算法设计与分析

作业（三）

姓　　名	熊恪峥
学　　号	22920202204622
日　　期	2022年3月7日
学　　院	信息学院
课程名称	算法设计与分析

作业（三）

目录

1 题4.2

算法1使用多数投票算法查找出现次数最多的元素。

算法 1 查找数组中出现次数最多的元素

Output: 元组(m, c)，其中m是出现最多的元素，c是出现次数

```
1:  procedure FindMost(A)
2:      c ← 0
3:      m ← 0
4:      for j ← 1 to n do
5:          if c == 0 then
6:              c ← 1
7:              m ← A[j]
8:          if A[j] == m then
9:              c ←c+1
10:         else
11:             c ←c-1
12:     c ← 0
13:     for j ← 1 to n do
14:         if A[j] == m then
15:             c ←c+1
        return (m,c)
```

该算法前10行使用投票法找到出现次数最多的元素，然后统计出现的次数。时间复杂度是$O(n)$，空间复杂度是$O(1)$。

2 题4.4

$Perm(m)$的循环不变式是

$$L_m = \text{每次执行}Perm(m)\text{前，}P[1 \ldots m-1]\text{是}m-1\text{个元素的一个排列。}$$

初始：当$m = 1$时，$P[1 \ldots 0]$中是0个元素的排列，即空集。

归纳：调用$Perm(m)$时，$P[1 \ldots m-1]$是$m-1$个元素的一个排列，算法$Perm(m)$将$P[j]$中放入第$m+1 \ldots n$中的一个元素，得到m个数字的排列。

终止：当$m = n$时，即$m = n+1$执行前，$P[1 \ldots n]$是n个元素的一个排列。

3 题4.6

初始：$GeneratingPerm2$第一次调用$Perm2(n)$时，$P[1 \ldots n]$有n个0。

归纳：若调用$Perm2(m)$时有m个0，$Perm2(m)$先在其中一个0的位置以m替代0，然后调用$Perm2(m-1)$，此时有$m-1$个0。

因此在每一次调用$Perm2(m)$时P中都有m个0。

$Perm(2)$中的for循环将每一个0的位置以m替代0然后调用$Perm2(m-1)$，因此$Perm2(m-1)$调用了m次。

4 题4.11

令

$$n = 2^m$$

则

$$S(m) = T(2^m) = 2T(2^{\frac{m}{2}}) + 1 = 2S(\frac{m}{2}) + 1$$

猜测$S(m) = O(m)$，则需要证$S(m) \leq cn - b$

当$m = 1$，$S(m) = 1 \leq c - b$，当$c \geq 1 + b$。

若$S(\frac{m}{2}) \leq c\frac{m}{2} - b$，则

$$\begin{aligned} S(m) &= 2S(\frac{m}{2}) + 1 \\ &= 2c\frac{m}{2} + 1 - 2b \\ &\leq cm - b \end{aligned}$$

当

$$b \geq 1$$

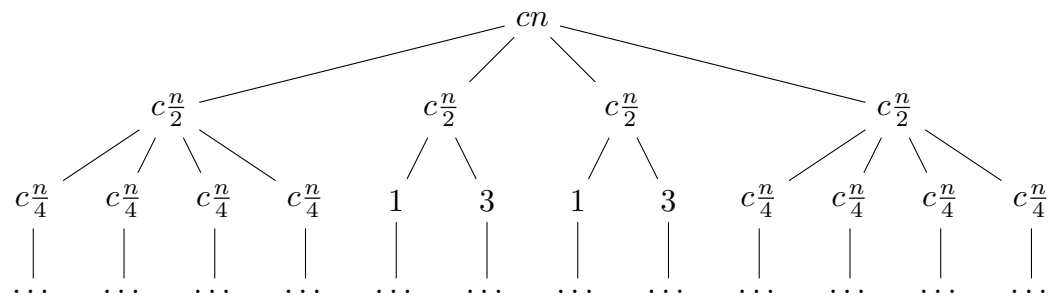
所以

$$S(m) \leq cm \iff S(m) = O(m)$$

则

$$T(n) = T(2^m) = S(m) = O(m) = O(\log n)$$

5 题4.12



由树状图可得$k = \log_2 n$

则

$$
\begin{aligned}
&cn + 4 \cdot \frac{1}{2}n + 16 \cdot \frac{1}{16}n + \dots \\
&= n \sum_{i=1}^{\log_2 n} 2^i \\
&= n\frac{2(1-n)}{1-2} \\
&= 2n^2 - 2n
\end{aligned}
$$

因此

$$T(n) = O(n^2)$$

证明： $T(n) \leq kn^2 - bn$

当$n = 1$，$T(n) = c \leq k - b$，当$k \geq c + b$

若$T(\frac{n}{2}) < k(\frac{n}{2})^2 - b\frac{n}{2}$

$$
\begin{aligned}
T(n) &\leq 4k(\frac{n}{2})^2 - b\frac{n}{2} + cn \\
&= kn^2 - (\frac{b}{2} - c)n \\
&\leq kn^2 - bn
\end{aligned}
$$

当

$$b \leq 2c$$

6 Leetcode 144

6.1 思路分析

求先序遍历，就先输出根节点的值，然后对左右子树分别进行先序遍历。运行结果如图6.2。代码如1，递归的终止条件是左右子树都是空。具体实现见附录：编程题代码。

6.2 运行结果

图 1: 运行结果

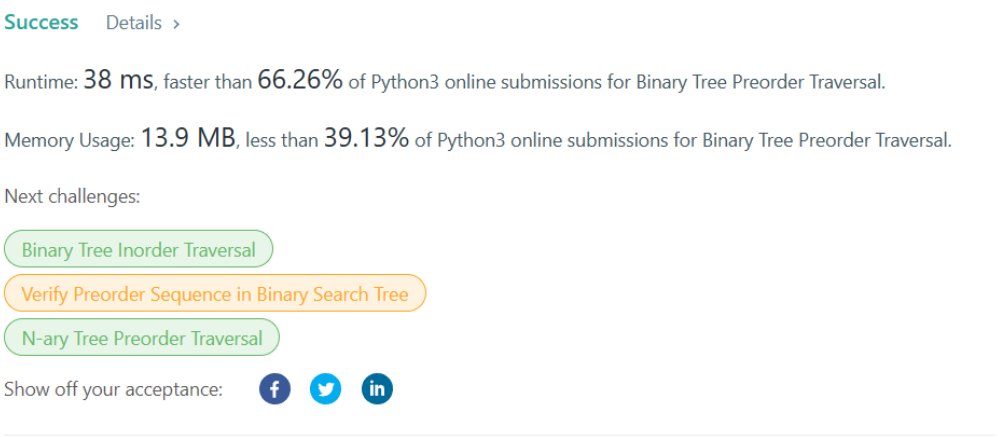
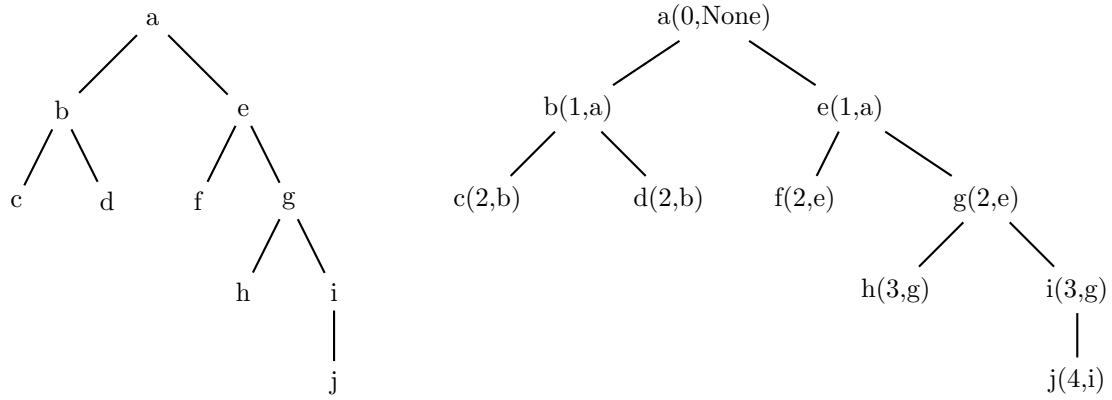

Success
Details ›
Runtime: 38 ms, faster than 66.26% of Python3 online submissions for Binary Tree Preorder Traversal.
Memory Usage: 13.9 MB, less than 39.13% of Python3 online submissions for Binary Tree Preorder Traversal.
Next challenges:
Binary Tree Inorder Traversal
Verify Preorder Sequence in Binary Search Tree
N-ary Tree Preorder Traversal
Show off your acceptance:

图 2: 一棵二叉树



(a) 原始二叉树 (b) 预处理后的二叉树

7 Leetcode 236

LCA是一种有各种算法的问题。常见的方法比如使用了并查集的Tarjan算法能较快地给出多组LCA查询的答案。但我写了一个暴力的算法，因为我想体验一下暴力算法的写法。

7.1 思路分析

首先假设有如图2(a)所示的二叉树，假设我们对点h和d求LCA，那么我们先对节点递归地标注高度，并求出节点的父节点，然后得到2(b)，之后按照如下方法进行迭代

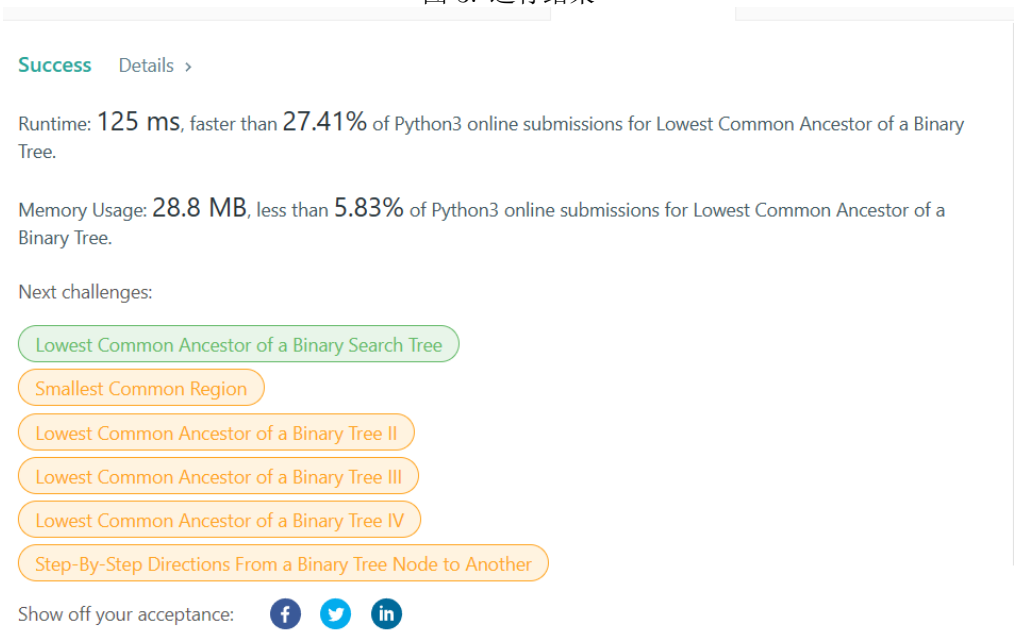
- 先让高度较低（高度值较大的节点沿父节点方向向上移动到两个节点处在同一层
- 两节点同时向上移动直到首次达到相等

然后就得到了LCA是a的答案。按照这个思路实现代码2，具体见附录：编程题代码。

7.2 运行结果

运行结果如图7.2。可以发现暴力算法确实运行较慢，而且并不比Tarjan算法好写，看来还是写Tarjan算法比较好。

图 3: 运行结果


Success
Details ›
Runtime: 125 ms, faster than 27.41% of Python3 online submissions for Lowest Common Ancestor of a Binary Tree.
Memory Usage: 28.8 MB, less than 5.83% of Python3 online submissions for Lowest Common Ancestor of a Binary Tree.
Next challenges:
Lowest Common Ancestor of a Binary Search Tree
Smallest Common Region
Lowest Common Ancestor of a Binary Tree II
Lowest Common Ancestor of a Binary Tree III
Lowest Common Ancestor of a Binary Tree IV
Step-By-Step Directions From a Binary Tree Node to Another
Show off your acceptance:

A 附录：编程题代码

代码 1: 求先序遍历

```
class Solution:
        def __init__(self):
                self. list = []

        def traversal(self, root: Optional[TreeNode]):
                if root is None:
                        return

                self. list.append(root.val)
                self.traversal(root.left)
                self.traversal(root.right)

        def preorderTraversal(self, root: Optional[TreeNode]) -> List[ int]:
                self.traversal(root)
                return self. list
```

代码 2: 求LCA

```
class Solution:
        def __init__(self):
                self.heights =  dict()

        def height(self, root: 'TreeNode', anc: Optional[TreeNode], h):
                root.ancestor = anc
                self.heights[root] = h

        if root.left is not None:
                self.height(root.left, root, h + 1)

        if root.right is not None:
                self.height(root.right, root, h + 1)

        def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') -> 'TreeNode'
            :
                self.height(root, None, 1)

                if self.heights[p] == self.heights[q]:
                        while p.val != q.val:
                        p = p.ancestor
                        q = q.ancestor

                return p

                (low, high, high_h) = (p, q, self.heights[q]) if self.heights[p] > self.heights[q]
                    else (q, p, self.heights[p])

                while self.heights[low] > high_h:
                        low = low.ancestor

                while not (low is None) and not (high is None) and low.val != high.val:
                        low = low.ancestor
                        high = high.ancestor
```

```
33
34                  return low
```