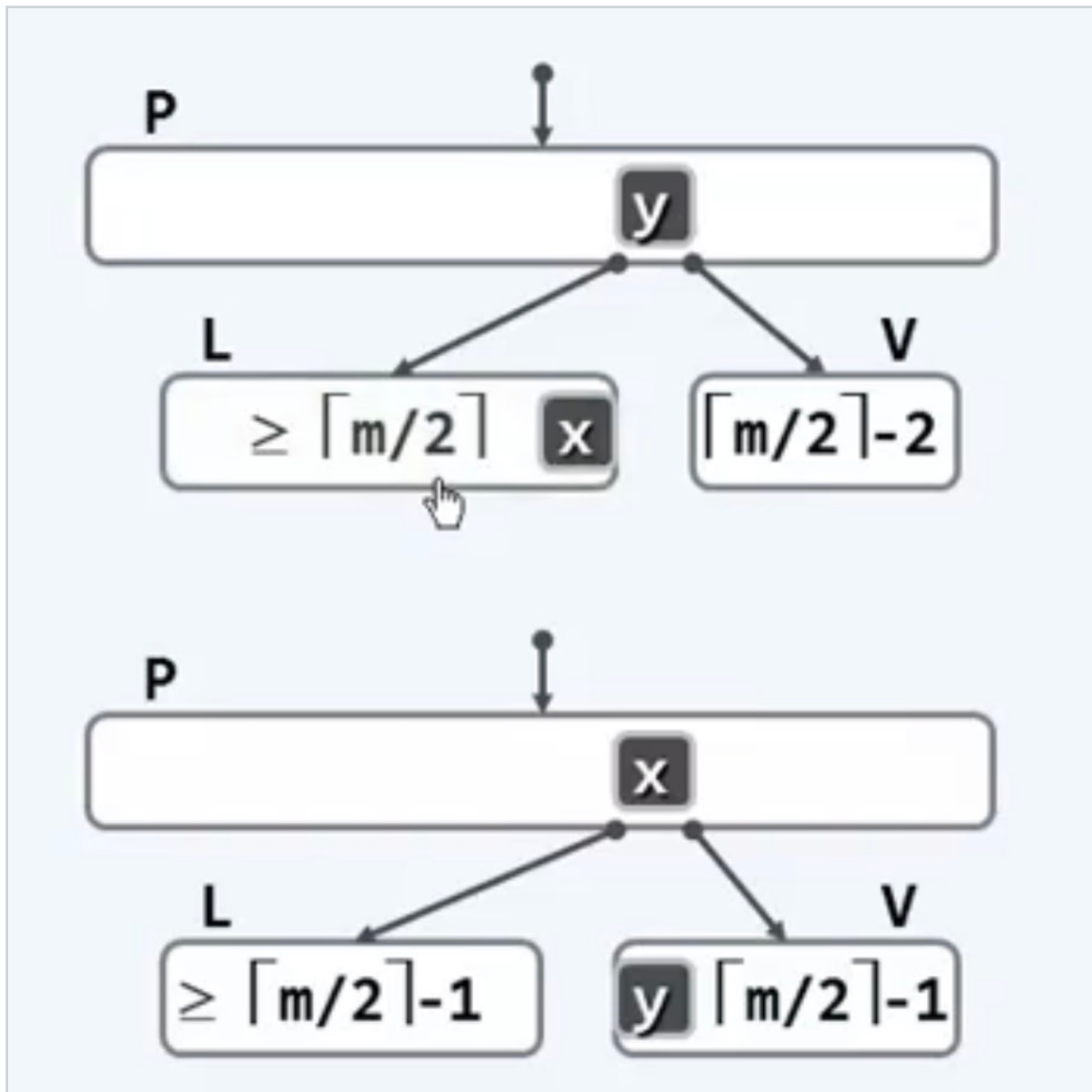
```

_size--;
solveUnderFlow(v);
return true;
}

```

旋转

向左/右边借一下 如果能借

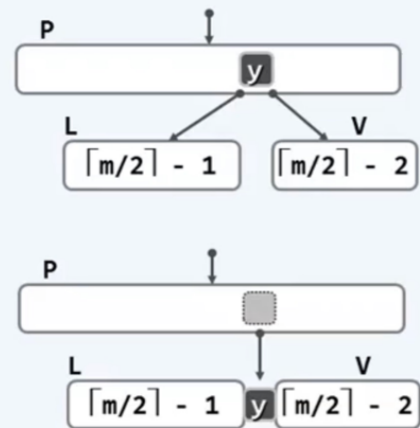


合并

合并

3) L 和 R 或者不存在, 或者所含的关键码均不足 $\lceil m/2 \rceil$ 个

注意, L 和 R 仍必有其一, 且恰含 $\lceil m/2 \rceil - 1$ 个关键码 (不妨以 L 为例)



即便继续发生上溢出, 最多也就 $O(h)$