

**1. Duplicate Keys 06A2**

如果允许BST中节点的关键码**重复**，你现在能想到什么解决方案？对应的时间、空间效率如何？

**2. Besides Inorder 06A2**

BST可以看作是以关键码为序组织起来的一个数据集合，这种次序与BST的**中序**遍历序列相对应。如果改为与**先序、后序、层次**遍历序列相对应，会分别遇到什么问题？

**3. virtual BST::search() 06A3**

我们看到，Class BST作为BST家族公共的ADT接口，用关键词virtual来修饰search()、insert()、remove()等操作接口，以便该家族中的各变种通过**重写**，来具体定义并实现对应的算法。然而，其中的search()属于**静态**操作（不会修改数据集的内容），似乎应是统一公用的，为何也要如此修饰？

**4. BST::search() 06B1**

我们记得在Vector::search()二分查找算法里，每一步迭代中所做比较的**次序、次数**都颇为讲究，并做过仔细测算与调校。然而，为何在此处实现的BST::search()，却没有过多地考虑这些？

**5. Inserting The First Node 06B2**

讲义中实现的BST::insert()算法，会首先调用search()算法确定一个位置x==NULL及其父节点\_hot，然后在此局部完成新节点的创建及接入。那么特别地，这种机制是如何保证BST中**首个**节点的插入呢？

- 编译运行示例代码中的BST工程，**确认**这类情况可以得到正确处置；
- 阅读并跟踪代码，**了解**其原理。

**6. \_hot in BST 06B2**

我们看到，让BST::search()算法在内部记录一个父节点\_hot，可使insert()、remove()等接口简捷而高效地实现（尽管不易理解）。如果不维护\_hot之类的信息，还有什么**其他**的方式也可以达到这一目的？

**7. 2-Branch Case In removeAT() 06B3**

在待删除结点x的左、右孩子并存时，我们会找到其直接**后继**w，并在交换二者的data之后摘除w。为此，又需进而找到w的父亲u，并将w替换为其（唯一可能非空的）右子树w->rc。在代码中我们看到，该子树既可能作为**左子树**联接至u，也可能作为**右子树**联接。

- 为何可能出现这样的**两种**情况？
- 这两种情况分别在**什么**情况下出现？

**8. Time Cost Of BST::remove() 06B3**

我们知道，调用insert()将一个节点引入BST，所需的时间取决于该节点的**深度**。那么，调用remove()从BST中摘除一个节点所需的运行时间，除了节点的深度还与**哪些**因素有关？

**9. updateHeightAbove() In remove() 06B3**

从示例代码可见：一方面，remove()总是会在调用removeAt()之后，再调用updateHeightAbove(\_hot)来更新祖先节点的高度；然而另一方面，在双分支情况下**实际**被摘除的节点的深度会更大。

- 这样是否会有一些节点**不能**及时更新高度？
- 试阅读示例代码，**确认**你的判断。

**10. Concurrent remove()****06B3**

我们知道, `remove()` 算法为处理双分支的情况, 不得不在一段时间窗口内, 在局部**违反**BST的中序顺序性。然而在**并发**环境中, 同一棵BST需要支持**多个**应用的**同时**访问。不难理解, 尽管这类时间窗口通常很短, 但仍然非常有可能造成其它应用的操作出错, 即便限制其他应用此时只能做`search()`, 也有这类风险。

- a) 为**避免**这类问题, 你觉得可以采取什么措施?
- b) 按照你的方法, 在并发环境中BST各操作接口的性能, 还将受到哪些**因素**的影响?
- c) 针对这些因素, 你的方法还需做哪些**改进**?

**11. Insertion Sequences****06C1**

本节介绍了此前有人为估计出BST的期望高度, 将一组关键码的每一排列作为一个插入序列, 并分别统计由此生成的BST高度。然而即便从讲义中的简单实例我们也可看出, **同一棵**BST可能对应于**多个**插入序列。

- a) 试说明, **越是平衡**的BST, 所对应的插入序列也**越多**;
- b) 试通过编程并实验统计, **验证**上述结论。

**12. Number Of BST's****06C1**

试证明: 由 $n$ 个互异节点组成的BST, 共有 $catalan(n)$ 棵。

**13. Random Sequence****06C1**

讲义中指出, 理想随机的插入序列很难出现, 实际上其中往往含有明显的**非随机性**。试分别验证, 讲义中罗列的各种非随机性 (单调性、局部性、周期性, 等等), 都会倾向于导致BST的**不平衡**。

**14.  $O(1)$ -Time Rotate****06C3**

试验证: 无论在BST中的任何位置做一次zig/zag旋转, 都可以在**常数**时间内完成。

**15. Transform By Rotates****06C3**

- a) 试证明: 借助适当的zig/zag旋转, 任何一棵都可以**转换**为另一棵;
- b) 在最坏情况下, 这样的转换需做**多少次**旋转?
- c) 试设计一个算法, 通过**尽量少**的旋转, 完成上述转换;
- d) 编程实现你的算法, 并**验证**你的结论。

**16. Asymptotically Balanced****06D1**

本节证明了, AVL树的高度不会超过 $O(\log n)$ , 也就是所谓的**渐近平衡**。

- a) 如果将AVL的准则再退让**一步**, 即改为:  $\forall v \in \text{AVL}, |\text{balFac}(v)| \leq 2$ , 是否还是渐近平衡的?
- b) 如果退让**常数步**呢? 试证明你的结论。

**17. Size vs. Height****06D1**

我们知道, AVL树中任何一对兄弟子树, **高度**都彼此接近——然而, 这并不意味着二者的**规模**也相近。

- a) 在高度为 $h$ 的AVL树中, 左、右子树的规模之差**最大**可能达到多少?
- b) 编程、手绘或借助演示工具, 为你的答案提供佐证。

**18. Size/Weight Balanced****06D1**

我们已经看到, 通过限制所有节点的平衡因子有界, AVL成功地保证了整体的渐近平衡。但稍加推敲便不

难发现，其对平衡因子的定义未免显得**迂回**。实际上，人们最早尝试的是一种更自然更**直觉**的尺度，即用左、右子树的**规模**（而非高度）之差作为平衡/失衡程度的度量：

$$\text{balFac}(v) = \text{size}(\text{lc}(v)) - \text{size}(\text{rc}(v))$$

这也称作**规模平衡**（size-balanced）或**权重平衡**（weight-balanced）。

有趣的是，这方面的探索后来都逐渐湮没于学术历史的长河中，你认为这可能是什么原因？

## 19. remove() In Fibonaccian Trees

06D1

本节考查过**最瘦的**AVL树，这类树中所有内部节点的平衡因子都是+1，且它们的规模（大致）呈Fibonacci数列规律而增长的，所以也称作**Fibonacci树**。

- 实际上，在高度为h的Fibonacci树中删除一个节点，有可能会（相继地）引发h/2个节点失衡，相应地也需做h/2次旋转调整。试指出能导致这一极端情况的节点**位置**；
- 试对任何 $n = \text{fib}(h + 3) - 1$ 设计一个插入序列，由它可生成一棵高度为h、规模为n的**Fibonacci树**；
- 试说明，在高度为2h的AVL中删除一个节点后，可能需要旋转2h次；
- 试设计一个插入序列，由它生成的BST就是一棵且具有上述性质的AVL树。

## 20. AVL::insert()

06D2

考查在AVL中（按常规BST的算法）刚刚**接入**一个节点，尚未重平衡的时刻。

- 为何只有其祖先**才可能**失衡？
- 为何其父亲（\_hot）**不会**失衡？
- 何时除父亲之外的**所有**祖先可能同时失衡？
- 如果祖先a1与a2失衡，它们之间的其他祖先是否**必然**失衡？

## 21. AVL::remove()

06D2

考查在AVL中（按常规BST的算法）刚刚**摘除**一个节点，尚未重平衡的时刻。

- 为何只有祖先**才可能**失衡？
- 为何**至多**只有一个祖先失衡？
- 何时父亲**就可能**失衡？

## 22. AVL::insert()

06D3

考查在AVL中（按常规BST的算法）刚刚**接入**一个节点，尚未重平衡的时刻。

- g和p的平衡因子可能有哪些**组合**？
- 二者会否因平衡因子为0，而导致tallerChild()的**歧义**？
- 如果出现歧义，你认为应该如何**破解**？示例代码是这样处理的吗？

## 23. AVL::remove()

06D4

考查在AVL中（按常规BST的算法）刚刚**摘除**一个节点，尚未重平衡的时刻。

- g和p的平衡因子可能有哪些**组合**？
- 二者会否因平衡因子为0，而导致tallerChild()的**歧义**？
- 如果出现歧义，你认为应该如何**破解**？示例代码是这样处理的吗？

**24. AVL::connect34()****06D5**

- a) 试对照AVL树插入、删除的各种情况验证：无论如何，的确都对应于局部**三个节点、四棵**子树的重构；
- b) 对于上述tallerChild()的**歧义**情况，connect34()的是如何**破解**的？