

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**

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

- 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？
- 如何支持节点的**接入**？
- 如何支持子树的**删除**？
- 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。

- 反过来，任何一棵二叉树是否也会对应于某棵多叉树？
- 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。

- 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？
- 在某个叶节点被删除之后呢？
- 在某棵子树接入 (attach()) 之后呢？
- 在某棵子树分离 (secede()) 之后呢？
- 基于以上结论，可如何**优化**updateHeightAbove()算法？
- 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。

- 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。
- 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**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。

- 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

05E2

试通过数学归纳法, 证明本节的迭代式**先序**遍历算法是正确的, 亦即, 其**功能**与递归版完全一致。

(比如, 可对二叉树的**规模**作归纳。考查左侧路径末端节点刚被访问完的时刻: 此时, 原树的遍历问题被分解为两棵子树的遍历问题; 两棵子树的规模均有所**削减**, 且两个子问题在某种意义上彼此**独立**。)

10. Storage Cost Of Iterative 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。

- 试证明: 该算法的空间复杂度不会超过 T 的高度;
- 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;
- 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 ;
- 试构造另一极端的 T 来说明: 两个上界也可能**相等**;
- 两个上界之比值的数学期望是多大? 为什么?

11. Correctness Of Iterative Inorder

05F[2+3]

试通过数学归纳法, 证明本节的迭代式**中序**遍历算法是正确的, 亦即, 其**功能**与递归版完全一致。

12. 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);
```

- 扩充并实现`BinTree::first()`接口, 并完成测试;
- 按上述思路实现中序遍历接口, 并完成测试;
- 新接口的时间复杂度, 是否还是 $\mathcal{O}(n)$? 空间呢?
- 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;
- 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。

13. Left/Right Parent/Ancessor

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。

14. IsLChild() & IsRChild

05F4

在`BinNode::succ()`算法的后一支中, 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?

15. 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?

16. 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?

17. Amortization**05G4**

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

18. 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?

19. 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:

- 每次迭代中入队的节点 (若存在), 都是出队节点的孩子;
- 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;
- 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;
- 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;
- 每个节点入、出队恰好各一次, 故知整体只需 $\mathcal{O}(n)$ 时间。

20. Storage Cost Of Level-Order Traversal**05H1**

- 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?
- 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?

21. Complete Binary Tree**05H2**

- 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。
- 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**?

22. 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。

23. Recontruaction By Augmented Sequences**05I**

- 试证明: 由先序增强序列, 可以构造出对应的二叉树;
- 试编程实现对应的算法;
- 中序增强序列呢?
- 后序增强序列呢?

24. 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。

25. Greedy Huffman: Lower/Higher Frequency Lower/Higher**05J1**

- 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**。
- 下降的**数值**取决于哪些因素? 为什么?

26. 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**。

27. 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？

28. 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。

29. 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**。