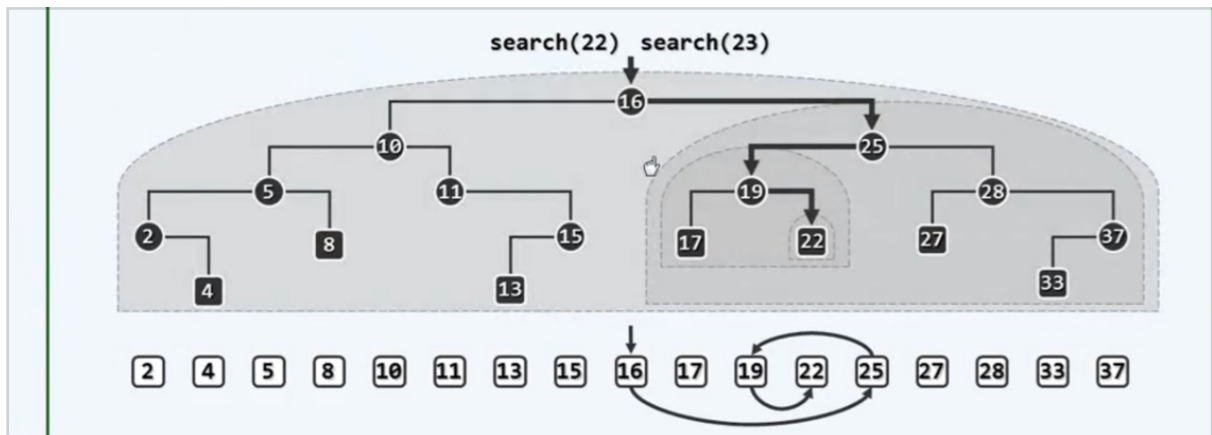


08B1 BST:查找

#数据结构邓神

算法



理解

这就是树的二分查找？

好像跟向量的二分查找差不多

查找 算法

❖ **减而治之** 从根节点出发，逐步地缩小查找范围，直到发现目标（成功），或查找范围缩小至空树（失败）

search(22) search(23)

❖ 对照中序遍历序列可见，整个过程可视作是在仿效有序向量的二分查找

Data Structures & Algorithms / Fall 2013

实现

查找：实现

```
❖ template <typename T> BinNodePosi(T) & BST<T>::search(const T & e) //演示
{ return searchIn( _root, e, _hot = NULL ); } //从根节点启动查找

❖ static BinNodePosi(T) & searchIn( //典型的尾递归，可改为迭代版
    BinNodePosi(T) & v, //当前（子）树根
    const T & e, //目标关键码
    BinNodePosi(T) & hot) //记忆热点
{
    if ( !v || (e == v->data) ) return v; //足以确定失败、成功，或者
    hot = v; //先记下当前（非空）节点，然后再...
    return searchIn( ( ( e < v->data ) ? v->lChild : v->rChild ), e, hot );
} //运行时间正比于返回节点v的深度，不超过树高 $O(h)$ 
```

```
// search
template <typename T> BinNodePosi<T> & BST<T>::search(const T & e){
    return searchIn(_root, e, _hot = nullptr);
}

// searchIn
template <typename T> static BinNodePosi<T> & searchIn(BinNodePosi<T> & v,
    const T & e, BinNodePosi<T> & hot){
    if ( !v || (e == v->data) ){
        return v;
    }
    hot = v;
    return searchIn((e < v->data)?v->lChild:v->rChild, e, hot);
}
```

语义 | hot 有什么用

查找：接口语义

- ❖ 返回的引用值：成功时，指向一个关键码为 e 且真实存在的节点
失败时，指向最后一次试图转向的空节点`NULL`



- ❖ 失败时，不妨假想地将此空节点，转换为一个数值为 e 的哨兵节点
如此，依然满足BST的充要条件；而且更重要地，

- ❖ 无论成功与否：返回值总是等效地指向命中节点，而`_hot`总是指向命中节点的父亲