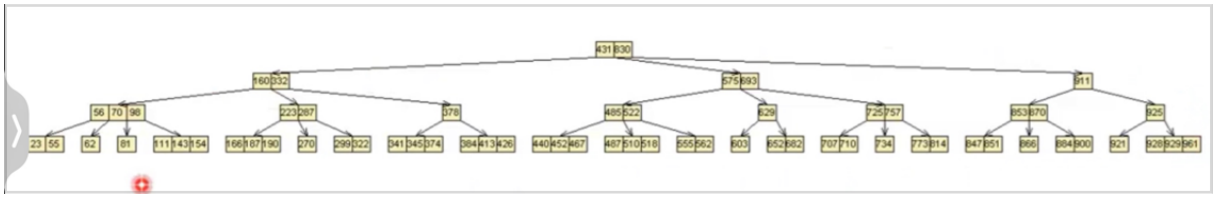
10B2 B-树 结构

#数据结构邓神

体验

RPReplay_Final163919789…
2021年12月11日

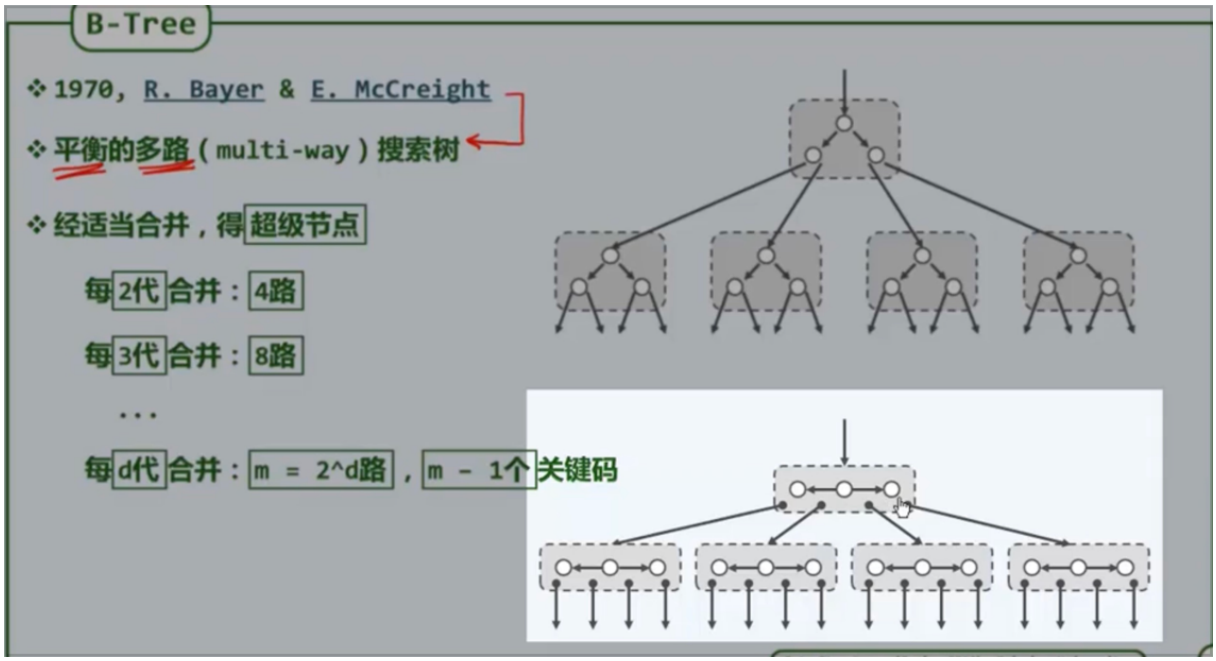


每个节点可能有多个分叉，每个底层节点的深度完全一致。

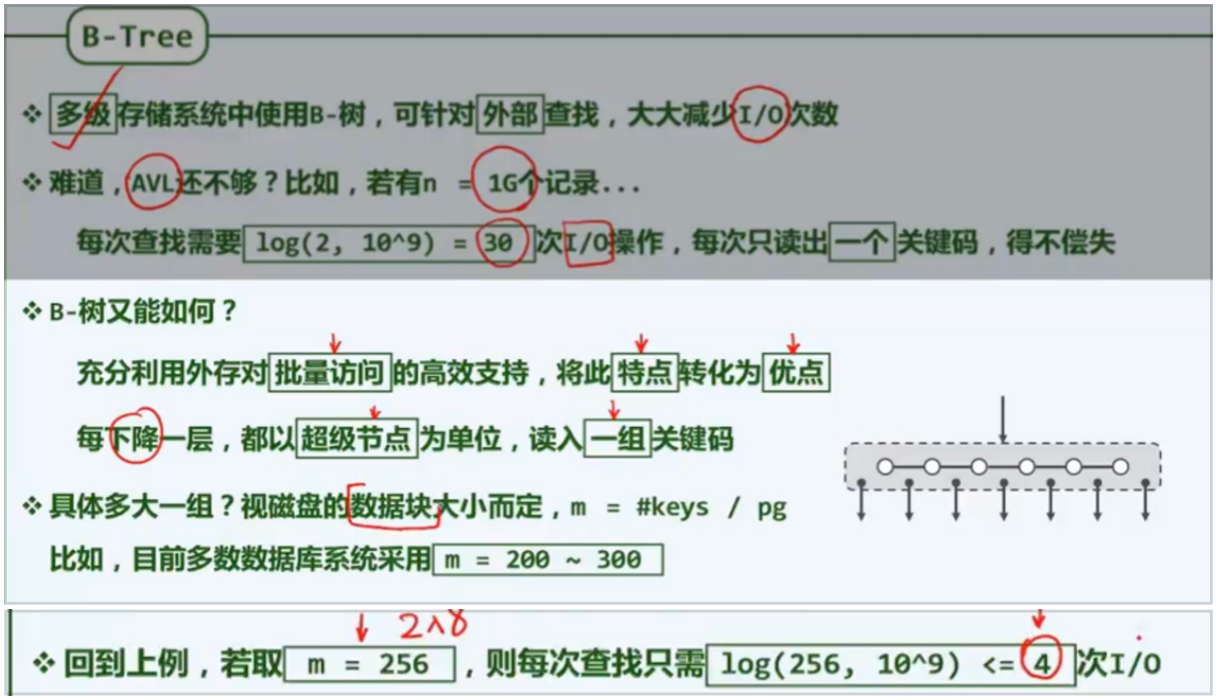
相对于常规的二叉查找树，B树会更宽更矮， 为什么呢?

多路平衡

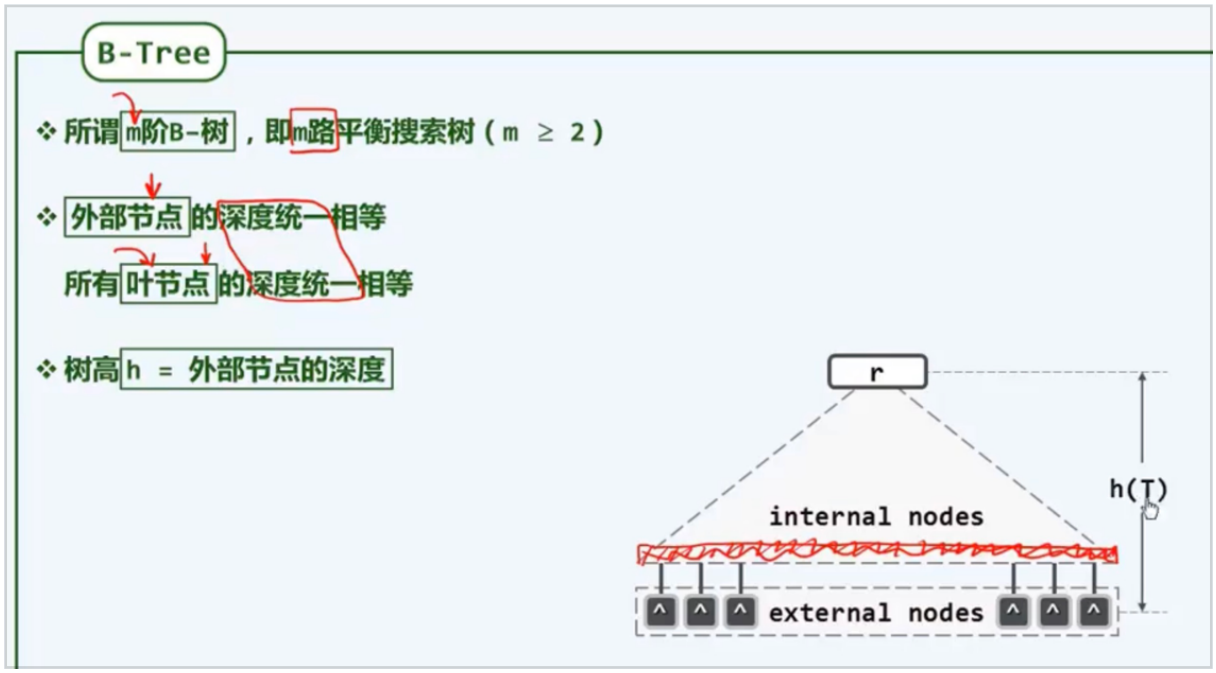
平衡的多数(multi-way)搜索树



还是IO



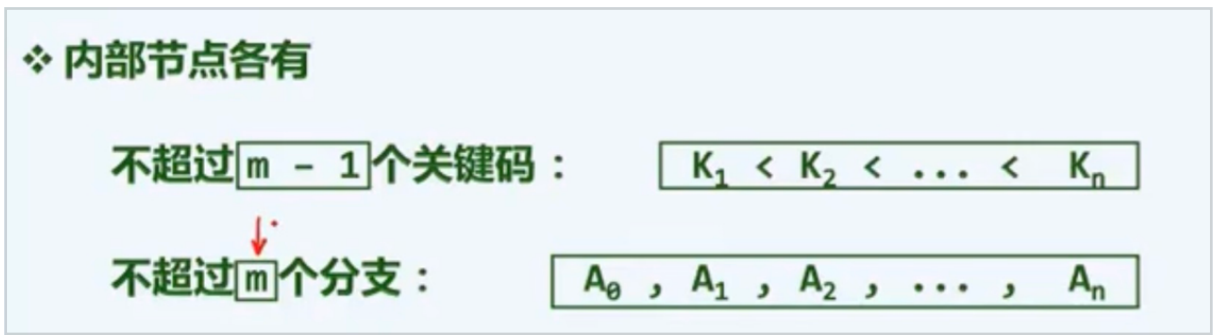
深度统一

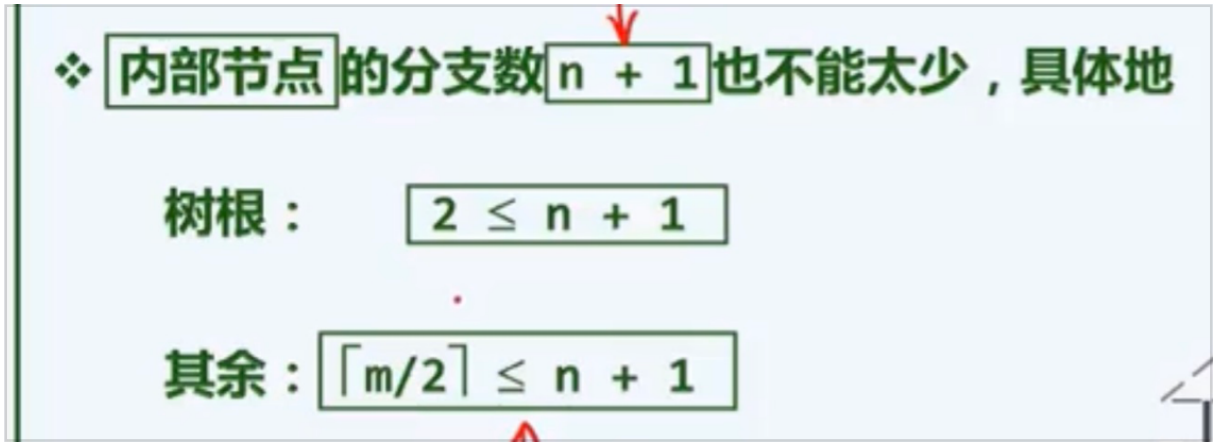


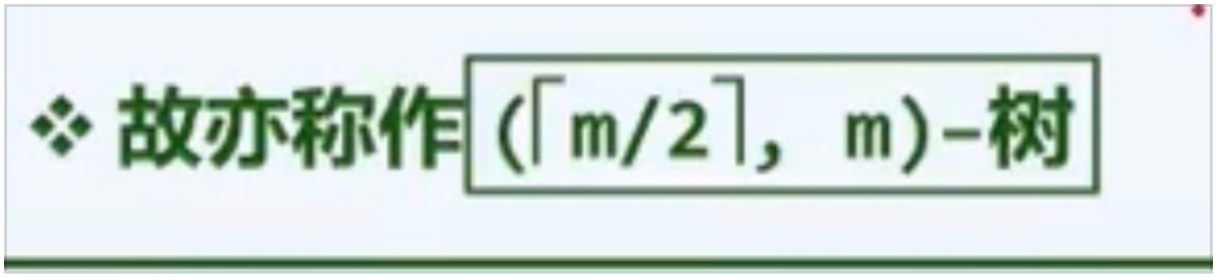
外部节点：就是叶子节点还不存在的孩子节点

高度定义有变化

阶次含义（m）

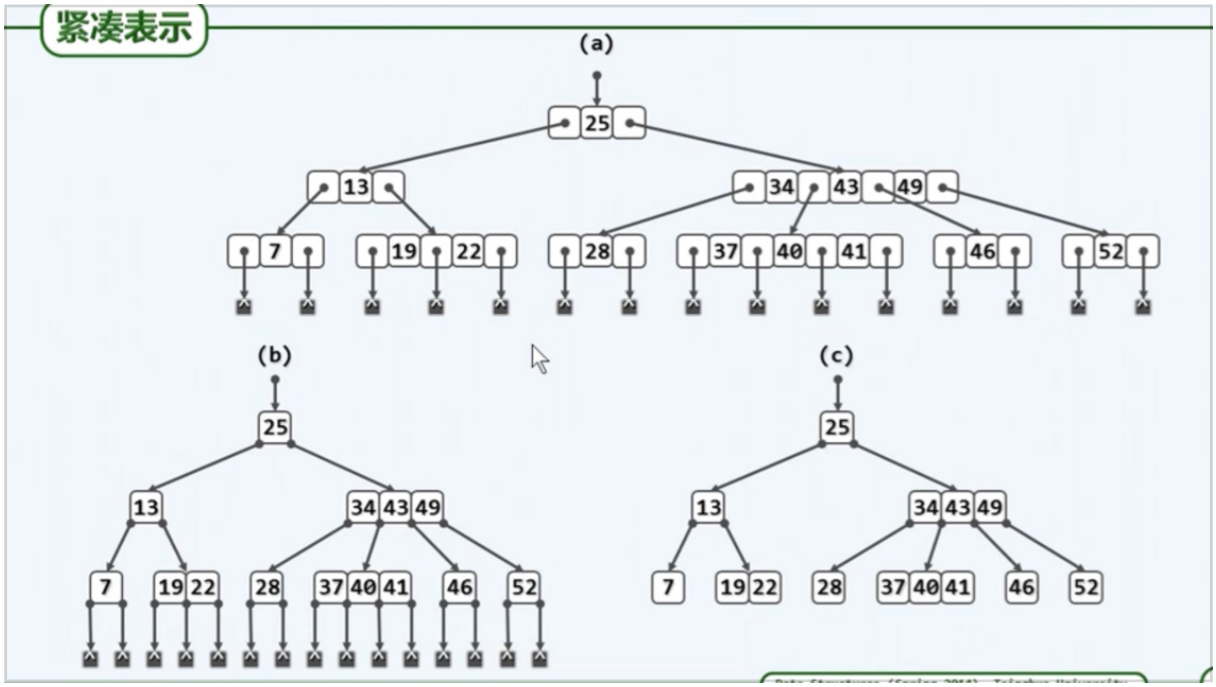






(2,4)树和红黑树有解不开的渊源

紧凑表示



A 是完整表示为每一个关键码分别留出一个左右分支引用的位置 (a ➡ b)

我们可以讲所有引用简化为一个点

同时我们也可以简化掉 x (b ➡ c)

BTNode

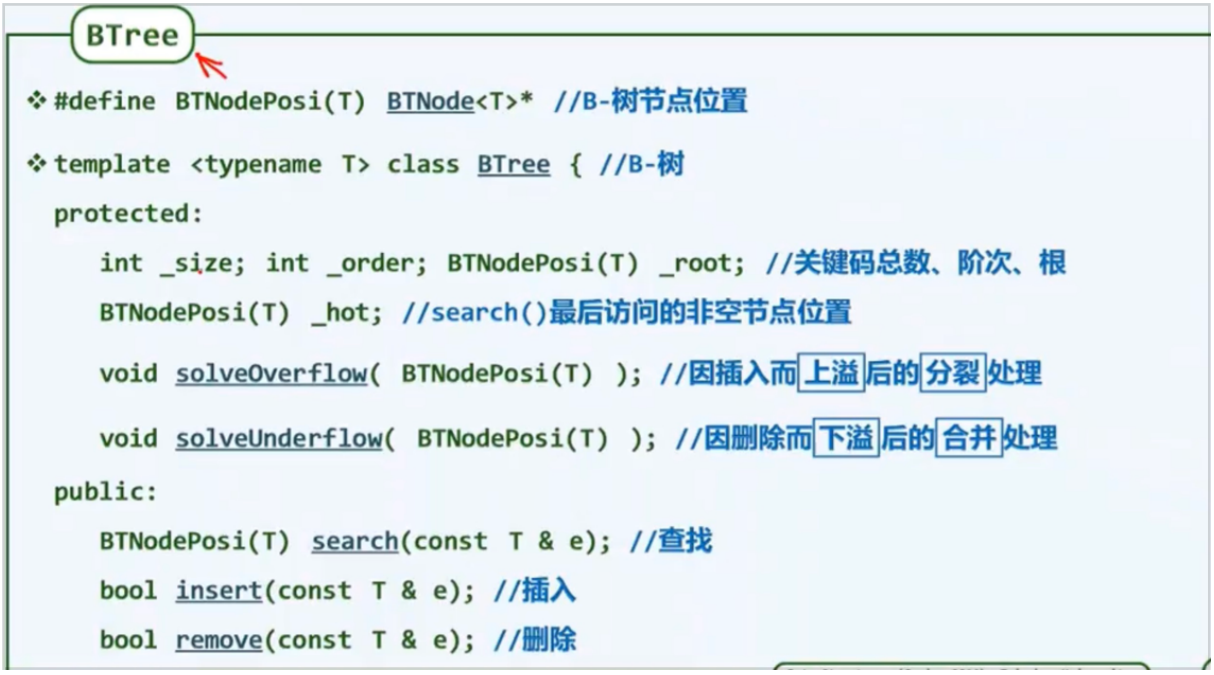
BTNode

```
template <typename T> struct BTNode { //B-树节点
    BTNodePosi(T) parent; //父
    Vector<T> key; //数值向量
    Vector< BTNodePosi(T) > child; //孩子向量（其长度总比key多一）
    BTNode() { parent = NULL; child.insert( 0, NULL ); }
    BTNode( T e, BTNodePosi(T) lc = NULL, BTNodePosi(T) rc = NULL ) {
        parent = NULL; //作为根节点，而且初始时
        key.insert( 0, e ); //仅一个关键码，以及
        child.insert( 0, lc ); child.insert( 1, rc ); //两个孩子
        if ( lc ) lc->parent = this; if ( rc ) rc->parent = this;
```

```
#define BTNodePosi(T) BTNode<T>*
template <typename T> struct BTNode{
    BTNodePosi(T) parent;

    vector<T> key;
    vector<BTNodePosi(T)> child;

    BTNode(){
        parent = nullptr;
        child.insert(0,nullptr);
    }
    BTNode(T e,BTNodePosi(T) lc = nullptr,BTNodePosi(T) rc =nullptr){
        parent = nullptr;
        key.insert(0,e);
        child.insert(0,lc);
        child.insert(1,rc);

        if (lc) {
            lc->parent = this;
        }
        if (rc){
            rc->parent = this;
        }
    }
```

```
};
```

BTree




```
template <typename T> class BTree {
protected:
    int _size;
    int _order;
    BTNodePosi(T) root;
    BTNodePosi(T) _hot;
    void solveOverFlow(BTNodePosi(T));
    void solveUnderFlow(BTNodePosi(T));
public:
    BTNodePosi(T) search(const T & e);
    bool insert(const T & e);
    bool remove(const T & e);
};
```