class Solution:

def isLongPressedName(self, name: str, typed: str) -> bool:

np, tp = 0,0

while np < len(name) and tp < len(typed):

if name[np] == typed[tp]:

np += 1

tp += 1

elif tp >= 1 and typed[tp] == typed[tp-1]:

tp += 1

else:

return False

if np != len(name):

return False

else:

while tp < len(typed):

if typed[tp] != typed[tp-1]:

return False

tp += 1

return True

class Solution:

def isLongPressedName(self, name: str, typed: str) -> bool:

tmp = []

tmp.append(name[0])

store = []

for i in range(1,len(name)):

if name[i] == tmp[-1]:

tmp.append(name[i])

else:

store.append((tmp[-1],len(tmp)))

tmp = []

tmp.append(name[i])

store.append((tmp[-1],len(tmp)))

tmp1 = []

tmp1.append(typed[0])

store1 = []

for i in range(1,len(typed)):

if typed[i] == tmp1[-1]:

tmp1.append(typed[i])

else:

store1.append((tmp1[-1],len(tmp1)))

tmp1 = []

tmp1.append(typed[i])

store1.append((tmp1[-1],len(tmp1)))

if len(store) != len(store1):

return False

for i in range(len(store)):

print(store[i][0])

if store[i][0] != store1[i][0] or store[i][1]>store1[i][1]:

return False

return True

def isLongPressedName(self, name: str, typed: str) -> bool:

n=len(name)

m=len(typed)

i=j=0

while(i<n and j<m):

if name[i]==typed[j]:

count1=count2=0

temp=name[i]

i+=1

while(i<n and name[i]==temp):

i+=1

count1+=1

temp=typed[j]

j+=1

while(j<m and typed[j]==temp):

count2+=1

j+=1

if count1>count2:

return False

else:

return False

if i>=n and j>=m:

return True

return False

class Solution:

def isLongPressedName(self, name: str, typed: str) -> bool:

# typed = "abcdd" name = "abcdefg"

if len(typed) < len(name):

return False

j = 0

# creating a freq map takes care of characters

# that are present in the name but not in typed

# eg: typed = abcdde name = abcdef

mp = {}

for ch in name:

mp[ch] = mp.get(ch, 0) + 1

# basically check,

# if a character at an index position in typed matches a correspoding index position in name

# if it does, increment index counter for name (j) and decrement frequency of the ch from the map

# or does it match a previous character in typed itself

# using these checks i'm basically trying to see if name can be reconstructed from type

for i in range(len(typed)):

if j < len(name) and typed[i] == name[j]:

j+=1

mp[typed[i]] -= 1

elif i > 0 and typed[i] == typed[i-1]:

continue

else:

return False

return sum(mp.values()) == 0

class Solution:

def isLongPressedName(self, name: str, typed: str) -> bool:

if len(typed) < len(name):

return False

j = 0

mp = {}

for ch in name:

mp[ch] = mp.get(ch, 0) + 1

for i in range(len(typed)):

if j < len(name) and typed[i] == name[j]:

j+=1

mp[typed[i]] -= 1

elif i > 0 and typed[i] == typed[i-1]:

continue

else:

return False

return sum(mp.values()) == 0

Coins solution:

class Solution:

def coinChange(self, coins: List[int], amount: int) -> int:

if not coins:

return 0

cache = {}

def dp(amount):

if amount in cache:

return cache[amount]

if amount == 0:

return 0

tmp = []

for coin in coins:

if amount - coin >= 0:

tmp.append(dp(amount - coin))

else:

tmp.append(float('inf'))

min\_coins = min(tmp) + 1

cache[amount] = min\_coins

return min\_coins

result = dp(amount)

if result != float('inf'):

return result

return -1

Another coins solution:

def coinChange(coins, amount):

mapOfOccuredNums = {}

coinsUsed = 0

if amount == 0: return coinsUsed

queue = [-1, amount]

while len(queue) > 1:

if queue[0] == -1:

coinsUsed += 1

queue.append(-1)

queue.pop(0)

else:

for coin in coins:

num = queue[0]-coin

if num == 0: return coinsUsed

if num > 0 and num not in mapOfOccuredNums:

mapOfOccuredNums[num] = None

queue.append(num)

queue.pop(0)

return -1

Q. Given an array of non-negative integers arr, your task is to count the number of pairs (i, j)  
such that i ≤ j and arr[i] + rev(arr[j]) = arr[j] + rev(arr[i])

Sample input and output

for A=>[1,20,2,11],  
output is 7  
2.for A [4,8,20,3,10,18,5,0,13,13]  
output is 21

def rev(n):

s = str(n)

return int(s[::-1])

def count\_pairs(arr):

hash\_map = {}

answer = 0

for i in arr:

if i - rev(i) not in hash\_map:

hash\_map[i-rev(i)] = 1

else:

hash\_map[i-rev(i)] += 1

for k,v in hash\_map.items():

answer += (v+1)\*v/2

return answer

# Driver code

arr = [1, 20, 2, 11]

#n = len(arr)

print(count\_pairs(arr))

def func(array):

result = list()

for index in range(len(array) - 2):

sum\_one = array[index + 1] + array[index + 2]

sum\_two = array[index] + array[index + 2]

sum\_three = array[index] + array[index + 1]

if array[index] < sum\_one and array[index + 1] < sum\_two and array[index + 2] < sum\_three:

result.append(1)

else:

result.append(0)

return result

**Q. Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the non-zero elements.**

class Solution:

def moveZeroes(self, nums: List[int]) -> None:

"""

Do not return anything, modify nums in-place instead.

"""

pos = 0

for i in range(len(nums)):

if nums[i] != 0:

nums[i], nums[pos] = nums[pos], nums[i]

pos += 1

return

Q. Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one sorted array.

The number of elements initialized in nums1 and nums2 are m and n respectively. You may assume that nums1 has a size equal to m + n such that it has enough space to hold additional elements from nums2.

**Input:** nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3

**Output:** [1,2,2,3,5,6]

class Solution:

def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:

if n == 0:

return nums1

else:

del nums1[-n:]

nums1.extend(nums2)

nums1.sort()

return nums1

another

def mergeArray(arr1, arr2):

if n == 0:

return arr1

else:

#del arr1[-n:]

arr1.extend(arr2)

arr1.sort()

return arr1

arr1 = [2, 4, 6, 8]

arr2 = [3, 5, 7, 9]

m = len(arr1)

n = len(arr2)

print(mergeArray(arr1,arr2))

Q. Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.\

**Input:** nums = [1,3,5,6], target = 5

**Output:** 2

class Solution:

def searchInsert(self, nums: List[int], target: int) -> int:

for i in nums:

if target in nums:

return nums.index(target)

else:

nums.append(target)

nums.sort()

return nums.index(target)

Q. Given an array nums containing n distinct numbers in the range [0, n], return the only number in the range that is missing from the array.

**Input:** nums = [3,0,1]

**Output:** 2

**Explanation:** n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it does not appear in nums.

**Input:** nums = [0,1]

**Output:** 2

**Explanation:** n = 2 since there are 2 numbers, so all numbers are in the range [0,2]. 2 is the missing number in the range since it does not appear in nums.

Brute

class Solution:

def missingNumber(self, nums: List[int]) -> int:

a = len(nums)

for i in range(0, a):

if i not in nums:

return i

else:

i += 1

return i

alternate soln:

class Solution:

def missingNumber(self, nums: List[int]) -> int:

ans = len(nums)

for i in range(ans):

ans += i - nums[i]

return ans

Q. Given a string s, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

**Input:** s = "A man, a plan, a canal: Panama"

**Output:** true

**Explanation:** "amanaplanacanalpanama" is a palindrome.

class Solution:

def isPalindrome(self, s: str) -> bool:

s = [i for i in s.lower() if i.isalnum()]

s = ''.join(s)

if str(s)[::-1] == str(s):

return True

else:

return False

#def palindrome\_check(s):

# return s[::-1] == s

#s= 'malayalm'

#print(palindrome\_check(s))

def isPalindrome(str):

# Run loop from 0 to len/2

for i in range(0, int(len(str)/2)):

if str[i] != str[len(str)-i-1]:

return False

return True

# main function

s = "malayalam"

ans = isPalindrome(s)

if (ans):

print("Yes")

else:

print("No")

Q. Write a function that reverses a string. The input string is given as an array of characters s.

**Input:** s = ["h","e","l","l","o"]

**Output:** ["o","l","l","e","h"]

class Solution:

def reverseString(self, s: List[str]) -> None:

left,right = 0,len(s)-1

while left < right:

s[left],s[right] = s[right],s[left]

left += 1

right -= 1

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

**Input:** num1 = "11", num2 = "123"

**Output:** "134"

class Solution:

def addStrings(self, num1: str, num2: str) -> str:

a = int(num1) + int(num2)

return str(a)

class Solution:

def addStrings(self, num1: str, num2: str) -> str:

s, t = num1, num2

a = int(s)+ int(t)

return str(a)

Q. Given a sequence of integers as an array, determine whether it is possible to obtain a strictly increasing sequence by removing no more than one element from the array.

*Note:* sequence a0, a1, ..., an is considered to be a strictly increasing if a0 < a1 < ... < an. Sequence containing only one element is also considered to be strictly increasing.

* For sequence = [1, 3, 2, 1], the output should be  
  almostIncreasingSequence(sequence) = false.

There is no one element in this array that can be removed in order to get a strictly increasing sequence.

* For sequence = [1, 3, 2], the output should be  
  almostIncreasingSequence(sequence) = true.

You can remove 3 from the array to get the strictly increasing sequence [1, 2]. Alternately, you can remove 2 to get the strictly increasing sequence [1, 3].

def check\_increasing(seq):

# This will check if it is increasing:

# If it is return -1 else return the element at which is it not increasing

for i in range(len(seq)-1):

if seq[i] >= seq[i+1]:

return i

return -1

def almostIncreasingSequence(sequence):

check = check\_increasing(sequence)

# List is increasing

if check == -1:

return True

# Check if removing an item will make a strictly increasing list

if check\_increasing(sequence[check-1:check] + sequence[check+1:]) == -1 or check\_increasing(sequence[check:check+1] + sequence[check+2:]) == -1:

return True

# If not return False, since more than 1 element needs to be removed

return False

Q. array summation: arrays inputs [-1,8,3] and tests:[3,7,2] return True because -1+3=2, 2 appears in the tests array and -1 and 3 appear in the input array. if any 2 of the numbers in the first array add up to any of the numbers in the second array else return False

def arraySum(arr, test):

for i in arr:

if arr[i] + arr[i + 1] in test:

return True

else:

return False

arr = [-1, 8, 3]

test = [3, 7, 2]

print(arraySum(arr, test))

Q. Given a string as an input, design a function that returns the string with the first letter to every word capitalized.

Example input: "problem example word"  
output --> "Problem Example Word"

def capitalizeString(string):

result = ' '.join(elem.capitalize() for elem in string.split())

return result

string = "problem example word"

print(capitalizeString(string))

1. Valid Parenthesis

Jpmc first round online test[summer:software engineering intern]-  
There were 2 coding questions:  
1.Tool List- Given strings of tools, startindex and target need to find minimum moves from target[kind of circular linkedlist]  
2.Binary number in Linkedlist[ <https://leetcode.com/problems/convert-binary-number-in-a-linked-list-to-integer/> ]

class Solution:

def getDecimalValue(self, head: ListNode) -> int:

total = 0

while(head):

total = total \* 2 + head.val

head = head.next

return total

class Solution:

def getDecimalValue(self, head: ListNode) -> int:

a = []

while(head!=None):

a.append(str(head.val))

head = head.next

n = int(''.join(a),2)

return (n

Time: 75 mins

I would say the coding challenge wasn't that bad. There was another post that helped me prepare for it.

1. <https://leetcode.com/problems/decode-ways/>

def numDecodings(self, s):

"""

:type s: str

:rtype: int

"""

dp = {}

def f(s):

if s in dp: return dp[s]

if not s:

return 1

res = 0

if len(s)>=1 and 1<=int(s[0])<=9:

res+=f(s[1:])

if len(s)>=2 and 10<=int(s[:2])<=26:

res+=f(s[2:])

dp[s] = res

return res

return f(s)

class Solution:

def numDecodings(s):

if not s or s[0] == '0':

return 0

dp = [0 for i in range(len(s)+1)]

dp[0] = 1

for i in range(len(s)):

if 1 <= int(s[i]) <= 9:

dp[i+1] = dp[i]

if i > 0 and 10 <= int(s[i-1:i+1]) <= 26:

dp[i+1] += dp[i-1]

return dp[-1]

#s = "12"

print( numDecodings("12"))

1. <https://leetcode.com/problems/happy-number/>

class Solution:

def isHappy(self, n: int) -> bool:

res = 0

if n == 1:

return True

if n == 4:

return False

for i in str(n):

res += int(i)\* int(i)

n = res

return Solution.isHappy(self,n)

class Solution(object):

def isHappy(self, n):

"""

:type n: int

:rtype: bool

"""

examples = []

while n:

if n == 1:

return True

n = self.helper(n)\*\*\*\*

if n in examples:

return False

examples.append(n)

def helper(self, n):

new = [int(i)\*\*2 for i in str(n)]

return sum(new)

Q. Given an integer n, break it into the sum of k **positive integers**, where k >= 2, and maximize the product of those integers.

Return *the maximum product you can get*.

**Example 1:**

**Input:** n = 2

**Output:** 1

**Explanation:** 2 = 1 + 1, 1 × 1 = 1.

**Example 2:**

**Input:** n = 10

**Output:** 36

**Explanation:** 10 = 3 + 3 + 4, 3 × 3 × 4 = 36.

class Solution:

def integerBreak(self, n: int) -> int:

# base Checks

if n == 2:

return 1

if n == 3:

return 2

count\_two = 0

count\_three = n // 3

if n % 3 == 1:

count\_three -= 1

count\_two += 2

elif n % 3 == 2:

count\_two += 1

product = 3 \*\* count\_three

if count\_two:

product \*= 2 \*\* count\_two

return product

Q. Design Linked list

class Node(object):

def \_\_init\_\_(self, val):

self.val = val

self.next = None

class MyLinkedList:

def \_\_init\_\_(self):

"""

Initialize your data structure here.

"""

self.head = None

self.tail = None

self.size = 0

def get(self, index: int) -> int:

"""

Get the value of the index-th node in the linked list. If the index is invalid, return -1.

"""

if self.head is None or index >= self.size:

# print "I am here"

return -1

current = self.head

for i in range(index):

current = current.next

return current.val

def addAtHead(self, val: int) -> None:

"""

Add a node of value val before the first element of the linked list. After the insertion, the new node will be the first node of the linked list.

"""

if not self.head:

self.head = Node(val)

self.tail = self.head

else:

new\_node = Node(val)

new\_node.next = self.head

self.head = new\_node

self.size += 1

def addAtTail(self, val: int) -> None:

"""

Append a node of value val to the last element of the linked list.

"""

if not self.head:

self.head = Node(val)

self.tail = self.head

else:

new\_node = Node(val)

self.tail.next = new\_node

self.tail = new\_node

self.size += 1

def addAtIndex(self, index: int, val: int) -> None:

"""

Add a node of value val before the index-th node in the linked list. If index equals to the length of linked list, the node will be appended to the end of linked list. If index is greater than the length, the node will not be inserted.

"""

if index > self.size:

return

if index == 0:

self.addAtHead(val)

else:

current = self.head

for i in range(index - 1):

current = current.next

new\_node = Node(val)

new\_node.next = current.next

current.next = new\_node

if new\_node.next is None:

self.tail = new\_node

self.size += 1

def deleteAtIndex(self, index: int) -> None:

"""

Delete the index-th node in the linked list, if the index is valid.

"""

if not self.head or index >= self.size:

return -1

if index == 0:

self.head = self.head.next

if not self.head:

self.tail = None

else:

current = self.head

for i in range(index - 1):

if current.next is None:

return

current = current.next

current.next = current.next.next

if current.next is None:

self.tail = current

self.size -= 1

# Your MyLinkedList object will be instantiated and called as such:

# obj = MyLinkedList()

# param\_1 = obj.get(index)

# obj.addAtHead(val)

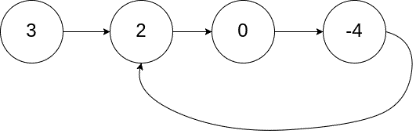
# obj.addAtTail(val)

# obj.addAtIndex(index,val)

# obj.deleteAtIndex(index)

Q. Given head, the head of a linked list, determine if the linked list has a cycle in it.

**Example 1:**



**Input:** head = [3,2,0,-4], pos = 1

**Output:** true

**Explanation:** There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).

# Definition for singly-linked list.

# class ListNode:

# def \_\_init\_\_(self, x):

# self.val = x

# self.next = None

class Solution:

def hasCycle(self, head: ListNode) -> bool:

s = set()

while head:

if head in s:

return True

s.add(head)

head = head.next

return False

**using two pointer method**

# Definition for singly-linked list.

# class ListNode:

# def \_\_init\_\_(self, x):

# self.val = x

# self.next = None

class Solution:

def hasCycle(self, head: ListNode) -> bool:

if head is None:

return False

slow = head

fast = head.next

while slow!= fast:

if fast is None or fast.next is None:

return False

slow = slow.next

fast = fast.next.next

return True

Q. Given a linked list, return the node where the cycle begins. If there is no cycle, return null.

**Example 1:**



**Input:** head = [3,2,0,-4], pos = 1

**Output:** tail connects to node index 1

**Explanation:** There is a cycle in the linked list, where tail connects to the second node.

# Definition for singly-linked list.

# class ListNode:

# def \_\_init\_\_(self, x):

# self.val = x

# self.next = None

class Solution:

def detectCycle(self, head: ListNode) -> ListNode:

s = set()

while head:

if head in s:

return head

s.add(head)

head = head.next

return None

Round 1: General Aptitude Test (On-site) - 45 mins (approx). Round 2: Problem Solving on Paper without panel. You will get set of problems on DS & Algorithms. Write the solution on paper. Round 3 (Continuing Round 2): Explain and optimize the code. a. String manipulation problem. b. Print Binary Tree in ZigZag manner. c. Reverse every k-nodes linkedlist. d. Provided an array find if sum of two elements in the array gives k O(n). Round 4: a. Spring related questions - DI & IOC. b. Print Inorder without recursion. c. Producer Consumer problem, wait & sleep, synchronized keyword and multiple questions. d. Provided sorted rotatated array find the smallest element. e. Difference between angular and React. f. Singleton design pattern and it's variations and synchronization issues. g. Factory design patterns.

Q. Given the heads of two singly linked-lists headA and headB, return the node at which the two lists intersect. If the two linked lists have no intersection at all, return null.

# Definition for singly-linked list.

# class ListNode:

# def \_\_init\_\_(self, x):

# self.val = x

# self.next = None

class Solution:

def getIntersectionNode(self, headA: ListNode, headB: ListNode) -> ListNode:

def get\_len(node):

l = 0

while node is not None:

node = node.next

l += 1

return l

len1 = get\_len(headA)

len2 = get\_len(headB)

for i in range(len1 - len2):

headA = headA.next

for i in range(len2 - len1):

headB = headB.next

while headA and headB:

if headA == headB:

return headA

headA = headA.next

headB = headB.next

return None

Q. Given the head of a linked list, remove the nth node from the end of the list and return its head.

**Input:** head = [1,2,3,4,5], n = 2

**Output:** [1,2,3,5]

**Example 2:**

**Input:** head = [1], n = 1

**Output:** []

**Example 3:**

**Input:** head = [1,2], n = 1

**Output:** [1]

# Definition for singly-linked list.

# class ListNode:

# def \_\_init\_\_(self, val=0, next=None):

# self.val = val

# self.next = next

class Solution:

def removeNthFromEnd(self, head: ListNode, n: int) -> ListNode:

fast, slow = head, head

for \_ in range(n): fast = fast.next

if not fast: return head.next

while fast.next: fast, slow = fast.next, slow.next

slow.next = slow.next.next

return head

Q. Given the head of a singly linked list, reverse the list, and return the reversed list.

**Input:** head = [1,2,3,4,5]

**Output:** [5,4,3,2,1]

**Input:** head = [1,2]

**Output:** [2,1]

**Input:** head = []

**Output:** []

# Definition for singly-linked list.

# class ListNode:

# def \_\_init\_\_(self, val=0, next=None):

# self.val = val

# self.next = next

class Solution:

def reverseList(self, head: ListNode) -> ListNode:

curr = head

prev = None

next = None

while curr!= None:

next = curr.next

curr.next = prev

prev = curr

curr = next

return prev

Q. Given the head of a linked list and an integer val, remove all the nodes of the linked list that has Node.val == val, and return *the new head*.

**Input:** head = [1,2,6,3,4,5,6], val = 6

**Output:** [1,2,3,4,5]

**Example 2:**

**Input:** head = [], val = 1

**Output:** []

**Example 3:**

**Input:** head = [7,7,7,7], val = 7

**Output:** []

# Definition for singly-linked list.

# class ListNode:

# def \_\_init\_\_(self, val=0, next=None):

# self.val = val

# self.next = next

class Solution:

def removeElements(self, head: ListNode, val: int) -> ListNode:

h = head

while head is not None and head.val==val:

head = head.next

while h is not None and h.next is not None:

if h.next.val == val:

h.next = h.next.next

else:

h = h.next

return head

Q. Given the head of a singly linked list, group all the nodes with odd indices together followed by the nodes with even indices, and return the reordered list.

**nput:** head = [1,2,3,4,5]

**Output:** [1,3,5,2,4]

**Input:** head = [2,1,3,5,6,4,7]

**Output:** [2,3,6,7,1,5,4]

# Definition for singly-linked list.

# class ListNode:

# def \_\_init\_\_(self, val=0, next=None):

# self.val = val

# self.next = next

class Solution:

def oddEvenList(self, head: ListNode) -> ListNode:

if not head or not head.next:

return head

even = head

odd = head.next

curr = head

while curr and curr.next:

nxt = curr.next

curr.next = curr.next.next

curr = nxt

curr = even

while curr and curr.next:

curr = curr.next

curr.next = odd

return head

Q. Given the root of a binary tree, return the preorder traversal of its nodes' values.

Recursive and iterative solutions:

# Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution:

def preorderTraversal(self, root: TreeNode) -> List[int]:

if root is None:

return []

return [root.val] + self.preorderTraversal(root.left) + self.preorderTraversal(root.right)

#2

class Solution:

def preorderTraversal(self, root: TreeNode) -> List[int]:

if root is None:

return []

stack =[root]

result = []

while stack != []:

root = stack.pop()

result.append(root.val)

if root.right is not None:

stack.append(root.right)

if root.left is not None:

stack.append(root.left)

return result

Q. Given the root of a binary tree, return the inorder traversal of its nodes' values.

Sol 1:

# Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution:

def inorderTraversal(self, root: TreeNode) -> List[int]:

if root is None:

return []

return self.inorderTraversal(root.left)+ [root.val] + self.inorderTraversal(root.right)

Sol2:

class Solution:

def inorderTraversal(self, root: TreeNode) -> List[int]:

stack =[]

result = []

while root is not None or stack != []:

while root is not None:

stack.append(root)

root = root.left

root = stack.pop()

result.append(root.val)

root = root.right

return result

Q. Given the root of a binary tree, return the postorder traversal of its nodes' values.

Sol1 recursion:

# Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution:

def postorderTraversal(self, root: TreeNode) -> List[int]:

if root is None:

return []

return self.postorderTraversal(root.left) + self.postorderTraversal(root.right) + [root.val]

Sol2: iterative

class Solution:

def postorderTraversal(self, root: TreeNode) -> List[int]:

if root is None:

return None

stack =[root]

result = []

while stack != []:

temp = stack.pop()

result.insert(0,temp.val)

if temp.left is not None:

stack.append(temp.left)

if temp.right is not None:

stack.append(temp.right)

return result

Sol3: reverse output soln

class Solution:

def postorderTraversal(self, root: TreeNode) -> List[int]:

if root is None:

return []

stack =[root]

result = []

while stack != []:

root = stack.pop()

result.append(root.val)

if root.left is not None:

stack.append(root.left)

if root.right is not None:

stack.append(root.right)

return result[::-1]

Q. Given the root of a binary tree, return the level order traversal of its nodes' values. (i.e., from left to right, level by level).

Sol:

class Solution:

def levelOrder(self, root: TreeNode) -> List[List[int]]:

if root is None:

return []

queue = [root]

next\_queue = []

level = []

result = []

while queue != []:

for root in queue:

level.append(root.val)

if root.left is not None:

next\_queue.append(root.left)

if root.right is not None:

next\_queue.append(root.right)

result.append(level)

level = []

queue = next\_queue

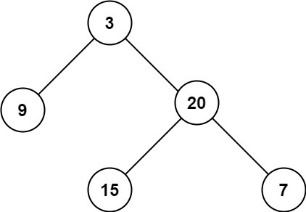
next\_queue = []

return result

Q. Given the root of a binary tree, return *its maximum depth*.

A binary tree's **maximum depth** is the number of nodes along the longest path from the root node down to the farthest leaf node.

**Example 1:**



**Input:** root = [3,9,20,null,null,15,7]

**Output:** 3

**Example 2:**

**Input:** root = [1,null,2]

**Output:** 2

**Example 3:**

**Input:** root = []

**Output:** 0

**Example 4:**

**Input:** root = [0]

**Output:** 1

Soln :

# Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution:

def maxDepth(self, root: TreeNode) -> int:

if root is None:

return 0

left = self.maxDepth(root.left)

right = self.maxDepth(root.right)

return 1+max(left, right)

Q. Given the root of a binary tree, *check whether it is a mirror of itself* (i.e., symmetric around its center).

**Example 1:**



**Input:** root = [1,2,2,3,4,4,3]

**Output:** true

Soln:

# Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution:

def isSymmetric(self, root: TreeNode) -> bool:

if root is None:

return 0

return self.checkMirror(root.left, root.right)

def checkMirror(self, root1, root2):

if not root1 and not root2:

return True

if not root1 and root2:

return False

if root1 and not root2:

return False

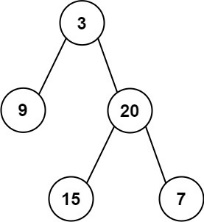
if root1 and root2 and root1.val != root2.val:

return False

return self.checkMirror(root1.left, root2.right) and self.checkMirror(root1.right, root2.left)

Q. Given two integer arrays inorder and postorder where inorder is the inorder traversal of a binary tree and postorder is the postorder traversal of the same tree, construct and return *the binary tree*.

**Example 1:**



**Input:** inorder = [9,3,15,20,7], postorder = [9,15,7,20,3]

**Output:** [3,9,20,null,null,15,7]

**Example 2:**

**Input:** inorder = [-1], postorder = [-1]

**Output:** [-1]

Soln:

class Solution:

def buildTree(self, inorder: List[int], postorder: List[int]) -> TreeNode:

order = {}

for i, e in enumerate(inorder):

order[e] = i

def tree(low, high):

if low > high or not postorder:

return

root = TreeNode(postorder.pop())

mid = order[root.val]

root.right = tree(mid+1, high) # Construct right tree

root.left = tree(low, mid-1) # Construct left tree

return root

return tree(0, len(postorder)-1)

Soln2:

class Solution:

def buildTree(self, inorder: List[int], postorder: List[int]) -> TreeNode:

if not postorder or not inorder:

return None

value = postorder[-1]

root = TreeNode(value)

index = inorder.index(value)

root.left = self.buildTree(inorder[:index], postorder[:index])

root.right = self.buildTree(inorder[index+1:], postorder[index: -1])

return root

Given two integer arrays preorder and inorder where preorder is the preorder traversal of a binary tree and inorder is the inorder traversal of the same tree, construct and return *the binary tree*.

**Example 1:**



**Input:** preorder = [3,9,20,15,7], inorder = [9,3,15,20,7]

**Output:** [3,9,20,null,null,15,7]

# Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution:

def buildTree(self, preorder: List[int], inorder: List[int]) -> TreeNode:

if not preorder or not inorder:

return None

value = preorder[0]

root = TreeNode(value)

index = inorder.index(value)

root.left = self.buildTree(preorder[index: index+1], inorder[:index])

root.right = self.buildTree( preorder[index+1:], inorder[index+1:])

return root

Q. You are given a license key represented as a string s that consists of only alphanumeric characters and dashes. The string is separated into n + 1 groups by n dashes. You are also given an integer k.

We want to reformat the string s such that each group contains exactly k characters, except for the first group, which could be shorter than k but still must contain at least one character. Furthermore, there must be a dash inserted between two groups, and you should convert all lowercase letters to uppercase.

Return the reformatted license key.

**Example 1:**

**Input:** s = "5F3Z-2e-9-w", k = 4

**Output:** "5F3Z-2E9W"

**Explanation:** The string s has been split into two parts, each part has 4 characters.

Note that the two extra dashes are not needed and can be removed.

**Example 2:**

**Input:** s = "2-5g-3-J", k = 2

**Output:** "2-5G-3J"

**Explanation:** The string s has been split into three parts, each part has 2 characters except the first part as it could be shorter as mentioned above.

Soln: class Solution:

def licenseKeyFormatting(self, s: str, k: int) -> str:

s\_new = []

for i in reversed(range(len(s))):

if s[i] == '-':

continue

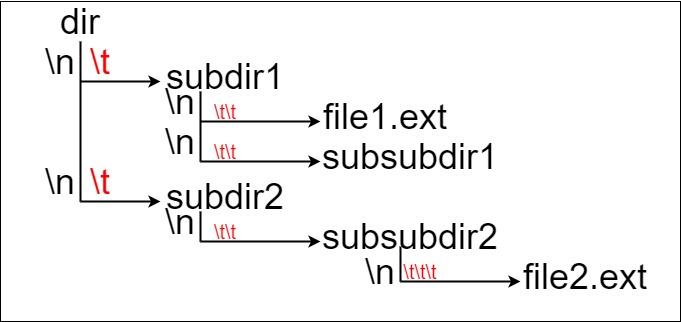
if len(s\_new) % (k+1) == k:

s\_new += '-'

s\_new += s[i].upper()

return ''.join(reversed(s\_new))

Q. Suppose we have a file system that stores both files and directories. An example of one system is represented in the following picture:



Here, we have dir as the only directory in the root. dir contains two subdirectories, subdir1 and subdir2. subdir1 contains a file file1.ext and subdirectory subsubdir1. subdir2 contains a subdirectory subsubdir2, which contains a file file2.ext.

In text form, it looks like this (with ⟶ representing the tab character):

dir

⟶ subdir1

⟶ ⟶ file1.ext

⟶ ⟶ subsubdir1

⟶ subdir2

⟶ ⟶ subsubdir2

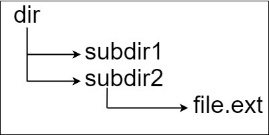
⟶ ⟶ ⟶ file2.ext

If we were to write this representation in code, it will look like this: "dir\n\tsubdir1\n\t\tfile1.ext\n\t\tsubsubdir1\n\tsubdir2\n\t\tsubsubdir2\n\t\t\tfile2.ext". Note that the '\n' and '\t' are the new-line and tab characters.

Every file and directory has a unique **absolute path** in the file system, which is the order of directories that must be opened to reach the file/directory itself, all concatenated by '/'s. Using the above example, the **absolute path** to file2.ext is "dir/subdir2/subsubdir2/file2.ext". Each directory name consists of letters, digits, and/or spaces. Each file name is of the form name.extension, where name and extension consist of letters, digits, and/or spaces.

Given a string input representing the file system in the explained format, return *the length of the****longest absolute path****to a****file****in the abstracted file system*. If there is no file in the system, return 0.

**Example 1:**

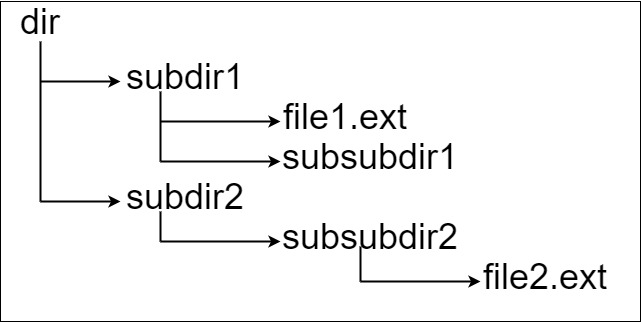


**Input:** input = "dir\n\tsubdir1\n\tsubdir2\n\t\tfile.ext"

**Output:** 20

**Explanation:** We have only one file, and the absolute path is "dir/subdir2/file.ext" of length 20.

**Example 2:**



**Input:** input = "dir\n\tsubdir1\n\t\tfile1.ext\n\t\tsubsubdir1\n\tsubdir2\n\t\tsubsubdir2\n\t\t\tfile2.ext"

**Output:** 32

**Explanation:** We have two files:

"dir/subdir1/file1.ext" of length 21

"dir/subdir2/subsubdir2/file2.ext" of length 32.

We return 32 since it is the longest absolute path to a file.

**Example 3:**

**Input:** input = "a"

**Output:** 0

**Explanation:** We do not have any files, just a single directory named "a".

**Example 4:**

**Input:** input = "file1.txt\nfile2.txt\nlongfile.txt"

**Output:** 12

**Explanation:** There are 3 files at the root directory.

Since the absolute path for anything at the root directory is just the name itself, the answer is "longfile.txt" with length 12.

Soln: class Solution:

def lengthLongestPath(self, input: str) -> int:

stack = [(-1,0)]

max\_len = 0

for p in input.split("\n"):

depth = p.count("\t")

p = p.replace("\t", "")

while stack and depth <= stack[-1][0]:

stack.pop()

if '.' not in p:

stack.append((depth, len(p)+ stack[-1][1]+1))

else:

max\_len = max(max\_len, len(p) + stack[-1][1])

return max\_len

Q. Given an alphanumeric string s, return *the****second largest****numerical digit that appears in*s*, or*-1*if it does not exist*.

An **alphanumeric**string is a string consisting of lowercase English letters and digits.

**Example 1:**

**Input:** s = "dfa12321afd"

**Output:** 2

**Explanation:** The digits that appear in s are [1, 2, 3]. The second largest digit is 2.

**Example 2:**

**Input:** s = "abc1111"

**Output:** -1

**Explanation:** The digits that appear in s are [1]. There is no second largest digit.

Soln:

class Solution:

def secondHighest(self, s: str) -> int:

s\_new = []

for i in s:

if i.isnumeric():

s\_new.append(i)

s\_new = sorted(set(s\_new))

if len(s\_new) > 1:

return s\_new[-2]

else:

return -1

Q. Write a method to check if a tree is a balance tree. And then explain the time complexity of your method.

Q.

#Input:

# "Lemons 3:1 Ducks",

# "Ducks 2:0 Baboons",

# "Lemons 1:1 Baboons",

# "Rays 1:0 Limes"

#Scores:

# Win: 3 points

# Tie: 1 point

# Loss: 0 points

#Expected output:

# Lemons: 4

# Ducks: 3

# Rays: 3

# Baboons: 1

# Limes: 0

Approach:

def string(arr):

team1\_count = 0

team2\_count = 0

for i in range(len(arr)):

team = arr.split(':')

print(team)

l1 = team[0]

l2 = team[1]

print(l1)

print(l2)

l = l1.split(' ')

m = l2.split(' ')

print(l)

print(m)

a = int(l[1])

b = int(m[0])

print(a)

print(b)

if a > b:

team1\_count += 3

team2\_count -= 0

if a < b:

team1\_count -= 0

team2\_count += 3

if a == b:

team1\_count += 1

team2\_count += 1

return team1\_count

arr = "Lemons 3:1 Duck"

print(string(arr))

Determine the basic blocks. Apply optimizations and specify the optimization techniques used. Describe some data flow analysis done by the compiler.

The performance of a program is slower than the same program before the backend of the compiler has changed. What direction would you look into this issue and solve it?

Various ways of testing a compiler

Question about binary search tree

How would you narrow down bugs, if you came in one day and found 10 new failures with your run?

Q. Given matrix, a rectangular matrix of integers, where each value represents the cost of the room, your task is to return the total sum of all rooms that are suitable for the CodeBots (ie: add up all the values that don't appear below a 0).

**Example**

* For
* matrix = [[0, 1, 1, 2],
* [0, 5, 0, 0],
* [2, 0, 3, 3]]

the output should be  
matrixElementsSum(matrix) = 9.



There are several haunted rooms, so we'll disregard them as well as any rooms beneath them. Thus, the answer is 1 + 5 + 1 + 2 = 9.

Soln: def matrixElementsSum(matrix):

    row\_len = len(matrix)

    col\_len = len(matrix[0])

    cols = []

    for c in range(col\_len):

        col = []

        for r in range(row\_len):

            col.append(matrix[r][c])

        cols.append(col)

    s = 0

    for arr in cols:

        for n in arr:

            if n == 0: break

            s += n

    return s

Q. Given an array of strings, return another array containing all of its longest strings.

Example

For inputArray = ["aba", "aa", "ad", "vcd", "aba"], the output should be  
allLongestStrings(inputArray) = ["aba", "vcd", "aba"]

Soln: def allLongestStrings(inputArray):

    new = []

    for i in inputArray:

        new.append(len(i))

    count = max(new)

    ar = []

    for i in inputArray:

        if len(i) == count:

            ar.append(i)

    return ar

Q. Given a fixed length array arr of integers, duplicate each occurrence of zero, shifting the remaining elements to the right.

Note that elements beyond the length of the original array are not written.

Do the above modifications to the input array **in place**, do not return anything from your function.

**Example 1:**

**Input:** [1,0,2,3,0,4,5,0]

**Output:** null

**Explanation:** After calling your function, the **input** array is modified to: [1,0,0,2,3,0,0,4]

**Example 2:**

**Input:** [1,2,3]

**Output:** null

**Explanation:** After calling your function, the **input** array is modified to: [1,2,3]

Sol: class Solution:

def duplicateZeros(self, arr: List[int]) -> None:

"""

Do not return anything, modify arr in-place instead.

"""

i = 0

l = len(arr)

while i<l:

if arr[i] == 0:

arr.pop(-1)

arr.insert(i+1, 0)

i+=1

i+=1

Q. Given two sorted integer arrays nums1 and nums2, merge nums2 into nums1 as one sorted array.

The number of elements initialized in nums1 and nums2 are m and n respectively. You may assume that nums1 has a size equal to m + n such that it has enough space to hold additional elements from nums2.

**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]

**Example 2:**

**Input:** nums1 = [1], m = 1, nums2 = [], n = 0

**Output:** [1]

Sol:

class Solution:

def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:

if n == 0:

return nums1

else:

del nums1[-n:]

nums1.extend(nums2)

nums1.sort()

return nums1

Q. Given an array nums and a value val, remove all instances of that value [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) and return the new length.

Sol: class Solution:

def removeElement(self, nums: List[int], val: int) -> int:

i=0

while i< len(nums):

if nums[i]==val:

nums.pop(i)

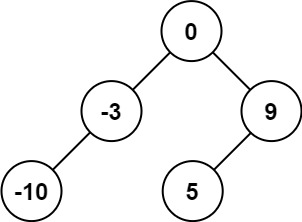
else:

i+=1

Q. Given an integer array nums where the elements are sorted in **ascending order**, convert *it to a****height-balanced****binary search tree*.

A **height-balanced** binary tree is a binary tree in which the depth of the two subtrees of every node never differs by more than one.

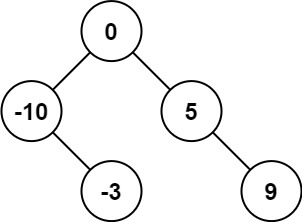
**Example 1:**



**Input:** nums = [-10,-3,0,5,9]

**Output:** [0,-3,9,-10,null,5]

**Explanation:** [0,-10,5,null,-3,null,9] is also accepted:



Sol: # Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution:

def sortedArrayToBST(self, nums: List[int]) -> TreeNode:

if len(nums)==0:

return

mid = len(nums)//2

root = TreeNode(nums[mid])

root.left = self.sortedArrayToBST(nums[:mid])

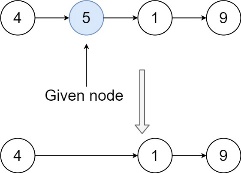
root.right = self.sortedArrayToBST(nums[mid+1:])

return root

Q. Write a function to **delete a node** in a singly-linked list. You will **not** be given access to the head of the list, instead you will be given access to **the node to be deleted** directly.

It is **guaranteed** that the node to be deleted is **not a tail node** in the list.

**Example 1:**



**Input:** head = [4,5,1,9], node = 5

**Output:** [4,1,9]

**Explanation:** You are given the second node with value 5, the linked list should become 4 -> 1 -> 9 after calling your function.

Sol: # Definition for singly-linked list.

# class ListNode:

# def \_\_init\_\_(self, x):

# self.val = x

# self.next = None

class Solution: updated current node value to next node value and point current node towards the next of next node

def deleteNode(self, node):

"""

:type node: ListNode

:rtype: void Do not return anything, modify node in-place instead.

"""

node.val = node.next.val

node.next = node.next.next

Q. Given an array, rotate the array to the right by k steps, where k is non-negative.

**Example 1:**

**Input:** nums = [1,2,3,4,5,6,7], k = 3

**Output:** [5,6,7,1,2,3,4]

**Explanation:**

rotate 1 steps to the right: [7,1,2,3,4,5,6]

rotate 2 steps to the right: [6,7,1,2,3,4,5]

rotate 3 steps to the right: [5,6,7,1,2,3,4]

Sol:

class Solution:

def rotate(self, nums: List[int], k: int) -> None:

"""

Do not return anything, modify nums in-place instead.

"""

n = len(nums)

k = k%n

if k ==0:

return

nums[:]= nums[n-k:n] + nums[:n-k]

Q. Given an integer array nums, return true if any value appears **at least twice** in the array, and return false if every element is distinct.

**Example 1:**

**Input:** nums = [1,2,3,1]

**Output:** true

**Example 2:**

**Input:** nums = [1,2,3,4]

**Output:** false

Sol: class Solution:

def containsDuplicate(self, nums: List[int]) -> bool:

nums.sort()

for i in range(0,len(nums)-1):

if nums[i] == nums[i+1]:

return True

return False

**using set**

class Solution:

def containsDuplicate(self, nums: List[int]) -> bool:

new= set()

for n in nums:

if n not in new:

new.add(n)

else:

return True

return False

Q. Given a sorted array *nums*, remove the duplicates [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) such that each element appears only *once* and returns the new length.

Do not allocate extra space for another array, you must do this by **modifying the input array**[**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm) with O(1) extra memory.

**Input:** nums = [1,1,2]

**Output:** 2, nums = [1,2]

**Explanation:** Your function should return length = **2**, with the first two elements of *nums* being **1** and **2** respectively. It doesn't matter what you leave beyond the returned length.

Sol:

Using sets

class Solution:

def removeDuplicates(self, nums: List[int]) -> int:

num = set(nums)

nums.clear()

for i in num:

nums.append(i)

nums.sort()

return len(nums)

return (nums)

Q. You are given an array prices where prices[i] is the price of a given stock on the ith day.

Find the maximum profit you can achieve. You may complete as many transactions as you like (i.e., buy one and sell one share of the stock multiple times).

**Note:** You may not engage in multiple transactions simultaneously (i.e., you must sell the stock before you buy again).

**Example 1:**

**Input:** prices = [7,1,5,3,6,4]

**Output:** 7

**Explanation:** Buy on day 2 (price = 1) and sell on day 3 (price = 5), profit = 5-1 = 4.

Then buy on day 4 (price = 3) and sell on day 5 (price = 6), profit = 6-3 = 3.

Sol:

class Solution:

def maxProfit(self, prices: List[int]) -> int:

minima = float('inf')

maxima = 0

for p in prices:

if p < minima:

minima = p

else:

maxima += p - minima

minima = p

return maxima

Sometimes this works:

class Solution:

def maxProfit(self, prices: List[int]) -> int:

minima = float('inf')

maxima = 0

for p in prices:

minima = min(minima,p)

maxima = max(maxima, p-minima)

return maxima

Dynamic programming

class Solution:

def maxProfit(self, prices: List[int]) -> int:

n = len(prices)

maxima,current=0,0

for i in range(1,n):

current = max(current + prices[i] - prices[i-1], 0)

maxima = max(current, maxima)

return maxima

Q. hackerrank buy sell stock

def stockmax(p):

    ind\_max = p.index(max(p))

    inv = sum(p[:ind\_max])

    pf = len(p[:ind\_max])\*p[ind\_max] - inv

    if len(p[ind\_max+1:]) > 0:

        pf += stockmax(p[ind\_max+1:])

    return pf

    profit = stockmax(p)

Q. Given a **non-empty** array of integers nums, every element appears *twice* except for one. Find that single one.

You must implement a solution with a linear runtime complexity and use only constant extra space.

**Example 1:**

**Input:** nums = [2,2,1]

**Output:** 1

Sol:

class Solution:

def singleNumber(self, nums: List[int]) -> int:

for i in nums:

if nums.count(i)==1:

return i

Q. Given two integer arrays nums1 and nums2, return *an array of their intersection*. Each element in the result must appear as many times as it shows in both arrays and you may return the result in **any order**.

**Example 1:**

**Input:** nums1 = [1,2,2,1], nums2 = [2,2]

**Output:** [2,2]

Sol: class Solution:

def intersect(self, nums1: List[int], nums2: List[int]) -> List[int]:

res=[]

for i in nums1:

if i in nums2:

res.append(i)

nums2.remove(i)

return res

Q. Given a **non-empty** array of decimal digits representing a non-negative integer, increment one to the integer.

The digits are stored such that the most significant digit is at the head of the list, and each element in the array contains a single digit.

You may assume the integer does not contain any leading zero, except the number 0 itself.

**Example 1:**

**Input:** digits = [1,2,3]

**Output:** [1,2,4]

**Explanation:** The array represents the integer 123.

Sol:

Solution 1

class Solution:

def plusOne(self, digits: List[int]) -> List[int]:

a = ''.join(map(str,digits))

b = int(a)+1

c=str(b)

return list(map(int,c))

Sol2

class Solution:

def plusOne(self, digits: List[int]) -> List[int]:

str1 = "".join([str(d) for d in digits])

plusOne = int(str1)+1

return [int(d) for d in str(plusOne)]

Q. Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the non-zero elements.

**Note** that you must do this in-place without making a copy of the array.

**Example 1:**

**Input:** nums = [0,1,0,3,12]

**Output:** [1,3,12,0,0]

Sol:

class Solution:

def moveZeroes(self, nums: List[int]) -> None:

"""

Do not return anything, modify nums in-place instead.

"""

l = 0

for i in range(len(nums)):

if nums[i]!=0:

nums[l], nums[i] = nums[i], nums[l]

l+=1

return nums

Q. Two sum hash

Sol:

class Solution:

def twoSum(self, nums: List[int], target: int) -> List[int]:

hashmp={}

for i, n in enumerate(nums):

diff= target-n

if diff not in hashmp:

hashmp[n]=i

else:

return [hashmp[diff], i]

Q. Write a function that reverses a string. The input string is given as an array of characters s.

 Do not allocate extra space for another array. You must do this by modifying the input array [in-place](https://en.wikipedia.org/wiki/In-place_algorithm) with O(1) extra memory.

**Example 1:**

**Input:** s = ["h","e","l","l","o"]

**Output:** ["o","l","l","e","h"]

Sol: class Solution:

def reverseString(self, s: List[str]) -> None:

"""

Do not return anything, modify s in-place instead.

"""

i=0

j=len(s)-1

while i<=j:

s[i], s[j]= s[j],s[i]

i+=1

j-=1

return s

Q. Given a signed 32-bit integer x, return x*with its digits reversed*. If reversing x causes the value to go outside the signed 32-bit integer range [-231, 231 - 1], then return 0.

**Assume the environment does not allow you to store 64-bit integers (signed or unsigned).**

**Example 1:**

**Input:** x = 123

**Output:** 321

**Example 2:**

**Input:** x = -123

**Output:** -321

Sol: class Solution:

def reverse(self, x: int) -> int:

if x>0:

a=int(str(x)[::-1])

if x<=0:

a= -1 \* int(str(x\*-1)[::-1])

mina = -2\*\*31

maxa = 2\*\*31 - 1

if a not in range(mina,maxa):

return 0

else:

return a

Q. Given a string s, return *the first non-repeating character in it and return its index*. If it does not exist, return -1.

**Example 1:**

**Input:** s = "leetcode"

**Output:** 0

**Example 2:**

**Input:** s = "loveleetcode"

**Output:** 2

Sol:

1 Using Counter

class Solution:

def firstUniqChar(self, s: str) -> int:

c=Counter(s)

for i in s:

if c[i]==1:

return s.index(i)

return -1

using dictionary

class Solution:

def firstUniqChar(self, s: str) -> int:

dicts={}

count=[]

#create a dictionary with letters in the string and it's count

for i in s:

if i in dicts:

dicts[i] +=1

else:

dicts[i] =1

#print(dicts)

#get the index of letter which only counted once i.e. first unique character

for i in range(len(s)):

if dicts[s[i]] == 1:

return(i)

#if there are no unique character then return -1

return ('-1')

List comprehension

class Solution(object):

def firstUniqChar(self, s):

"""

:type s: str

:rtype: int

"""

if len(s) > 0:

idx = [s.index(x) for x in 'abcdefghijklmnopqrstuvwxyz' if s.count(x) == 1]

return min(idx) if len(idx) > 0 else -1

return -1

Q. Given two strings s and t, return true *if* t *is an anagram of* s*, and* false *otherwise*.

**Example 1:**

**Input:** s = "anagram", t = "nagaram"

**Output:** true

**Example 2:**

**Input:** s = "rat", t = "car"

**Output:** false

Sol:

class Solution:

def isAnagram(self, s: str, t: str) -> bool:

new\_s= ''.join(sorted(s))

new\_t= ''.join(sorted(t))

if new\_s == new\_t:

return True

else:

return False

using tuple

class Solution:

def isAnagram(self, s: str, t: str) -> bool:

if tuple(sorted(s)) == tuple(sorted(t)):

return True

else:

return False

using hashmaps:

def isAnagram(self, s, t):

return collections.Counter(s) == collections.Counter(t)

Q. Given a string s, determine if it is a palindrome, considering only alphanumeric characters and ignoring cases.

**Example 1:**

**Input:** s = "A man, a plan, a canal: Panama"

**Output:** true

**Explanation:** "amanaplanacanalpanama" is a palindrome.

Sol:

class Solution:

def isPalindrome(self, s: str) -> bool:

s = re.sub(r'[\W\_]+', '', s).lower()

print(s)

return str(s)==str(s)[::-1]

**using isalnum()**

class Solution:

def isPalindrome(self, s: str) -> bool:

s= [e for e in s.lower() if e.isalnum()]

return s==s[::-1]

Q. **String to Integer (atoi)**

Implement the myAtoi(string s) function, which converts a string to a 32-bit signed integer (similar to C/C++'s atoi function).

The algorithm for myAtoi(string s) is as follows:

1. Read in and ignore any leading whitespace.
2. Check if the next character (if not already at the end of the string) is '-' or '+'. Read this character in if it is either. This determines if the final result is negative or positive respectively. Assume the result is positive if neither is present.
3. Read in next the characters until the next non-digit charcter or the end of the input is reached. The rest of the string is ignored.
4. Convert these digits into an integer (i.e. "123" -> 123, "0032" -> 32). If no digits were read, then the integer is 0. Change the sign as necessary (from step 2).
5. If the integer is out of the 32-bit signed integer range [-231, 231 - 1], then clamp the integer so that it remains in the range. Specifically, integers less than -231 should be clamped to -231, and integers greater than 231 - 1 should be clamped to 231 - 1.
6. Return the integer as the final result.
7. **Input:** s = "42"
8. **Output:** 42
9. **Explanation:** The underlined characters are what is read in, the caret is the current reader position.
10. Step 1: "42" (no characters read because there is no leading whitespace)
11. ^
12. Step 2: "42" (no characters read because there is neither a '-' nor '+')
13. ^
14. Step 3: "42" ("42" is read in)
15. ^
16. The parsed integer is 42.
17. Since 42 is in the range [-231, 231 - 1], the final result is 42.

Sol:

class Solution:

def myAtoi(self, s: str) -> int:

i=0

number=0

negative= False

length=len(s)

if length>0:

while i<=length-1 and s[i] is ' ':

i+=1

if i<=length-1:

if s[i] is '-':

negative=True

i+=1

elif s[i] is '+':

i+=1

while i<=length-1 and s[i]=='0':

i+=1

if i< length:

for j in range(i,len(s),1):

if not s[j].isdigit():

break

else:

number=number\*10 +int(s[j])

number = -number if (negative==True) else number

if number> 2147483647:

number = 2147483647

if number< -2147483648:

number = -2147483648

return number

Q. Implement [strStr()](http://www.cplusplus.com/reference/cstring/strstr/" \t "_blank).

Return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

**Example 1:**

**Input:** haystack = "hello", needle = "ll"

**Output:** 2

**Example 2:**

**Input:** haystack = "aaaaa", needle = "bba"

**Output:** -1

Sol: class Solution:

def strStr(self, haystack: str, needle: str) -> int:

if needle not in haystack:

return -1

else:

return haystack.index(needle)

class Solution:

def strStr(self, haystack: str, needle: str) -> int:

if needle == "":

return 0

if needle in haystack:

return haystack.find(needle)

else:

return -1

class Solution:

def strStr(self, haystack: str, needle: str) -> int:

needle\_len = len(needle)

for i in range(len(haystack) - needle\_len + 1):

if haystack[i:i + needle\_len] == needle:

return i

return -1

Q. Write a function to find the longest common prefix string amongst an array of strings.

If there is no common prefix, return an empty string "".

**Example 1:**

**Input:** strs = ["flower","flow","flight"]

**Output:** "fl"

Sol: class Solution:

def longestCommonPrefix(self, strs: List[str]) -> str:

pre=''

if len(strs)==0:

return pre

res=''

strs=sorted(strs)

for i in strs[0]:

if strs[-1].startswith(res+i):

res+=i

else:

break

return res

Q. Merge two sorted linked lists and return it as a **sorted** list. The list should be made by splicing together the nodes of the first two lists.

**nput:** l1 = [1,2,4], l2 = [1,3,4]

**Output:** [1,1,2,3,4,4]

**Example 2:**

**Input:** l1 = [], l2 = []

**Output:** []

**Example 3:**

**Input:** l1 = [], l2 = [0]

**Output:** [0]

Sol:

# Definition for singly-linked list.

# class ListNode:

# def \_\_init\_\_(self, val=0, next=None):

# self.val = val

# self.next = next

class Solution:

def mergeTwoLists(self, l1: ListNode, l2: ListNode) -> ListNode:

# if not l1 or l2:

# return l1 or l2

c1=[]

c2=[]

while l1:

c1+=[l1.val]

l1=l1.next

while l2:

c2+=[l2.val]

l2=l2.next

s = sorted(c1+c2)

point = head = ListNode(0)

for k in s:

point.next= ListNode(k)

point = point.next

return head.next

Q. Given the root of a binary tree, determine if it is a valid binary search tree (BST).

A **valid BST** is defined as follows:

* The left subtree of a node contains only nodes with keys **less than** the node's key.
* The right subtree of a node contains only nodes with keys **greater than** the node's key.
* Both the left and right subtrees must also be binary search trees
* Diagram

  Description automatically generated
* **Input:** root = [2,1,3]
* **Output:** true

Sol:

# Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution:

def isValidBST(self, root: TreeNode) -> bool:

if root is None:

return None

stack=[]

prev=None

while (root!=None or len(stack)!=0):

while root!=None:

stack.append(root)

root=root.left

popped = stack.pop()

if prev!=None and prev>=popped.val:

return False

prev = popped.val

root = popped.right

return True

Q. First bad version

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have n versions [1, 2, ..., n] and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API bool isBadVersion(version) which returns whether version is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

Example 1:

Input: n = 5, bad = 4

Output: 4

Explanation:

call isBadVersion(3) -> false

call isBadVersion(5) -> true

call isBadVersion(4) -> true

Then 4 is the first bad version.

Example 2:

Input: n = 1, bad = 1

Output: 1

Sol:

# The isBadVersion API is already defined for you.

# @param version, an integer

# @return an integer

# def isBadVersion(version):

class Solution:

def firstBadVersion(self, n):

low,high=1,n

while low < high:

mid = (low+high)//2

if isBadVersion(mid):

high = mid

else:

low = mid+1

return low

Q. Given the head of a sorted linked list, *delete all duplicates such that each element appears only once*. Return *the linked list****sorted****as well*.

**Example 1:**

Diagram

Description automatically generated

**Input:** head = [1,1,2]

**Output:** [1,2]

Sol:

# Definition for singly-linked list.

# class ListNode:

# def \_\_init\_\_(self, val=0, next=None):

# self.val = val

# self.next = next

class Solution:

def deleteDuplicates(self, head: ListNode) -> ListNode:

curr = head

while curr!=None and curr.next !=None:

if curr.next.val == curr.val:

curr.next = curr.next.next

else:

curr = curr.next

return head

* Time complexity : O(n)*O*(*n*). Because each node in the list is checked exactly once to determine if it is a duplicate or not, the total run time is O(n)*O*(*n*), where n*n* is the number of nodes in the list.
* Space complexity : O(1)*O*(1). No additional space is used
* the head will never be a duplicate since it's the first thing you've seen, and since it isn't getting changed it's still the beginning of the list. By the time you've looped through the linked list, current will be pointing to the final node (with it's next pointer pointing to null) which is why it isn't the correct return value

Other : class Solution:

def deleteDuplicates(self, head):

dummy = ListNode(0, head)

while head:

while head and head.next and head.val == head.next.val:

head.next = head.next.next

head = head.next

return dummy.next

Q. Dynamic programming

You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

**Example 1:**

**Input:** n = 2

**Output:** 2

**Explanation:** There are two ways to climb to the top.

1. 1 step + 1 step

2. 2 steps

Sol:

class Solution:

def climbStairs(self,n) :

if n==1:

return 1

dp = [0]\*(n+1) # intialization of the DP matrix

dp[1]=1

dp[2]=2

for i in range(3, n + 1) :

dp[i] = dp[i - 1] + dp[i - 2]

return dp[n]

iterative method:

x, y = 0, 1

for i in range(n):

x, y = y, x + y

return y

Q. Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return *its sum*.

**Example 1:**

**Input:** nums = [-2,1,-3,4,-1,2,1,-5,4]

**Output:** 6

**Explanation:** [4,-1,2,1] has the largest sum = 6.

Sol:

**Dp**

class Solution:

def maxSubArray(self, nums: List[int]) -> int:

dp = [0]\*(len(nums))

dp[0]=nums[0]

for i in range(1,len(nums)):

if nums[i]> (dp[i-1]+nums[i]):

dp[i]=nums[i]

else:

dp[i]=dp[i-1]+nums[i]

return max(dp)

**another sol:**

total\_sum = max\_sum = nums[0]

for i in nums[1:]:

total\_sum = max(i, total\_sum+i)

max\_sum = max(max\_sum,total\_sum)

return max\_sum

**kadane’s sol:**

class Solution:

def maxSubArray(self, nums: List[int]) -> int:

total\_sum = max\_sum = nums[0]

for i in range(1,len(nums)-1):

total\_sum = max(nums[i], total\_sum+nums[i])

if total\_sum> max\_sum:

max\_sum = total\_sum

return max\_sum

Q. find largest number in the array that has both positive and negative values in the array.

Example: arr=[-3,-2,1,2,3,5]

Op: 3

Sol: iterative method

def largestVal(arr):

n = len(arr)

a = []

for i in range(n):

for j in range(i+1, n):

if (abs(arr[i]) == abs(arr[j])):

a.append(abs(arr[i]))

if (len(a)==0):

return

a.sort()

return max(a)

if \_\_name\_\_ == "\_\_main\_\_":

arr = [ -3, -2, 1, 2, 3, 5]

print(largestVal(arr))

Sol using set :

def largestVal(arr):

val = set()

ar = []

for i in arr:

if abs(i) in val:

ar.append(abs(i))

else:

val.add(abs(i))

ar.sort()

return max(ar)

if \_\_name\_\_ == "\_\_main\_\_":

arr = [-3, -2, 1, 2, 3, 5]

print(largestVal(arr))

java code:

import java.util.\*;

import java.lang.\*;

class GFG {

// Print pair with negative and positive value

public static void largestVal(int arr[] , int n)

{

Vector<Integer> v = new Vector<Integer>();

// For each element of array.

for (int i = 0; i < n; i++)

// Try to find the negative value of

// arr[i] from i + 1 to n

for (int j = i + 1; j < n; j++)

// If absolute values are equal

// print pair.

if (Math.abs(arr[i]) ==

Math.abs(arr[j]))

v.add(Math.abs(arr[i]));

// If size of vector is 0, therefore there

// is no element with positive negative

// value, print "0"

if (v.size() == 0)

return;

// Sort the vector

Collections.sort(v);

// Print the pair with negative positive

// value.

//

System.out.println("The maximum element is: " + Collections.max(v)); }

// Driven Program

public static void main(String[] args)

{

int arr[] = { 4, 8, 9, -4, 1, -1 };

int n = arr.length;

largestVal(arr, n);

}

}

Q. Count the number of prime numbers less than a non-negative number, n.

**Example 1:**

**Input:** n = 10

**Output:** 4

**Explanation:** There are 4 prime numbers less than 10, they are 2, 3, 5, 7.

**Example 2:**

**Input:** n = 0

**Output:** 0

Sol:

class Solution:

def countPrimes(self, n: int) -> int:

if n <= 2:

return 0

a={}

for i in range(2,int(sqrt(n))+1):

if i not in a:

for multiple in range(i\*i, n, i):

a[multiple]=1

return n - len(a) -2

Works but time limit exceeds

class Solution:

def isPrime(self,n):

prime= True

for i in range(2, n):

if n%i==0:

prime=False

break

return prime

def countPrimes(self,n):

count =0

for i in range(2,n):

if self.isPrime(i):

count+=1

return count

Q.House robber

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.

Given an integer array nums representing the amount of money of each house, return *the maximum amount of money you can rob tonight****without alerting the police***.

**Example 1:**

**Input:** nums = [1,2,3,1]

**Output:** 4

**Explanation:** Rob house 1 (money = 1) and then rob house 3 (money = 3).

Total amount you can rob = 1 + 3 = 4.

Sol:

dp[i] is storing max sum from i -> n , but we don't know at any i, arr[i] is contributing in max sum or not.  
But it is 100% sure that for any i, dp[i+2] definately didn't included either arr[i] or arr [i+1] , since we dp[i+2] is sum from i+2 -> n. That's how we got this relation ..  
dp[ i ] = ( arr[i] + dp[i+2] , dp [i+1] )

def rob(self, arr: List[int]) -> int:

n = len(arr)

dp = [0]\*(n+1)

for i in range(n-1, -1, -1):

if i+1 < n and arr[i] + dp[i+2] > dp[i+1]: dp[i] = arr[i] + dp[i+2]

else: dp[i] = max(dp[i+1], arr[i])

return dp[0]

Q. You are given an n x n 2D matrix representing an image, rotate the image by **90** degrees (clockwise).

You have to rotate the image [**in-place**](https://en.wikipedia.org/wiki/In-place_algorithm), which means you have to modify the input 2D matrix directly. **DO NOT** allocate another 2D matrix and do the rotation.

**Input:** matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]

**Output:** [[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]

**Example 3:**

**Input:** matrix = [[1]]

**Output:** [[1]]

Sol:

List comprehension

class Solution:

def rotate(self, matrix: List[List[int]]) -> None:

"""

Do not return anything, modify matrix in-place instead.

"""

matrix[:] = [[j[i] for j in matrix ][::-1] for i in range(len(matrix))]

Q. The **count-and-say** sequence is a sequence of digit strings defined by the recursive formula:

* countAndSay(1) = "1"
* countAndSay(n) is the way you would "say" the digit string from countAndSay(n-1), which is then converted into a different digit string.

To determine how you "say" a digit string, split it into the **minimal** number of groups so that each group is a contiguous section all of the **same character.** Then for each group, say the number of characters, then say the character. To convert the saying into a digit string, replace the counts with a number and concatenate every saying.

For example, the saying and conversion for digit string "3322251":

Text

Description automatically generated

Given a positive integer n, return the nth term of the ***count-and-say*** sequence.

**Example 1:**

**Input:** n = 1

**Output:** "1"

**Explanation:** This is the base case.

Sol:

class Solution:

def countAndSay(self, n: int) -> str:

if n == 1:

return '1'

result = ''

count = 0

previous = self.countAndSay(n-1)

for i in range(len(previous)):

count+=1

if i+1>=len(previous) or previous[i]!=previous[i+1]:

result += str(count)+ str(previous[i])

count=0

return result

Q. Shuffle an array

Given an integer array nums, design an algorithm to randomly shuffle the array.

Implement the Solution class:

* Solution(int[] nums) Initializes the object with the integer array nums.
* int[] reset() Resets the array to its original configuration and returns it.
* int[] shuffle() Returns a random shuffling of the array.

**Example 1:**

**Input**

["Solution", "shuffle", "reset", "shuffle"]

[[[1, 2, 3]], [], [], []]

**Output**

[null, [3, 1, 2], [1, 2, 3], [1, 3, 2]]

**Explanation**

Solution solution = new Solution([1, 2, 3]);

solution.shuffle(); // Shuffle the array [1,2,3] and return its result. Any permutation of [1,2,3] must be equally likely to be returned. Example: return [3, 1, 2]

solution.reset(); // Resets the array back to its original configuration [1,2,3]. Return [1, 2, 3]

solution.shuffle(); // Returns the random shuffling of array [1,2,3]. Example: return [1, 3, 2]

Sol:

class Solution:

def \_\_init\_\_(self, nums: List[int]):

self.array = nums

self.original = list(nums)

def reset(self) -> List[int]:

"""

Resets the array to its original configuration and return it.

"""

self.array = self.original

self.original = list(self.original)

return self.array

def shuffle(self) -> List[int]:

"""

Returns a random shuffling of the array.

"""

nums2 = list(self.array)

for index in range(len(self.array)):

r\_index = random.randrange(len(nums2))

self.array[index] = nums2.pop(r\_index)

return self.array

# Your Solution object will be instantiated and called as such:

# obj = Solution(nums)

# param\_1 = obj.reset()

# param\_2 = obj.shuffle()

Q. Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the MinStack class:

* MinStack() initializes the stack object.
* void push(val) pushes the element val onto the stack.
* void pop() removes the element on the top of the stack.
* int top() gets the top element of the stack.
* int getMin() retrieves the minimum element in the stack.

**Example 1:**

**Input**

["MinStack","push","push","push","getMin","pop","top","getMin"]

[[],[-2],[0],[-3],[],[],[],[]]

**Output**

[null,null,null,null,-3,null,0,-2]

**Explanation**

MinStack minStack = new MinStack();

minStack.push(-2);

minStack.push(0);

minStack.push(-3);

minStack.getMin(); // return -3

minStack.pop();

minStack.top(); // return 0

minStack.getMin(); // return -2

Sol:

class MinStack:

def \_\_init\_\_(self):

"""

initialize your data structure here.

"""

self.stack = []

def push(self, val: int) -> None:

self.stack.append(val)

def pop(self) -> None:

self.stack.pop()

def top(self) -> int:

return self.stack[-1]

def getMin(self) -> int:

mini = float('inf')

if len(self.stack)==0:

return

else:

for i in range(len(self.stack)):

mini = min(mini, self.stack[i])

return mini

# Your MinStack object will be instantiated and called as such:

# obj = MinStack()

# obj.push(val)

# obj.pop()

# param\_3 = obj.top()

# param\_4 = obj.getMin()

**Another method**

Here I have used two formula  
1 . In Push function (2*x - minvalue) -------- here "x" is the given val in the push function  
2. In Pop function (2*minvalue - self.stack[-1] ) ------------- here "self.stack[-1 ] " is the top of the stack

**Solution**

class MinStack:

def \_\_init\_\_(self):

"""

initialize your data structure here.

"""

self.stack = []

self.minvalue = float("-inf")

def push(self, val: int) -> None:

if len(self.stack) == 0:

self.stack.append(val)

self.minvalue = val

else:

if val >= self.minvalue:

self.stack.append(val)

#print(self.stack)

else:

ans = 2\*val - self.minvalue

self.stack.append(ans)

self.minvalue = val

def pop(self) -> None:

if len(self.stack) == 0:

return

else:

if self.stack[-1] >= self.minvalue:

self.stack.pop()

else:

if self.stack[-1] < self.minvalue:

ans = 2\*self.minvalue - self.stack[-1]

self.minvalue = ans

self.stack.pop()

def top(self) -> int:

if len(self.stack) == 0:

return

else:

if self.stack[-1] >= self.minvalue:

return self.stack[-1]

else:

if self.stack[-1] < self.minvalue:

return self.minvalue

def getMin(self) -> int:

if len(self.stack) == 0:

return

return self.minvalue

The space comlexity will be O(1) , as we are just using the single variable name "minvalue" which take O(1) space  
and I am bit confuse with time complexity but I'm expecting it to be O(n)

Q. power of three

Given an integer n, return *true if it is a power of three. Otherwise, return false*.

An integer n is a power of three, if there exists an integer x such that n == 3x.

**Example 1:**

**Input:** n = 27

**Output:** true

**Example 2:**

**Input:** n = 0

**Output:** false

Sol:

class Solution:

def isPowerOfThree(self, n: int) -> bool:

while n:

if n==1 or n ==3:

return True

n = n/3

if 1<n<3:

return False

recursion

less runtime

class Solution:

def isPowerOfThree(self, n: int) -> bool:

while n:

if n==1 or n ==3:

return True

elif n<=0:

return False

else:

if n%3==0:

return self.isPowerOfThree(n//3)

else:

return False

Q. Roman to integers

Roman numerals are represented by seven different symbols: I, V, X, L, C, D and M.

For example, 2 is written as II in Roman numeral, just two one's added together. 12 is written as XII, which is simply X + II. The number 27 is written as XXVII, which is XX + V + II.

Given a roman numeral, convert it to an integer.

**Example 1:**

**Input:** s = "III"

**Output:** 3

**Example 2:**

**Input:** s = "IV"

**Output:** 4

**Example 3:**

**Input:** s = "IX"

**Output:** 9

Sol:

class Solution:

def romanToInt(self, s: str) -> int:

roman = {'I':1, 'V':5,'X':10,'L':50,'C':100,'D':500, 'M':1000, 'IV':4,'IX':9,'XL':40, 'XC':90, 'CD':400, 'CM':900}

i = 0

num =0

while i<len(s):

if i+1<len(s) and s[i:i+2] in roman:

num+=roman[s[i:i+2]]

i+=2

else:

num+=roman[s[i]]

i+=1

return num

using reverse and dictionary

IV = -1 + 5  
VI = 5 + 1  
XL = -10 + 50  
LX = 50 + 10  
So, if a smaller number appears before a larger number, it indicates that the number is smaller by a quantity used as a suffix to it, which made going backwards seem easy.

class Solution:

def romanToInt(self, s: str) -> int:

roman = {'I':1, 'V':5,'X':10,'L':50,'C':100,'D':500, 'M':1000}

result, temp = 0,0

for i in reversed(s):

if roman[i]>=temp:

result=result+roman[i]

else:

result=result-roman[i]

temp = roman[i]

return result

Another soln:

class Solution:

def romanToInt(self, s: str) -> int:

roman = {'I':1, 'V':5,'X':10,'L':50,'C':100,'D':500, 'M':1000}

result=0

for i,n in enumerate(s): #scanning each roman symbol

result+=roman[n]

if i and (roman[n]>roman[s[i-1]]): #for IV, IX, XL, XC, CD, CM

result-=2\*roman[s[i-1]]

return result

another soln:

class Solution:

def romanToInt(self, s: str) -> int:

roman = {'I':1, 'V':5, 'X':10, 'L':50, 'C':100, 'D':500, 'M':1000}

n = len(s)

i= n-1

output=0

while i >=0:

if i<n-1 and roman[s[i]] < roman[s[i+1]]:

output -= roman[s[i]]

else:

output += roman[s[i]]

i-=1

return output

Q. Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid. Parentheses.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.

**Example 1:**

**Input:** s = "()"

**Output:** true

**Example 2:**

**Input:** s = "()[]{}"

**Output:** true

Sol:

class Solution:

def isValid(self, s: str) -> bool:

bracket\_map = {"(": ")", "[": "]", "{": "}"}

#open\_par = set(["(", "[", "{"])

stack = []

for i in s:

if i in bracket\_map:

stack.append(bracket\_map[i])

elif not stack or (stack and stack.pop() != i):

return False

return not stack

Q. Given an integer n, return a string array answer (**1-indexed**) where:

* answer[i] == "FizzBuzz" if i is divisible by 3 and 5.
* answer[i] == "Fizz" if i is divisible by 3.
* answer[i] == "Buzz" if i is divisible by 5.
* answer[i] == i if non of the above conditions are true.

**Example 1:**

**Input:** n = 3

**Output:** ["1","2","Fizz"]

Sol:

class Solution:

def fizzBuzz(self, n: int) -> List[str]:

answer=[]

for num in range(1,n+1):

three = (num%3==0)

five = (num%5==0)

if three and five:

answer.append("FizzBuzz")

elif three:

answer.append("Fizz")

elif five:

answer.append("Buzz")

else:

answer.append(str(num))

return answer

using hash

class Solution:

def fizzBuzz(self, n: int) -> List[str]:

answer=[]

fizzbuzz = {3 : "Fizz", 5 : "Buzz"}

for num in range(1,n+1):

num\_ans = ""

for key in fizzbuzz.keys():

if num%key==0:

num\_ans += fizzbuzz[key]

if not num\_ans:

num\_ans=str(num)

answer.append(num\_ans)

return answer

Q. For two strings s and t, we say "t divides s" if and only if s = t + ... + t  (t concatenated with itself 1 or more times)

Given two strings str1 and str2, return the largest string x such that x divides both str1 and str2.

**Example 1:**

**Input:** str1 = "ABCABC", str2 = "ABC"

**Output:** "ABC"

Sol:

Using recursion

class Solution:

def gcdOfStrings(self, str1: str, str2: str) -> str:

if not str1: return str2

if not str2: return str1

str1, str2 = (str1, str2) if len(str1) <= len(str2) else (str2, str1)

if str2[:len(str1)] == str1:

return self.gcdOfStrings(str2[len(str1):],str1)

return ''

return the largest string that divides both of the strings.

* temp := shorter string between A and B
* m := length of temp
* x := 1
* res is an array and insert “” into the res
* while A and B has substring of size x, then add the substring into the res, and increase x by 1
* finally return the last element in the res array.

class Solution(object):

   def gcdOfStrings(self, str1, str2):

      if len(str1)<=len(str2):

         temp = str1

      else:

         temp = str2

      m = len(temp)

      x = 1

      res=[""]

      while x<=m:

         if m%x==0 and temp[:x] \* (len(str1)//x) == str1 and temp[:x] \* (len(str2)//x) == str2:

            res.append(temp[:x])

         x+=1

      return res[-1]

ob1 = Solution()

print(ob1.gcdOfStrings("ABABAB","ABAB"))

Q. Common words among a tuple string in python

Sol:

When it is required to find common words among the tuple strings, the 'join' method, the 'set' method, the '&' operator and the 'split' method is used.

The 'join' method can be used to join multiple values based on a specific value,

Python comes with a datatype known as 'set'. This 'set' contains elements that are unique only. The set is useful in performing operations such as intersection, difference, union and symmetric difference.

The 'split' function splits the given data into multiple sections depending on the value on which it needs to be split.

The '&' operator performs multiplication, i.e AND operation.

Sol:

my\_tuple\_1 = ('Hi there', 'Hi Will,', 'Hi ', 'Hi there')

print("The tuple is : ")

print(my\_tuple\_1)

my\_result = ", ".join(sorted(set(my\_tuple\_1[0].split()) & set(my\_tuple\_1[1].split()) &

   set(my\_tuple\_1[2].split())))

print("Common words among the tuples are : ")

print(my\_result)

Q. iven two stings ransomNote and magazine, return true if ransomNote can be constructed from magazine and false otherwise.

Each letter in magazine can only be used once in ransomNote.

**Example 1:**

**Input:** ransomNote = "a", magazine = "b"

**Output:** false

**Example 3:**

**Input:** ransomNote = "aa", magazine = "aab"

**Output:** true

Sol:

class Solution:

def canConstruct(self, ransomNote: str, magazine: str) -> bool:

dict1 = Counter(magazine)

dict2 = Counter(ransomNote)

for i in dict2:

if dict2[i]> dict1[i]:

return False

return True

solution using set

class Solution:

def canConstruct(self, ransomNote: str, magazine: str) -> bool:

string = set(ransomNote)

for i in string:

if ransomNote.count(i)> magazine.count(i):

return False

return True

Q. Given an array of meeting time intervals consisting of start and end times [[s1,e1],[s2,e2],…] (si < ei), find the minimum number of conference rooms required.

**Example 1**

Input: [[0, 30],[5, 10],[15, 20]]

Output: 2

**Example 2**

Input: [[7,10],[2,4]]

Output: 1

Greedy: We need to wrap the end time of interval as one room and append it to rooms. First we sort the interval. Then we iterate the given interval, if the current interval’s start time is bigger or equal than any room’s end time in rooms, we will append the current interval’s end time to this room; if the current interval’s start time is not bigger or not equal than any room’s end time in rooms, we will wrap this end time of the current interval as one room and append it to rooms. Finally, the result will be the size of the rooms.

Sol:

**def** **minMeetingRooms**(self, intervals: List[List[int]]) **->** int:

size **=** len(intervals)

**if** size**<=**1: **return** size

sorted\_intervals **=** sorted(intervals)

rooms**=**[[sorted\_intervals[0][1]]]

**for** i **in** range(1,size):

booked **=** False

**for** room **in** rooms:

**if** sorted\_intervals[i][0]**>=**room[**-**1]:

room.append(sorted\_intervals[i][1])

booked **=** True

**break**

**if** **not** booked:rooms.append([sorted\_intervals[i][1]])

**return** len(rooms)

Q. H index

Given an array of integers citations where citations[i] is the number of citations a researcher received for their ith paper, return compute the researcher's h**-index**.

According to the [definition of h-index on Wikipedia](https://en.wikipedia.org/wiki/H-index): A scientist has an index h if h of their n papers have at least h citations each, and the other n − h papers have no more than h citations each.

If there are several possible values for h, the maximum one is taken as the h**-index**.

**Example 1:**

**Input:** citations = [3,0,6,1,5]

**Output:** 3

**Explanation:** [3,0,6,1,5] means the researcher has 5 papers in total and each of them had received 3, 0, 6, 1, 5 citations respectively.

Since the researcher has 3 papers with at least 3 citations each and the remaining two with no more than 3 citations each, their h-index is 3.

**Example 2:**

**Input:** citations = [1,3,1]

**Output:** 1

Sol:

class Solution:

def hIndex(self, citations: List[int]) -> int:

sort\_cite = sorted(citations, reverse=True)

for i in range(len(sort\_cite)):

if i+1 > sort\_cite[i]:

return i

return len(sort\_cite)

class Solution:

def hIndex(self, citations: List[int]) -> int:

ans=0

sort\_cite = sorted(citations, reverse=True)

for i, n in enumerate(sort\_cite):

ans= max(ans, min(i+1,n))

return ans

class Solution:

def hIndex(self, citations: List[int]) -> int:

citations.sort()

m = len(citations)

for i, n in enumerate(citations):

if n >= m -i:

return m-i

return 0

O(n log n)

Sc : O(1)

Q. Arrange coins

You have n coins and you want to build a staircase with these coins. The staircase consists of k rows where the ith row has exactly i coins. The last row of the staircase **may be** incomplete.

Given the integer n, return *the number of****complete rows****of the staircase you will build*.

A picture containing calendar

Description automatically generated

**Input:** n = 5

**Output:** 2

**Explanation:** Because the 3rd row is incomplete, we return 2.

Sol:

class Solution:

def arrangeCoins(self, n: int) -> int:

left, right = 0, n

while left <= right:

k = (left + right) // 2

current = k \* (k+1)// 2

if current == n:

return k

if n < current:

right = k - 1

else:

left = k + 1

return right

Time complexity: O(log n)

Space complexity: O(1)

Q. Given two binary strings a and b, return *their sum as a binary string*.

**Example 1:**

**Input:** a = "11", b = "1"

**Output:** "100"

**Example 2:**

**Input:** a = "1010", b = "1011"

**Output:** "10101"

Sol:

class Solution:

def addBinary(self, a: str, b: str) -> str:

aa = int(a,2)

bb = int(b,2)

c = bin(aa+bb) [2:]

return c

class Solution:

def addBinary(self, a: str, b: str) -> str:

carry =0

result=''

a,b = list(a),list(b)

while a or b or carry == 1:

if a:

carry += int(a.pop())

if b:

carry += int(b.pop())

result += str(carry % 2)

carry = carry//2

return result[::-1]

class Solution:

def addBinary(self, a: str, b: str) -> str:

return bin(int(a,2) + int(b,2))[2:]

Q. Given a non-negative integer x, compute and return *the square root of* x.

Since the return type is an integer, the decimal digits are **truncated**, and only **the integer part** of the result is returned.

**Note:**You are not allowed to use any built-in exponent function or operator, such as pow(x, 0.5) or x \*\* 0.5.

**Example 1:**

**Input:** x = 4

**Output:** 2

Sol:

class Solution:

def mySqrt(self, x: int) -> int:

if x == 0:

return 0

elif x == 1 or x == 2:

return 1

stack = []

for i in range(x):

if i\*i > x:

return stack[i-1]

stack.append(i)

Q. Given the roots of two binary trees p and q, write a function to check if they are the same or not.

Two binary trees are considered the same if they are structurally identical, and the nodes have the same value.

 Same tree

**Example 1:**

Diagram

Description automatically generated

**Input:** p = [1,2,3], q = [1,2,3]

**Output:** true

Sol:

# Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution:

def isSameTree(self, p: TreeNode, q: TreeNode) -> bool:

if not p and not q:

return True

if not p or not q:

return False

if p.val!=q.val:

return False

return self.isSameTree(p.right, q.right) and self.isSameTree(p.left, q.left)

Q. minimum depth of a binary tree

Given a binary tree, find its minimum depth.

The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

**Note:** A leaf is a node with no children.

**Example 1:**

Diagram, shape

Description automatically generated

**Input:** root = [3,9,20,null,null,15,7]

**Output:** 2

Sol:

# Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution:

def minDepth(self, root: TreeNode) -> int:

if root is None: return 0

if root.left is None and root.right is None:

return 1

elif not root.left and root.right:

return 1 + self.minDepth(root.right)

elif root.left and not root.right:

return 1 + self.minDepth(root.left)

else:

return min(self.minDepth(root.left), self.minDepth(root.right)) + 1

Q. Pascals triangle

Given an integer numRows, return the first numRows of **Pascal's triangle**.

In **Pascal's triangle**, each number is the sum of the two numbers directly above it as shown:



**Example 1:**

**Input:** numRows = 5

**Output:** [[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]]

**Example 2:**

**Input:** numRows = 1

**Output:** [[1]]

Sol:

class Solution:

def generate(self, numRows: int) -> List[List[int]]:

if numRows == 0: return []

elif numRows == 1: return [[1]]

Tri = [[1]]

for i in range(1,numRows):

row = [1]

for j in range(1,i):

row.append(Tri[i-1][j-1] + Tri[i-1][j])

row.append(1)

Tri.append(row)

return Tri

Q. Two sum 2

Given an array of integers numbers that is already ***sorted in non-decreasing order***, find two numbers such that they add up to a specific target number.

Return*the indices of the two numbers (****1-indexed****) as an integer array*answer*of size*2*, where*1 <= answer[0] < answer[1] <= numbers.length.

The tests are generated such that there is **exactly one solution**. You **may not** use the same element twice.

**Example 1:**

**Input:** numbers = [2,7,11,15], target = 9

**Output:** [1,2]

**Explanation:** The sum of 2 and 7 is 9. Therefore index1 = 1, index2 = 2.

Sol:

class Solution:

def twoSum(self, numbers: List[int], target: int) -> List[int]:

start = 0

end = len(numbers) - 1

sum = 0

while start<=end:

sum = numbers[start] + numbers[end]

if sum>target:

end-=1

elif sum<target:

start+=1

else:

return [start+1, end+1]

Q. reverse vowels in string

Given a string s, reverse only all the vowels in the string and return it.

The vowels are 'a', 'e', 'i', 'o', and 'u', and they can appear in both cases.

**Example 1:**

**Input:** s = "hello"

**Output:** "holle"

**Example 2:**

**Input:** s = "leetcode"

**Output:** "leotcede"

Sol:

class Solution:

def reverseVowels(self, s: str) -> str:

vowels = ['a','e','i','o','u', 'A','E','I','O','U']

string\_vowels= [i for i in s if i in vowels]

s = list(s)

for i in range(0,len(s)):

if s[i] in vowels:

s[i] = string\_vowels.pop()

return ''.join(s)

another sol:

``````vowels= "aeiouAEIOU"

lst = list(s)

l,r = 0, len(s)-1

while l<r:

if lst[l] not in vowels:

l+=1

elif lst[r] not in vowels:

r-=1

else:

lst[l], lst[r]= lst[r], lst[l]

l+=1

r-=1

return ''.join(lst)

Q. number of segments in a string

You are given a string s, return *the number of segments in the string*.

A **segment** is defined to be a contiguous sequence of **non-space characters**.

**Example 1:**

**Input:** s = "Hello, my name is John"

**Output:** 5

**Explanation:** The five segments are ["Hello,", "my", "name", "is", "John"]

Sol:

class Solution:

def countSegments(self, s: str) -> int:

return len(s.split())

TC: O(n), sc : O(n)

class Solution:

def countSegments(self, s):

segment\_count = 0

for i in range(len(s)):

if (i == 0 or s[i-1] == ' ') and s[i] != ' ':

segment\_count += 1

return segment\_count

TC: O(n), sc : O(1)

Q. Repeated substring pattern

Given a string s, check if it can be constructed by taking a substring of it and appending multiple copies of the substring together.

**Example 1:**

**Input:** s = "abab"

**Output:** true

**Explanation:** It is the substring "ab" twice.

Sol:

class Solution:

def repeatedSubstringPattern(self, s: str) -> bool:

for i in range(1, len(s)):

repeat = len(s)// i

if i \* repeat > len(s):

return False

if s[0:i]\*repeat==s:

return True

return False

Q. Fibonacci number

The **Fibonacci numbers**, commonly denoted F(n) form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from 0 and 1. That is,

F(0) = 0, F(1) = 1

F(n) = F(n - 1) + F(n - 2), for n > 1.

Given n, calculate F(n).

**Example 1:**

**Input:** n = 2

**Output:** 1

**Explanation:** F(2) = F(1) + F(0) = 1 + 0 = 1.

Sol:

class Solution:

def fib(self, n: int) -> int:

if n < 0:

return None

elif n == 0:

return 0

elif n == 1 or n ==2:

return 1

else:

return (self.fib(n-1) + self.fib(n-2))

class Solution:

def fib(self, N: int) -> int:

if N <= 1:

return N

return self.fib(N-1) + self.fib(N-2)

TC: O(2N) This is the slowest way to solve the Fibonacci Sequence because it takes exponential time. The amount of operations needed, for each level of recursion, grows exponentially as the depth approaches N.

Sc: O(N)

Iterative top down

class Solution:

def fib(self, N: int) -> int:

if (N <= 1):

return N

if (N == 2):

return 1

current = 0

prev1 = 1

prev2 = 1

# Since range is exclusive and we want to include N, we need to put N+1.

for i in range(3, N+1):

current = prev1 + prev2

prev2 = prev1

prev1 = current

return current

TC: O(N)

Sc: O(N)

Q. Minimum Add to Make Parentheses Valid

Given a string s of '(' and ')' parentheses, we add the minimum number of parentheses ( '(' or ')', and in any positions ) so that the resulting parentheses string is valid.

Formally, a parentheses string is valid if and only if:

* It is the empty string, or
* It can be written as AB (A concatenated with B), where A and B are valid strings, or
* It can be written as (A), where A is a valid string.

Given a parentheses string, return the minimum number of parentheses we must add to make the resulting string valid.

**Example 1:**

**Input:** s = "())"

**Output:** 1

Keep track of the *balance* of the string: the number of '(''s minus the number of ')''s. A string is valid if its balance is 0, plus every prefix has non-negative balance.

The above idea is common with matching brackets problems, but could be difficult to find if you haven't seen it before.

Now, consider the balance of every prefix of S. If it is ever negative (say, -1), we must add a '(' bracket. Also, if the balance of S is positive (say, +B), we must add B ')' brackets at the end.

Sol:

class Solution:

def minAddToMakeValid(self, s: str) -> int:

ans = bal = 0

for paren in s:

bal += 1 if paren == '(' else -1

if bal == -1:

ans +=1

bal +=1

return bal+ans

TC: *O*(*N*), where Nis the length of S.

SC:O(1)

Q. Given a string **s** and two words **w1** and **w2** that are present in S. The task is to find the minimum distance between **w1** and **w2**. Here, distance is the number of steps or words between the first and the second word.

**Examples:**

***Input :****s = “geeks for geeks contribute practice”, w1 = “geeks”, w2 = “practice”****Output : 1****There is only one word between the closest occurrences of w1 and w2.*

***Input :****s = “the quick the brown quick brown the frog”, w1 = “quick”, w2 = “frog”****Output :****2*

Sol:

def distance(s, w1, w2):

    if w1 == w2 :

       return 0

    # get individual words in a list

    words = s.split(" ")

    # assume total length of the string as

    # minimum distance

    min\_dist = len(words)+1

    # traverse through the entire string

    for index in range(len(words)):

        if words[index] == w1:

            for search in range(len(words)):

                if words[search] == w2:

                    # the distance between the words is

                    # the index of the first word - the

                    # current word index

                    curr = abs(index - search) - 1;

                    # comparing current distance with

                    # the previously assumed distance

                    if curr < min\_dist:

                       min\_dist = curr

    # w1 and w2 are same and adjacent

    return min\_dist

# Driver code

s = "geeks for geeks contribute practice"

w1 = "geeks"

w2 = "practice"

print(distance(s, w1, w2))

**Q. sliding window**

**Given an array and an integer K, find the maximum for each and every contiguous subarray of size k.**

**Examples :**

**Input: arr[] = {1, 2, 3, 1, 4, 5, 2, 3, 6}, K = 3**

**Output: 3 3 4 5 5 5 6**

**Explanation:**

Maximum of 1, 2, 3 is 3

Maximum of 2, 3, 1 is 3

Maximum of 3, 1, 4 is 4

Maximum of 1, 4, 5 is 5

Maximum of 4, 5, 2 is 5

Maximum of 5, 2, 3 is 5

Maximum of 2, 3, 6 is 6

**Input**: arr[] = {8, 5, 10, 7, 9, 4, 15, 12, 90, 13}, K = 4

**Output**: 10 10 10 15 15 90 90

**Explanation:**

Maximum of first 4 elements is 10, similarly for next 4

elements (i.e from index 1 to 4) is 10, So the sequence

generated is 10 10 10 15 15 90 90

Sol:

**Create a nested loop, the outer loop from starting index to n – k th elements. The inner loop will run for k iterations.**

**Create a variable to store the maximum of k elements traversed by the inner loop.**

**Find the maximum of k elements traversed by the inner loop.**

**Print the maximum element in every iteration of outer loop**

def printMax(arr, n, k):

    max = 0

    for i in range(n - k + 1):

        max = arr[i]

        for j in range(1, k):

            if arr[i + j] > max:

                max = arr[i + j]

        print(str(max) + " ", end = "")

# Driver method

if \_\_name\_\_=="\_\_main\_\_":

    arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

    n = len(arr)

    k = 3

    printMax(arr, n, k)

* **Time Complexity:**O(N \* K).   
  The outer loop runs n-k+1 times and the inner loop runs k times for every iteration of outer loop. So time complexity is O((n-k+1)\*k) which can also be written as **O(N \* K)**.
* **Space Complexity:**O(1).   
  No extra space is required.

1. **Q.** Create a deque to store k elements.
2. Run a loop and insert first k elements in the deque. Before inserting the element, check if the element at the back of the queue is smaller than the current element , if it is so remove the element from the back of the deque, until all elements left in the deque are greater than the current element. Then insert the current element, at the back of the deque.
3. Now, run a loop from k to end of the array.
4. Print the front element of the deque.
5. Remove the element from the front of the queue if they are out of the current window.
6. Insert the next element in the deque. Before inserting the element, check if the element at the back of the queue is smaller than the current element , if it is so remove the element from the back of the deque, until all elements left in the deque are greater than the current element. Then insert the current element, at the back of the deque.
7. Print the maximum element of the last window.

from collections import deque

# A Deque (Double ended queue) based

# method for printing maximum element

# of all subarrays of size k

def printMax(arr, n, k):

    """ Create a Double Ended Queue, Qi that

    will store indexes of array elements.

    The queue will store indexes of useful

    elements in every window and it will

    maintain decreasing order of values from

    front to rear in Qi, i.e., arr[Qi.front[]]

    to arr[Qi.rear()] are sorted in decreasing

    order"""

    Qi = deque()

    # Process first k (or first window)

    # elements of array

    for i in range(k):

        # For every element, the previous

        # smaller elements are useless

        # so remove them from Qi

        while Qi and arr[i] >= arr[Qi[-1]] :

            Qi.pop()

        # Add new element at rear of queue

        Qi.append(i);

    # Process rest of the elements, i.e.

    # from arr[k] to arr[n-1]

    for i in range(k, n):

        # The element at the front of the

        # queue is the largest element of

        # previous window, so print it

        print(str(arr[Qi[0]]) + " ", end = "")

        # Remove the elements which are

        # out of this window

        while Qi and Qi[0] <= i-k:

            # remove from front of deque

            Qi.popleft()

        # Remove all elements smaller than

        # the currently being added element

        # (Remove useless elements)

        while Qi and arr[i] >= arr[Qi[-1]] :

            Qi.pop()

        # Add current element at the rear of Qi

        Qi.append(i)

    # Print the maximum element of last window

    print(str(arr[Qi[0]]))

# Driver code

if \_\_name\_\_=="\_\_main\_\_":

    arr = [12, 1, 78, 90, 57, 89, 56]

    k = 3

    printMax(arr, len(arr), k)

**Q, number of islandds**

def dfs(grid, r,c ):

grid[r][c]= '0'

list = [(r-1, c), (r+1, c), (r, c-1), (r, c+1)]

for row, col in list:

if row>=0 and col>=0 and row < len(grid) and col< len(grid[row]) and grid[row][col] == '1':

dfs(grid, row, col)

def numberOfIslands(grid):

islands = 0

for r in range(len(grid)):

for c in range(len(grid[r])):

if grid[r][c] == '1':

dfs(grid, r, c)

islands += 1

return islands

grid = [

["1","1","0","0","0"],

["1","1","0","0","0"],

["0","0","1","0","0"],

["0","0","0","1","1"]

]

print(numberOfIslands(grid))

m times n if we are given an m by n array

Q. meeting rooms one

def meetingRoom(intervals):

#0--5--10--15--20--25--30

intervals.sort()

last\_end = -1

for start, end in intervals:

if last\_end <= start:

last\_end = end

else:

return False

return True

intervals = [[7,10], [2, 4]]

print(meetingRoom(intervals))

#TC o(nlogn)

#SC o(1)

Q. meeting rooms

import heapq

def meetingRoom(intervals):

intervals.sort()

meeting\_rooms = 1

heap = [intervals[0][1]]

for start, end in intervals[1:]:

if heap[0] <= start:

heapq.heappop(heap)

heapq.heappush(heap, end)

meeting\_rooms = max(meeting\_rooms, len(heap))

return meeting\_rooms

#intervals = [[0,15], [0, 30], [15, 20], [21, 25]]

intervals = [[7,10], [2, 4]]

print(meetingRoom(intervals))

Q. star alphabet

def star(n):

num = 65

for i in range(0, n):

for j in range(0, i+1):

al = chr(num)

print(al , end=" ")

num = num + 1

print("\r")

n = 5

star(n)

Q. star pattern

def star(n):

for i in range(0, n):

for j in range(0, i+1):

print(" \* ", end="")

print("\r")

n = 5

star(n)

using list

def pypart(n):

    myList = []

    for i in range(1,n+1):

        myList.append("\*"\*i)

    print("\n".join(myList))

# Driver Code

n = 5

pypart(n)

180

def pypart2(n):

    # number of spaces

    k = 2\*n - 2

    # outer loop to handle number of rows

    for i in range(0, n):

        # inner loop to handle number spaces

        # values changing acc. to requirement

        for j in range(0, k):

            print(end=" ")

        # decrementing k after each loop

        k = k - 2

        # inner loop to handle number of columns

        # values changing acc. to outer loop

        for j in range(0, i+1):

            # printing stars

            print("\* ", end="")

        # ending line after each row

        print("\r")

# Driver Code

n = 5

pypart2(n)

triangle star

def triangle(n):

    # number of spaces

    k = n - 1

    # outer loop to handle number of rows

    for i in range(0, n):

        # inner loop to handle number spaces

        # values changing acc. to requirement

        for j in range(0, k):

            print(end=" ")

        # decrementing k after each loop

        k = k - 1

        # inner loop to handle number of columns

        # values changing acc. to outer loop

        for j in range(0, i+1):

            # printing stars

            print("\* ", end="")

        # ending line after each row

        print("\r")

# Driver Code

n = 5

triangle(n)

number triangle

def numpat(n):

    # initialising starting number

    num = 1

    # outer loop to handle number of rows

    for i in range(0, n):

        # re assigning num

        num = 1

        # inner loop to handle number of columns

            # values changing acc. to outer loop

        for j in range(0, i+1):

                # printing number

            print(num, end=" ")

            # incrementing number at each column

            num = num + 1

        # ending line after each row

        print("\r")

# Driver code

n = 5

numpat(n)

number pattern ascending

def contnum(n):

    # initializing starting number

    num = 1

    # outer loop to handle number of rows

    for i in range(0, n):

        # not re assigning num

        # num = 1

        # inner loop to handle number of columns

        # values changing acc. to outer loop

        for j in range(0, i+1):

            # printing number

            print(num, end=" ")

            # incrementing number at each column

            num = num + 1

        # ending line after each row

        print("\r")

n = 5

# sending 5 as argument

# calling Function

contnum(n)

Q. binary search array

def binary\_search(arr, target):

l = 0

r = len(arr) - 1

while l <= r:

mid = (l+r) // 2

if target == arr[mid]:

return mid

elif target < arr[mid]:

r = mid - 1

else:

l = mid + 1

return -1

arr = [-2, 3, 4, 7, 8, 9, 11, 13]

target = 11

print(binary\_search(arr, target))

recursive binary search

def binary\_search\_recursive(array, element, start, end):

if start > end:

return -1

mid = (start + end) // 2

if element == array[mid]:

return mid

if element < array[mid]:

return binary\_search\_recursive(array, element, start, mid-1)

else:

return binary\_search\_recursive(array, element, mid+1, end)

arr = [-2, 3, 4, 7, 8, 9, 11, 13]

target = 11

print(binary\_search\_recursive(arr, target, 0, len(arr)))

Q.Invert a binary tree

# A node contains the value, left and right pointers

class newNode:

def \_\_init\_\_(self,data):

self.data = data

self.left = self.right = None

def invert(node):

if (node == None):

return

else:

temp = node

# recursive calls

invert(node.left)

invert(node.right)

# swap the pointers in this node

temp = node.left

node.left = node.right

node.right = temp

# print InOrder binary tree traversal.

def print\_tree(node) :

if (node == None):

return

print\_tree(node.left)

print(node.data, end=" ")

print\_tree(node.right)

root = newNode(2)

root.left = newNode(1)

root.right = newNode(4)

root.right.left = newNode(3)

root.right.right = newNode(5)

# Print inorder traversal of the input tree

print("Inorder traversal of the constructed tree is")

print\_tree(root)

# Convert tree to its mirror

invert(root)

# Print inorder traversal of the mirror tree

print("\nInorder traversal of the mirror tree is")

print\_tree(root)

Q. Given an array and an integer **K**, find the maximum for each and every contiguous subarray of size k.

1. Create a nested loop, the outer loop from starting index to n – k th elements. The inner loop will run for k iterations.
2. Create a variable to store the maximum of k elements traversed by the inner loop.
3. Find the maximum of k elements traversed by the inner loop.
4. Print the maximum element in every iteration of outer loop

def printMax(arr, n, k):

    max = 0

    for i in range(n - k + 1):

        max = arr[i]

        for j in range(1, k):

            if arr[i + j] > max:

                max = arr[i + j]

        print(str(max) + " ", end = "")

# Driver method

if \_\_name\_\_=="\_\_main\_\_":

    arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

    n = len(arr)

    k = 3

    printMax(arr, n, k)

* **Time Complexity:**O(N \* K).   
  The outer loop runs n-k+1 times and the inner loop runs k times for every iteration of outer loop. So time complexity is O((n-k+1)\*k) which can also be written as **O(N \* K)**.
* **Space Complexity:**O(1).   
  No extra space is required.

Q. merge two sorted arrays

def mergeArrays(arr1, arr2, n1, n2):

    arr3 = [None] \* (n1 + n2)

    i = 0

    j = 0

    k = 0

    # Traverse both array

    while i < n1 and j < n2:

        # Check if current element

        # of first array is smaller

        # than current element of

        # second array. If yes,

        # store first array element

        # and increment first array

        # index. Otherwise do same

        # with second array

        if arr1[i] < arr2[j]:

            arr3[k] = arr1[i]

            k = k + 1

            i = i + 1

        else:

            arr3[k] = arr2[j]

            k = k + 1

            j = j + 1

    # Store remaining elements

    # of first array

    while i < n1:

        arr3[k] = arr1[i];

        k = k + 1

        i = i + 1

    # Store remaining elements

    # of second array

    while j < n2:

        arr3[k] = arr2[j];

        k = k + 1

        j = j + 1

    print("Array after merging")

    for i in range(n1 + n2):

        print(str(arr3[i]), end = " ")

# Driver code

arr1 = [1, 3, 5, 7]

n1 = len(arr1)

arr2 = [2, 4, 6, 8]

n2 = len(arr2)

mergeArrays(arr1, arr2, n1, n2);

Q. diagonal traverse

class Solution:

def findDiagonalOrder(self, matrix: List[List[int]]) -> List[int]:

# Check for empty matrices

if not matrix or not matrix[0]:

return []

# Variables to track the size of the matrix

N, M = len(matrix), len(matrix[0])

# The two arrays as explained in the algorithm

result, intermediate = [], []

# We have to go over all the elements in the first

# row and the last column to cover all possible diagonals

for d in range(N + M - 1):

# Clear the intermediate array everytime we start

# to process another diagonal

intermediate.clear()

# We need to figure out the "head" of this diagonal

# The elements in the first row and the last column

# are the respective heads.

r, c = 0 if d < M else d - M + 1, d if d < M else M - 1

# Iterate until one of the indices goes out of scope

# Take note of the index math to go down the diagonal

while r < N and c > -1:

intermediate.append(matrix[r][c])

r += 1

c -= 1

# Reverse even numbered diagonals. The

# article says we have to reverse odd

# numbered articles but here, the numbering

# is starting from 0 :P

if d % 2 == 0:

result.extend(intermediate[::-1])

else:

result.extend(intermediate)

return result

Q. **Excel Sheet Column Title**

class Solution:

def convertToTitle(self, columnNumber: int) -> str:

s, i = '', 26

while columnNumber != 0:

columnNumber /= (26/i)

d = int(columnNumber) % 26

if d == 0:

s = 'Z' + s

columnNumber -= 26

else:

s = chr(d + 64) + s

columnNumber -= d

i = 1

return(s)

class Solution:

def convertToTitle(self, n: int) -> str:

column = ''

while n:

n -= 1

column += chr(n % 26 + 65)

n //= 26

return column[::-1]

Q. binary tree paths

Given the root of a binary tree, return *all root-to-leaf paths in****any order***.

A **leaf** is a node with no children.

**Example 1:**

A picture containing text, clipart

Description automatically generated

**Input:** root = [1,2,3,null,5]

**Output:** ["1->2->5","1->3"]

**Example 2:**

**Input:** root = [1]

**Output:** ["1"]

Sol:

# Definition for a binary tree node.

# class TreeNode:

# def \_\_init\_\_(self, val=0, left=None, right=None):

# self.val = val

# self.left = left

# self.right = right

class Solution:

def binaryTreePaths(self, root: TreeNode) -> List[str]:

result=[]

def dfs(root,string):

if not root: return

if not (root.left or root.right):

result.append(string+str(root.val))

if root.left:

dfs(root.left, string+str(root.val)+"->")

if root.right:

dfs(root.right, string+str(root.val)+"->")

dfs(root, "")

return result

Q. word pattern

Given a pattern and a string s, find if s follows the same pattern.

Here **follow** means a full match, such that there is a bijection between a letter in pattern and a **non-empty** word in s.

**Example 1:**

**Input:** pattern = "abba", s = "dog cat cat dog"

**Output:** true

**Example 2:**

**Input:** pattern = "abba", s = "dog cat cat fish"

**Output:** false

**Example 3:**

**Input:** pattern = "aaaa", s = "dog cat cat dog"

**Output:** false

Sol:

class Solution:

def wordPattern(self, pattern: str, s: str) -> bool:

s = s.split(" ")

if len(s) != len(pattern):

return False

d = {}

for i in range(len(s)):

if s[i] not in d:

d[s[i]]=pattern[i]

elif d[s[i]]!=pattern[i]:

return False

return len(set(pattern)) == len(set(d.keys()))

pattern = "aabac"

s = "dog dog cat dog mao"

print(wordPattern(pattern, s))

Q. Find the difference

You are given two strings s and t.

String t is generated by random shuffling string s and then add one more letter at a random position.

Return the letter that was added to t.

**Example 1:**

**Input:** s = "abcd", t = "abcde"

**Output:** "e"

**Explanation:** 'e' is the letter that was added.

**Example 2:**

**Input:** s = "", t = "y"

**Output:** "y"

**Example 3:**

**Input:** s = "a", t = "aa"

**Output:** "a"

Sol:

Using counter

class Solution:

def findTheDifference(self, s: str, t: str) -> str:

s= Counter(s)

t= Counter(t)

for i, n in t.items():

if s[i]==n:

continue

return i

using set

class Solution:

def findTheDifference(self, s: str, t: str) -> str:

for i in set(t):

if s.count(i)!=t.count(i):

return i

Q. is subsequence

Given two strings s and t, return true*if*s*is a****subsequence****of*t*, or*false*otherwise*.

A **subsequence** of a string is a new string that is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (i.e., "ace" is a subsequence of "abcde" while "aec" is not).

**Example 1:**

**Input:** s = "abc", t = "ahbgdc"

**Output:** true

**Example 2:**

**Input:** s = "axc", t = "ahbgdc"

**Output:** false

Sol:

Using two pointer

class Solution:

def isSubsequence(self, s: str, t: str) -> bool:

sid, tid = 0,0

while (sid<len(s)) and (tid<len(t)):

if t[tid] == s[sid]:

sid +=1

tid+=1

return sid == len(s)

TC: O(n)

Sc: o(1)

class Solution(object):

def isSubsequence(self, s, t):

i = 0

for c in t:

if i < len(s) and s[i] == c:

i += 1

return i == len(s)

Q. longest palindrome

Given a string s which consists of lowercase or uppercase letters, return *the length of the****longest palindrome*** that can be built with those letters.

Letters are **case sensitive**, for example, "Aa" is not considered a palindrome here.

**Example 1:**

**Input:** s = "abccccdd"

**Output:** 7

**Explanation:**

One longest palindrome that can be built is "dccaccd", whose length is 7.

**Example 2:**

**Input:** s = "a"

**Output:** 1

**Example 3:**

**Input:** s = "bb"

**Output:** 2

Sol:

class Solution:

def longestPalindrome(self, s: str) -> int:

pairs=0

unpaired = set()

for i in s:

if i in unpaired:

pairs+=1

unpaired.remove(i)

else:

unpaired.add(i)

return pairs \* 2 + 1 if unpaired else pairs\*2

Q. keyboard row

Given an array of strings words, return the words that can be typed using letters of the alphabet on only one row of American keyboard like the image below.

In the **American keyboard**:

* the first row consists of the characters "qwertyuiop",
* the second row consists of the characters "asdfghjkl", and
* the third row consists of the characters "zxcvbnm".
* **Example 1:**
* **Input:** words = ["Hello","Alaska","Dad","Peace"]
* **Output:** ["Alaska","Dad"]
* **Example 2:**
* **Input:** words = ["omk"]
* **Output:** []
* **Example 3:**
* **Input:** words = ["adsdf","sfd"]
* **Output:** ["adsdf","sfd"]

Sol:

Using set

class Solution:

def findWords(self, words: List[str]) -> List[str]:

set1 = {'q','w','e','r','t','y','u','i','o','p'}

set2 = {'a','s','d','f','g','h','j','k','l'}

set3 = {'z','x','c','v','b','n','m'}

result=[]

for i in words:

wordset = set(i.lower())

if (wordset&set1== wordset) or (wordset&set2==wordset) or (wordset&set3==wordset):

result.append(i)

return result

Another solution(don’t attempt niki)

class Solution:

def findWords(self, words: List[str]) -> List[str]:

querty = ['qwertyuiop', 'asdfghjkl', 'zxcvbnm']

return ([word for word in words if any([all([letter in row for letter in word.lower()]) for row in querty])])

Explain : Return <word in list> if \*\*all\*\*<letter in word> contained by \*\*any\*\*<row of querty>

* <word in list> : [**word** for **word** in **words**]
* <letter in word> : [**letter** in row **for letter in word**.lower()]
* <row of querty>: [all([letter in **row** for letter in word.lower()]) **for row in querty**])
* all <letter in word> in a row : **all(** [letter in row for letter in word.lower()] **)**
* all <letter in word> in any row : **any(** [all([letter in row for letter in word.lower()]) for row in querty])] **)**  
  Good luck :D

same as the first solution:

class Solution:

def findWords(self, words: List[str]) -> List[str]:

set1 = set('qwertyuiop')

set2 = set('asdfghjkl')

set3 = set('zxcvbnm')

result=[]

for i in words:

w = set(i.lower())

if w.issubset(set1) or w.issubset(set2) or w.issubset(set3):

result.append(i)

return result

Q. Detect capital

We define the usage of capitals in a word to be right when one of the following cases holds:

* All letters in this word are capitals, like "USA".
* All letters in this word are not capitals, like "leetcode".
* Only the first letter in this word is capital, like "Google".

Given a string word, return true if the usage of capitals in it is right.

**Example 1:**

**Input:** word = "USA"

**Output:** true

**Example 2:**

**Input:** word = "FlaG"

**Output:** false

Sol:

class Solution:

def detectCapitalUse(self, word: str) -> bool:

return word in [ word.upper(), word.lower(), word.capitalize()]

Another similar

class Solution:

def detectCapitalUse(self, word: str) -> bool:

n = len(word)

if len(word) == 1:

return True

# case 1: All capital

if word[0].isupper() and word[1].isupper():

for i in range(2, n):

if not word[i].isupper():

return False

# case 2 and case 3

else:

for i in range(1, n):

if word[i].isupper():

return False

# if pass one of the cases

return True

* Time complexity: O(n)*O*(*n*), where n is the length of the word. We only need to check each char at most constant times.
* Space complexity : O(1)*O*(1). We only need constant spaces to store our variables.

Q. **Longest Uncommon Subsequence I**

Given two strings a and b, return the length of the ***longest uncommon subsequence*** between a and b. If the longest uncommon subsequence does not exist, return -1.

An **uncommon subsequence** between two strings is a string that is a **subsequence of one but not the other**.

A **subsequence** of a string s is a string that can be obtained after deleting any number of characters from s.

* For example, "abc" is a subsequence of "aebdc" because you can delete the underlined characters in "aebdc" to get "abc". Other subsequences of "aebdc" include "aebdc", "aeb", and "" (empty string).

**Example 1:**

**Input:** a = "aba", b = "cdc"

**Output:** 3

**Explanation:** One longest uncommon subsequence is "aba" because "aba" is a subsequence of "aba" but not "cdc".

Note that "cdc" is also a longest uncommon subsequence.

Sol:

class Solution:

def findLUSlength(self, a: str, b: str) -> int:

if a == b:

return -1

else:

return max(len(a),len(b))

Q. Reversed string 2

Given a string s and an integer k, reverse the first k characters for every 2k characters counting from the start of the string.

If there are fewer than k characters left, reverse all of them. If there are less than 2k but greater than or equal to k characters, then reverse the first k characters and left the other as original.

**Example 1:**

**Input:** s = "abcdefg", k = 2

**Output:** "bacdfeg"

**Example 2:**

**Input:** s = "abcd", k = 2

**Output:** "bacd"

Sol:

class Solution:

def reverseStr(self, s: str, k: int) -> str:

a = list(s)

for i in range(0, len(a), 2\*k):

a[i:i+k] = a[i:i+k][::-1]

return "".join(a)

using inbuilt fnction

class Solution:

def reverseStr(self, s: str, k: int) -> str:

s=list(s)

for i in range(0,len(s),2\*k):

# print(s[i:i+k:-1])

s[i:i+k]=reversed(s[i:i+k])

return "".join(s)

Q. Student attendance record

You are given a string s representing an attendance record for a student where each character signifies whether the student was absent, late, or present on that day. The record only contains the following three characters:

* 'A': Absent.
* 'L': Late.
* 'P': Present.

The student is eligible for an attendance award if they meet **both** of the following criteria:

* The student was absent ('A') for **strictly** fewer than 2 days **total**.
* The student was **never** late ('L') for 3 or more **consecutive** days.

Return true if the student is eligible for an attendance award, or false otherwise.

**Example 1:**

**Input:** s = "PPALLP"

**Output:** true

**Explanation:** The student has fewer than 2 absences and was never late 3 or more consecutive days.

**Example 2:**

**Input:** s = "PPALLL"

**Output:** false

**Explanation:** The student was late 3 consecutive days in the last 3 days, so is not eligible for the award.

Sol:

class Solution:

def checkRecord(self, s: str) -> bool:

count=0

for i in range(len(s)):

if s[i]=='A':

count+=1

return count<2 and 'LLL' not in s

Another soln

class Solution:

def checkRecord(self, s: str) -> bool:

return s.count('A')<=1 and not 'LLL' in s

using hash table

class Solution:

def checkRecord(self, s: str) -> bool:

ht = {}

for i in range(len(s)):

ht[s[i]] = ht.get(s[i],0) + 1

if s[i] == "L":

if ht[s[i]] > 2:

return False

if i < len(s) - 1 and s[i+1] != "L":

ht[s[i]] = 0

return ht.get("A",0) <= 1 and ht.get("L",0) <= 2

Q. **Reverse Words in a String III**

Given a string s, reverse the order of characters in each word within a sentence while still preserving whitespace and initial word order.

**Example 1:**

**Input:** s = "Let's take LeetCode contest"

**Output:** "s'teL ekat edoCteeL tsetnoc"

**Example 2:**

**Input:** s = "God Ding"

**Output:** "doG gniD"

Sol:

class Solution:

def reverseWords(self, s: str) -> str:

a = s.split(' ')

res = [i[::-1] for i in a]

op = " ".join(res)

return op

Q. **Minimum Index Sum of Two Lists**

Suppose Andy and Doris want to choose a restaurant for dinner, and they both have a list of favorite restaurants represented by strings.

You need to help them find out their **common interest** with the **least list index sum**. If there is a choice tie between answers, output all of them with no order requirement. You could assume there always exists an answer.

**Example 1:**

**Input:** list1 = ["Shogun","Tapioca Express","Burger King","KFC"], list2 = ["Piatti","The Grill at Torrey Pines","Hungry Hunter Steakhouse","Shogun"]

**Output:** ["Shogun"]

**Explanation:** The only restaurant they both like is "Shogun".

**Example 2:**

**Input:** list1 = ["Shogun","Tapioca Express","Burger King","KFC"], list2 = ["KFC","Shogun","Burger King"]

**Output:** ["Shogun"]

**Explanation:** The restaurant they both like and have the least index sum is "Shogun" with index sum 1 (0+1).

Sol:

class Solution:

def findRestaurant(self, list1: List[str], list2: List[str]) -> List[str]:

l3 = set(list1)&set(list2)

mini,l= 2\*\*31,[]

for i in l3:

sum\_idx = list1.index(i) + list2.index(i)

if mini>sum\_idx:

l=[i]

mini=sum\_idx

elif mini==sum\_idx:

l.append(i)

return l

using dict

from collections import defaultdict

class Solution:

def findRestaurant(self, list1: List[str], list2: List[str]) -> List[str]:

ind1 = {}

ind2 = {}

for i, val in enumerate(list1):

ind1[val] = i

result = defaultdict(list)

min\_idx = float("inf")

for i, val in enumerate(list2):

if val in ind1:

ind = i + ind1[val]

min\_idx = min(min\_idx, ind)

result[ind].append(val)

return result[min\_idx]

Q. Word search

Given an m x n grid of characters board and a string word, return true *if* word *exists in the grid*.The word can be constructed from letters of sequentially adjacent cells, where adjacent cells are horizontally or vertically neighboring. The same letter cell may not be used more than once.

**Example 1:**

Calendar

Description automatically generated

**Input:** board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCCED"

**Output:** true

Sol:

class Solution:

def exist(self, board: List[List[str]], word: str) -> bool:

Rows, Cols = len(board), len(board[0])

path = set() #cant revisit the same char within path twice

def dfs(r,c,i): #nested dfs so we dont have to pass board or word variable

if i == len(word): # if we reach end or we know we found the word

return True

if (r<0 or c<0 or r>=Rows or c>=Cols or word[i]!= board[r][c] or (r,c) in path): # if we go out of bounds of the entire board

return False

path.add((r,c))

res=(dfs(r+1, c, i+1) or dfs(r-1, c, i+1) or dfs(r, c+1, i+1) or dfs(r, c-1, i+1))

path.remove((r,c)) #remove the position we just added to the path bcause we are no longer visiting that position

return res

#brute force, go through every single position in our grid and run dfs function on it

for r in range(Rows):

for c in range(Cols):

if dfs(r,c,0): #0 cause starting position as we are starting from begining

return True

return False

#TC: O(m\*n\* dfs) 4^len(word) = O(m\*n\*4^n)

Q. **Find All Anagrams in a String**

Given two strings s and p, return *an array of all the start indices of*p*'s anagrams in*s. You may return the answer in **any order**.

**Example 1:**

**Input:** s = "cbaebabacd", p = "abc"

**Output:** [0,6]

**Explanation:**

The substring with start index = 0 is "cba", which is an anagram of "abc".

The substring with start index = 6 is "bac", which is an anagram of "abc".

Sol:

class Solution:

def findAnagrams(self, s: str, p: str) -> List[int]:

s\_len, p\_len= len(s), len(p)

if s\_len < p\_len:

return []

s\_count = Counter()

p\_count = Counter(p)

result=[]

for i in range(s\_len):

s\_count[s[i]]+=1

if i >=p\_len:

if s\_count[s[i-p\_len]]==1:

del s\_count[s[i-p\_len]]

else:

s\_count[s[i-p\_len]]-=1

if p\_count == s\_count:

result.append(i-p\_len+1)

return result

* Time complexity: \mathcal{O}(N\_s + N\_p)O(*Ns*​+*Np*​) since it's one pass along both strings.
* Space complexity: \mathcal{O}(1)O(1), because pCount and sCount contain not more than 26 elements.

Q. One edit distance

Given two strings s and t, return true if they are both one edit distance apart, otherwise return false.

A string s is said to be one distance apart from a string t if you can:

* Insert **exactly one** character into s to get t.
* Delete **exactly one** character from s to get t.
* Replace **exactly one** character of s with **a different character** to get t.

**Example 1:**

**Input:** s = "ab", t = "acb"

**Output:** true

**Explanation:** We can insert 'c' into s to get t.

Sol:

class Solution:

def isOneEditDistance(self, s: str, t: str) -> bool:

s\_len, t\_len= len(s), len(t)

if s\_len>t\_len:

return self.isOneEditDistance(t,s)

if t\_len-s\_len> 1:

return False

for i in range(s\_len):

if s[i]!=t[i]:

if s\_len == t\_len:

return s[i+1:] == t[i+1:]

else:

return s[i:] == t[i+1:]

return s\_len + 1 == t\_len

* Time complexity : \mathcal{O}(N)O(*N*) in the worst case when string lengths are close enough abs(ns - nt) <= 1, where N is a number of characters in the longest string. \mathcal{O}(1)O(1) in the best case when abs(ns - nt) > 1.
* Space complexity : \mathcal{O}(N)O(*N*) because strings are immutable in Python and Java and to create substring costs \mathcal{O}(N)O(*N*) space.

My 2 pointer approach:  
Time complexity: O(max(m,n))  
Space complexity: O(1)

class Solution(object):

def isOneEditDistance(self, s, t):

if abs(len(s) - len(t)) > 1 or s == t:

return False

found\_inequality = False

i = j = 0

while i < len(s) and j < len(t):

if s[i] != t[j]:

if found\_inequality: return False

found\_inequality = True

if len(s) < len(t): i -= 1

elif len(s) > len(t): j -= 1

i += 1

j += 1

return True

Q. Decode string

Given an encoded string, return its decoded string.

The encoding rule is: k[encoded\_string], where the encoded\_string inside the square brackets is being repeated exactly k times. Note that k is guaranteed to be a positive integer.

You may assume that the input string is always valid; No extra white spaces, square brackets are well-formed, etc.

Furthermore, you may assume that the original data does not contain any digits and that digits are only for those repeat numbers, k. For example, there won't be input like 3a or 2[4].

**Example 1:**

**Input:** s = "3[a]2[bc]"

**Output:** "aaabcbc"

Sol:

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

class Solution:

def decodeString(self, s: str) -> str:

stack=[["",1]]

num=''

nums = [str(x) for x in range(10)]

for char in s:

if char in nums:

num += char

elif char == '[':

stack.append(["", int(num)])

num = ""

elif char == ']':

string\_, k = stack.pop()

stack[-1][0]+=string\_ \* k

else:

stack[-1][0] += char

return stack[-1][0]

Q. Robot searches for battery  
Given all rooms modeled as a grid (matrix), each position (i, j) represents a room.   
A room could be locked or not locked (accessible or not). An unlocked room could be empty or with a battery.   
Given a robot at position (x, y) with remaining power of T units, check whether the robot can find a battery before out of power.   
Note that the robot can only move top/bottom/left/right to its neighbor and it uses one unit of power for one move.

Q. Related to bash command line: brace expansion 2  
Write a function that takes in a string such and returns an array of strings based on the elements in the bracket:

Input: "file{1,2,3,4,5}.txt"  
output: ["file1.txt", "file2.txt", .......]

Input: "{Jan}month{march}month"  
output: ["Janmonth", "marchmonth"]

Q. Two words are [*anagrams*](https://en.wikipedia.org/wiki/Anagram) of one another if their letters can be rearranged to form the other word.

Given a string, split it into two contiguous substrings of equal length. Determine the minimum number of characters to change to make the two substrings into anagrams of one another.

**Example**

Break  into two parts: 'abc' and 'cde'. Note that all letters have been used, the substrings are contiguous and their lengths are equal. Now you can change 'a' and 'b' in the first substring to 'd' and 'e' to have 'dec' and 'cde' which are anagrams. Two changes were necessary.

**Sample Input**

6

aaabbb

ab

abc

mnop

xyyx

xaxbbbxx

**Sample Output**

3

1

-1

2

0

1

Sol:

from collections import Counter

for \_ in range(int(input())):

s = input()

l = len(s)

if l % 2 == 1:

print(-1)

else:

count = 0

s1, s2 = Counter(s[:l//2]), Counter(s[l//2:])

for char in s2:

current = s2[char] - s1.get(char,0)

if current > 0:

count += current

print(count)

Q. group anagrams

Given an array of strings strs, group **the anagrams** together. You can return the answer in **any order**.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

**Example 1:**

**Input:** strs = ["eat","tea","tan","ate","nat","bat"]

**Output:** [["bat"],["nat","tan"],["ate","eat","tea"]]

Sol:

class Solution:

def groupAnagrams(self, strs: List[str]) -> List[List[str]]:

"""

use dictionary for sorted elemnts

Sort the strs[i] in the list

iterate through list and check if sorted el in dictionary

if it's in dictionary then append it as the value of the el

return the values of the key from dictionary as a list

aet: eat, ate, tea

ant: tan,nat

abt: bat

"""

dict\_of\_anagram={}

for string in strs:

element= ''.join(sorted(string))

if element in dict\_of\_anagram:

dict\_of\_anagram[element].append(string)

else:

dict\_of\_anagram[element] = [string]

return dict\_of\_anagram.values()

class Solution:

def groupAnagrams(self, strs: List[str]) -> List[List[str]]:

ans = collections.defaultdict(list)

for s in strs:

ans[tuple(sorted(s))].append(s)

return ans.values()

* Time Complexity: O(NK \log K)*O*(*NK*log*K*), where N*N* is the length of strs, and K*K* is the maximum length of a string in strs. The outer loop has complexity O(N)*O*(*N*) as we iterate through each string. Then, we sort each string in O(K \log K)*O*(*K*log*K*) time.
* Space Complexity: O(NK)*O*(*NK*), the total information content stored in ans.

Using count

class Solution:

def groupAnagrams(self, strs: List[str]) -> List[List[str]]:

ans = collections.defaultdict(list)

for s in strs:

count = [0]\*26

for cnt in s:

count[ord(cnt) - ord('a')] += 1

ans[tuple(count)].append(s)

return ans.values()

* Time Complexity: O(NK)*O*(*NK*), where N*N* is the length of strs, and K*K* is the maximum length of a string in strs. Counting each string is linear in the size of the string, and we count every string.
* Space Complexity: O(NK)*O*(*NK*), the total information content stored in ans.

Another sol

class Solution:

def groupAnagrams(self, strs: List[str]) -> List[List[str]]:

d = {}

for s in strs:

res=str(''.join(sorted(s)))

if not res in d:

d[res]=[s]

else:

d[res].append(s)

return list(d.values())

Q. Jump game 2

Given an array of non-negative integers nums, you are initially positioned at the first index of the array.

Each element in the array represents your maximum jump length at that position.

Your goal is to reach the last index in the minimum number of jumps.

You can assume that you can always reach the last index.

**Example 1:**

**Input:** nums = [2,3,1,1,4]

**Output:** 2

**Explanation:** The minimum number of jumps to reach the last index is 2. Jump 1 step from index 0 to 1, then 3 steps to the last index.

**Example 2:**

**Input:** nums = [2,3,0,1,4]

**Output:** 2

Sol:

A picture containing text

Description automatically generated

class Solution:

def jump(self, nums: List[int]) -> int:

jumps = 0

start=0

end=0

while end<= len(nums)-1:

max\_jump=0

for i in range(start, end+1):

max\_jump = max(max\_jump, i+nums[i])

start = end+1

end = max\_jump

jumps+=1

return jumps

TC : O(n)

Sc: O(1)

Se

Q. **Course Schedule II**

There are a total of numCourses courses you have to take, labeled from 0 to numCourses - 1. You are given an array prerequisites where prerequisites[i] = [ai, bi] indicates that you **must** take course bi first if you want to take course ai.

* For example, the pair [0, 1], indicates that to take course 0 you have to first take course 1.

Return *the ordering of courses you should take to finish all courses*. If there are many valid answers, return **any** of them. If it is impossible to finish all courses, return **an empty array**.

**Example 1:**

**Input:** numCourses = 2, prerequisites = [[1,0]]

**Output:** [0,1]

**Explanation:** There are a total of 2 courses to take. To take course 1 you should have finished course 0. So the correct course order is [0,1].

Sol:

#1 build an adjacency list of prereqs

#2 iterate through course prereq pair in our prereq list

#3 append pre to prereq[crs]

#4 create o/p list, have two sets, visit and cycle sets, (to check if crs is already vsisted)

#5 define dfs function

#6 to check if cycle, if crs in cycle set. If cycle return False

If crs in visit, return true

Add crs to cycle

#7 go through every prereq of this course

Run dsf on the pre and if it returns false, we know we just detected a cycle

So return false

If it return true we continue to go through all pre and run dfs

#8 remove crs from cycle because as it’s no longer along the path that we are going and also add it to visit as we just went through this crs and all its prereqs

#9 and since it is visited we can finally append it to our output, which was our ultimate goal. Since we added it to our output we know that it’s a crs we are allowed to take and we can return True not False

#10 now to execute this code we have to go through every course in the given number of courses, we’ll run dfs on evry single course.

But we have to check if the return value of any of them happens to be false, what that means is we detected a cycle and we are forced to return an empty list. We are not gonna return output bcause we cant take the given courses.

But if is not false we will return output

class Solution:

def findOrder(self, numCourses: int, prerequisites: List[List[int]]) -> List[int]:

prereq= { c:[] for c in range(numCourses)}

for crs, pre in prerequisites:

prereq[crs].append(pre)

output=[]

visit, cycle = set(), set()

def dfs(crs):

if crs in cycle:

return False

if crs in visit:

return True

cycle.add(crs)

for pre in prereq[crs]:

if dfs(pre) == False:

return False

cycle.remove(crs)

visit.add(crs)

output.append(crs)

return True

for c in range(numCourses):

if dfs(c) == False:

return []

return output

TC: O(E+V) e: edges, visits or nodes

i.e O(p+n)

Q. **Insert Delete GetRandom O(1)**

Implement the RandomizedSet class:

* RandomizedSet() Initializes the RandomizedSet object.
* bool insert(int val) Inserts an item val into the set if not present. Returns true if the item was not present, false otherwise.
* bool remove(int val) Removes an item val from the set if present. Returns true if the item was present, false otherwise.
* int getRandom() Returns a random element from the current set of elements (it's guaranteed that at least one element exists when this method is called). Each element must have the **same probability** of being returned.

You must implement the functions of the class such that each function works in **average** O(1) time complexity.

**Input**

["RandomizedSet", "insert", "remove", "insert", "getRandom", "remove", "insert", "getRandom"]

[[], [1], [2], [2], [], [1], [2], []]

**Output**

[null, true, false, true, 2, true, false, 2]

Sol:

class RandomizedSet:

def \_\_init\_\_(self):

"""

Initialize your data structure here.

"""

self.dict={}

self.list=[]

def insert(self, val: int) -> bool:

"""

Inserts a value to the set. Returns true if the set did not already contain the specified element.

"""

if val in self.dict:

return False

self.dict[val]= len(self.list)

self.list.append(val)

return True

def remove(self, val: int) -> bool:

"""

Removes a value from the set. Returns true if the set contained the specified element.

"""

if val in self.dict:

last\_element, index = self.list[-1], self.dict[val]

self.list[index], self.dict[last\_element] = last\_element, index

self.list.pop()

del self.dict[val]

return True

return False

def getRandom(self) -> int:

"""

Get a random element from the set.

"""

return choice(self.list)

# Your RandomizedSet object will be instantiated and called as such:

# obj = RandomizedSet()

# param\_1 = obj.insert(val)

# param\_2 = obj.remove(val)

# param\_3 = obj.getRandom()

* Time complexity. GetRandom is always \mathcal{O}(1)O(1). Insert and Delete both have \mathcal{O}(1)O(1) average time complexity, and \mathcal{O}(N)O(*N*) in the worst-case scenario when the operation exceeds the capacity of currently allocated array/hashmap and invokes space reallocation.
* Space complexity: \mathcal{O}(N)O(*N*), to store N elements.

**Insert**

Add value -> its index into dictionary, average \mathcal{O}(1)O(1) time.

Append value to array list, average \mathcal{O}(1)O(1) time as well.

**Delete**

Retrieve an index of element to delete from the hashmap.

Move the last element to the place of the element to delete, \mathcal{O}(1)O(1) time.

Pop the last element out, \mathcal{O}(1)O(1) time.

**GetRandom**

GetRandom could be implemented in \mathcal{O}(1)O(1) time with the help of standard random.choice in Python

Q. **Longest Substring with At Most K Distinct Characters**

Given a string s and an integer k, return *the length of the longest substring of* s *that contains at most* k ***distinct****characters*.

**Example 1:**

**Input:** s = "eceba", k = 2

**Output:** 3

**Explanation:** The substring is "ece" with length 3.

Text

Description automatically generated

Text

Description automatically generated

Sol:

class Solution:

def lengthOfLongestSubstringKDistinct(self, s: str, k: int) -> int:

n = len(s) #store the length of the string in variable n

if n\*k == 0: #also let's take care of the edge case , if either of the inputs or both the inputs are zero it will return zero

return 0

l,r=0,0 #now lets set up our pointer variable whci help us generate teh strings, left is the left side of the window and r is r side

hashmp= defaultdict() # initialize the hashmap for key values, when we itetrate, th ekey will stor the specific character in theinput string and the value is the index

max\_len = 1 # any char in string is valid substring

while r < n: #right pointer which opens the sliding window is going to iterate till the end of the string

hashmp[s[r]]=r # everytime it moves to a new position we are going to put the new chararcter in the map

r+=1 #increement

if len(hashmp) == k+1: # now lets look at the constrainst, we know tht our substring can only have a lenghth of k, but we are constantly putting a new character into the map each time we iterate, so we'll check the hashmap size when it is k+1 with an if statement

del\_idx = min(hashmp.values()) #we are going to find the smallest number out of the values in the map, this smallest number represents the index of the character of the input string that we want to delete from our map

del hashmp[s[del\_idx]]

l = del\_idx+1 #set the left pointer equal to the dletion index we just used + 1

max\_len = max(max\_len, r-l)

return max\_len

* Time complexity : \mathcal{O}(N)O(*N*) in the best case of k distinct characters in the string and \mathcal{O}(N k)O(*Nk*) in the worst case of N distinct characters in the string.
* Space complexity : \mathcal{O}(k)O(*k*) since additional space is used only for a hashmap with at most k + 1 elements.

Diagram

Description automatically generated

Q. **Reconstruct Itinerary**

You are given a list of airline tickets where tickets[i] = [fromi, toi] represent the departure and the arrival airports of one flight. Reconstruct the itinerary in order and return it.

All of the tickets belong to a man who departs from "JFK", thus, the itinerary must begin with "JFK". If there are multiple valid itineraries, you should return the itinerary that has the smallest lexical order when read as a single string.

* For example, the itinerary ["JFK", "LGA"] has a smaller lexical order than ["JFK", "LGB"].

You may assume all tickets form at least one valid itinerary. You must use all the tickets once and only once.

**Input:** tickets = [["MUC","LHR"],["JFK","MUC"],["SFO","SJC"],["LHR","SFO"]]

**Output:** ["JFK","MUC","LHR","SFO","SJC"]

Sol:

class Solution:

def findItinerary(self, tickets: List[List[str]]) -> List[str]:

self.adj={}

tickets.sort(key = lambda x: x[1]) #sort for lexicographically first

for u, v in tickets:

if u in self.adj: self.adj[u].append(v)

else: self.adj[u]=[v]

self.result = []

self.dfs("JFK")

return self.result[::-1]

def dfs(self,s):

while s in self.adj and len(self.adj[s]) >0:

v = self.adj[s][0]

self.adj[s].pop(0)

self.dfs(v)

self.result.append(s)

s = Solution()

tickets = [["MUC","LHR"],["JFK","MUC"],["SFO","SJC"],["LHR","SFO"]]

print(s.findItinerary(tickets))

Another sol:

class Solution(object):

def findItinerary(self, tickets):

"""

:type tickets: List[List[str]]

:rtype: List[str]

"""

from collections import defaultdict

self.flightMap = defaultdict(list)

for ticket in tickets:

origin, dest = ticket[0], ticket[1]

self.flightMap[origin].append(dest)

# sort the itinerary based on the lexical order

for origin, itinerary in self.flightMap.items():

# Note that we could have multiple identical flights, i.e. same origin and destination.

itinerary.sort(reverse=True)

self.result = []

self.DFS('JFK')

# reconstruct the route backwards

return self.result[::-1]

def DFS(self, origin):

destList = self.flightMap[origin]

while destList:

#while we visit the edge, we trim it off from graph.

nextDest = destList.pop()

self.DFS(nextDest)

self.result.append(origin)

Time Complexity: \mathcal{O}(|E| \log{\frac{|E|}{|V|}})O(∣*E*∣log∣*V*∣∣*E*∣​) where |E|∣*E*∣ is the number of edges (flights) in the input.

* As one can see from the above algorithm, during the DFS process, we would traverse each edge once. Therefore, the complexity of the DFS function would be |E|∣*E*∣.
* However, before the DFS, we need to sort the outgoing edges for each vertex. And this, unfortunately, dominates the overall complexity.

Q. Decode Ways

A message containing letters from A-Z can be **encoded** into numbers using the following mapping:

'A' -> "1"

'B' -> "2"

...

'Z' -> "26"

To **decode** an encoded message, all the digits must be grouped then mapped back into letters using the reverse of the mapping above (there may be multiple ways). For example, "11106" can be mapped into:

* "AAJF" with the grouping (1 1 10 6)
* "KJF" with the grouping (11 10 6)

Note that the grouping (1 11 06) is invalid because "06" cannot be mapped into 'F' since "6" is different from "06".

Given a string s containing only digits, return the ***number*** of ways to ***decode*** it.

The answer is guaranteed to fit in a **32-bit** integer.

**Example 1:**

**Input:** s = "12"

**Output:** 2

**Explanation:** "12" could be decoded as "AB" (1 2) or "L" (12).

**Example 2:**

**Input:** s = "226"

**Output:** 3

**Explanation:** "226" could be decoded as "BZ" (2 26), "VF" (22 6), or "BBF" (2 2 6).

Text

Description automatically generated

Calendar

Description automatically generated with medium confidence

Chart

Description automatically generated

A screen shot of a computer

Description automatically generated with low confidence

A screenshot of a computer

Description automatically generated with medium confidence

Memorization reduces TC to O(N)

Sol:

class Solution:

def numDecodings(self, s: str) -> int:

def dfs(index, map):

if index == len(s): #end of string

return 1

if s[index] == '0': #if we find 0 in string return 0

return 0

if index in map: #memoization : if index already in map, simply return value at index, will give ways to decode string at that particular index and don’t have to check again n again.

return map[index]

if index+2 <= len(s) and int(s[index:index+2])<= 26: #for double digit index+2, the length has to be less than equal to string and the value of the double digit has to be less than 26

single = dfs(index+1, map) #two variables, single for single digit.

double = dfs(index+2, map) # passing params index+1&2 for single & double &map which is empty dictionary

map[index] = single+double #add values obtained to map

else:

map[index] = dfs(index+1, map) #if > 26 then only add single

return map[index] #simply return map

map={}

return dfs(0, map) #return function with params 0 as index and empty map

* Time Complexity: O(N)*O*(*N*), where N*N* is length of the string. Memoization helps in pruning the recursion tree and hence decoding for an index only once. Thus this solution is linear time complexity.
* Space Complexity: O(N)*O*(*N*). The dictionary used for memoization would take the space equal to the length of the string. There would be an entry for each index value. The recursion stack would also be equal to the length of the string.

using dp

Text, timeline

Description automatically generated

class Solution:

def numDecodings(self, s: str) -> int:

n = len(s)

dp=[0 for i in range(n)]

if s[0]!="0":

dp[0]=1

for i in range(1,n):

x=int(s[i])

y=int(s[i-1:i+1])

if x>=1 and x<=9:

dp[i]+=dp[i-1]

if y>=10 and y<=26:

if i-2>=0:

dp[i]+=dp[i-2]

else:

dp[i]+=1

return dp[-1] #last value in dp

* Time Complexity: O(N)*O*(*N*), where N*N* is length of the string. We iterate the length of dp array which is N+1*N*+1.
* Space Complexity: O(N)*O*(*N*). The length of the DP array.

Q. LRU cache

class LRUCache:

def \_\_init\_\_(self, capacity: int):

self.dict = {}

self.capacity = capacity

def get(self, key: int) -> int:

if key not in self.dict:

return -1

val = self.dict.pop(key) #Remove it first before inserting it at the end again

self.dict[key] = val

return val

def put(self, key: int, value: int) -> None:

if key in self.dict:

self.dict.pop(key)

else:

if len(self.dict) == self.capacity:

del self.dict[next(iter(self.dict))] # The **Python iter**() function returns an iterator for the given object. The **iter**() function creates an object which can be iterated one element at a time.

self.dict[key] = value

# Your LRUCache object will be instantiated and called as such:

# obj = LRUCache(capacity)

# param\_1 = obj.get(key)

# obj.put(key,value)

Q. cheapest flight k stops

**Input:** n = 3, flights = [[0,1,100],[1,2,100],[0,2,500]], src = 0, dst = 2, k = 1

**Output:** 200

**Explanation:** The graph is shown.

The cheapest price from city 0 to city 2 with at most 1 stop costs 200, as marked red in the picture.

from collections import defaultdict  
import heapq  
class Solution:

def findCheapestPrice(self, n: int, flights: List[List[int]], src: int, dst: int, K: int) -> int:

flight\_list = defaultdict(list) #store the flight destination and cost

for s,d,c in flights:

flight\_list[s].append([c,d])

search\_list = [(0,src,0)] #starting position

visited = defaultdict(int)

while(search\_list):

distance, curr, stops = heapq.heappop(search\_list) #always pop the one with smallest cost(i named it as distance, it doesnt matter)

if curr == dst and stops<=K+1:

return distance

elif stops > K+1:

continue

if curr not in visited or visited[curr]>stops:

visited[curr] = stops

for cost, des in flight\_list[curr]: #for every destination it connects, push into the heap queue

if des not in visited or visited[des]>stops:

heapq.heappush(search\_list, (distance+cost,des, stops+1))

return -1

Q. Minimum area rectangle

Similar idea as above, but here we take (x1, y1) as lower left and try to find the upper right (x2, y2) using (x1, y2) and (x2, y1). We can sort of treat all x-coordinates at the same y-coordinate as a 'row' and all y-coordinates at the same x-coordinate as a 'column'. Use x1 to see what y2s are in this column, and y1 to see what x2s are in this row. The trick is that once we find a valid (x2, y2) combo, it *must* be the smallest rectangle for that (x1, y1), (x2, y2). Break from iterating through y2s and move on to the next x2. This has a huge time savings for these test cases despite a (roughly?) equivalent time complexity. I'm actually not 100% sure how to evaluate time complexity here (we pick up O(NlogN) from sorting and N binary searches), but I'm assuming it is still quadratic in the worst case.  
O(N^2) @ 370ms, beat 94%.

def minAreaRect(self, points: List[List[int]]) -> int:

# Collect all y coordinates per x coordinate into a list (and vice versa).

x\_to\_y = collections.defaultdict(list)

y\_to\_x = collections.defaultdict(list)

for x, y in points:

x\_to\_y[x].append(y)

y\_to\_x[y].append(x)

# Sort all lists.

for x, y\_list in x\_to\_y.items():

y\_list.sort()

for y, x\_list in y\_to\_x.items():

x\_list.sort()

# For each x1, y1 in points,

points = set([tuple(point) for point in points])

smallest = float('inf')

for x1, y1 in points:

# Get all y2 coordinates for this x1 (and vice versa).

y\_list = x\_to\_y[x1]

x\_list = y\_to\_x[y1]

# But only consider the y2 coordinates that are greater than y1.

# Meaning, lets only consider rectangles from lower left to upper right.

y\_idx = bisect.bisect\_right(y\_list, y1)

x\_idx = bisect.bisect\_right(x\_list, x1)

ys\_above = y\_list[y\_idx:]

xs\_right = x\_list[x\_idx:]

for x2 in xs\_right:

for y2 in ys\_above:

# Here, we know (x1, y2) and (y1, x2) are points because they were

# in x\_to\_y and y\_to\_x. If (x2, y2) is a point, we have a rectangle.

if (x2, y2) in points:

smallest = min(smallest, (x2 - x1) \* (y2 - y1))

# Key to efficiency: Because the lists were sorted, we have found

# the smallest rectangle for this (x2, y2). Move to next x2.

break

if smallest <= (x2 - x1) \* (y2 - y1):

break

return smallest if smallest != float('inf') else 0

Q. shortest bridge

**Input:** grid = [[0,1],[1,0]]

**Output:** 1

# TIme = O(m x n)

# Space = O(m x n)

from collections import deque

class Solution:

def shortestBridge(self, grid: List[List[int]]) -> int:

# data structure

q = deque()

self.dirs = [[1,0],[-1,0],[0,1],[0,-1]]

steps = 0

# Step1) Find first island.

for i in range(len(grid)):

found = False

for j in range(len(grid[0])):

if grid[i][j] == 1:

self.dfs(grid, i, j, q)

found = True

break

if found:

break

while q:

size = len(q)

for \_ in range(size):

i, j = q.popleft()

for pair in self.dirs:

r = i + pair[0]

c = j + pair[1]

if 0 <= r < len(grid) and 0 <= c < len(grid[0]):

if grid[r][c] == 1:

# we found the second bridge

return steps

if grid[r][c] == 2:

continue

grid[r][c] = 2

q.append((r, c))

steps += 1

return steps

def dfs(self, grid, i, j, q):

# base case

# logic

grid[i][j] = 2

q.append((i,j))

for pair in self.dirs:

r = i + pair[0]

c = j + pair[1]

if 0 <= r < len(grid) and 0 <= c < len(grid[0]) and grid[r][c] == 1:

self.dfs(grid, r, c, q)

Q. 4 sum

We can implement k - 2 loops using a recursion. We will pass the starting point and k as the parameters. When k == 2, we will call twoSum, terminating the recursion.

1. For the main function:
   * Sort the input array nums.
   * Call kSum with start = 0, k = 4, and target, and return the result.
2. For kSum function:
   * Check if the sum of k smallest values is greater than target, or the sum of k largest values is smaller than target. Since the array is sorted, the smallest value is nums[start], and largest - the last element in nums.
     + If so, no need to continue - there are no k elements that sum to target.
   * If k equals 2, call twoSum and return the result.
   * Iterate i through the array from start:
     + If the current value is the same as the one before, skip it.
     + Recursively call kSum with start = i + 1, k = k - 1, and target - nums[i].
     + For each returned set of values:
       - Include the current value nums[i] into set.
       - Add set to the result res.
   * Return the result res.
3. For twoSum function:
   * Set the low pointer lo to start, and high pointer hi to the last index.
   * While low pointer is smaller than high:
     + If the sum of nums[lo] and nums[hi] is less than target, increment lo.
       - Also increment lo if the value is the same as for lo - 1.
     + If the sum is greater than target, decrement hi.
       - Also decrement hi if the value is the same as for hi + 1.
     + Otherwise, we found a pair:
       - Add it to the result res.
       - Decrement hi and increment lo.
   * Return the result res.

Sol:

class Solution:

def fourSum(self, nums: List[int], target: int) -> List[List[int]]:

def kSum(nums: List[int], target: int, k: int) -> List[List[int]]:

res = []

if len(nums) == 0 or nums[0] \* k > target or target > nums[-1] \* k:

return res

if k == 2:

return twoSum(nums, target)

for i in range(len(nums)):

if i == 0 or nums[i - 1] != nums[i]:

for \_, set in enumerate(kSum(nums[i + 1:], target - nums[i], k - 1)):

res.append([nums[i]] + set)

return res

def twoSum(nums: List[int], target: int) -> List[List[int]]:

res = []

lo, hi = 0, len(nums) - 1

while (lo < hi):

sum = nums[lo] + nums[hi]

if sum < target or (lo > 0 and nums[lo] == nums[lo - 1]):

lo += 1

elif sum > target or (hi < len(nums) - 1 and nums[hi] == nums[hi + 1]):

hi -= 1

else:

res.append([nums[lo], nums[hi]])

lo += 1

hi -= 1

return res

nums.sort()

return kSum(nums, target, 4)

class Solution:

def fourSum(self, nums: List[int], target: int) -> List[List[int]]:

# First sort the array

nums = sorted(nums)

# A set to store the quadruplets

quadruplets = set()

# Store the length of 'nums' to 'lenNums'

lenNums = len(nums)

# Initialize values

i = 0

j = i+1

# Traverse till the 4th last element

while i < lenNums-3:

# print(i)

j = i+1

# For each value of 'i' traverse from i+1 to the 3rd last element

while j < lenNums-2:

# For each pair value of 'i' & 'j', take two pointers at j+1 and at lenNums-1

low = j+1

high = lenNums-1

# Traverse until 'low' is less than 'high'

while low < high:

# Store the sum of the quadruplets

fourSum = nums[i] + nums[j] + nums[low] + nums[high]

# IF the summed value is equal to 'target'

if fourSum == target:

# Then add the quadruplet to the list

quadruplets.add(tuple(sorted([nums[i], nums[j], nums[low], nums[high]])))

# Check for duplicates and increase 'low'

while low < high and nums[low] == nums[low+1]:

low += 1

# Check for duplicates and decrease 'high'

while high > low and nums[high] == nums[high-1]:

high -= 1

# Change both the values of 'low' & 'high'

low += 1

high -= 1

# ELSE IF the summed value is greater then 'target', decrease 'high'

elif fourSum > target:

high -= 1

# ELSE, increase 'low'

else:

low += 1

# END while

# Check for duplicates

while j < lenNums-2 and nums[j] == nums[j+1]:

j += 1

j += 1

# END while

# Check for duplicates

while i < lenNums-3 and nums[i] == nums[i+1]:

i += 1

i += 1

# END while

return quadruplets

'''

Sample Inputs:

[1,0,-1,0,-2,2]

0

[0,0,0,0]

0

[1,5,-5,9,-7,2]

3

[-3,-2,-1,0,0,1,2,3]

0

[0,0,1,2,4,7,9]

5

[-1,0,1,2,-1,-4]

-1

Expected Outputs:

[[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]

[[0,0,0,0]]

[[-5,1,2,5]]

[[-3,-2,2,3],[-3,-1,1,3],[-3,0,0,3],[-3,0,1,2],[-2,-1,0,3],[-2,-1,1,2],[-2,0,0,2],[-1,0,0,1]]

[[0,0,1,4]]

[[-4,0,1,2],[-1,-1,0,1]]

'''

Q. brace expansion 2

Recursively iterate through expression handling 4 cases, when expression[i] is:

1. An opening bracket { - **distribute** our current output with the inner expression.
2. A comma , - **concatenate** our current output with inner expression.
3. A closing bracket } or we reach end of the expression - return output.
4. A lowercase letter - build up the word until we run into one of the previous 3 cases. We should just be able to add this to output and keep going, except when we have something like {a,b,c}d. When we get to d we have to distribute, so we need to see if the previous character was a closing bracket.

When **distributing**, we have to keep an output set within that recursive call and overwrite it. This keeps things grouped as they should.

When **concatenating**, each letter/word gets its own recursive call because thats easier.

Using a "global" i is a bit annoying but IMO its useful here since this can become deeply recursive and we only want to process each character once.

class Solution:

def braceExpansionII(self, expression: str) -> List[str]:

def distribute(L1, L2):

if L1 and L2:

output = set()

for elem1 in L1:

for elem2 in L2:

output.add(elem1 + elem2)

return output

else:

return L1 or L2

def build\_set():

nonlocal i

output = set()

while i < len(expression):

if expression[i].isalpha():

prev = expression[i-1] if i != 0 else None

word = ""

while i < len(expression) and expression[i].isalpha():

word += expression[i]

i += 1

if prev == "}":

output = distribute(output, {word})

else:

output.add(word)

elif expression[i] == ",":

i += 1

return output | build\_set()

elif expression[i] == "{":

i += 1

output = distribute(output, build\_set())

else:

i += 1

return output

return output

i = 0

return sorted(list(build\_set()))

s = Solution()

expression = "{Jan}month,{march}month"

print(s.braceExpansionII(expression))

Time Complexity of parsing is O(N), sorting is O(NlogN).

Q. word search 2

board = [["o","a","a","n"],["e","t","a","e"],["i","h","k","r"],["i","f","l","v"]], words = ["oath","pea","eat","rain"]

def findWords(self, board: List[List[str]], words: List[str]) -> List[str]:

self.board = board

self.m = len(board)

self.n = len(board[0])

self.word = words

found = []

def word\_search(word):

for row in range(self.m):

for col in range(self.n):

if self.search(row, col, word, 0):

found.append(word)

return

for word in words:

word\_search(word)

return found

def search(self, row, col, word, w\_idx):

if not self.board[row][col] == word[w\_idx]:

return False

if w\_idx == len(word) - 1:

return True

self.board[row][col] = '#'

ret = False

surrounds = [(row-1, col), (row, col+1), (row+1, col), (row, col-1)]

for i, j in surrounds:

if i>=0 and i <self.m and j >= 0 and j < self.n:

ret = self.search(i, j, word, w\_idx+1)

if ret:

break

self.board[row][col] = word[w\_idx]

return ret

2 sum

class Solution:  
def twoSum(self, nums: List[int], target: int) -> List[int]:  
res = []  
for i in range(len(nums) - 1):  
for j in range(i + 1,len(nums)):  
if nums[i] + nums[j] == target:  
res.append(i)  
res.append(j)  
return res

class Solution:

def twoSum(self, nums: List[int], target: int) -> List[int]:

hashmp={}

for i, n in enumerate(nums):

diff= target-n

if diff not in hashmp:

hashmp[n]=i

else:

return [hashmp[diff], i]

''''

Machine is delivery shirt @ 0.3 seconds ( M seconds) , it's making a new shirt

Workers[N] 1, 1.5 , 2 , 2.5 , 1 , 1.5 amount of seconds each work takes

Worker # 1 , 1 second to do the job

Worker #2 takes 1.5 seconds to the job

Worker # 3 , 2 second to do the job

Worker #4 takes 2.5 seconds to the job

If N= 5

Worker # 5 , 1 second to do the job

...

W= 1, 1.5 , 2 , 2.5 , 1 , 1.5

0.3--|0.6

W= 1

W2= 1.5

W= 1, 1, 2, 1.5, 1.5, 2.5

0.3, 0.6, 0.9, 1.2,

hash = 1:0.3

Worker\_duration[] = { 1, 1.5 , 2 , 2.5}

Worker\_time\_remaining = {Worker\_duration[0],}

Clock += 0.1 second

while 1 :

    if(FindFreeWorker() == true) :

        Assign the task to free worker()

    else :

        Pause the machine()

    Check if Worker Done with the job()

    for i = 0 to range of Workers

        Woreker\_time\_remaining[i] -= 0.1 // till it is not zero

Write a program which dispatches the shirt making to the workers

and if all the workers are busy then it pauses the machine

All workers are equal , so no preference to the worker

Print Shirt ## being worked by Worker ##

Print "Pausing machine"

Print "Resuming machine"

N = 3 , N= 4

M = 0.3 seconds

'''

**def** find\_free\_worker(n,m):

*# for i in n:*

*#     if n[i]==1*

*# else:*

    worker\_duration = [x **for** x **in** **range**(n)]

    work\_remaining = 0

**for** i **in** **range**(n):

        work\_remaining[i] = worker\_duration[]

    clock = 0.3

**for** i **in** n:

**while** n[i]==1:

            work\_remaining

Q. reverse bits

Reverse bits of a given 32 bits unsigned integer.

**Input:** n = 00000010100101000001111010011100

**Output:** 964176192 (00111001011110000010100101000000)

**Explanation:** The input binary string **00000010100101000001111010011100** represents the unsigned integer 43261596, so return 964176192 which its binary representation is **00111001011110000010100101000000**.

Text

Description automatically generated

Start from last digit i.e 0

And output will start with value 0

We need to have a way to check if bit has value of 1

So we use &

Eg 0&1=0, 1&1=1

Perform left shift for output

And right shift for input i.e add 0 to start

Go to next last elemt and check if value 1

Next we come accrors 1&1 =1

So left shift will have value 0 in output but we encounter 1 so we add 1 in output

Keep going till we reach first i.e 32 times

Sol:

class Solution:

def reverseBits(self, n: int) -> int:

output = 0

for \_ in range(0,32):

output<<=1

if n&1:

output+=1

n>>=1

return output

* Time Complexity: \mathcal{O}(1)O(1). Though we have a loop in the algorithm, the number of iteration is fixed regardless the input, since the integer is of fixed-size (32-bits) in our problem.

Q. **Number of 1 Bits**

Write a function that takes an unsigned integer and returns the number of '1' bits it has (also known as the [Hamming weight](http://en.wikipedia.org/wiki/Hamming_weight)).

**Input:** n = 00000000000000000000000000001011

**Output:** 3

**Explanation:** The input binary string **00000000000000000000000000001011** has a total of three '1' bits.

Sol:

class Solution:

def hammingWeight(self, n: int) -> int:

count = 0

x = 1

for \_ in range(0,32):

if n&x:

count+=1

x<<=1

return count

the time complexity is O(1)*O*(1).

The space complexity is O(1)*O*(1), since no additional space is allocated.

Q. longest palindrome substring

s = "babad"

**Output:** "bab"

**Note:** "aba" is also a valid answer

Two pointer approach

Graphical user interface, text, application

Description automatically generated

b a b a d

|--|

|-----|

|---------|

|-------------|

b a b a d

1

|------|

|-------------|

|-------|

|-------------|

"""

Sol:

class Solution:

def longestPalindrome(self, s: str) -> str:

def helper(l,r):

while l>=0 and r < len(s) and s[l]==s[r]:

l -= 1

r += 1

return s[l+1:r]

res=''

for i in range(len(s)):

check = helper(i,i)

if len(check) > len(res):

res= check

check = helper(i,i+1)

if len(check) > len(res):

res= check

return res

Q. **Remove Vowels from a String**

Given a string s, remove the vowels 'a', 'e', 'i', 'o', and 'u' from it, and return the new string.

**Example 1:**

**Input:** s = "leetcodeisacommunityforcoders"

**Output:** "ltcdscmmntyfrcdrs"

Sol:

class Solution:

def removeVowels(self, s: str) -> str:

vowels="aeiouAEIOU"

res=""

for i in s:

if i not in vowels:

res += i

return res

Q. **Longest Substring Without Repeating Characters**

Given a string s, find the length of the **longest substring** without repeating characters.

**Example 1:**

**Input:** s = "abcabcbb"

**Output:** 3

**Explanation:** The answer is "abc", with the length of 3.

**Sol:**

**class Solution:**

**def lengthOfLongestSubstring(self, s: str) -> int:**

**d = {}**

**i = 0**

**res = 0**

**for j in range(0, len(s)):**

**if s[j] in d:**

**i = max(i, d[s[j]]+1)**

**res = max(res, (j - i + 1))**

**d[s[j]] = j**

**return res**

**TC: O(N)**

**Q. Happy number**

Write an algorithm to determine if a number n is happy.

A **happy number** is a number defined by the following process:

* Starting with any positive integer, replace the number by the sum of the squares of its digits.
* Repeat the process until the number equals 1 (where it will stay), or it **loops endlessly in a cycle** which does not include 1.
* Those numbers for which this process **ends in 1** are happy.

Return true if n is a happy number, and false if not.

**Example 1:**

**Input:** n = 19

**Output:** true

**Explanation:**

12 + 92 = 82

82 + 22 = 68

62 + 82 = 100

12 + 02 + 02 = 1

**Letter

Description automatically generated**

**Text, letter

Description automatically generated**

**Sol**

**:**

**class Solution:**

**def isHappy(self, n: int) -> bool:**

**def sum(num):**

**res=0**

**while num>0:**

**r = num%10**

**res = res + r\*r**

**num = num//10**

**# print(res)**

**return res**

**seen= set()**

**while sum(n) not in seen:**

**sum1 = sum(n)**

**if sum1 == 1:**

**return True**

**else:**

**seen.add(sum1)**

**n=sum1**

**return False**

**Q. Longest Increasing Subsequence**

iven an integer array nums, return the length of the longest strictly increasing subsequence.

A **subsequence** is a sequence that can be derived from an array by deleting some or no elements without changing the order of the remaining elements. For example, [3,6,2,7] is a subsequence of the array [0,3,1,6,2,2,7].

**Example 1:**

**Input:** nums = [10,9,2,5,3,7,101,18]

**Output:** 4

**Explanation:** The longest increasing subsequence is [2,3,7,101], therefore the length is 4.

**Diagram

Description automatically generated**

**A screenshot of a computer

Description automatically generated with low confidence**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Sol:**

**class Solution:**

**def lengthOfLIS(self, nums: List[int]) -> int:**

**LIS = [1]\* len(nums) #**

**for i in range(len(nums)-1,-1,-1):**

**for j in range(i+1, len(nums)):**

**if nums[i] < nums[j]:**

**LIS[i] = max(LIS[i], 1+ LIS[j])**

**return max(LIS)**

**TC: O(N2)**

**Q. Factorial Trailing Zeroes**

Given an integer n, return *the number of trailing zeroes in n!*.

**Follow up:**Could you write a solution that works in logarithmic time complexity?

**Example 1:**

**Input:** n = 3

**Output:** 0

**Explanation:** 3! = 6, no trailing zero.

**Example 2:**

**Input:** n = 5

**Output:** 1

**Explanation:** 5! = 120, one trailing zero.

**Compute the factorial**

Recall that if a number has a zero on the end of it, then it is divisible by 1010. Dividing by 1010 will remove that zero, and shift all the other digits to the right by one place. We can, therefore, count the number of zeroes by repeatedly checking if the number is divisible by 1010, and if it is then dividing it by 1010. The number of divisions we're able to do is equal to the number of 0's on the end of it.

define function factorial(n):

n\_factorial = 1

for i from 1 to n (inclusive):

n\_factorial = n\_factorial \* i

return n\_factorial

define function zero\_count(x):

zero\_count = 0

while x is divisible by 10:

zero\_count += 1

x = x / 10

return zero\_count

**Sol:**

**This method works but exceeds time limit in leetcode**

**class Solution:**

**def trailingZeroes(n):**

**fact = 1**

**for i in range(2, n+1):**

**fact \*= i**

**count=0**

**while fact%10 == 0:**

**count+=1**

**fact//=10**

**return count**

**n = 5**

**print(trailingZeroes(n))**

Time complexity : Worse than O(n ^ 2)*O*(*n*2).

Space complexity : O(\log \, n!) = O(n \, \log \, n)*O*(log*n*!)=*O*(*n*log*n*).

**2nd slow method**

**def trailingZeroes(self, n: int) -> int:**

**zero\_count = 0**

**for i in range(5, n + 1, 5):**

**current = i**

**while current % 5 == 0:**

**zero\_count += 1**

**current //= 5**

**return zero\_count**

Alternatively, instead of dividing by 55 each time, we can check each power of 55 to count how many times 55 is a factor. This works by checking if i is divisible by 55, then 2525, then 125125, etc. We stop when this number does not divide into i without leaving a remainder. The number of times we can do this is equivalent to the number of 55 factors in i.

**def trailingZeroes(self, n: int) -> int:**

**zero\_count = 0**

**for i in range(5, n + 1, 5):**

**power\_of\_5 = 5**

**while i % power\_of\_5 == 0:**

**zero\_count += 1**

**power\_of\_5 \*= 5**

**return zero\_count**

Time complexity : O(n)*O*(*n*).

Space complexity : O(1)*O*(1).

Another sol:

class Solution:

def trailingZeroes(self, n: int) -> int:

# iteration -- since all zeros come from 2\*5, so we count how many 2 and 5 in given n. since we have enough 2, so we only need to count how many 5 we have in n.

res = 0

while n > 0:

n //= 5

res += n

return res

Q. **Second Highest Salary**

Write a SQL query to get the second highest salary from the Employee table.

+----+--------+

| Id | Salary |

+----+--------+

| 1 | 100 |

| 2 | 200 |

| 3 | 300 |

+----+--------+

For example, given the above Employee table, the query should return 200 as the second highest salary. If there is no second highest salary, then the query should return null.

Sol:

# Write your MySQL query statement below

SELECT

(SELECT DISTINCT

Salary

FROM

Employee

ORDER BY Salary DESC

LIMIT 1 OFFSET 1) AS SecondHighestSalary

;

**Q.  Invert Binary Tree**

Given the root of a binary tree, invert the tree, and return *its root*.

**Example 2:**

A picture containing text, clipart

Description automatically generated

**Input:** root = [2,1,3]

**Output:** [2,3,1]

**Sol:**

**Using recursion**

**# Definition for a binary tree node.**

**# class TreeNode:**

**# def \_\_init\_\_(self, val=0, left=None, right=None):**

**# self.val = val**

**# self.left = left**

**# self.right = right**

**class Solution:**

**def invertTree(self, root: TreeNode) -> TreeNode:**

**if root is None:**

**return None**

**root.left, root.right = self.invertTree(root.right), self.invertTree(root.left)**

**return root**

**same as above**

**class Solution:**

**def invertTree(self, root: TreeNode) -> TreeNode:**

**if root is None:**

**return None**

**root.left, root.right = root.right, root.left**

**self.invertTree(root.right)**

**self.invertTree(root.left)**

**return root**

**Q. Binary search**

**Input:** nums = [-1,0,3,5,9,12], target = 9

**Output:** 4

**Explanation:** 9 exists in nums and its index is 4

**Soln:**

**class Solution:**

**def search(self, nums: List[int], target: int) -> int:**

**low, high = 0, len(nums)-1**

**while low<=high:**

**mid = (low + high) // 2**

**if nums[mid]==target:**

**return mid**

**if target < nums[mid]:**

**high = mid-1**

**else:**

**low = mid + 1**

**return -1**

Time complexity : \mathcal{O}(\log N)O(log*N*).

* Space complexity : \mathcal{O}(1)O(1) since it's a constant space solution.

**Q. Power of Two**

Given an integer n, return *true if it is a power of two. Otherwise, return false*.

An integer n is a power of two, if there exists an integer x such that n == 2x.

**Example 1:**

**Input:** n = 1

**Output:** true

**Explanation:** 20 = 1

**Example 2:**

**Input:** n = 16

**Output:** true

**Explanation:** 24 = 16

**Example 3:**

**Input:** n = 3

**Output:** false

**Sol:**

**class Solution(object):**

**def isPowerOfTwo(self, n):**

**if n == 0:**

**return False**

**while n % 2 == 0:**

**n /= 2**

**return n == 1**

O(log*N*) time

Another sol:

class Solution:

def isPowerOfTwo(self, n: int) -> bool:

x=0

while 2\*\*x<=n:

if(2\*\*x==n):

return True

x+=1

return False