08 二叉搜索树 | 08A 概述

#数据结构邓神

Vector and List ? | Binary Tree ?

显示的力不从心，不能兼得鱼和熊掌不可兼得

Binary Search Tree 二叉搜索树 | 神

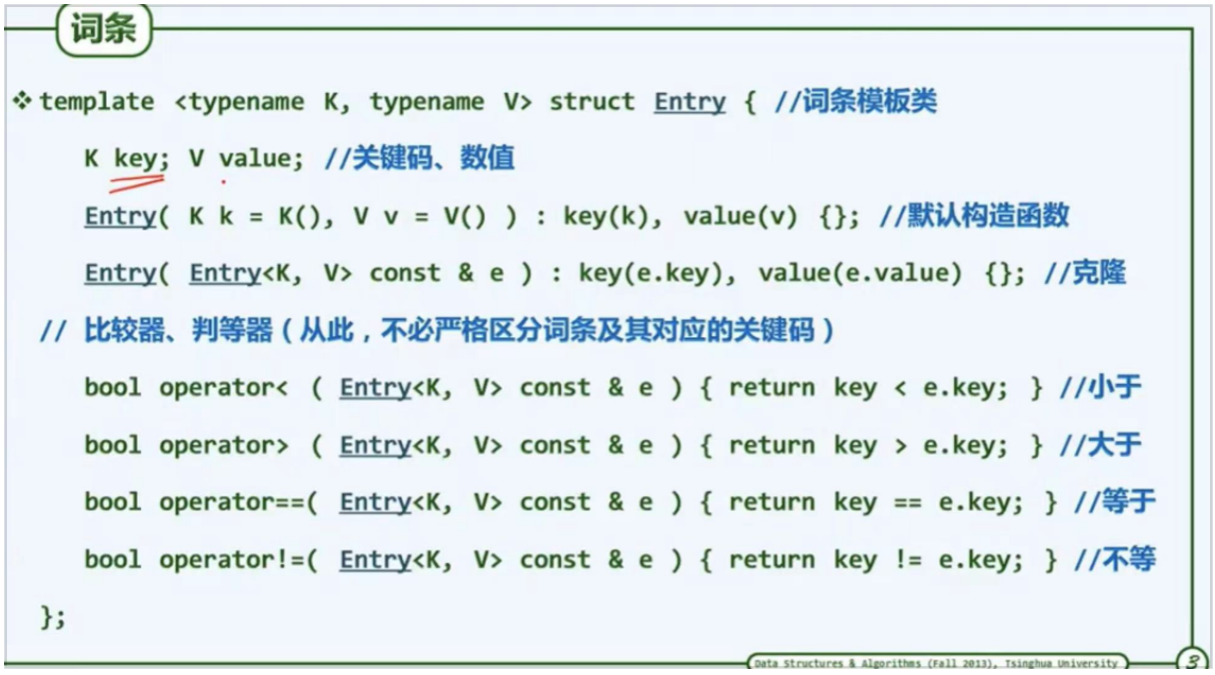
在形式上使用了List的特点，而在很多方面也有Vector上面的借鉴！

Balanced Binary Search Tree 平衡二叉搜索树 | 神中神

- 定义？
- 特点？
- 规范？

循关键码访问

词条

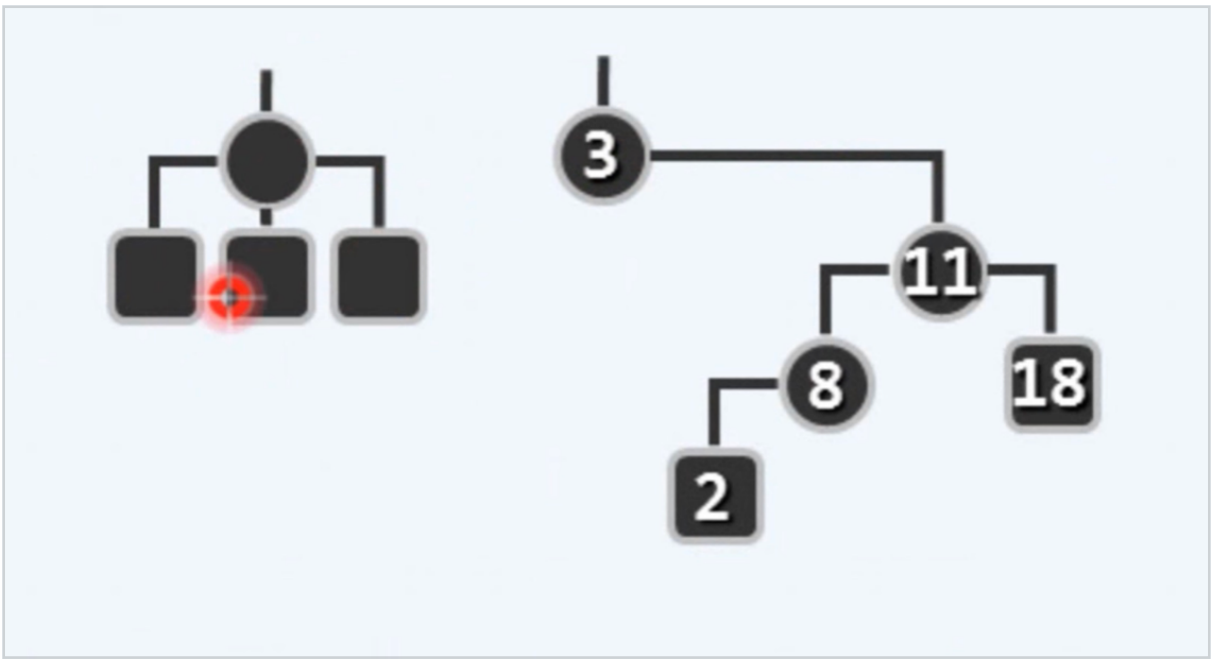



```cpp
template <typename K,typename V> struct Entry {
    K key;
    V value; // <key, Value> pair

    Entry(K k = K(),V v = V()): key(k),value(v){};
    Entry(Entry<K,V> const & e):key(e.key),value(e.value){}; // 克隆构造函数

    // 比较器和判等器，以后默认就比较关键码key 而不是 value;
    bool operator< (Entry<K,V> const & e){
        return key < e.key;
    }
    bool operator> (Entry<K,V> const & e){
        return key > e.key;
    }
    bool operator== (Entry<K,V> const & e){
        return key == e.key;
    }
    bool operator!= (Entry<K,V> const & e){
        key != e.key;
    }
};
```

节点 == 词条 == 关键码

BST：： 任一节点均不小于/不大于其左/右后代

反例



BST必须为二叉树

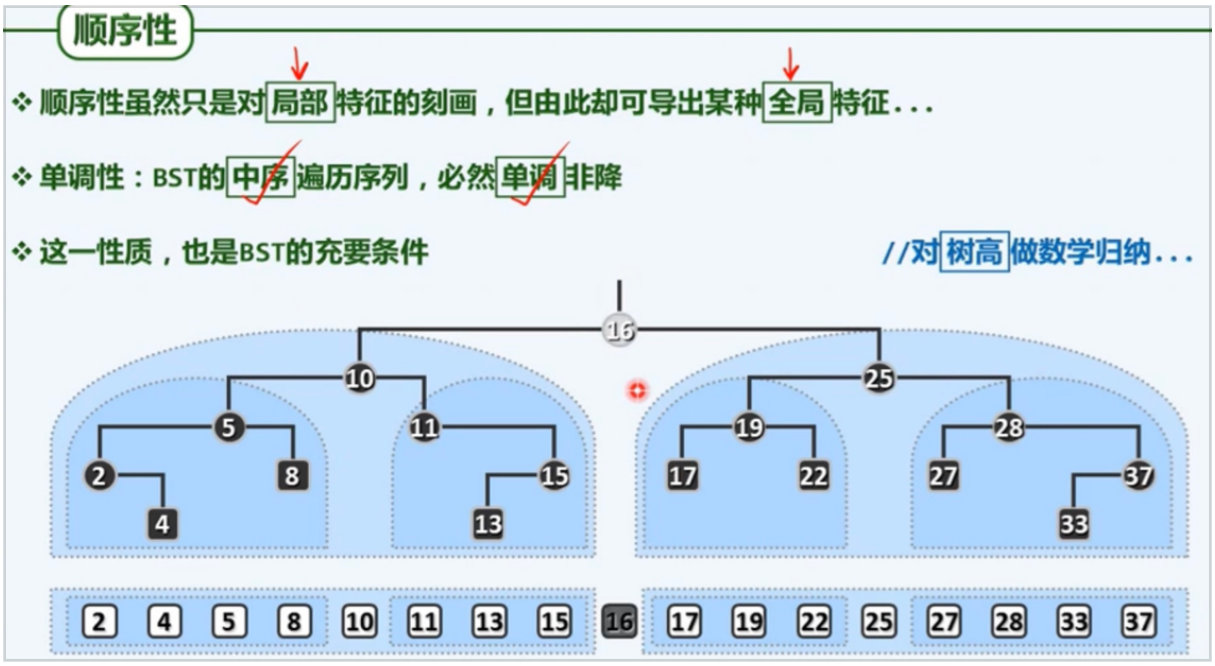
BST：： 任一节点均不小于/不大于其左/右后代

但是 3 > 2

出于简化问题等靠考虑，禁止存在重复词条

这种简化：应用极不自然，算法上也毫无必要

单调性



这个是否是可以看作为一种二分查找？哈哈哈为查找提供方便，同时插入和排序都是O(1)

中序遍历就是所有节点的垂直投影！！！！

在微观处处满足顺序性，在宏观上处处满足单调性

接口


```
template <typename T> class BST : public BinTree<T>{
public:
    virtual BinNodePosi<T> & search(const T &);
    virtual BinNodePosi<T> insert(const T &);
    virtual bool remove(const T &);

protected:
    BinNodePosi<T> _hot;
    BinNodePosi<T> connect34(BinNodePosi<T>,BinNodePosi<T>,BinNodePosi<T>
            ,BinNodePosi<T>,BinNodePosi<T>,BinNodePosi<T>,BinNodePosi<T>);
    BinNodePosi<T> rotateAt(BinNodePosi<T>);
};
```