1. Path Length 05A

将路径的长度定义为**边**的总数而非**顶点**的总数，相对而言有何便利之处？

2. Depth & Height 05A

a) 试证明：树中任何节点v都满足：$depth(v) + height(v) \leq height(T)$；

b) 何时取等号？

3. Tree Representation 05B

在**父节点**、**孩子节点**、**父节点-孩子节点**表示法中

a) 如何判定一对节点是否为**祖先-后代**关系？为此需要花费多少时间？

b) 如何支持节点的**接入**？

c) 如何支持子树的**删除**？

d) 如何找出任意一对节点的**最低公共祖先**（Lowest Common Ancestor）？

4. Binary Tree/Forest 05C

我们在课上已看到，在**有根**、**有序**的前提下，任何一棵多叉树经过变换都对应于一棵二叉树。

a) 反过来，任何一棵二叉树是否也会对应于某棵多叉树？

b) 试在由有根、有序树构成的所有**有序森林**，与所有的**二叉树**之间建立一个一一对应的关系。

5. Proper Binary Tree 05C

讲义中指出，任何一棵二叉树均可转化为一棵**真**二叉树。反过来，这种转换是否也总是可行？

6. updateHeightAbove() 05D

为确保所有祖先的高度都能更新，示例代码中实现的算法会一直逐层上溯到根。

a) 在某节点作为叶子插入之后，如果某个祖先的高度没有变化，是否**更高的**祖先们也必然不会变化？

b) 在某个叶节点被删除之后呢？

c) 在某棵子树接入（attach()）之后呢？

d) 在某棵子树分离（secede()）之后呢？

e) 基于以上结论，可如何**优化**updateHeightAbove()算法？

f) 如何评估你的优化效果？

7. Energetic vs. Lazy 05D + 05E1

二叉树的height、size等信息有两种记录方式，在我们的示例代码中，前者采用“**勤奋**策略”，每个节点的高度一旦有变化，便立即更新；后者则采用“**懒惰**策略”，直到需要时才通过接口size()递归地统计。

a) 试颠倒过来，分别修改代码，用另一种策略来记录height、size，并完成测试。

b) 两种策略各有什么优点、缺点，分别**适用**于哪些应用场合？

8. Iterative Preorder Traversal 05E1

讲义中介绍的先序遍历的一种迭代算法，思路是沿着**左侧通路**逐层地分解二叉树。自然地，也应该可以对称地沿着**右侧通路**来分解。

a) 试按照后一思路，实现先序遍历的另一个迭代算法；

b） 相对于讲义中的算法，新的这个算法有何优点、缺点？

9. Correctness Of Iterative Preorder Traversal 05E2

a） 试通过对二叉树的**规模**做数学归纳，证明本节给出的迭代式**先序**遍历算法是正确的。

提示：考查藤上所有节点均被访问之后的时刻——此时，原树接下来的遍历过程，可分解为对**多棵**右子树的遍历；这些子树的规模均有所**削减**，且彼此**独立**。

b） 对藤的长度做数学归纳，是否同样可以证明？

如果不行，试说明理由；否则，给出你的证明。

10. Storage Cost Of Interative Preorder Traversal 05E2

在二叉树T中若节点v是节点a的**左**后代，则称a是v的**左**祖先。设按照讲义中的算法对T做迭代式**先序**遍历。

a） 试证明：该算法的空间复杂度不会超过T的高度；

b） 试证明：空间复杂度更精确的估计是$\mathcal{O}(\ \max\{\ la(v)\mid v\in T\ \}\ )$，其中$la(v)$为节点$v$所有左祖先的总数；

c） 当然，后一上界不致超过前者，但二者的**差距**最大可能多大？试构造出这样极端的T；

d） 试构造另一极端的T来说明：两个上界也可能**相等**；

e） 两个上界之比值的数学期望是多大？为什么？

11. Tail Recursion 05E2

考查二叉树的**递归式**先序遍历算法。

a） 试验证：在二叉树退化成**单链**（所有节点至多只有一个孩子）时，遍历的过程相当于**尾递归**；

b） 此时，时间、空间复杂度可以如何**优化**？

c） 如果BinNode**没有**设置parent引用，你的优化是否依然可行？

12. Correctness Of Iterative Inorder Traversal 05F[2+3]

a） 试通过对二叉树的**规模**做数学归纳，证明本节给出的迭代式**中序**遍历算法是正确的。

提示：考查藤的末端节点刚被访问之后的时刻——此时，原树接下来的遍历过程，可分解为对**两棵**子树的遍历；这两棵子树的规模均有所**削减**，且彼此**独立**。

b） 对藤的长度做数学归纳，是否同样可以证明？

如果不行，试说明理由；否则，给出你的证明。

13. Head/Tail Recursion 05E2

考查二叉树的**递归式**中序遍历算法。

a） 对于什么样的二叉树，遍历的过程会相当于**尾递归**？

b） 对于什么样的二叉树，遍历的过程会相当于**头递归**（递归发生在函数中**最靠前**的位置，仅次于递归基）？

14. Standard Iterator 05F4

我们针对**中序**遍历，实现了一个BinNode::**succ**()接口。如果还能实现接口BinTree::**first**()，确定遍历的起始节点，那么二叉树T的中序遍历就可以简明地描述并实现为如下循环：

```
for ( BinNodePosi v = T.first(); v; v = v.succ() )
   visit(v->data);
```

a） 扩充并实现BinTree::first()接口，并完成测试；

b) 按上述思路实现中序遍历接口，并完成测试；

c) 新接口的时间复杂度，是否还是$\mathcal{O}(n)$？空间呢？

d) 试实现**先序**遍历所对应的succ()和first()接口，并分析其时间、空间复杂度；

e) 试实现**后序**遍历所对应的succ()和first()接口，并分析其时间、空间复杂度。

15. Left/Right Parent/Ancestor 05F4

从本节开始，同一父节点相对于**左**/**右**孩子而论时，称作**左**/**右**父亲。请注意，这一定义与我们的直觉恰好相反，**左**/**右**父亲其实位于孩子节点之**右**/**左**。对于祖先，可以照此以**左**/**右**来称谓。

16. IsLChild() & IsRChild 05F4

在BinNode::succ()算法的后一分支中，我们用到了IsRChild()——与IsLChild()类似，它们都是示例代码中定义的宏。无论是从一般的逻辑覆盖来讲，还是就此处循环终止的条件而言，我们都必须处理**树根**（既**非左**孩子，亦**非右**孩子）这一特殊情况。我们的示例代码对此具体是如何处理的？

17. Last Inorder Node 05F4

无论何种遍历，每一棵二叉树都有且仅有一个**最终**被访问的节点。当然，对于这个节点而言的succ()应该返回**NULL**。本节针对中序遍历所实现的succ()算法，是如何**落实**这一功能的？

18. Parent in postOrder() 05G2

讲义中实现的迭代式后序遍历算法需要借助parent信息，来**区分**"返回父节点"或"展开右子树并遍历之"这两种情况。如果BinNode结构**没有**记录parent信息，该算法可否在经过适当调整之后依然可行？

19. Correctness Of Iterative Postorder Traversal 05G4

仿照先序、中序的方法，证明本节给出的迭代式后序遍历算法是正确的。

20. Amortization 05G4

任选一种**分摊分析**的方法以证明，迭代式先序、中序、后序遍历算法的时间复杂度均为$\mathcal{O}(n)$。

21. RPN ~ Postorder 05G5

我们知道只含一元、二元运算符的合法表达式都可转换为二叉树，并进而通过后序遍历得到对应的RPN。那么反过来，可否由PRN得出对应的二叉树呢？若可以，方法如何？若不可以，原因何在？

22. Level-Order Traversal 05H1

试证明层次遍历的如下性质，并由此确立算法的正确性及复杂度：

a) 每次迭代中入队的节点（若存在），都是出队节点的孩子；

b) 辅助队列中的各节点，在任何时刻都按深度单调排列，而且深度相差不超过1层；

c) 所有节点迟早都会入队，而且更高/低的节点，更早/晚入队；更左/右的节点，更早/晚入队；

d) 每次迭代中尽管入队节点数目（从0至2）不定，但总是恰有一个节点出队并接受访问；

e) 每个节点入、出队恰好各一次，故知整体只需$\mathcal{O}(n)$时间。

23. Storage Cost Of Level-Order Traversal 05H1

a) 层次遍历的**空间**成本主要消耗于辅助队列，那么这一成本与二叉树的**结构**有何关系？

b) 在规模同为n的所有二叉树中，哪一棵的层次遍历需要使用**最多**的空间？

24. Complete Binary Tree 05H2

a） 本节指出，完全二叉树可以向量的形式，紧凑而高效地**物理**实现。试动手完成这一任务。

b） 这种实现方式能否充分发挥**系统缓存**的作用？也就是说，连续访问的节点是否在物理上通常会彼此**临近**？

25. Reconstruction Of Proper Binary Tree 05I

本节指出，仅凭其先序与后序遍历序列，依然可以重构一棵**真**二叉树，试编程实现这一算法。

26. Recontruction By Augmented Sequences 05I

a） 试证明：由先序增强序列，可以构造出对应的二叉树；

b） 试编程实现对应的算法；

c） 中序增强序列呢？

d） 后序增强序列呢？

27. More Reconstructions 05I

各种遍历序列的其它组合，是否可以完成对原二叉树的重构？试逐一考查验证。

28. Greedy Huffman: Lower/Higher Frequency Lower/Higher 05J1

a） 试证明Huffman贪心策略的正确性：

任何一对子树**交换**位置，只要能使频率高/低者更高/低，编码成本便会**下降。**

b） 下降的**数值**取决于哪些因素？为什么？

29. Huffman Tree Is Optimal 05J2

试按照讲义中的思路证明：尽管最优编码树未必**唯一**，但Huffman算法所构造出来的必属**其一。**

30. Implementation Of Huffman ALgorithm 05J3

试阅读示例代码中Huffman算法的部分，重点厘清以下方面：

a） 算法的**主体框架**是在何处、如何描述的？

b） 目前是如何实现Huffman**森林**的？

c） 算法框架的描述，如何能够**独立**于具体所选用的数据结构？

d） 日后实现Huffman森林的其他更**高级**的数据结构，是如何与此框架自然**接驳**的？

31. Huffman Tree Using A Stack And A Queue 05J4

试按照本节介绍的方法，借助栈和队列来简明实现一个复杂度为$\mathcal{O}(n\log n)$的Huffman算法。

32. Reduction & Lower Bound 05K2

针对讲义中所列的一系列计算问题，试按照提示分别通过建立**适当**的规约，确定其复杂度**下界。**