Table of content

Window sliding 1,2

Pre-built Key creator in obj 5

The key with the highest value 7

Sequence questions 6,7

جا به جایی هم زمان دو عضو از آرایه 8,9

List slicing in python 9

شمردن تعداد عضو های یک لیست 11

11حلقه for با بیش از یک اندیس

*12شمارنده تعداد هر عضو آرایه یا رشته*

*f a new method for populating a Dictionary (unique value count) 17-22*

iterate through two arrays or tuples at the same time 18

21 شمارش قطاری

Set 23

Window Sliding Technique

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)

This technique shows how a nested for loop in some problems can be converted to a single for loop to reduce the time complexity.  
Let’s start with a problem for illustration where we can apply this technique –

Given an array of integers of size **‘n’**.

Our aim is to calculate the maximum sum of **‘k’**

consecutive elements in the array.

import sys

print "GFG"

# O(n \* k) solution for finding

# maximum sum of a subarray of size k

INT\_MIN = -sys.maxsize - 1

# Returns maximum sum in a

# subarray of size k.

def maxSum(arr, n, k):

# Initialize result

max\_sum = INT\_MIN

# Consider all blocks

# starting with i.

for i in range(n - k + 1):

m = 0

current\_sum = 0

for j in range(k):

current\_sum = current\_sum + arr[i + j]

# Update result if required.

max\_sum = max(current\_sum, m)

m = max\_sum

current\_sum=0

return m

# Driver code

arr = [1, 4, 2, 10, 2,

3, 1, 0, 20]

k = 4

n = len(arr)

print(maxSum(arr, n, k))

It can be observed from the above code that the time complexity is **O(k\*n)** as it contains two nested loops.

**Window Sliding Technique**

The technique can be best understood with the window pane in bus, consider a window of length **n** and the pane which is fixed in it of length **k**. Consider, initially the pane is at extreme left i.e., at 0 units from the left. Now, co-relate the window with array arr[] of size n and pane with current\_sum of size k elements. Now, if we apply force on the window such that it moves a unit distance ahead. The pane will cover next **k** consecutive elements.   
Consider an array **arr[]** = {5, 2, -1, 0, 3} and value of **k** = 3 and **n** = 5  
**Applying sliding window technique**:

1. We compute the sum of first k elements out of n terms using a linear loop and store the sum in variable window\_sum.
2. Then we will graze linearly over the array till it reaches the end and simultaneously keep track of maximum sum.
3. To get the current sum of block of k elements just subtract the first element from the previous block and add the last element of the current block .

The below representation will make it clear how the window slides over the array.  
This is the initial phase where we have calculated the initial window sum starting from index 0 . At this stage the window sum is 6. Now, we set the maximum\_sum as current\_window i.e 6. 



Now, we slide our window by a unit index. Therefore, now it discards 5 from the window and adds 0 to the window. Hence, we will get our new window sum by subtracting 5 and then adding 0 to it. So, our window sum now becomes 1. Now, we will compare this window sum with the maximum\_sum. As it is smaller we wont the change the maximum\_sum. 



Similarly, now once again we slide our window by a unit index and obtain the new window sum to be 2. Again we check if this current window sum is greater than the maximum\_sum till now. Once, again it is smaller so we don’t change the maximum\_sum.  
Therefore, for the above array our maximum\_sum is 6.



**code for the above description :**

# O(n) solution for finding

# maximum sum of a subarray of size k

def maxSum(arr, k):

    # length of the array

    n = len(arr)

    # n must be greater than k

    if n < k:

        print("Invalid")

        return -1

    # Compute sum of first window of size k

    window\_sum = sum(arr[:k])

    # first sum available

    max\_sum = window\_sum

    # Compute the sums of remaining windows by

    # removing first element of previous

    # window and adding last element of

    # the current window.

    for i in range(n - k):

        window\_sum = window\_sum - arr[i] + arr[i + k]

        max\_sum = max(window\_sum, max\_sum)

    return max\_sum

# Driver code

arr = [1, 4, 2, 10, 2, 3, 1, 0, 20]

k = 4

print(maxSum(arr, k))

2 - Given an array of integers nums and an integer k, return the total number of continuous subarrays whose sum equals to *k*. (hard!)  
  
**naïve answer:**

arr=[-1, -2, 3, 2, 1, -1, 4, -2 ]

k = 2

# arr = [ 10, 2, -2, -20, 10 ]

n = len(arr)

# k = -10

res = 0

# Calculate all subarrays

for i in range(n):

summ = 0

for j in range(i, n):

# Calculate required sum

summ += arr[j]

# Check if sum is equal to

# required sum

if summ == k:

res += 1

print(res)

better solution:

intro:

from collections import defaultdict

somedict = {}

print(somedict[3]) # KeyError

someddict = defaultdict(int)

print(someddict[3])

defaultdict means that if a key is not found in the dictionary, then instead of a KeyError being thrown, a new entry is created. The type of this new entry is given by the argument of defaultdict.

دیفالت دیک : کار آن ساخت مقدار پیش فرض 0 برای عضوی که که از قبل وجود ندارد است.

someddict = defaultdict(lambda:10)

لاندا هم به آن مقدار پیش فرض میدهد. یعنی مثلا اینجا   
someddic[3]   
برابر با ده میشود نه صفر.  
  
from collections import defaultdict

def findSubarraySum(arr, n, Sum):

# Dictionary to store number of subarrays

# starting from index zero having

# particular value of sum.

prevSum = defaultdict(lambda : 0)

res = 0

# Sum of elements so far.

currsum = 0

for i in range(0, n):

# Add current element to sum so far.

currsum += arr[i]

# If currsum is equal to desired sum,

# then a new subarray is found. So

# increase count of subarrays.

if currsum == Sum:

res += 1

# currsum exceeds given sum by currsum - sum.

# Find number of subarrays having

# this sum and exclude those subarrays

# from currsum by increasing count by

# same amount.

if (currsum - Sum) in prevSum:

res += prevSum[currsum - Sum]

# Add currsum value to count of

# different values of sum.

prevSum[currsum] += 1

return res

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

arr = [10, 2, -2, -20, 10]

Sum = -10

n = len(arr)

print(findSubarraySum(arr, n, Sum))

print(findSubarraySum([1,2,3,-1,1,5], 6, 5))

Given a binary array, find the maximum number of consecutive 1s in this array.

**Example 1:**

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

**Output:** 3

**Explanation:** The first two digits or the last three digits are consecutive 1s.

The maximum number of consecutive 1s is 3.

**Note:**

* The input array will only contain 0 and 1.
* The length of input array is a positive integer and will not exceed 10,000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

یه سوال مکمل هم میتونیم طراحی کنیم که در آن بیشترین توالی رو پیدا کند مثلا بیشترین تعداد تکرار عضوهای مشابه و متوالی  
  
[ 1, 2, 2, 2, 2, 0, 0, 4, 4, 4, 4, 4, 4]=> {‘4’:6}

solution:

def ones(arr):

obj = {}

for i in range(len(arr)):

if arr[i] == 1:

obj[i] =1

for j in range(i, len(arr)):

if j+1 < len(arr) and arr[j] == arr[j+1]:

obj[i] += 1

else:

break

MaxKey = max(obj, key=obj.get)

return obj[MaxKey]

print(ones([1,1,0,1,1,1,1,9,9,1]))  
  
 **better solution:**

class Solution:

def findMaxConsecutiveOnes(self, x: List[int]) -> int:

count = 0

compare=0

for i in range (len(x)):

if x[i]==1:

count +=1

else:

compare= max(compare,count)

count = 0

return(max(compare,count))

Given an unsorted array of integers nums, return *the length of the longest consecutive elements sequence.*

**Example 1:**

**Input:** nums = [100,4,200,1,3,2]

**Output:** 4

**Explanation:** The longest consecutive elements sequence is [1, 2, 3, 4]. Therefore its length is 4.

**Example 2:**

**Input:** nums = [0,3,7,2,5,8,4,6,0,1]

**Output:** 9

print('Hello, world!')

def long(x):

x.sort()

if x == []:

return 0

count=1

compare = 0

for i in range (len(x)):

if i+1 < len(x) and x[i]==x[i+1]-1:

count += 1

elif i+1 < len(x) and x[i]==x[i+1]:

continue

else:

compare = max(compare, count)

count = 1

print("here's compare:",compare)

return(max(compare, count))

print(long([1,2,500,300,4,4,6,5]))

print(long([100,4,200,1,3,2]))

print(long([2]))

print(long([2,4,6]))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#next permutation

Implement **next permutation**, which rearranges numbers into the lexicographically next greater permutation of numbers.

If such an arrangement is not possible, it must rearrange it as the lowest possible order (i.e., sorted in ascending order).

The replacement must be [**in place**](http://en.wikipedia.org/wiki/In-place_algorithm) and use only constant extra memory.

print('Hello, world!')

def nextPerm(x):

n = len(x)

y = sorted(x,reverse=True)

print("this is the list with a descending order :",y)

if x == y:

x.sort()

return x

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

if (i-1 > -1 and x[i] < x[i-1]) or (i-1 > 1 and x[i]==x[i-1]):

continue

else:

sub1 = x[:i]

sub2 = x[i:n]

sub2.sort()

for j in range(len(sub2)):

if i-1 > -1 and sub1[i-1] < sub2[j]:

tt = sub1[i-1]

sub1[i-1]= sub2[j]

sub2[j] = tt

sub1.extend(sub2)

return sub1

print(nextPerm([1,2,3]))

print(nextPerm([2]))

print(nextPerm([2,3,6,5,1]))

print(nextPerm([5,4,0]))

**better solution:**

class Solution:

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

index = len(nums) - 1

while index - 1 >= 0 and nums[index] <= nums[index - 1]:

index -= 1

if index == 0:

nums.reverse()

return

greater = index

while greater + 1 < len(nums) and nums[greater + 1] > nums [index - 1]:

greater += 1

nums[greater], nums[index - 1] = nums[index - 1], nums[greater]

nums[index :] = nums[index :][::-1]

**Slicing in python with step:**

>>> "ABCD"[1:]

'BCD'

>>> "ABCD"[:3]

'ABC'

>>> "ABCD"[1:3]

'BC'

>>> "ABCD"[1:3:]

'BC'

>>> "ABCD"[::2]

'AC'

>>> "ABCD"[::]

'ABCD'

>>> "ABCD"[:]

'ABCD

**Reverse strings and lists:**

>>> "ABCD"[::-1]

'DCBA'

>>> [0, 1, 2, 3][::-1]

[3, 2, 1, 0]

**Replace multiple items all at once:**

>>> # slices can be used to replace multiple items

>>> l = [0, 1, 2, 3]

>>> l[:2] = ("AB", "CD")

>>> l

['AB', 'CD', 2, 3]

>>> # when using extended slice syntax both chunks must match

>>> l = [0, 1, 2, 3]

>>> l[::2] = "ABCD"

Traceback (most recent call last):

File "<interactive input>", line 1, in <module>

ValueError: attempt to assign sequence of size 4 to extended slice of size 2

**Remove multiple items all at once:**

>>> # deleting items

>>> l = [0, 1, 2, 3]

>>> del l[::2]

>>> l

[1, 3]

There is an integer array nums sorted in ascending order (with **distinct** values).

Prior to being passed to your function, nums is **rotated** at an unknown pivot index k (0 <= k < nums.length) such that the resulting array is [nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]] (**0-indexed**). For example, [0,1,2,4,5,6,7] might be rotated at pivot index 3 and become [4,5,6,7,0,1,2].

Given the array nums **after** the rotation and an integer target, return *the index of*target*if it is in*nums*, or*-1*if it is not in*nums.  
(foolishly simple)

**Example 1:**

**Input:** nums = [4,5,6,7,0,1,2], target = 0

**Output:** 4

**Example 2:**

**Input:** nums = [4,5,6,7,0,1,2], target = 3

**Output:** -1

**Example 3:**

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

**Output:** -1

class Solution:

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

low = 0

high = len(nums)-1

while low<=high:

mid = (low+high)//2

if nums[mid] == target:

return mid

if nums[mid]>=nums[low]:

if target>=nums[low] and target<nums[mid]:

high = mid-1

else:

low = mid+1

else:

if target>nums[mid] and target<=nums[high]:

low = mid+1

else:

high = mid-1

return -1

**class Solution:**

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

**if target in nums:**

**return nums.index(target)**

**else:**

**return (-1)**

Given an integer array nums where every element appears **three times** except for one, which appears **exactly once**. *Find the single element and return it*.

**Example 1:**

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

**Output:** 3

**Example 2:**

**Input:** nums = [0,1,0,1,0,1,99]

**Output:** 99

**Constraints:**

* 1 <= nums.length <= 3 \* 104
* -231 <= nums[i] <= 231 - 1
* Each element in nums appears exactly **three times** except for one element which appears **once**.

**Follow up:** Your algorithm should have a linear runtime complexity. Could you implement it without using extra memory?

**List comprehension :**

newlist = [x for x in fruits if x != "apple"]

///

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if x != "apple"]

print(newlist)

result: ['banana', 'cherry', 'kiwi', 'mango']

The expression can also contain conditions, not like a filter, but as a way to manipulate the outcome:

newlist = [x if x != "banana" else "orange" for x in fruits]

"Return the item if it is not banana, if it is banana return orange".

Answer:

class Solution:

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

return [i for i in nums if nums.count(i)<3][0]

شمردن اعضای یک لیست به طور جداگانه:

fruits = ["apple", "banana", "cherry"]

x = fruits.count("cherry")

print(x)

حلقه for با بیش از یک اندیس:

test2 = [

("dog", "god", True, "hi"),

("abcd", "bacd", True,"good")

]

for str1, str2, ex, y in test2:

print (str1,str2,y)

for x in test2:

print x

Answer:

('dog', 'god', 'hi')

('abcd', 'bacd', 'good')

('dog', 'god', True, 'hi')

('abcd', 'bacd', True, 'good')

**Palindrome Permutation**: Given a string, write a function to check if it is a permutation of a palindrome.

A palindrome is a word or phrase that is the same forwards and backwards. A permutation

is a rearrangement of letters. The palindrome does not need to be limited to just dictionary words.

Palindrome examples:

*Level, defied, radar, naan, rotor*

*شمارنده تعداد هر عضو:*

*from collections import Counter*

*ll = ["a","b","a"]*

*c = Counter(ll)*

*for i in c:*

*print(i)*

*// a, b*

*for val in c.values():*

*print(val)*

*// 2, 1*

*print(c)*

***solution:***

*from collections import Counter*

*def is\_palindrome\_permutation\_pythonic(phrase):*

*"""function checks if a string is a permutation of a palindrome or not"""*

*counter = Counter(phrase.replace(" ", "").lower())*

*return sum(val % 2 for val in counter.values()) <= 1*

*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Leetcode contest:*

### 5723. Finding the Users Active Minutes

[My Submissions](https://leetcode.com/contest/weekly-contest-235/problems/finding-the-users-active-minutes/submissions/)[Back to Contest](https://leetcode.com/contest/weekly-contest-235/)

* **User Accepted:**616
* **User Tried:**636
* **Total Accepted:**616
* **Total Submissions:**648
* **Difficulty:Medium**

You are given the logs for users' actions on LeetCode, and an integer k. The logs are represented by a 2D integer array logs where each logs[i] = [IDi, timei] indicates that the user with IDi performed an action at the minute timei.

**Multiple users** can perform actions simultaneously, and a single user can perform **multiple actions** in the same minute.

The **user active minutes (UAM)** for a given user is defined as the **number of unique minutes** in which the user performed an action on LeetCode. A minute can only be counted once, even if multiple actions occur during it.

You are to calculate a **1-indexed** array answer of size k such that, for each j (1 <= j <= k), answer[j] is the **number of users** whose **UAM** equals j.

Return *the array*answer*as described above*.

**Example 1:**

**Input:** logs = [[0,5],[1,2],[0,2],[0,5],[1,3]], k = 5

**Output:** [0,2,0,0,0]

**Explanation:**

The user with ID=0 performed actions at minutes 5, 2, and 5 again. Hence, they have a UAM of 2 (minute 5 is only counted once).

The user with ID=1 performed actions at minutes 2 and 3. Hence, they have a UAM of 2.

Since both users have a UAM of 2, answer[2] is 2, and the remaining answer[j] values are 0.

**Example 2:**

**Input:** logs = [[1,1],[2,2],[2,3]], k = 4

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

**Explanation:**

The user with ID=1 performed a single action at minute 1. Hence, they have a UAM of 1.

The user with ID=2 performed actions at minutes 2 and 3. Hence, they have a UAM of 2.

There is one user with a UAM of 1 and one with a UAM of 2.

Hence, answer[1] = 1, answer[2] = 1, and the remaining values are 0.

**Constraints:**

* 1 <= logs.length <= 104
* 0 <= IDi <= 109
* 1 <= timei <= 105
* k is in the range [The maximum **UAM** for a user, 105].

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\

You are given two positive integer arrays nums1 and nums2, both of length n.

The **absolute sum difference** of arrays nums1 and nums2 is defined as the **sum** of |nums1[i] - nums2[i]| for each 0 <= i < n (**0-indexed**).

You can replace **at most one** element of nums1 with **any** other element in nums1 to **minimize** the absolute sum difference.

Return the *minimum absolute sum difference****after****replacing at most one**element in the array nums1.* Since the answer may be large, return it **modulo** 109 + 7.

|x| is defined as:

* x if x >= 0, or
* -x if x < 0.

**Example 1:**

**Input:** nums1 = [1,7,5], nums2 = [2,3,5]

**Output:** 3

**Explanation:** There are two possible optimal solutions:

- Replace the second element with the first: [1,**7**,5] => [1,**1**,5], or

- Replace the second element with the third: [1,**7**,5] => [1,**5**,5].

Both will yield an absolute sum difference of |1-2| + (|1-3| or |5-3|) + |5-5| = 3.

**Example 2:**

**Input:** nums1 = [2,4,6,8,10], nums2 = [2,4,6,8,10]

**Output:** 0

**Explanation:** nums1 is equal to nums2 so no replacement is needed. This will result in an

absolute sum difference of 0.

**Example 3:**

**Input:** nums1 = [1,10,4,4,2,7], nums2 = [9,3,5,1,7,4]

**Output:** 20

**Explanation:** Replace the first element with the second: [**1**,10,4,4,2,7] => [**10**,10,4,4,2,7].

This yields an absolute sum difference of |10-9| + |10-3| + |4-5| + |4-1| + |2-7| + |7-4| = 20

### Number of Different Subsequences GCDs

[My Submissions](https://leetcode.com/contest/weekly-contest-235/problems/number-of-different-subsequences-gcds/submissions/)[Back to Contest](https://leetcode.com/contest/weekly-contest-235/)

* **User Accepted:**42
* **User Tried:**221
* **Total Accepted:**46
* **Total Submissions:**346
* **Difficulty:Hard**

You are given an array nums that consists of positive integers.

The **GCD** of a sequence of numbers is defined as the greatest integer that divides **all** the numbers in the sequence evenly.

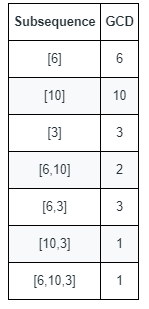
* For example, the GCD of the sequence [4,6,16] is 2.

A **subsequence** of an array is a sequence that can be formed by removing some elements (possibly none) of the array.

* For example, [2,5,10] is a subsequence of [1,2,1,**2**,4,1,**5**,**10**].

Return the ***number*** of ***different*** GCDs among all ***non-empty*** subsequences of nums.

**Example 1:**



**Input:** nums = [6,10,3]

**Output:** 5

**Explanation:** The figure shows all the non-empty subsequences and their GCDs.

The different GCDs are 6, 10, 3, 2, and 1.

**Example 2:**

**Input:** nums = [5,15,40,5,6]

**Output:** 7

**End of the first contest  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**621. Task Scheduler**  
Given a characters array tasks, representing the tasks a CPU needs to do, where each letter represents a different task. Tasks could be done in any order. Each task is done in one unit of time. For each unit of time, the CPU could complete either one task or just be idle.

However, there is a non-negative integer n that represents the cooldown period between two **same tasks** (the same letter in the array), that is that there must be at least n units of time between any two same tasks.

Return *the least number of units of times that the CPU will take to finish all the given tasks*.

**Example 1:**

**Input:** tasks = ["A","A","A","B","B","B"], n = 2

**Output:** 8

**Explanation:**

A -> B -> idle -> A -> B -> idle -> A -> B

There is at least 2 units of time between any two same tasks.

**Example 2:**

**Input:** tasks = ["A","A","A","B","B","B"], n = 0

**Output:** 6

**Explanation:** On this case any permutation of size 6 would work since n = 0.

["A","A","A","B","B","B"]

["A","B","A","B","A","B"]

["B","B","B","A","A","A"]

...

And so on.

**Example 3:**

**Input:** tasks = ["A","A","A","A","A","A","B","C","D","E","F","G"], n = 2

**Output:** 16

**Explanation:**

One possible solution is

A -> B -> C -> A -> D -> E -> A -> F -> G -> A -> idle -> idle -> A -> idle -> idle -> A

my solution :

class Solution:

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

for i in range(len(arr)):

if i+1 <len(arr) and arr[i] != arr[i+1]:

continue

else:

for j in range(i+1,len(arr)):

if arr[i] != arr[j]:

arr[i+1],arr[j]=arr[j],arr[i+1]

break

# return arr

# print arr

rest=0

for i in range(len(arr)):

if i+1 < len(arr) and arr[i] == arr[i+1]:

rest += n \*(len(arr)-1-i)

break

else:

return n+len(arr)

return rest+ len(arr)

correct solution:

<https://leetcode.com/problems/task-scheduler/discuss/699297/Python-Very-detailed-explanation-with-examples>

<https://leetcode.com/problems/task-scheduler/discuss/761070/Python-or-Heavily-visualized-%2B-Detailed-explanation>

**4sum**

## Return Value from get()

get() method returns:

* the value for the specified key if key is in dictionary.
* None if the key is not found and value is not specified.
* value if the key is not found and value is specified.

dictionary.get(keyname, default\_value)

در صورتی که کلید مورد نظر مقداری نداشته باشد مقدار دیفالت(پارامتر دوم) در نظر گرفته میشود در غیر این صورت این مقدار پیشفرض هر چه که باشد به حساب نمی آید و ملاک خود دیکشنری اصلی است:

**person = {'name': 'Phill', 'age': 22, 'boz':11}**

**print('Salary: ', person.get('boz',1000))**

**// 🡺 salary 11**

**Question:**

Given four lists A, B, C, D of integer values, compute how many tuples (i, j, k, l) there are such that A[i] + B[j] + C[k] + D[l] is zero.

To make problem a bit easier, all A, B, C, D have same length of N where 0 ≤ N ≤ 500. All integers are in the range of -228 to 228 - 1 and the result is guaranteed to be at most 231 - 1.

**Example:**

**Input:**

A = [ 1, 2]

B = [-2,-1]

C = [-1, 2]

D = [ 0, 2]

**Output:**

2

**Explanation:**

The two tuples are:

1. (0, 0, 0, 1) -> A[0] + B[0] + C[0] + D[1] = 1 + (-2) + (-1) + 2 = 0

2. (1, 1, 0, 0) -> A[1] + B[1] + C[0] + D[0] = 2 + (-1) + (-1) + 0 = 0

class Solution:

def fourSumCount(self, A: List[int], B: List[int], C: List[int], D: List[int]) -> int:

sum\_cd = {}

res = 0

for i in C:

for j in D:

sum\_cd[i+j] = sum\_cd.get(i+j,0) + 1

for i in A:

for j in B:

res += sum\_cd.get(-i-j,0)

return res

**One Away:** There are three types of edits that can be performed on strings: insert a character,

remove a character, or replace a character. Given two strings, write a function to check if they are

one edit (or zero edits) away.

EXAMPLE

pale, ple -> true

pales, pale -> true

pale, bale -> true

pale, bake -> false

zip:

to iterate through two arrays or tuples at the same time:

a = ("John", "Charles", "Mike")

b = ("Jenny", "Christy", "Monica", "Vicky")

x = zip(a, b)

// 🡺 (('John', 'Jenny'), ('Charles', 'Christy'), ('Mike', 'Monica'))

#use the tuple() function to display a readable version of the result:

print(tuple(x))

*def* one\_away(*s1*, *s2*):

    '''Check if a string can converted to another string with a single edit'''

    if len(s1) == len(s2):

        return one\_edit\_replace(s1, s2)

    elif len(s1) + 1 == len(s2):

        return one\_edit\_insert(s1, s2)

    elif len(s1) - 1 == len(s2):

        return one\_edit\_insert(s2, s1)

    return False

*def* one\_edit\_replace(*s1*, *s2*):

    edited = False

    for c1, c2 in zip(s1, s2):

        if c1 != c2:

            if edited:

                return False

            edited = True

    return True

*def* one\_edit\_insert(*s1*, *s2*):

    edited = False

    i, j = 0, 0

    while i < len(s1) and j < len(s2):

        if s1[i] != s2[j]:

            if edited:

                return False

            edited = True

            j += 1

        else:

            i += 1

            j += 1

    return True

You are given row x col grid representing a map where grid[i][j] = 1 represents land and grid[i][j] = 0 represents water.

Grid cells are connected **horizontally/vertically** (not diagonally). The grid is completely surrounded by water, and there is exactly one island (i.e., one or more connected land cells).

The island doesn't have "lakes", meaning the water inside isn't connected to the water around the island. One cell is a square with side length 1. The grid is rectangular, width and height don't exceed 100. Determine the perimeter of the island.



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

**Output:** 16

**Explanation:** The perimeter is the 16 yellow stripes in the image above.

**Example 2:**

**Input:** grid = [[1]]

**Output:** 4

**Example 3:**

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

**Output:** 4

class Solution:

def islandPerimeter(self, grid):

res = 0

for i in range(len(grid)):

for j in range(len(grid[i])):

upNeighbor=0

downNeighbor = 0

leftNeighbor = 0

rightNeighbor = 0

if grid[i][j] == 1:

if(i-1> -1):

# print("this is g[i-1]",grid[i-1][j])

upNeighbor = grid[i-1][j]

if(i+1 < len(grid)):

# print("this is g[i+1]",grid[i+1])

downNeighbor = grid[i+1][j]

if(j-1 > -1):

leftNeighbor = grid[i][j-1]

if(j+1 < len(grid[i])):

rightNeighbor = grid[i][j+1]

res += 4- (upNeighbor + downNeighbor + leftNeighbor + rightNeighbor)

return res

Given a binary array, find the maximum length of a contiguous subarray with equal number of 0 and 1.

**Example 1:**

**Input:** [0,1]

**Output:** 2

**Explanation:** [0, 1] is the longest contiguous subarray with equal number of 0 and 1.

**Example 2:**

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

**Output:** 2

**Explanation:** [0, 1] (or [1, 0]) is a longest contiguous subarray with equal number of 0 and 1.

Cracking:

String Compression: Implement a method to perform basic string compression using the counts

of repeated characters. For example, the string aabcccccaaa would become a2blc5a3. If the

"compressed" string would not become smaller than the original string, your method should return

the original string. You can assume the string has only uppercase and lowercase letters (a - z).

**string compression:**

Given an input string of a certain length, design an algorithm that compresses the string. The string should be compressed such that consecutive duplicates of characters are replaced with the character and followed by the number of consecutive duplicates.

For example, if the input string is “wwwwaaadexxxxxx”, then the function should return “w4a3dex6”.

# X is the input string

*def* stringCompression(*x*):

  obj={}

  string1=""

  for letter in x:

    obj[letter]=obj.get(letter,0)+1

    # if letter not in obj:

    #   obj[letter] =1

    # else:

    #   obj[letter] += 1

  print(obj)

  for letter in x:

    if letter in string1:

      continue

    if obj[letter] != 1:

      string1 +=letter + *str*(obj[letter])

    else:

      string1 += letter

  print(string1)

stringCompression("permanent")

stringCompression("hello")

stringCompression("baby")

stringCompression("wwwwaaab")

cracking solution:  
(a bit of diff concept)

شمارش قطاری

#another one acooding to the book's example

print('Hello, world!')

*def* compress\_string(*string*):

    compressed = []

    counter = 0

    for i in range(len(string)):  # noqa

        if i != 0 and string[i] != string[i - 1]:

            compressed.append(string[i - 1] + *str*(counter))

            counter = 0

        counter += 1

    # add last repeated character

    if counter:

        compressed.append(string[-1] + *str*(counter))

    # returns original string if compressed string isn't smaller

    print("this is the result:","".join(compressed))

    return min(string, "".join(compressed), *key*=len)

print(compress\_string("hello"))

print(compress\_string("permanent"))

print(compress\_string("aabcccccaaa"))

**Get method in dictionaries:  
روشی برای ساخت عضو در دیکشنری**

**car = {**

**"brand": "Ford",**

**"model": "Mustang",**

**"year": 1964**

**}**

**x = car.get("price", 15000)**

**car["price"]=car.get("price",14000)**

**car["price"]=car.get("price",999)**

**print(x)**

**print(car)**

**جواب ها:**

15000

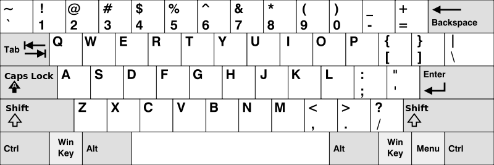
{'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'price': 14000}

می بینیم که گت فقط در صورت نبود مقدار -مقدار میدهد. اگر کلیدی مقدار داشته باشد مقدار گت کار نخواهد کرد. بسیار ایده آل برای پاپیولیت کردن دیکشنری

**500-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"]

class Solution:

def findWords(self, words: List[str]) -> List[str]:

row1 ="qwertyuiop"

row2 = "asdfghjkl"

row3 = "zxcvbnm"

arr=[]

for val in words:

obj1={}

for i in range(len(val)):

if val[i] in row1:

obj1["row1"]=obj1.get("row1",0)+1

if val[i] in row2:

obj1["row2"] = obj1.get("row2",0)+1

if val[i] in row3:

obj1["row3"] = obj1.get("row3",0)+1

if len(obj1) == 1:

# arr.append(obj1)

arr.append(val)

return(arr)

روش بهتر

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'}

res = []

for i in words:

wordset = set(i.lower())

if (wordset&set1 == wordset) or (wordset&set2 == wordset) or (wordset&set3 == wordset):

res.append(i)

return res

معرفی set

**Note:** Sets are unordered, so you cannot be sure in which order the items will appear.

## **Set Items**

Set items are unordered, unchangeable, and do not allow duplicate values.

Duplicate values will be ignored:

thisset = {"apple", "banana", "cherry", "apple"}

print(thisset)

res:

{'banana', 'cherry', 'apple'}

print(len(thisset)) = 3 not 4!

set1 = {"apple", "banana", "cherry"}

set2 = {1, 5, 7, 9, 3}

set3 = {True, False, False}

print(set1)

print(set2)

print(set3)

جواب

{'cherry', 'apple', 'banana'}

{1, 3, 5, 7, 9}

{False, True}

**Set() constructor:**

thisset = set(("apple", "banana", "cherry")) # note the double round-brackets  
print(thisset)

result: {“banana”, ”apple”, “cherry”}

**for:**

thisset = {"apple", "banana", "cherry"}  
  
for x in thisset:  
  print(x)

**existence:**

Check if "banana" is present in the set:

thisset = {"apple", "banana", "cherry"}  
  
print("banana" in thisset)

sets are immutable:

Once a set is created, you cannot change its items, but you can add new items.

**Add:**

thisset = {"apple", "banana", "cherry"}  
  
thisset.add("orange")  
  
print(thisset)

**Add set to set:**

thisset = {"apple", "banana", "cherry"}

tropical = {"pineapple", "mango", "papaya", "apple"}

thisset.update(tropical)

print(thisset)

## **Add Any Iterable**

The object in the update() method does not have be a set, it can be any iterable object (tuples, lists, dictionaries etc.).

### Example

Add elements of a list to at set:

thisset = {"apple", "banana", "cherry"}  
mylist = ["kiwi", "orange"]  
  
thisset.update(mylist)  
  
print(thisset)

(result is still aa set)

## **Remove Item**

To remove an item in a set, use the remove(), or the discard() method.

### Example

Remove "banana" by using the remove() method:

thisset = {"apple", "banana", "cherry"}  
  
thisset.remove("banana")  
  
print(thisset)

Discard is better

### Example

Remove "banana" by using the discard() method:

thisset = {"apple", "banana", "cherry"}  
  
thisset.discard("banana")  
  
print(thisset)

[Try it Yourself »](https://www.w3schools.com/python/trypython.asp?filename=demo_set_discard)

**Note:** If the item to remove does not exist, discard() will **NOT** raise an error.

**Note:** Sets are unordered, so when using the pop() method, you do not know which item that gets removed.

**Clear all at once**

The clear() method empties the set:

thisset = {"apple", "banana", "cherry"}  
  
thisset.clear()  
  
print(thisset)

**Delete completely**

The del keyword will delete the set completely:

thisset = {"apple", "banana", "cherry"}  
  
del thisset  
  
print(thisset)

## **Join Two Sets**

There are several ways to join two or more sets in Python.

You can use the union() method that returns a new set containing all items from both sets, or the update() method that inserts all the items from one set into another:

### Example

The union() method returns a new set with all items from both sets:

set1 = {"a", "b" , "c"}  
set2 = {1, 2, 3}  
  
set3 = set1.union(set2)  
print(set3)

(New set is created unlike update where the second set is added to the first one)

تساوی در ست ها:  
جایگاه اهمیتی نداره.

Print({1,3} == {3,1})🡺

True

print({1,2,3} == {3,2,1,1,2,1}) 🡺True

اگر مستقیما درون ست یک رشته قرار دهیم آن رشته را به اجرایش تجزیه میکند.

(یعنی روش خوبی برای پر کردن ست با کلمات نیست)

w4 = set("hello")

print(w4)

Amazon Top Questions:

Given an array of integers nums and an integer target, return *indices of the two numbers such that they add up to target*.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

**1. Two Sum**

Easy

**Example 1:**

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

**Output:** [0,1]

**Output:** Because nums[0] + nums[1] == 9, we return [0, 1].

**Example 2:**

**Input:** nums = [3,2,4], target = 6

**Output:** [1,2]

**Example 3:**

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

**Output:** [0,1]

**Constraints:**

* 2 <= nums.length <= 104
* -109 <= nums[i] <= 109
* -109 <= target <= 109
* **Only one valid answer exists.**

**Follow-up:**Can you come up with an algorithm that is less than O(n2) time complexity?

class Solution:

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

hashmap = {}

for i in range(len(nums)):

complement = target - nums[i]

if complement in hashmap:

return [i, hashmap[complement]]

hashmap[nums[i]] = i

**56. Merge Intervals**

Medium

Given an array of intervals where intervals[i] = [starti, endi], merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input*.

**Example 1:**

**Input:** intervals = [[1,3],[2,6],[8,10],[15,18]]

**Output:** [[1,6],[8,10],[15,18]]

**Explanation:** Since intervals [1,3] and [2,6] overlaps, merge them into [1,6].

**Example 2:**

**Input:** intervals = [[1,4],[4,5]]

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

**Explanation:** Intervals [1,4] and [4,5] are considered overlapping.

**Constraints:**

* 1 <= intervals.length <= 104
* intervals[i].length == 2
* 0 <= starti <= endi <= 104

class Solution:

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

intervals.sort(key=lambda x: x[0])

merged = []

print("not merged:\t", not merged)

for interval in intervals:

# if the list of merged intervals is empty or if the current

# interval does not overlap with the previous, simply append it.

if len(merged) == 0 or merged[-1][1] < interval[0]:

merged.append(interval)

else:

# otherwise, there is overlap, so we merge the current and previous

# intervals.

merged[-1][1] = max(merged[-1][1], interval[1])

return merged

**696. Count Binary Substrings**

Easy

Give a binary string s, return the number of non-empty substrings that have the same number of 0's and 1's, and all the 0's and all the 1's in these substrings are grouped consecutively.

Substrings that occur multiple times are counted the number of times they occur.

**Example 1:**

**Input:** s = "00110011"

**Output:** 6

**Explanation:** There are 6 substrings that have equal number of consecutive 1's and 0's: "0011", "01", "1100", "10", "0011", and "01".

Notice that some of these substrings repeat and are counted the number of times they occur.

Also, "00110011" is not a valid substring because all the 0's (and 1's) are not grouped together.

**Example 2:**

**Input:** s = "10101"

**Output:** 4

**Explanation:** There are 4 substrings: "10", "01", "10", "01" that have equal number of consecutive 1's and 0's.

**Constraints:**

* 1 <= s.length <= 105
* s[i] is either '0' or '1'.

#### **Approach #1: Group By Character [Accepted]**

**Intuition**

We can convert the string s into an array groups that represents the length of same-character contiguous blocks within the string. For example, if s = "110001111000000", then groups = [2, 3, 4, 6].

For every binary string of the form '0' \* k + '1' \* k or '1' \* k + '0' \* k, the middle of this string must occur between two groups.

Let's try to count the number of valid binary strings between groups[i] and groups[i+1]. If we have groups[i] = 2, groups[i+1] = 3, then it represents either "00111" or "11000". We clearly can make min(groups[i], groups[i+1]) valid binary strings within this string. Because the binary digits to the left or right of this string must change at the boundary, our answer can never be larger.

**Algorithm**

Let's create groups as defined above. The first element of s belongs in it's own group. From then on, each element either doesn't match the previous element, so that it starts a new group of size 1; or it does match, so that the size of the most recent group increases by 1.

Afterwards, we will take the sum of min(groups[i-1], groups[i])

class Solution(object):

def countBinarySubstrings(self, s):

groups = [1]

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

if s[i-1] != s[i]:

groups.append(1)

else:

groups[-1] += 1

ans = 0

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

ans += min(groups[i-1], groups[i])

return ans

**2nd approach (brute force- mine)**

print("Hello world")

def x(s):

arr=[]

sum = 0

for i in range(len(s)):

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

arr.append(s[i:j])

print(arr)

for i in range(len(arr)):

if arr[i].count("0") == arr[i].count("1"):

print("hi sub",arr[i][0: (len(arr[i]))//2 ])

if len( set(arr[i][0:(len(arr[i]))//2]) ) ==1:

sum += 1

return(sum)

print(x("1100"))

print(x( "110001111000000" ))

**49. Group Anagrams**

Medium

7071257Add to ListShare

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"]]

**Example 2:**

**Input:** strs = [""]

**Output:** [[""]]

**Example 3:**

**Input:** strs = ["a"]

**Output:** [["a"]]

**Constraints:**

* 1 <= strs.length <= 104
* 0 <= strs[i].length <= 100
* strs[i] consists of lowercase English letters.

Brute Force Solution:

def g(strs):

res=[]

obj={}

for i in range(len(strs)):

if ''.join(sorted(strs[i])) not in obj:

obj[''.join(sorted(strs[i]))]=[strs[i]]

for j in range(i+1, len(strs)):

if ''.join(sorted(strs[i])) == ''.join(sorted(strs[j])):

obj[''.join(sorted(strs[i]))].append(strs[j])

print("sorted elements of the string array:\t",''.join(sorted(strs[i])))

res.append(obj[''.join(sorted(strs[i]))])

else:

continue

return(res)

print(g(["eat","tea","tan","ate","nat","bat"]))

better solution:

class Solution(object):

def groupAnagrams(self, strs):

ans = collections.defaultdict(list)

for s in strs:

ans[tuple(sorted(s))].append(s)

return ans.values()

**1710. Maximum Units on a Truck**

Easy

103680Add to ListShare

You are assigned to put some amount of boxes onto **one truck**. You are given a 2D array boxTypes, where boxTypes[i] = [numberOfBoxesi, numberOfUnitsPerBoxi]:

* numberOfBoxesi is the number of boxes of type i.
* numberOfUnitsPerBoxiis the number of units in each box of the type i.

You are also given an integer truckSize, which is the **maximum** number of **boxes** that can be put on the truck. You can choose any boxes to put on the truck as long as the number of boxes does not exceed truckSize.

Return *the****maximum****total number of****units****that can be put on the truck.*

**Example 1:**

**Input:** boxTypes = [[1,3],[2,2],[3,1]], truckSize = 4

**Output:** 8

**Explanation:** There are:

- 1 box of the first type that contains 3 units.

- 2 boxes of the second type that contain 2 units each.

- 3 boxes of the third type that contain 1 unit each.

You can take all the boxes of the first and second types, and one box of the third type.

The total number of units will be = (1 \* 3) + (2 \* 2) + (1 \* 1) = 8.

**Example 2:**

**Input:** boxTypes = [[5,10],[2,5],[4,7],[3,9]], truckSize = 10

**Output:** 91

**1603. Design Parking System**

Easy

456217Add to ListShare

Design a parking system for a parking lot. The parking lot has three kinds of parking spaces: big, medium, and small, with a fixed number of slots for each size.

Implement the ParkingSystem class:

* ParkingSystem(int big, int medium, int small) Initializes object of the ParkingSystem class. The number of slots for each parking space are given as part of the constructor.
* bool addCar(int carType) Checks whether there is a parking space of carType for the car that wants to get into the parking lot. carType can be of three kinds: big, medium, or small, which are represented by 1, 2, and 3 respectively. **A car can only park in a parking space of its**carType. If there is no space available, return false, else park the car in that size space and return true.

**Example 1:**

**Input**

["ParkingSystem", "addCar", "addCar", "addCar", "addCar"]

[[1, 1, 0], [1], [2], [3], [1]]

**Output**

[null, true, true, false, false]

**Explanation**

ParkingSystem parkingSystem = new ParkingSystem(1, 1, 0);

parkingSystem.addCar(1); // return true because there is 1 available slot for a big car

parkingSystem.addCar(2); // return true because there is 1 available slot for a medium car

parkingSystem.addCar(3); // return false because there is no available slot for a small car

parkingSystem.addCar(1); // return false because there is no available slot for a big car. It is already occupied.

class ParkingSystem:

def \_\_init\_\_(self, big: int, medium: int, small: int):

self.big = big

self.medium = medium

self.small = small

self.arr = [self.big,self.medium,self.small]

def addCar(self, carType: int) -> bool:

# print("this is arr:\t",arr)

if carType == 1:

self.arr[0] = self.arr[0]-1

if self.arr[0] >= 0:

return True

else:

return False

if carType == 2:

self.arr[1]=self.arr[1]-1

if self.arr[1] >= 0:

return True

else:

return False

if carType == 3:

self.arr[2]=self.arr[2]-1

if self.arr[2] >= 0:

return True

else:

return False

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_