1. Merge Sorted Array Easy

You are given two integer arrays nums1 and nums2, sorted in non-decreasing order, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

Merge nums1 and nums2 into a single array sorted in non-decreasing order.

The final sorted array should not be returned by the function, but instead be stored inside the array nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n.

Example 1:

Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3 Output: [1,2,2,3,5,6] Explanation: The arrays we are merging are [1,2,3] and [2,5,6]. The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1. Example 2:

Input: nums1 = [1], m = 1, nums2 = [], n = 0 Output: [1] Explanation: The arrays we are merging are [1] and []. The result of the merge is [1]. Example 3:

Input: nums1 = [0], m = 0, nums2 = [1], n = 1 Output: [1] Explanation: The arrays we are merging are [] and [1]. The result of the merge is [1]. Note that because m = 0, there are no elements in nums1. The 0 is only there to ensure the merge result can fit in nums1.

Constraints:

nums1.length == m + n nums2.length == n 0 <= m, n <= 200 1 <= m + n <= 200 -109 <= nums1[i], nums2[j] <= 109

In [ ]:
```
class Solution(object):
    def merge(self,num1,m,num2,n):
        #指向从 0 到 n的位置
        for i in range(n):
            #然后将 num2对应位置的数，写入num1 m位置的后面
            num1[i+m] = num2[i]
        #进行排序，默认是升序
        num1.sort()

num1 = [1,2,3,0,0,0]
num2 = [2,5,6]
m=3
n=3
Solution.merge(object,num1,m,num2,n)
num1
```

Out[ ]:
```
[1, 2, 2, 3, 5, 6]
```

In [ ]:
```
num1 = [1]
num2 = []
m=1
n=0
Solution.merge(object,num1,m,num2,n)
num1
```

Out[ ]:
```
[1]
```

In [ ]:
```
num1 = [0]
num2 = [1]
m=0
n=1
Solution.merge(object,num1,m,num2,n)
num1
```

Out[ ]:
```
[1]
```

1. Valid Palindrome Easy

A phrase is a palindrome if, after converting all uppercase letters into lowercase letters and removing all non-alphanumeric characters, it reads the same forward and backward. Alphanumeric characters include letters and numbers.

Given a string s, return true if it is a palindrome, or false otherwise.

Example 1:

Input: s = "A man, a plan, a canal: Panama" Output: true Explanation: "amanaplanacanalpanama" is a palindrome. Example 2:

Input: s = "race a car" Output: false Explanation: "raceacar" is not a palindrome. Example 3:

Input: s = " " Output: true Explanation: s is an empty string "" after removing non-alphanumeric characters. Since an empty string reads the same forward and backward, it is a palindrome.

Constraints:

1 <= s.length <= 2 * 105 s consists only of printable ASCII characters.

In [ ]:
```
#class 类，这个object,然后类里面的函数要带有self，去继承这个object
class Solution(object):
    def isPalindrome(self,s:str) -> bool:
        #isalnum() 查alpha & number，isnumeric() 查number,isalpha()查alpha
betic
        #filter(a,b)，对b进行a的判断，为true的就filter起来
        #这里是将string的字母和数字保留
        filtered_chars = filter(lambda ch: ch.isalnum(),s)
        #lower()转化小写
        #map将每一个在filtered_chars的字符匹配
        lowercase_filtered_chars = map(lambda ch: ch.lower(),filtered_ch
ars)

        filtered_chars_list = list(lowercase_filtered_chars)
        reversed_chars_list = filtered_chars_list[::-1]

        return filtered_chars_list == reversed_chars_list
```

In [ ]:
```
print(Solution.isPalindrome(object,"A man, a plan, a canal: Panama"))
print(Solution.isPalindrome(object,"race a car"))
print(Solution.isPalindrome(object," "))
```

```
True
False
True
```

1. Strobogrammatic Number Easy

Given a string num which represents an integer, return true if num is a strobogrammatic number.

A strobogrammatic number is a number that looks the same when rotated 180 degrees (looked at upside down).

Example 1:

Input: num = "69" Output: true Example 2:

Input: num = "88" Output: true Example 3:

Input: num = "962" Output: false

Constraints:

1 <= num.length <= 50 num consists of only digits. num does not contain any leading zeros except for zero itself.

In [ ]:
```
class Solution(object):
    def isStrobogrammatic(self,num:str) -> bool:
        #一个空的list，储存rotate之后的字符
        rotate_builder_list = []
        #直接指向string的每一个字符，因为是180度旋转，所以reverse string然后逐一判断
        for c in reversed(num):
            #这里首先要理解题目，0,1,2,3,4,5,6,7,8,9
            #180旋转后相同的 0,1,8
            #180旋转后相同的生成另一个数的 6,9
            #剩下的旋转后不能成为数字
            if c in {'0','1','8'}:
                rotate_builder_list.append(c)
            elif c == '6':
                rotate_builder_list.append('9')
            elif c == '9':
                rotate_builder_list.append('6')
            else:
                return False
        #将上面生成的每一个字符，将list的每个元素转化成字符，然后形成string
        rotated_string = "".join(rotate_builder_list)
        #旋转后的string和旋转前的string对比，是否相同，相同则为 strobogrammatic
        return rotated_string == num
```

In [ ]:
```
print(Solution.isStrobogrammatic(object,"69"))
print(Solution.isStrobogrammatic(object,"88"))
print(Solution.isStrobogrammatic(object,"962"))
```

```
True
True
False
```

1. Moving Average from Data Stream Easy

Given a stream of integers and a window size, calculate the moving average of all integers in the sliding window.

Implement the MovingAverage class:

MovingAverage(int size) Initializes the object with the size of the window size. double next(int val) Returns the moving average of the last size values of the stream.

Example 1:

Input ["MovingAverage", "next", "next", "next", "next"] [[3], [1], [10], [3], [5]] Output [null, 1.0, 5.5, 4.66667, 6.0]

Explanation MovingAverage movingAverage = new MovingAverage(3); movingAverage.next(1); // return 1.0 = 1 / 1 movingAverage.next(10); // return 5.5 = (1 + 10) / 2 movingAverage.next(3); // return 4.66667 = (1 + 10 + 3) / 3 movingAverage.next(5); // return 6.0 = (10 + 3 + 5) / 3

Constraints:

1 <= size <= 1000 -105 <= val <= 105 At most 104 calls will be made to next.

In [ ]:
```python
#因为我们用一下方式call class 和 class的函数
#定义class, def init, def next

class MovingAverage:
    #题目中提到了要initialize一个object with size of window
    #所以我们要写 def __init__, 参数为size
    def __init__(self, size:int):
        self.size = size
        #给一个初始的空list
        self.arr = []

    def next(self,val:int) -> float:
        if len(self.arr) == self.size:
            self.arr.pop(0)
        self.arr.append(val)
        return sum(self.arr)/len(self.arr)


    #这种方法也能解，但是不符合题目，因为在执行过程中我们扩大了window size
    # def next(self, val:int) -> float:
    #     #考虑下先做append，在做if判断，还是先做if判断，再做append
    #     self.arr.append(val)
    #     #判断现在arr的元素 和 我们的 window size是不是相等的
    #     if len(self.arr) > self.size:
    #         self.arr.pop(0)
    #     #求arr的和 除以 arr元素的个数
    #     return sum(self.arr)/len(self.arr)



# Your MovingAverage object will be instantiated and called as such:
# obj = MovingAverage(size)
# param_1 = obj.next(val)
```

In [ ]:
```python
obj = MovingAverage(3)
param_1 = obj.next(1)
print(param_1)
param_1 = obj.next(10)
print(param_1)
param_1 = obj.next(3)
print(param_1)
param_1 = obj.next(5)
print(param_1)
```

```
1.0
5.5
4.666666666666667
6.0
```

1. Valid Word Abbreviation Easy

A string can be abbreviated by replacing any number of non-adjacent, non-empty substrings with their lengths. The lengths should not have leading zeros.

For example, a string such as "substitution" could be abbreviated as (but not limited to):

"s10n" ("s ubstitutio n") "sub4u4" ("sub stit u tion") "12" ("substitution") "su3i1u2on" ("su bst i t u ti on") "substitution" (no substrings replaced) The following are not valid abbreviations:

"s55n" ("s ubsti tutio n", the replaced substrings are adjacent) "s010n" (has leading zeros) "s0ubstitution" (replaces an empty substring) Given a string word and an abbreviation abbr, return whether the string matches the given abbreviation.

A substring is a contiguous non-empty sequence of characters within a string.

Example 1:

Input: word = "internationalization", abbr = "i12iz4n" Output: true Explanation: The word "internationalization" can be abbreviated as "i12iz4n" ("i nternational iz atio n"). Example 2:

Input: word = "apple", abbr = "a2e" Output: false Explanation: The word "apple" cannot be abbreviated as "a2e".

Constraints:

1 <= word.length <= 20 word consists of only lowercase English letters. 1 <= abbr.length <= 10 abbr consists of lowercase English letters and digits. All the integers in abbr will fit in a 32-bit integer.

In [ ]:
```
#对比abbr 是否按照规则，缩写的 word
#两个对比，word 和 abbr，都是string的输入，考虑分别形成指针
#初始指针的位置

#指针判断的运行条件，指针只在word和abbr的长度内运行
    #指针运行后，如何执行对比；输入会存在两种情况，字母和数字
    #指针指向数字，只有abbr的指针会指向数字，
        # 开头数字不能为0，等于0 输出 false
        #指针获取两个参数，一个是只针对应的数字，一个指针对应的位置
        #abbr是数字，指针在len(abbr)的范围内
            #n 和 新的 p2
        #更新两个指针对应的位置
        #继续进行while，continue

    #指针指向字母
        #对比两个指针位置的字母是否一致
        #不一致出false
        #一致指针向下一位移动

    #两个指针分别增加一个位移位置

#return 判断最后的两个指针是不是都刚好在word 和 abbr的末尾，如果刚好在末尾，且完成了上述判断，那么就是对的

class Solution(object):
    def ValidWordAbbr(self,word:str,abbr:str) ->bool:
        p1,p2 = 0,0
        while p1< len(word) and p2< len(abbr):
            if abbr[p2].isdigit():
                if int(abbr[p2]) == 0: return False
                n , pp2 = abbr[p2],p2+1
                while pp2 < len(abbr) and abbr[pp2].isdigit():
                    n , pp2 = n+abbr[pp2], pp2+1
                p1 , p2 = p1+int(n), p2+len(n)
                continue
            if abbr[p2] != word[p1]: return False
            p1,p2 = p1+1,p2+1

        return p1 == len(word) and p2 == len(abbr)
```

In [ ]:
```
print(Solution.ValidWordAbbr(object,"internationalization","i12iz4n"))

print(Solution.ValidWordAbbr(object,"apple","a2e"))
```

```
True
False
```

1. Add Strings Easy

Given two non-negative integers, num1 and num2 represented as string, return the sum of num1 and num2 as a string.

You must solve the problem without using any built-in library for handling large integers (such as BigInteger). You must also not convert the inputs to integers directly.

Example 1:

Input: num1 = "11", num2 = "123" Output: "134" Example 2:

Input: num1 = "456", num2 = "77" Output: "533" Example 3:

Input: num1 = "0", num2 = "0" Output: "0"

Constraints:

1 <= num1.length, num2.length <= 104 num1 and num2 consist of only digits. num1 and num2 don't have any leading zeros except for the zero itself.

In [ ]:
```
#string 相加实际上是，两个指针对应的string转化成int相加
#存在进位和不进位的情况
#如果进位，返回的数值-10，同事需要记一个进位；carry = 1
#sum = d1 + d2 + carry
#sum = sum - 10 if sum >= 10 else sum
#return str(reversed(sum))


def AddString(num1:str,num2:str) -> str:
    p1, p2 = len(num1)-1 ,len(num2)-1

    carry = 0
    res = ""

    while p1 >=0 or p2 >=0 or carry >0:
        d1 = int(num1[p1]) if p1 >=0 else 0
        d2 = int(num2[p2]) if p2 >=0 else 0
        sum = d1 + d2 + carry
        carry = 1 if sum >= 10 else 0
        sum = sum -10 if sum >= 10 else sum
        res += str(sum)
        p1 -=1
        p2 -=1

    return res[::-1]
```

In [ ]:
```
print(AddString("456","77"))
print(AddString("0","0"))
```

```
533
0
```

1. Diameter of Binary Tree Easy

Given the root of a binary tree, return the length of the diameter of the tree.

The diameter of a binary tree is the length of the longest path between any two nodes in a tree. This path may or may not pass through the root.

The length of a path between two nodes is represented by the number of edges between them.

Example 1:

Input: root = [1,2,3,4,5] Output: 3 Explanation: 3 is the length of the path [4,2,1,3] or [5,2,1,3]. Example 2:

Input: root = [1,2] Output: 1

Constraints:

The number of nodes in the tree is in the range [1, 104]. -100 <= Node.val <= 100

In [ ]:
```
#定义tree的class
from typing import Optional

class TreeNode:
    def __init__(self,val = 0):
        self.val = val
        self.left = None
        self.right = None


class Solution:
    def DiameterOfBinaryTree(self,root: Optional[TreeNode]) -> int:
        diameter = 0

        def longest_path(Node):
            if not Node:
                return 0

            nonlocal diameter
            left_path = longest_path(Node.left)
            right_path = longest_path(Node.right)

            diameter = max(diameter,left_path+right_path)

            return max(left_path,right_path)+1

        longest_path(root)

        return diameter


#how to run tree test case

def fromlist(val):
    def create(it):
        value = next(it)
        return None if value is None else TreeNode(value)

    if not val:
        return None
    it = iter(val)
    root = TreeNode(next(it))
    nextlevel = [root]

    try:
        while nextlevel:
            level = nextlevel
            nextlevel = []
            for node in level:
                if node:
                    node.left = create(it)
                    node.right = create(it)
                    nextlevel += [node.left, node.right]
    except StopIteration:
        return root
    raise ValueError("Invalid List")
```

In [ ]:
```
root = [1,2,None,None,5]
s = Solution()

s.DiameterOfBinaryTree(fromlist(root))
```

Out[ ]: 2

1. Valid Palindrome II Easy

Given a string s, return true if the s can be palindrome after deleting at most one character from it.

Example 1:

Input: s = "aba" Output: true Example 2:

Input: s = "abca" Output: true Explanation: You could delete the character 'c'. Example 3:

Input: s = "abc" Output: false

Constraints:

1 <= s.length <= 105 s consists of lowercase English letters.

In [ ]:
```
class Solution:
    def ValidPalindrome(input: str) -> bool:
        isPalindrome = lambda s: s == s[::-1]
        leftpointer = 0
        rightpointer = len(input)-1

        while leftpointer <= rightpointer:
            if input[leftpointer] == input[rightpointer]:
                leftpointer +=1
                rightpointer -=1
            elif isPalindrome(input[leftpointer+1:rightpointer+1]) or is
Palindrome(input[leftpointer:rightpointer]) == True:
                print(f"you could remove {input[leftpointer+1:rightpoint
er+1]} or remove {input[leftpointer:rightpointer]} ")
                break
            else:
                return False

        return True

# s = "abcd"
# l=0, r=3, while True, s[0] == s[3] False, ispa(s[1:4]) False or ispa(s
[0:3]) False, False

# s = "abca"
# l=0, r=3, while True, s[0] == s[3] True,
# l=1, r=2, while True, s[1] == s[2] False, ispa(s[2:3]) True or ispa(s
[1:2]) True, True

# s = "abcdba"
# l=0, r=5, while True, s[0] == s[5] True,
# l=1, r=4, while True, s[1] == s[4] True,
# l=2, r=3, while True, s[2] == s[3] False, ispa(s[3:4]) True or ispa(s
[2:3]) True, True
```

In [ ]:
```
s1 = "aba"
s2 = "abca"
s3 = "abcfvcba"

print(Solution.ValidPalindrome(s1))
print(Solution.ValidPalindrome(s2))
print(Solution.ValidPalindrome(s3))
```

```
True
you could remove c or remove b
True
you could remove v or remove f
True
```

1. Toeplitz Matrix Easy

Given an m x n matrix, return true if the matrix is Toeplitz. Otherwise, return false.

A matrix is Toeplitz if every diagonal from top-left to bottom-right has the same elements.

Example 1:

Input: matrix = [[1,2,3,4],[5,1,2,3],[9,5,1,2]] Output: true Explanation: In the above grid, the diagonals are: "[9]", "[5, 5]", "[1, 1, 1]", "[2, 2, 2]", "[3, 3]", "[4]". In each diagonal all elements are the same, so the answer is True.
Example 2:

Input: matrix = [[1,2],[2,2]] Output: false Explanation: The diagonal "[1, 2]" has different elements.

Constraints:

m == matrix.length n == matrix[i].length 1 <= m, n <= 20 0 <= matrix[i][j] <= 99

Follow up:

What if the matrix is stored on disk, and the memory is limited such that you can only load at most one row of the matrix into the memory at once? What if the matrix is so large that you can only load up a partial row into the memory at once?

In [ ]:
```
"""
[1,2,3,4]
[5,1,2,3]
[9,5,1,2]
"""
from typing import List

class Solution:

    def ToeplitzM(matrix: List[List[int]]) -> bool:
        m = len(matrix)
        n = len(matrix[0])

        for i in range(m-1):
            for j in range(n-1):
                if matrix[i][j] != matrix[i+1][j+1]:
                    return False

        return True
```

In [ ]:
```
matrix = [[1,2,3,4],[5,1,2,3],[9,5,1,2]]
Solution.ToeplitzM(matrix)
```

Out[ ]: True

1. Range Sum of BST Easy

Given the root node of a binary search tree and two integers low and high, return the sum of values of all nodes with a value in the inclusive range [low, high].

Example 1:

Input: root = [10,5,15,3,7,null,18], low = 7, high = 15 Output: 32 Explanation: Nodes 7, 10, and 15 are in the range [7, 15]. 7 + 10 + 15 = 32. Example 2:

Input: root = [10,5,15,3,7,13,18,1,null,6], low = 6, high = 10 Output: 23 Explanation: Nodes 6, 7, and 10 are in the range [6, 10]. 6 + 7 + 10 = 23.

Constraints:

The number of nodes in the tree is in the range [1, 2 * 104]. 1 <= Node.val <= 105 1 <= low <= high <= 105 All Node.val are unique.

In [ ]:
```
#TreeNode
class TreeNode:
    def __init__(self, val = 0):
        self.val = val
        self.left = None
        self.right = None


#Solution
class Solution:
    def rangeBST(self, root: Optional[TreeNode], low:int, high:int) ->in
t:
        if not root:
            return 0

        res = 0
        if low <= root.val <= high:
            res += root.val
            res += self.rangeBST(root.left,low,high)
            res += self.rangeBST(root.right,low,high)
        elif root.val < low:
            res += self.rangeBST(root.right,low,high)
        elif root.val > high:
            res += self.rangeBST(root.left,low,high)
        return res


#Test
def fromlist(val):

    def createit(it):
        value = next(it)
        return None if value is None else TreeNode(value)

    if not val:
        return None

    it = iter(val)
    root = TreeNode(next(it))
    nextlevel = [root]

    try:
        while nextlevel:
            level = nextlevel
            nextlevel = []
            for node in level:
                if node:
                    node.left = createit(it)
                    node.right = createit(it)
                    nextlevel += [node.left,node.right]
    except StopIteration:
        return root
    raise ValueError("Invalid List")
```

In [ ]:
```
list = [10,5,15,3,7,None,18]
s = Solution()
s.rangeBST(fromlist(list),7,15)
```

Out[ ]: 32

In [ ]:
```

```