剑指offer（python版）61-66

61序列化二叉树

题目：请实现两个函数，分别用来序列化和反序列化二叉树

思路：把一棵二叉树按照某种遍历方式的结果以某种格式保存为**字符串**。需要注意的是，序列化二叉树的过程中，如果遇到空节点，需要以某种符号（这里用#）表示

序列化可以基于先序/中序/后序/按层等遍历方式进行，这里采用先序遍历的方式实现，字符串之间用 “，”隔开

反序列化二叉树：根据某种遍历顺序得到的序列化字符串，重构二叉树（就是按照同样的规则将一个序列还原为一颗二叉树）。具体思路是按前序遍历“根左右”的顺序，根节点位于其左右子节点的前面，即非空（#）的第一个节点是某子树的根节点，左右子节点在该根节点后，以空节点#为分隔符。

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** Serialize(self, root):
9. # write code here
10. valuelist=[]
11. self.preorder(root,valuelist)
12. **return** ','.join(map(str,valuelist))
13. **def** preorder(self,root,valuelist):
14. **if** **not** root:
15. valuelist.append('#')
16. **return** None
17. valuelist.append(root.val)
18. self.preorder(root.left,valuelist)
19. self.preorder(root.right,valuelist)
20. **def** Deserialize(self, s):
21. # write code here
22. valuelist=s.split(',')
23. root=self.preorders(valuelist)
24. **return** root
25. **def** preorders(self,valuelist):
26. **if** len(valuelist)==0 **or** valuelist[0]=='':
27. **return** None
28. **if** valuelist[0]=='#':
29. **del** valuelist[0]
30. **return** None
31. root=ListNode(int(valuelist[0]))
32. **del** valuelist[0]
33. root.left=self.preorders(valuelist)
34. root.right=self.preorders(valuelist)
35. **return** root

62二叉搜索树的第k个结点

题目：给定一棵二叉搜索树，请找出其中的第k小的结点。例如， （5，3，7，2，4，6，8）    中，按结点数值大小顺序第三小结点的值为4。

思路：二叉搜索树的中序遍历正好是一个递增的序列, 因此中序遍历的第K个结点就是二叉搜索树的第K个节点。

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. # 返回对应节点TreeNode
9. **def** KthNode(self, pRoot, k):
10. # write code here
11. **global** result
12. result=[]
13. **if** k==0:
14. **return** None
15. self.midnode(pRoot)
16. **if** k<=0 **or** len(result)<k:
17. **return** None
18. **else**:
19. **return** result[k-1]
20. **def** midnode(self,root):
21. **if** **not** root:
22. **return** None
23. self.midnode(root.left)
24. result.append(root)
25. self.midnode(root.right)

63数据流中的中位数

题目：如何得到一个数据流中的中位数？如果从数据流中读出奇数个数值，那么中位数就是所有数值排序之后位于中间的数值。如果从数据流中读出偶数个数值，那么中位数就是所有数值排序之后中间两个数的平均值。我们使用Insert()方法读取数据流，使用GetMedian()方法获取当前读取数据的中位数。

思路：insert函数是数据流的插入数据，然后排序，算索引就好了

代码：

1. # -\*- coding:utf-8 -\*-
2. **class** Solution:
3. **def** \_\_init\_\_(self):
4. self.nums=[]
5. **def** Insert(self, num):
6. # write code here
7. self.nums.append(num)
8. **def** GetMedian(self,nums):
9. # write code here
10. self.nums.sort()
11. **if** len(self.nums)%2==1:
12. **return** self.nums[(len(self.nums)-1)/2]
13. **else**:
14. **return** (self.nums[len(self.nums)/2]+self.nums[len(self.nums)/2-1])/2.0

64滑动窗口的最大值

题目：给定一个数组和滑动窗口的大小，找出所有滑动窗口里数值的最大值。例如，如果输入数组{2,3,4,2,6,2,5,1}及滑动窗口的大小3，那么一共存在6个滑动窗口，他们的最大值分别为{4,4,6,6,6,5}； 针对数组{2,3,4,2,6,2,5,1}的滑动窗口有以下6个： {[2,3,4],2,6,2,5,1}， {2,[3,4,2],6,2,5,1}， {2,3,[4,2,6],2,5,1}， {2,3,4,[2,6,2],5,1}， {2,3,4,2,[6,2,5],1}， {2,3,4,2,6,[2,5,1]}。

思路一：本题最简单的做法是定义一个求最大值的max函数，窗口每一次滑动求窗口内最大值，这种做法的时间复杂度为*O*((*n*−*k*−1)(*k*−1))=*O*(*nk*)O((n−k−1)(k−1))=O(nk)，*k*是滑动窗口大小。

代码：

1. # -\*- coding:utf-8 -\*-
2. **class** Solution:
3. **def** maxInWindows(self, num, size):
4. # write code here
5. maxqueue=[]
6. maxlist=[]
7. n=len(num)
8. **if** n==0 **or** size==0 **or** size>n:
9. **return** maxlist
10. **for** i **in** range(n):
11. **if** len(maxqueue)>0 **and** i-size>=maxqueue[0]:
12. maxqueue.pop(0)
13. **while** len(maxqueue)>0 **and** num[i]>num[maxqueue[-1]]:
14. maxqueue.pop()
15. maxqueue.append(i)
16. **if** i>=size-1:
17. maxlist.append(num[maxqueue[0]])
18. **return** maxlist

思路二：利用python的特性（算法题不太提倡）

代码：

1. **def** maxInWindows(self, num, size):
2. res = []
3. **if** len(num)>0 **and** size > 0:
4. endP = size
5. **while** endP <= len(num):
6. res.append(max(num[endP-size:endP]))
7. endP += 1
8. **return** res

65矩阵中的路径

题目：请设计一个函数，用来判断在一个矩阵中是否存在一条包含某字符串所有字符的路径。路径可以从矩阵中的任意一个格子开始，每一步可以在矩阵中向左，向右，向上，向下移动一个格子。如果一条路径经过了矩阵中的某一个格子，则之后不能再次进入这个格子。 例如 a b c e s f c s a d e e 这样的3 X 4 矩阵中包含一条字符串"bcced"的路径，但是矩阵中不包含"abcb"路径，因为字符串的第一个字符b占据了矩阵中的第一行第二个格子之后，路径不能再次进入该格子。

思路：本题是使用回溯法的典型题目。由于无法确定路径在字符矩阵中的起点，因此在矩阵中任选一格作为起点，进行搜索，判断是否存在所求路径。另外还需要与字符矩阵同样大小的标志数组，用来检查当前位置字符是否已经包含在路径中。矩阵中找到的第i个字符也一定对应着路径中第i个位置的字符。

代码：

1. # -\*- coding:utf-8 -\*-
2. **class** Solution:
3. **def** hasPath(self, matrix, rows, cols, path):
4. # write code here
5. assistMatrix = [True]\*rows\*cols
6. **for** i **in** range(rows):
7. **for** j **in** range(cols):
8. **if**(self.hasPathAtAStartPoint(matrix,rows,cols, i, j, path, assistMatrix)):
9. **return** True
10. **return** False
11. **def** hasPathAtAStartPoint(self, matrix, rows, cols, i, j, path, assistMatrix):
12. :param matrix:字符矩阵
13. :param rows:矩阵的行数
14. :param cols:矩阵的列数
15. :param path:需要寻找的路径
16. :param i:当前位置的横坐标(对应列数)
17. :param j:当前位置的纵坐标(对应行数)
18. :param visited:访问标志数组
19. :param pathlength:已经找到的路径长度
20. :return:是否存在路径
21. **if** **not** path:
22. **return** True
23. index = i\*cols+j
24. **if** i<0 **or** i>=rows **or** j<0 **or** j>=cols **or** matrix[index]!=path[0] **or** assistMatrix[index]==False:
25. **return** False
26. assistMatrix[index] = False
27. **if**(self.hasPathAtAStartPoint(matrix,rows,cols,i+1,j,path[1:],assistMatrix) **or** self.hasPathAtAStartPoint(matrix,rows,cols,i-1,j,path[1:],assistMatrix) **or** self.hasPathAtAStartPoint(matrix,rows,cols,i,j-1,path[1:],assistMatrix) **or** self.hasPathAtAStartPoint(matrix,rows,cols,i,j+1,path[1:],assistMatrix)):
28. **return** True
29. assistMatrix[index] = True
30. **return** False

66机器人的运动范围

题目：地上有一个m行和n列的方格。一个机器人从坐标0,0的格子开始移动，每一次只能向左，右，上，下四个方向移动一格，但是不能进入行坐标和列坐标的数位之和大于k的格子。 例如，当k为18时，机器人能够进入方格（35,37），因为3+5+3+7 = 18。但是，它不能进入方格（35,38），因为3+5+3+8 = 19。请问该机器人能够达到多少个格子

思路：回溯算法,当准备进入坐标(*i*,*j*)(i,j)是，通过检查坐标的数位来判断机器人能否进入。如果能进入的话，接着判断四个相邻的格子。

代码：

1. # -\*- coding:utf-8 -\*-
2. **class** Solution:
3. **def** movingCount(self, threshold, rows, cols):
4. # write code here
5. matrix = [[0 **for** i **in** range(cols)] **for** j **in** range(rows)]
6. count = self.findgrid(threshold, rows, cols, matrix, 0, 0)
7. **print**(matrix)
8. **return** count
9. **def** judge(self, threshold, i, j):
10. # sum(map(int, str(i) + str(j)))这一句简直精髓! 直接得到坐标位置的 位和! i,j是超过1位的数也可以完美解决!
11. **if** sum(map(int,str(i)+str(j))) <= threshold:
12. **return** True
13. **else**:
14. **return** False
15. **def** findgrid(self, threshold, rows, cols, matrix, i, j):
16. count = 0
17. **if** i<rows **and** j<cols **and** i>=0 **and** j>=0 **and** self.judge(threshold,i,j) **and** matrix[i][j]==0:
18. matrix[i][j] = 1# 表示已经走过了
19. count = 1 + self.findgrid(threshold, rows, cols, matrix, i, j+1) \
20. + self.findgrid(threshold, rows, cols, matrix, i, j-1) \
21. + self.findgrid(threshold, rows, cols, matrix, i+1, j) \
22. + self.findgrid(threshold, rows, cols, matrix, i-1, j)
23. **return** count

\+是什么意思？