二叉搜索树

概述：循关键码访问

06-A1

There's nothing in your head the sorting hat can't see.
So try me on and I will tell you where you ought to be.

邓俊辉
deng@tsinghua.edu.cn

循关键码访问

- 很遗憾，向量、列表并不能

  兼顾静态查找与动态修改

- 能否综合二者的优点?

- 各数据项依所持关键码

  而彼此区分：call-by-KEY

- 当然，关键码之间必须同时支持比较（大小）与比对（相等）

- 数据集中的数据项，统一地表示和实现为词条（entry）形式

基本结构	查找	插入/删除
无序向量	O(n)	O(n)
有序向量	O(logn)	O(n)
无序列表	O(n)	O(1)
有序列表	O(n)	O(n)

词条

```
template <typename K, typename V> struct Entry { //词条模板类
   K key; V value; //关键码、数值
   Entry( K k = K(), V v = V() ) : key(k), value(v) {}; //默认构造函数
   Entry( Entry<K, V> const & e ) : key(e.key), value(e.value) {}; //克隆
// 比较器、判等器（从此，不必严格区分词条及其对应的关键码）
   bool operator< ( Entry<K, V> const & e ) { return key < e.key; } //小于
   bool operator> ( Entry<K, V> const & e ) { return key > e.key; } //大于
   bool operator==( Entry<K, V> const & e ) { return key == e.key; } //等于
   bool operator!=( Entry<K, V> const & e ) { return key != e.key; } //不等
};
```

接口

```
template <typename T> class BST : public BinTree<T> { //由BinTree派生

public:     virtual BinNodePosi<T> & search( const T & ); //查找
            virtual BinNodePosi<T> insert( const T & ); //插入
            virtual bool remove( const T & ); //删除
-------------------------------------------------------------------------
protected:  BinNodePosi<T> _hot; //命中节点的父亲
            BinNodePosi<T> connect34( //3+4重构，稍晚再详解
               BinNodePosi<T>, BinNodePosi<T>, BinNodePosi<T>,
               BinNodePosi<T>, BinNodePosi<T>, BinNodePosi<T>, BinNodePosi<T> );
            BinNodePosi<T> rotateAt( BinNodePosi<T> ); //旋转调整
};
```