剑指offer(pytho版)-17-20

17树的子结构

题目：输入两棵二叉树A，B，判断B是不是A的子结构。（ps：我们约定空树不是任意一个树的子结构

思路：需要两个函数，一个用来判断是不是子结构，另外一个是用来进行初始化。

判断是否是子结构的时候，如果当前值相等，需要进行左右值是否相等的判断；如果当前值不等，则判断Root1的左右子树是否包含Root2

代码：

1. # -\*- coding:utf-8 -\*-
2. # class TreeNode:
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
7. **class** Solution:
8. **def** HasSubtree(self, pRoot1, pRoot2):
9. # write code here
10. **if** pRoot1 == None **or** pRoot2 == None:
11. **return** False
12. **return** self.isSubtree(pRoot1, pRoot2)
14. **def** isSubtree(self, p1, p2):
15. **if** p2 == None:
16. **return** True
17. **if** p1 == None:
18. **return** p1 == p2
19. res = False
20. **if** p1.val == p2.val:
21. res = self.isSubtree(p1.left, p2.left) **and** self.isSubtree(p1.right, p2.right)
22. **return** res **or** self.isSubtree(p1.left, p2) **or** self.isSubtree(p1.right, p2)

18二叉树的镜像

题目：操作给定的二叉树，将其变换为源二叉树的镜像。

思路：判断本身是否不为空，然后将左子树和右子树相互交换。然后判断是否有左子树，有的话递归函数；同理，判断是否有右子树，有的话递归函数。

代码：

1. # -\*- coding:utf-8 -\*-
2. # class TreeNode:
3. #     def \_\_init\_\_(self, x):
4. #         self.val = x
5. #         self.left = None
6. #         self.right = None
7. **class** Solution:
8. # 返回镜像树的根节点
9. **def** Mirror(self, root):
10. # write code here
11. **if** root==None:
12. **return**
13. root.right,root.left=root.left,root.right
14. self.Mirror(root.left)
15. self.Mirror(root.right)
16. **return** root

19顺时针打印矩阵

题目：输入一个矩阵，按照从外向里以顺时针的顺序依次打印出每一个数字，例如，如果输入如下4 X 4矩阵： 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 则依次打印出数字1,2,3,4,8,12,16,15,14,13,9,5,6,7,11,10

思路：这题看似简单，实际下手分析却越来越难。因为二维数组的下标变动太大。其实这题很像剑指Offer的problem4-二维数组的查找。分析过程很类似，有一个4\*4的二维矩阵，先依次打印第一行四个元素，打印完第一行以后第一行的数字就没有用了，可以不纳入考虑范围，接下来打印去掉第一行的3\*4矩阵的最后一列，这个时候最后一列只有三个元素，打印三个元素后。再考虑剩下的3\*3矩阵，倒序打印最后一行，然后再倒序打印剩下的2\*3矩阵的第一列。这个时候就完成了矩阵的最外圈顺时针打印，对于内层的打印方法一样。

代码：

1. # -\*- coding:utf-8 -\*-
2. **class** Solution:
3. # matrix类型为二维列表，需要返回列表
4. **def** printMatrix(self, matrix):
5. # write code here
6. out = []
7. # m, n分别表示行、列数
8. m = len(matrix)
9. **if** m == 0:
10. **return** out
11. n = len(matrix[0])
12. # x,y表示当前访问到第x行第y列
13. x = 0
14. y = -1
15. #　xnum， ynum限制当前访问方向上的元素个数
16. ynum = n
17. xnum = m - 1
18. **while** len(out) < m \* n **and** ynum >= 0 **and** xnum >= 0:
19. **for** i **in** range(ynum):
20. y += 1
21. out.append(matrix[x][y])
22. **for** i **in** range(xnum):
23. x += 1
24. out.append(matrix[x][y])
25. ynum -= 1
26. xnum -= 1
27. **if** xnum >= 0:
28. **for** i **in** range(ynum):
29. y -= 1
30. out.append(matrix[x][y])
31. ynum -= 1
32. **if** ynum >= 0:
33. **for** i **in** range(xnum):
34. x -= 1
35. out.append(matrix[x][y])
36. xnum -= 1
37. **return** out

20包含min函数的栈

题目：定义栈的数据结构，请在该类型中实现一个能够得到栈中所含最小元素的min函数（时间复杂度应为O（1））

思路：这里新建一个栈来存储当前的最小值：有更小的值入栈时，将其压入此栈，最小值为栈顶元素；当位于栈顶的最小值出栈后，此时栈顶的元素就顺应称为新的最小值。   
我看题目要求实现的top()函数定义什么作用，就偷懒用top()函数返回存储最小值的栈的栈顶元素了。

1. 代码：# -\*- coding:utf-8 -\*-
2. **import** sys
3. **class** Solution:
4. **def** \_\_init\_\_(self):
5. self.list=[]
6. self.minStack=[sys.maxsize]
7. **def** push(self, node):
8. # write code here
9. self.list.append(node)
10. **if** node<self.top():
11. self.minStack.append(node)
12. **def** pop(self):
13. # write code here
14. **if** self.list:
15. popnum=self.list.pop(-1)
16. **if** popnum==self.top():
17. self.minStack.pop(-1)
18. **return** popnum
19. **else**:
20. **return** None
21. **def** top(self):
22. # write code here
23. **if** self.minStack:
24. **return** self.minStack[-1]
25. **else**:
26. **return** None
27. **def** min(self):
28. # write code here
29. **return** self.top()