数据结构

第 15 周 – 二叉搜索树、平衡、堆；课程总结

Koczka Ferenc 博士 · 埃斯特哈齐·卡罗伊大学 · 2026

复习

- **二叉树：**至多两个孩子；链式表示和数组表示
- **遍历：**先序、中序、后序（递归），层序（用队列）
- **二分查找**（第 2 周）：在有序数组中 O(log n) – 但插入是 O(n)

今天的问题

- 怎样让查找**和**插入都很快?
- 为什么树**平衡**很重要?
- **堆**怎样提供快速的优先队列?

二叉搜索树

二叉搜索树

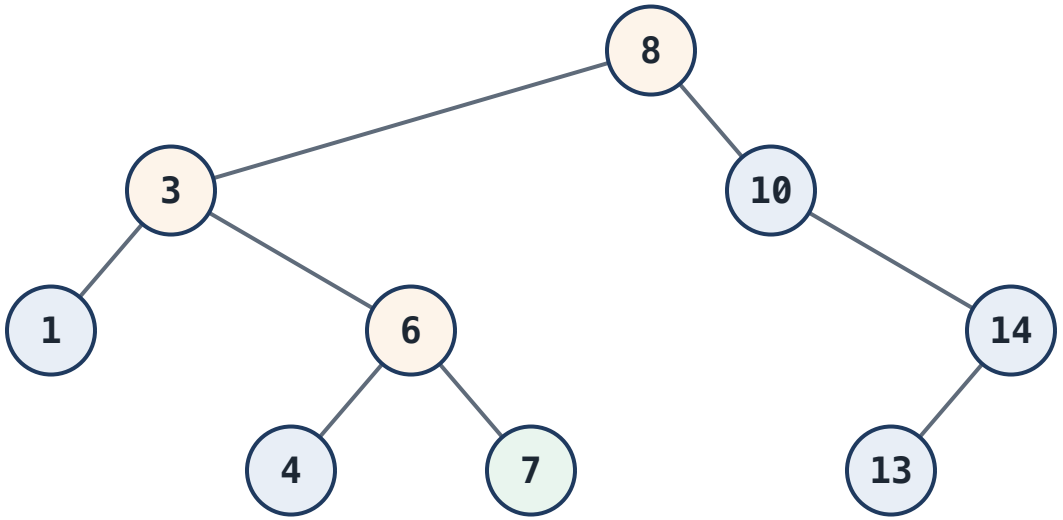
二叉搜索树（BST,**binary search tree**）：对**每一个**结点都满足以下条件的二叉树：

- 它的**左**子树中所有的键都**小于**该结点的键，
- 它的**右**子树中所有的键都**大于**该结点的键。

重要推论：中序遍历按**递增顺序**给出所有的键！
例：依次插入 8、3、10、1、6、14、4、7、13 之后，中序遍历为：**1 3 4 6 7 8 10 13 14**

只是每个结点大于/小于它的**直接**孩子还不够 – 条件针对的是**整棵**子树！

查找



查找：x = 7

7 < 8 → 向左

7 > 3 → 向右

7 > 6 → 向右

7 = 7 → 找到 ✓

每一步下降一层：

步数 = O（高度）

查找 – 算法

迭代实现

```
Function Search(Root, x)
  P ← Root
  While P ≠ Nil and Elem[P].Key ≠ x
    If x < Elem[P].Key then
      P ← Elem[P].Left
    else
      P ← Elem[P].Right
    End if
  End while
  Return P              // 未找到时为 Nil
End function
```

最小值：一直向左，直到不能再走

最大值：一直向右，直到不能再走

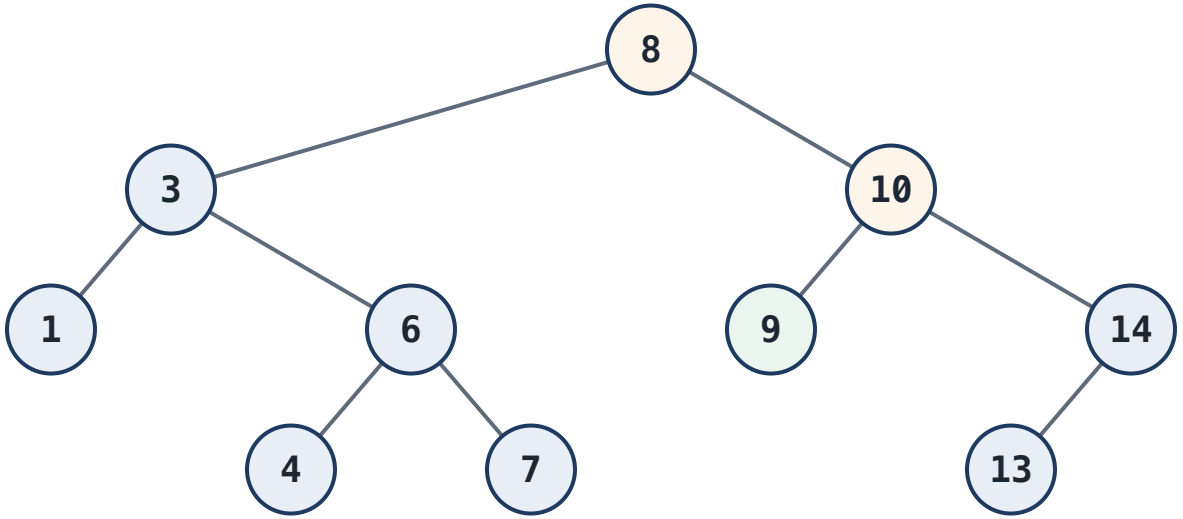
Java

```java
boolean contains(int x) {
    Node p = root;
    while (p != null && p.key != x)
        p = (x < p.key) ? p.left : p.right;
    return p != null;
}
```

Go

```go
func (t *BST) Contains(x int) bool {
    p := t.Root
    for p != nil && p.Key != x {
        if x < p.Key {
            p = p.Left
        } else {
            p = p.Right
        }
    }
    return p != nil
}
```

插入



插入：x = 9

9 > 8 → 向右

9 < 10 → 向左

空 → 放这里（新叶子）

新元素总是叶子。

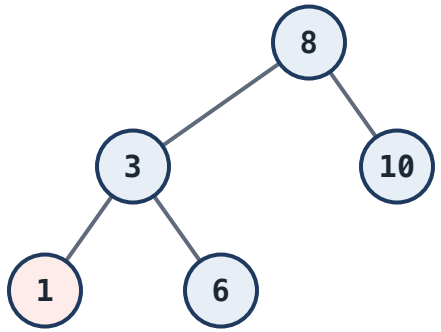
插入 – 算法

```
Function Insert(P, x)          // 返回新的子树
  If P = Nil then
    Reserve(Q) ; Elem[Q].Key ← x
    Elem[Q].Left ← Nil ; Elem[Q].Right ← Nil
    Return Q
  If x < Elem[P].Key then
    Elem[P].Left ← Insert(Elem[P].Left, x)
  else if x > Elem[P].Key then
    Elem[P].Right ← Insert(Elem[P].Right, x)
  Return P
End function
```

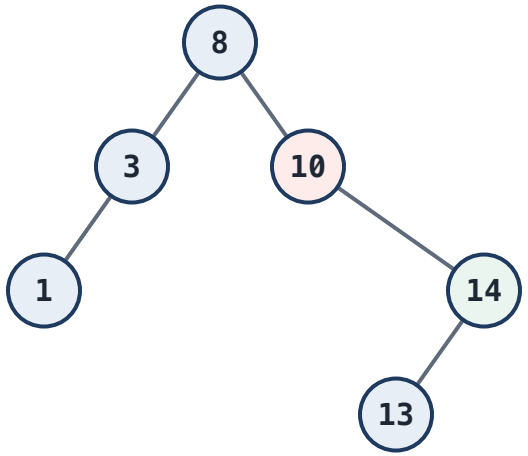
```
Node insert(Node p, int x) {
    if (p == null) return new Node(x);
    if (x < p.key)
        p.left = insert(p.left, x);
    else if (x > p.key)
        p.right = insert(p.right, x);
    return p;
}
```

调用：`root = insert(root, x);`

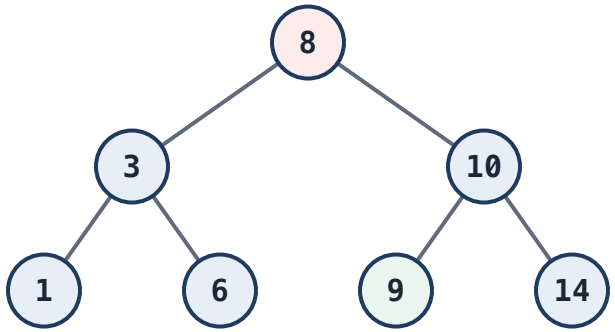
删除 – 三种情况



1. 叶子：
直接删除



2. 一个孩子：
由孩子顶替



3. 两个孩子：
复制中序后继

- **叶子：**直接删除
- **一个孩子：**孩子（连同它的整棵子树）顶替被删除的结点
- **两个孩子：**把右子树中**最小**的元素（**中序后继**）复制到它的位置，然后删除那个元素 – 它至多有一个孩子

删除 – 代码

Java

```java
Node delete(Node p, int x) {
    if (p == null) return null;
    if (x < p.key) p.left = delete(p.left, x);
    else if (x > p.key) p.right = delete(p.right, x);
    else {
        if (p.left == null) return p.right;    // 0 或 1 个孩子
        if (p.right == null) return p.left;
        Node s = p.right;                      // 2 个孩子:
        while (s.left != null) s = s.left;     // 中序后继
        p.key = s.key;
        p.right = delete(p.right, s.key);
    }
    return p;
}
```

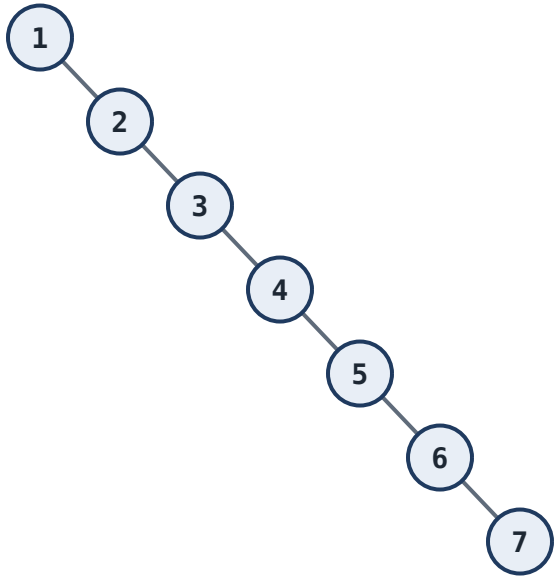
Go – 两个孩子的情况

```go
default:
    if p.Left == nil {
        return p.Right
    }
    if p.Right == nil {
        return p.Left
    }
    s := p.Right              // 中序后继
    for s.Left != nil {
        s = s.Left
    }
    p.Key = s.Key
    p.Right = del(p.Right, s.Key)
```

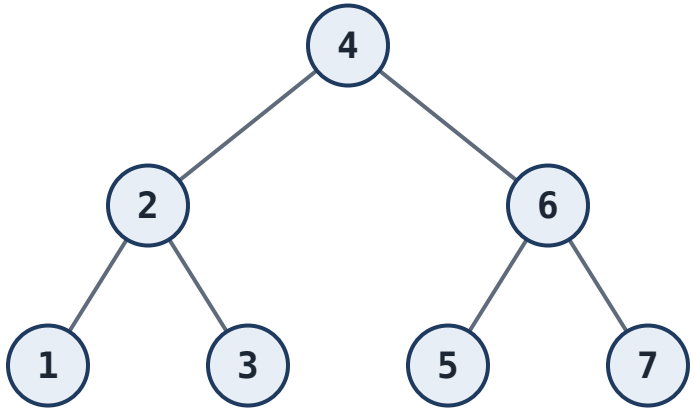
完整代码：`w15/bst`

平衡

二叉搜索树的效率取决于高度



按 1, 2, …, 7 插入：高度 6

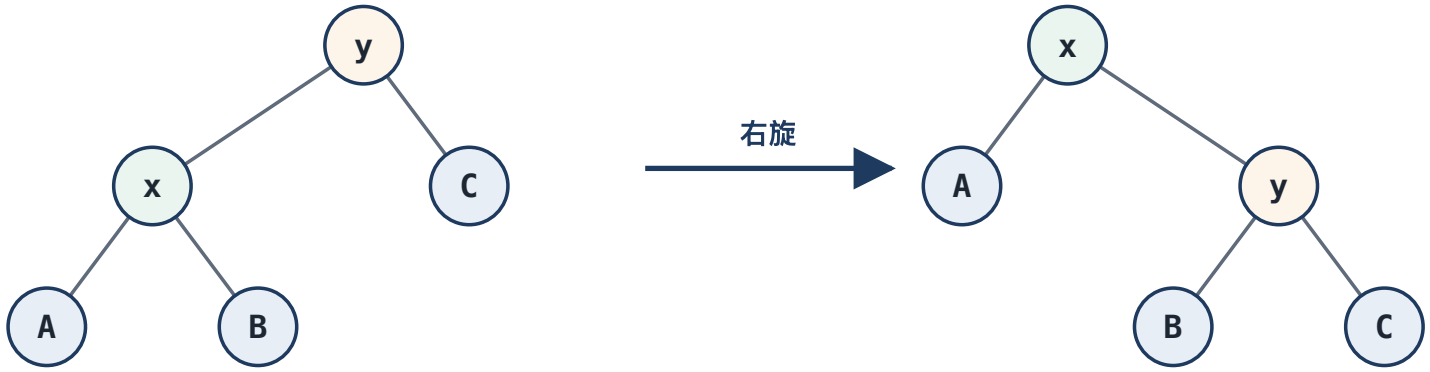


相同元素，平衡：高度 2

- 查找、插入、删除：**O(h)**，其中 **h** 是树的高度
- **平衡**的树：$h \approx \log_2 n \rightarrow$ **O(log n)** · **退化**的树：$h = n - 1 \rightarrow$ **O(n)**（就是链表！）
- **实测：**按有序顺序插入 1000 个元素 → 高度 **999** · 按随机顺序 → 高度 **18**

平衡二叉搜索树

平衡树：在每个结点上，左右子树的高度之差**至多为 1**（AVL 条件）。



A < x < B < y < C – 中序序列（BST 性质）保持不变

- 插入和删除之后用**旋转**恢复平衡 – 一次旋转是 O(1)
- **AVL 树**（1962）、**红黑树**（Java `TreeMap`、`TreeSet`）、**B 树**（数据库、文件系统）
- 保证查找、插入、删除都是 **O(log n)**

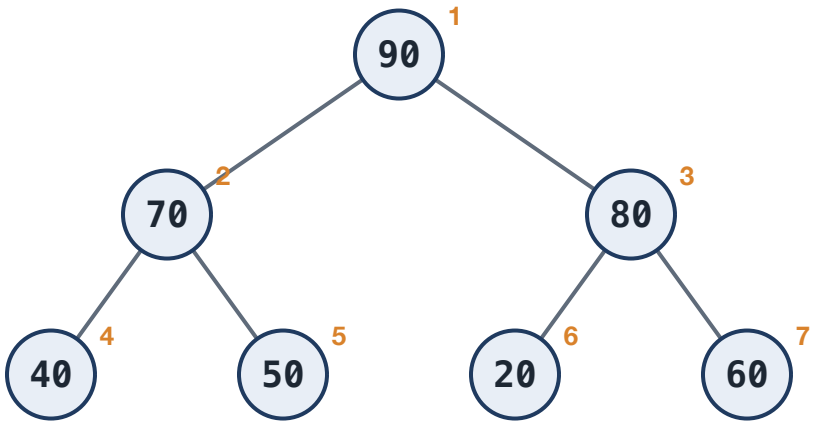
二叉搜索树、有序数组还是哈希表?

操作	有序数组	哈希表	平衡二叉搜索树
查找	O(log n)	平均 **O(1)**	O(log n)
插入、删除	O(n)	平均 **O(1)**	O(log n)
最小值、最大值	**O(1)**	O(n)	O(log n)
有序遍历	O(n)	O(n log n)	O(n)
区间 [a, b]	O(log n + k)	O(n)	O(log n + k)

如果只需要查找：**哈希表**。如果**顺序**也很重要（最小值、下一个、区间）：**二叉搜索树**。

堆

堆



A[1..7] – 用数组存放的完全二叉树（第 14 周）

90	70	80	40	50	20	60
1	2	3	4	5	6	7

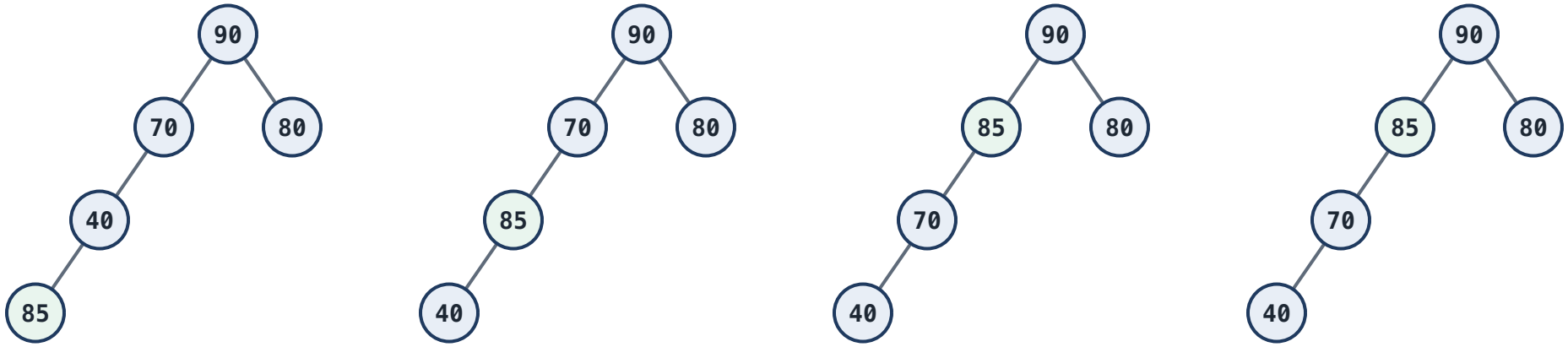
大顶堆：每个父结点 ≥ 它的孩子

`A[i] ≥ A[2i]` 且 `A[i] ≥ A[2i + 1]`

最大值总在根：A[1]

- **完全**二叉树，**用数组**存放（第 14 周：孩子 2i 和 2i + 1，父结点 i div 2）
- **堆性质：**每个父结点 ≥ 它的孩子（大顶堆） – 或 ≤（小顶堆）
- 兄弟之间**没有**顺序 – 堆**不是**二叉搜索树！

插入：上浮



- 新元素放在数组的**末尾**（树的下一个空位）
- 只要它大于父结点，就和父结点**交换** → 至多 **h** = $\log_2 n$ 次交换 → **O(log n)**

取出最大值：下沉

```
Function RemoveMax(A, n)
  max ← A[1]
  A[1] ← A[n] ; n ← n − 1
  i ← 1
  While 2·i ≤ n
    c ← 2·i                          // 左孩子
    If c + 1 ≤ n and A[c + 1] > A[c] then
      c ← c + 1                      // 较大的孩子
    If A[i] ≥ A[c] then exit
    swap(A[i], A[c]) ; i ← c
  End while
  Return max
End function
```

1. **最大值**在根上 – 把它返回
2. 把**最后一个**元素放到根上
3. 只要它小于较大的孩子，就**向下交换**

O(log n)

优先队列	插入	取出最大值
无序数组	O(1)	O(n)
有序数组	O(n)	O(1)
堆	**O(log n)**	**O(log n)**

Java `PriorityQueue` · Go `container/heap`

堆排序

1. **建堆：**在数组上原地建立大顶堆 – O(n)
2. 重复 **n** – 1 次：把根（最大值）**与最后一个元素交换**，把堆看作少了一个元素，然后让根**下沉**

```
[40, 70, 20, 90, 50, 80, 60, 10]   初始
[90, 70, 80, 40, 50, 20, 60, 10]   堆
[10, 20, 40, 50, 60, 70, 80, 90]   有序
```

性质	堆排序
步数	**O(n log n)** – 在所有情况下
额外空间	**O(1)** – 原地排序
稳定	否

课程总结

数据结构 – 总览

数据结构	访问	查找	插入	删除	周
数组	按下标 **O(1)**	O(n) / 有序时 O(log n)	O(n)	O(n)	3
动态数组	O(1)	O(n)	在末尾 O(1)*	O(n)	3
链表	O(n)	O(n)	已知位置 **O(1)**	已知位置 **O(1)**	9–10
栈、队列	只能访问端点	–	O(1)	O(1)	7、8、11
集合（位向量）	–	O(1)	O(1)	O(1)	6
哈希表	–	平均 **O(1)**	平均 **O(1)**	平均 **O(1)**	11
二叉搜索树（平衡）	–	O(log n)	O(log n)	O(log n)	15
堆	最大值：O(1)	O(n)	O(log n)	最大值：O(log n)	15

* 均摊

算法 – 总览

问题	算法	步数
查找	顺序 · 哨兵 · 二分	O(n) · O(n) · O(log n)
排序（简单）	冒泡 · 选择 · 插入	$O(n^2)$
排序（高效）	归并 · 快速 · 堆	O(n log n)
集合运算	有序数组的归并	O(n + m)
表达式	用栈计算后缀表达式	O(n)
模式匹配	朴素算法	O(n · m)
树的遍历	先序、中序、后序 · 层序	O(n)

准备第 16 周的考试

期末笔试（60 分钟，纸笔）

- 关于概念和效率的判断题和选择题
- 代码阅读：栈或队列输出什么？
- 伪代码填空（缺少的表达式、关系运算符）：链表、带哨兵的链表和二叉搜索树的操作
- 在纸上执行的算法：排序、二分查找、遍历、二叉搜索树的插入

上机考试（在计算机上）

- 用 **Java 或 Go** 实现并使用一个数据结构

> 复习每一周的**复习题**和**练习** – 笔试的题目与它们类似！

小结

- **二叉搜索树：**左 < 根 < 右；中序遍历是有序的
- 查找、插入、删除 **O(h)** – 删除有三种情况（叶子、一个孩子、两个孩子）
- **平衡**的树：h = O(log n)；有序插入 → 退化的树
- AVL 树、红黑树：用**旋转**保持树的平衡
- **堆：**用数组存放的完全二叉树，父结点 ≥ 孩子；插入和取出最大值 O(log n)
- **堆排序：**O(n log n)，原地

下周：期末笔试和上机考试。祝大家成功！

复习题

1. 向空的二叉搜索树中依次插入：50、30、70、20、40、60、80、35!画出这棵树，并写出中序遍历！
2. 从第 1 题的树中删除 30!树变成什么样?
3. 判断对错：二叉搜索树总是平衡的二叉树。
4. 判断对错：在二叉搜索树中查找可能和在链表中查找一样慢。
5. 向空的大顶堆中依次插入：15、40、30、50、10!写出每一步之后的数组！
6. 为什么堆不适合快速查找任意一个元素?

实践课

✎ 示例程序

程序	Java	Go	演示内容
二叉搜索树	`BST.java`	`w15/bst`	查找、插入、删除的三种情况、退化的树
堆	`MaxHeap.java`	`w15/maxheap`	上浮、下沉、内置优先队列
堆排序	`HeapSort.java`	`w15/heapsort`	建堆、排序、计时

✎ 练习

1. **是否为二叉搜索树?** 判断一棵二叉树是否是二叉搜索树！（注意：只看直接孩子是不够的！）
2. **第 k 小：** 找出二叉搜索树中第 **k** 小的键！
3. **区间：** 输出介于 a 和 b 之间的键，只进入必要的子树！
4. **最近公共祖先：** 用 O(h) 步找出二叉搜索树中两个键的最近公共祖先！
5. **向下取整：** 找出 ≤ **x** 的最大的键！
6. **前 k 大：** 用大小为 **k** 的小顶堆找出数组中最大的 **k** 个元素！步数是多少?

参考答案：`Exercises15.java`、`w15/exercises`

实践课的后半部分：**期末笔试练习题**（以各周的复习题为基础）。

✎ 课后作业（备考）

1. 为 `BST` 类增加一个 `successor(x)` 方法，返回大于 **x** 的最小的键！
2. 用二叉搜索树实现一个**电话簿**（姓名 → 号码），它还能按字母顺序输出所有条目！
3. 实现**小顶堆**，并用它实现一个简单的**任务调度器**（截止时间最早的任务最先执行）！
4. 复习各周的复习题，并在纸上完成旧试卷中的样题！