1. Line reflection

Даны n точек на двумерной плоскости, найдите, существует ли такая прямая, параллельная оси y, которая симметрично отражает данные точки, другими словами, ответьте, существует ли такая прямая, что после отражения всех точек через данную прямую множество исходных точек совпадает с множеством отраженных. Обратите внимание, что могут существовать повторяющиеся точки.

Идея: Такая прямая и существует ТОЛЬКО тогда когда самая крайняя левая точка отражается в самую правую крайнюю точку.

Тут возникает сложность в том, что хоть и координаты точек по условию целые числа, но координата прямой может быть не целой. Как с этим бороться? Нет смысла делить выражение (left_x + right_x) на два, так как мы всегда после этого умножаем значение прямой на два. Т.е **решение проблемы с вещественной координатой следующее: будем хранить не координату прямой, а удвоенной значение координаты.**

Создадим хэш-мапу где ключом будет является координаты точки (x,y) , а значением количество таких точек. Вспоминая условие задачи у нас могут быть повторяющиеся точки. Далее нам нужно пройтись по данной хэш-мапе, для каждого ключа определить точку в которую она отразится относительно нашего кандидата. Если отраженной точки нет в хэш мапе или же значение в отраженной точке не равно значению в отражаемой , то кандидат не подошел. И ответ False Если же мы прошли по всем точкам и для всех кандидат подошел, то ответ True

```
from collections import defaultdict
from math import inf
from typing import List

def is_reflected(points) -> bool:
    point_count = defaultdict(int)
    left_point = inf
    right_point = -inf

    for point in points:
        point_count[tuple(point)] += 1
        left_point = min(left_point, point[0])
        right_point = max(right_point, point[0])

    candidate = left_point + right_point
    for point in points:
        # это кортеж (candidate - point[0], point[1])
        reflected_point = candidate - point[0], point[1]
        if point_count[tuple(point)] != point_count[reflected_point]:
```

```
            return False
    return True
```

2. Longest subarray of 1s after deleting one element

Дан двоичный массив nums, вы можете удалить из него один элемент.

Возвращает размер самого длинного непустого подмассива, содержащего только единицы в результирующем массиве. Вернуть 0, если такого подмассива нет.

```
# O(N)time O(1)space

class Solution:
    def longestSubarray(self, nums):
        N = len(nums)

        all_ones = 0
        one_zero = 0
        res = 0

        for n in nums:
            if n == 1:
                all_ones += 1
                one_zero += 1
            else:
                one_zero = all_ones
                all_ones = 0

            res = max(res, all_ones, one_zero)

        return N - 1 if all_ones == N else res
```

3. Summary ranges

Вам дан отсортированный уникальный целочисленный массив nums.

Диапазон [a,b] — это набор всех целых чисел от a до b (включительно).

Возвращает наименьший отсортированный список диапазонов, точно покрывающий все числа в массиве. То есть каждый элемент nums покрывается ровно одним из диапазонов, и не существует целого числа x, такого что x находится в одном из диапазонов, но не в nums.

Каждый диапазон [a,b] в списке должен выводиться как:

"a->б", если a != б "a", если a == б

```
class Solution:
    def summaryRanges(self, nums):
        if nums == []:
```

```
            return nums

        elif len(nums) == 1:
            return [str(nums[0])]

        else:
            start = nums[0]
            end = nums[0]
            res = []
            for i in range(1, len(nums)):
                if nums[i] - nums[i-1] == 1:
                    end = nums[i]
                else:
                    res.append(self.to_str(start, end))
                    start = end = nums[i]

            res.append(self.to_str(start, end))

            return res

    def to_str(self, start, end):
        if start == end:
            return str(start)
        else:
            return str(start)+"->"+str(end)
```

4. String compression

["a","a","b","b","c","c","c"] -> ["a","2","b","2","c","3"]

```
# Time O(n), Space O(1)

class Solution:
    def compress(self, chars):
        i = 0
        res = 0
        while i < len(chars):
            group_length = 1
            while (i + group_length < len(chars) and chars[i +
group_length] == chars[i]):
                group_length += 1

            chars[res] = chars[i]
            res += 1

            if group_length > 1:
                str_repr = str(group_length)
                chars[res:res+len(str_repr)] = list(str_repr)
                res += len(str_repr)
```

```
            i += group_length
        return res
```

5. Zigzag Iterator

Input:

- v1 = [1,2]
- v2 = [3,4,5,6]

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

```
 class ZigzagIterator(object):

    def __init__(self, v1, v2):
        self.lists = list()
        if len(v1) > 0:
            self.lists.append(v1)
        if len(v2) > 0:
            self.lists.append(v2)

        self.numLists = len(self.lists)
        self.counters = [0] * self.numLists
        self.listCounter = 0

    def next(self):
        res = self.lists[self.listCounter]
[self.counters[self.listCounter]]

        if self.counters[self.listCounter] ==
len(self.lists[self.listCounter]) - 1:
            del self.counters[self.listCounter]
            del self.lists[self.listCounter]
            self.numLists -= 1
        else:
            self.counters[self.listCounter] += 1
            self.listCounter += 1
        if self.listCounter == self.numLists:
            self.listCounter = 0
        return res

    def hasNext(self):
        return self.numLists > 0


# Your ZigzagIterator object will be instantiated and called as such:
# i, v = ZigzagIterator(v1, v2), []
# while i.hasNext(): v.append(i.next())
```

6. Valid Palindrome

Фраза является палиндромом, если после преобразования всех прописных букв в строчные и удаления всех не буквенно-цифровых символов она читается одинаково вперед и назад. Буквенно-цифровые символы включают буквы и цифры.

Дана строка s, вернуть true, если это палиндром, или false в противном случае.

```
# Time O(n), Space: O(1)
class Solution:
    def isPalindrome(self, s: str) -> bool:
        l, r = 0, len(s)-1
        while l < r:
            while l < r and not s[l].isalnum():
                l += 1
            while l <r and not s[r].isalnum():
                r -= 1
            if s[l].lower() != s[r].lower():
                return False
            l +=1; r -= 1
        return True
```

7. One edit distance

Имея две строки s и t, вернуть true, если обе они находятся на расстоянии редактирования друг от друга, в противном случае вернуть false.

Говорят, что строка s находится на расстоянии одного расстояния от строки t, если вы можете:

- Вставить ровно один символ в s, чтобы получить t.
- Удалить ровно один символ из s, чтобы получить t.
- Заменить ровно один символ s другим символом, чтобы получить t.

```
# Time O(n), Space O(1)

class Solution:
    def isOneEditDistance(self, s, t):

        if abs(len(s) - len(t)) > 1:
            return False

        i = j = edits = 0
        while i < len(s) and j < len(t):
            if s[i] == t[j]:
                i, j = i + 1, j + 1
                continue

            edits += 1
```

```
            if edits > 1:
                return False

            if len(s) == len(t):
                i, j = i + 1, j + 1
            else:
                if len(s) > len(t):
                    i += 1
                else:
                    j += 1

        if i < len(s) or j < len(t):
            # Любые оставшиеся символы (максимум 1) должны быть
удалены
            edits += 1

        return edits == 1
```

8. Subarray sum equals K

Дан массив целых чисел nums и целое число k, **вернуть общее количество подмассивов, сумма которых равна k**.

Подмассив — это непрерывная непустая последовательность элементов внутри массива.

- Пройдитесь по массиву и отследите текущую текущую сумму до i-го индекса в переменной, скажем, sum.
- Кроме того, хешируйте различные значения суммы, полученные до сих пор, в хэш-карту.
- Если сумма равна k в любой точке массива, увеличьте количество подмассивов на 1.
- Если это значение суммы превысило k на значение суммы – k, мы можем найти количество подмассивов, найденных до сих пор с суммой = сумма – k, из нашей хэш-карты. Обратите внимание, что если эти подмассивы удалить из нашего текущего массива, мы снова получим сумму k. Итак, мы добавляем к нашему ответу количество подмассивов с суммой = сумма - k, найденных до сих пор из нашей хэш-карты.
- После обхода всего массива один раз и применения описанных выше шагов верните вычисленный результат.

```
class Solution:
    # Time: O(n), Space: O(1)
    def subarraySum(self, nums, k):
        cnt, total = 0, 0
        cache = {0: 1}
```

```
        for v in nums:
            total += v
            if cache.get(total-k):
                cnt += cache[total-k]

            if cache.get(total):
                cache[total] += 1
            else:
                cache[total] = 1
        return cnt
```

9. Move zeroes

[0,1,0,3,12] -> [1,3,12,0,0]

```
# Time O(N), Space O(1)
class Solution:
    def moveZeroes(self, nums):
        count = 0
        n = len(nums)

        for i in range(n):
            if nums[i] != 0:
                nums[count] = nums[i]
                count+=1

        while count < n:
            nums[count] = 0
            count += 1
```

10. Group anagrams

- Input: strs = ["eat","tea","tan","ate","nat","bat"]
- Output: [["bat"],["nat","tan"],["ate","eat","tea"]]

```
class Solution:
    def groupAnagrams(self, li):
        hash_map = {}

        for word in li:
            sorted_word = ''.join(sorted(word))

            if sorted_word not in hash_map:
                hash_map[sorted_word] = [word]

            else:
                hash_map[sorted_word] += [word]

        return [hash_map[i] for i in hash_map]
```

11. Insert Delete GetRandom O(1)

```python
import random

class RandomizedSet():
    def __init__(self):
        self.val_to_index = dict()
        self.val_list = list()

    def insert(self, val):
        if val in self.val_to_index:
            return False

        self.val_to_index[val] = len(self.val_list)
        self.val_list.append(val)
        return True

    def remove(self, val):
        if val not in self.val_to_index:
            return False

        last_val = self.val_list[-1]
        val_index = self.val_to_index[val]

        if last_val != val:
            self.val_to_index[last_val] = val_index
            self.val_list[val_index] = last_val

        self.val_list.pop()
        self.val_to_index.pop(val)
        return True

    def getRandom(self):
        return random.choice(self.val_list)
```

12. LRU Cache

```python
# Простое

from collections import deque

class LRUCache:
    def __init__(self, capacity: int):
        self.cache = deque()
        self.dict = {}
        self.capacity = capacity


    def get(self, key: int) -> int:
        if key in self.dict.keys():
            self.put(key, self.dict[key])
```

```
            return self.dict[key]
        else:return -1

    def put(self, key: int, value: int) -> None:
        if key in self.dict.keys():
            self.dict[key] = value
            self.cache.remove(key)
            self.cache.append(key)
        else:
            if len(self.dict) == self.capacity:
                keytoremove = self.cache[0]
                self.cache.remove(keytoremove)
                del(self.dict[keytoremove])

            self.cache.append(key)
            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)

# Время
# O(1)
# Поиск занимает всего O (1), так как мы храним в HashMap
# Добавление и удаление узла займет O (1), так как мы используем
двусвязный список.

# Память

# O(n)
# N для Hashmap для каждого узла
# N для N узлов в двусвязном списке

class Node:
    def __init__(self, key=None, value=None, next=None, prev=None):
        self.key = key
        self.value = value
        self.next = next
        self.prev = prev

class DoublyLinkedList:
    def __init__(self):
        self.head = Node()
        self.tail = Node()
        self.head.next = self.tail
        self.tail.next = self.head
```

```
    def remove_node(self, node):
        prev = node.prev
        next = node.next
        prev.next = next
        next.prev = prev
        return node

    def remove_from_tail(self):
        prev = self.tail.prev
        self.tail.prev = prev.prev
        prev.prev.next = self.tail
        return prev

    def add_to_head(self, node):
        next = self.head.next
        self.head.next = node
        node.prev = self.head
        node.next = next
        next.prev = node

    def move_to_head(self, node):
        node = self.remove_node(node)
        self.add_to_head(node)

class LRUCache:
    def __init__(self, capacity: int):
        self.capacity = capacity
        self.cache = {}
        self.dll = DoublyLinkedList()

    def get(self, key: int) -> int:
        if key in self.cache:
            node = self.cache[key]
            self.dll.move_to_head(node)
            return node.value
        else:
            return -1

    def put(self, key: int, value: int) -> None:
        if key in self.cache:
            node = self.cache[key]
            self.dll.move_to_head(node)
            node.value = value
        else:
            node = Node(key, value)
            if len(self.cache) == self.capacity:
                removed_node = self.dll.remove_from_tail()
                del self.cache[removed_node.key]
            self.dll.add_to_head(node)
```

```
            self.cache[key] = node


# Your LRUCache object will be instantiated and called as such:
# obj = LRUCache(capacity)
# param_1 = obj.get(key)
# obj.put(key,value)
# Your LRUCache object will be instantiated and called as such:
# obj = LRUCache(capacity)
# param_1 = obj.get(key)
# obj.put(key,value)
```

13. Generate Parentheses

```
# Time O((4^n)/sqrt(n))
# Он основан на понимании того, сколько элементов содержится в
функции.
# Это указывает на n-е каталонское число, которое асимптотически
ограничено C_n = (4^n)/(n*sqrt(n)).
# Поскольку каждая допустимая последовательность имеет максимум n
шагов,
# следовательно, временная сложность будет (4^n)/sqrt(n).

# Space (4^n)/sqrt(n) для рекурсивных вызовов и O(n) для хранения
последовательности.

class Solution:
    def generateParenthesis(self, n):

        def generate(result, s, _open, _close, n):
            if _open == n and _close == n:
                result.append(s)
                return
            if _open < n:
                generate(result, s + "(", _open + 1, _close, n)
            if _close < _open:
                generate(result, s + ")", _open, _close + 1, n)

        result = []

        generate(result, "", 0, 0, n)
        return result
```

14. Reverse Linked List

Односвязный

```
from typing import Optional

# Time O(n) Space O(1)
```

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

class Solution:
    def reverseList(self, head: Optional[ListNode]) ->
Optional[ListNode]:
        prev = None
        current = head

        while(current is not None):
            next = current.next
            current.next = prev
            prev = current
            current = next
        head = prev
        return head
```

Двусвязный

```
def reverse(head):
        temp = None
        current = head

        # Swap next and prev for all nodes
        # of doubly linked list

        while current is not None:
            temp = current.prev
            current.prev = current.next
            current.next = temp
            current = current.prev

        # Before changing head, check for the
        # cases like empty list and list with
        # only one node
        if temp is not None:
            head = temp.prev
```

15. Permutation in string

Вернуть true, если одна из перестановок s1 является подстрокой s2.

- Input: s1 = "ab", s2 = "eidbaooo"
- Output: true
- Input: s1 = "ab", s2 = "eidboaoo"
- Output: false

```
class Solution:
    def checkInclusion(self, s1: str, s2: str) -> bool:
```

```
        if len(s2) < len(s1):
            return False

        freq_s1, freq_s2 = [0]*26, [0]*26

        for x in s1:
            freq_s1[ord(x)-ord('a')] += 1

        for x in range(0, len(s2)-len(s1)+1):
            if x == 0:
                for y in range(x, x+len(s1)):
                    freq_s2[ord(s2[y]) - ord('a')] +=  1

            else:
                freq_s2[ord(s2[x+len(s1)-1]) - ord('a')] +=  1

            if freq_s2 == freq_s1:
                return True

            freq_s2[ord(s2[x]) - ord('a')] -=  1

        return False
```

16. Merge K Linked Lists

- Соедините k списков и объедините каждую пару.
- После первого спаривания k списков объединяются в k/2 списков со средней длиной 2N/k, затем k/4, k/8 и так далее.
- Повторяем эту процедуру, пока не получим окончательный отсортированный связанный список

Таким образом, мы будем проходить почти N узлов за соединение и слияние и повторять эту процедуру около logk раз.

```
# Time O(n*log(K))
# Space O(1)

class Solution:
    def mergeKLists(self, lists: List[Optional[ListNode]]) ->
Optional[ListNode]:
        amount = len(lists)
        interval = 1

        while interval < amount:
            for i in range(0, amount - interval, interval * 2):
                lists[i] = self.merge2Lists(lists[i], lists[i +
interval])
            interval *= 2

        return lists[0] if amount > 0 else None
```

```
    def merge2Lists(self, l1, l2):
        head = point = ListNode(0)
        while l1 and l2:
            if l1.val <= l2.val:
                point.next = l1
                l1 = l1.next
            else:
                point.next = l2
                l2 = l1
                l1 = point.next.next
            point = point.next

        if not l1:
            point.next = l2
        else:
            point.next = l1

        return head.next
```

17. Number of recent calls

Реализуйте класс RecentCounter:

- **RecentCounter()** Инициализирует счетчик с нулевыми последними запросами.
- **int ping(int t)** Добавляет новый запрос в момент времени t, где t представляет некоторое время в миллисекундах, и возвращает количество запросов, выполненных за последние 3000 миллисекунд (включая новый запрос). В частности, вернуть количество запросов, которые произошли в инклюзивном диапазоне [t - 3000, t].
- Гарантируется, что каждый **вызов ping использует строго большее значение t, чем предыдущий вызов**.

```
class RecentCounter:
    def __init__(self):
        self.data = []

    def ping(self, t):
        while self.data and t - self.data[0] > 3000:
            self.data.pop(0)

        self.data.append(t)
        return len(self.data)
```

18. Valid Parenthesis

Проверить баланс скобок

```
class Solution:
    # Time: O(n) / Space: O(n)
```

```
    def isValid(self, s: str) -> bool:
        stack = []
        for char in s:
            if char in ["(", "{", "["]:
                stack.append(char)
            else:
                if not stack:
                    return False
                current_char = stack.pop()
                if current_char == '(':
                    if char != ")":
                        return False
                if current_char == '{':
                    if char != "}":
                        return False
                if current_char == '[':
                    if char != "]":
                        return False

        if stack:
            return False
        return True
```

19. Max consecutive ones ii

Есть двоичный массив, найдите максимальное количество последовательных единиц в этом массиве, если вы можете превратить в 1 не более одного 0.

- Input: [1,0,1,1,0]
- Output: 4

```
class Solution:
    def findMaxConsecutiveOnes(self, nums) -> int:
        ans = 0
        last_zero_index = -1
        l = 0

        for r, num in enumerate(nums):
            if num == 0:
                l = last_zero_index + 1
                last_zero_index = r
            ans = max(ans, r - l + 1)

        return ans
```

20. Maximize Distance to Closest Person

- Вам дан массив, представляющий ряд мест, где seat[i] = 1 представляет человека, сидящего на i-м месте, а seat[i] = 0 означает, что i-е место пусто (индексировано 0).

- Есть по крайней мере одно свободное место и по крайней мере один сидящий человек.

Алекс хочет сесть на сиденье так, чтобы расстояние между ним и ближайшим к нему человеком было максимальным. Верните это максимальное расстояние ближайшему человеку.

- Input: seats = [1,0,0,0,1,0,1]
- Output: 2

Пояснение

- Если Алекс сидит на втором свободном месте (т. е. на месте[2]), то расстояние до ближайшего человека равно 2.
- Если Алекс сидит на любом другом свободном месте, расстояние до ближайшего человека равно 1.

Таким образом, максимальное расстояние до ближайшего человека равно 2.

```
# Time O(n); Space O(1)

# last - указатель последнего сидящего места

class Solution:
    def maxDistToClosest(self, seats) -> int:
        res, last, n = 0, -1, len(seats)
        for i in range(n):
            if seats[i]:
                res = max(res, i if last < 0 else (i - last) / 2)
                last = i
        return max(res, n - last - 1)
```

21. Design Hit Counter

Разработайте счетчик посещений, который подсчитывает количество посещений, полученных за последние 5 минут.

```
# Решение через очередь

class HitCounter:

    #TC: O(1)
    #SC: O(1)

    def __init__(self):
        """
        Initialize your data structure here.
        """
        self.Q = collections.deque([])
```

```
    def hit(self, timestamp: int) -> None:
        """
        Record a hit.
        @param timestamp - The current timestamp (in seconds
granularity).
        """
        self.Q.append(timestamp)

    def getHits(self, timestamp: int) -> int:
        """
        Return the number of hits in the past 5 minutes.
        @param timestamp - The current timestamp (in seconds
granularity).
        """
        while(len(self.Q) > 0 and self.Q[0] <= timestamp - 300):
            self.Q.popleft()
        return len(self.Q)

# T: O(1)
# S: O(1)

class HitCounter:
    def __init__(self):
        self.timestamps = [0] * 300
        self.hits = [0] * 300

    def hit(self, timestamp: int) -> None:
        i = timestamp % 300

        if self.timestamps[i] == timestamp:
            self.hits[i] += 1
        else:
            self.timestamps[i] = timestamp
            self.hits[i] = 1  # Reset hit count to 1

    def getHits(self, timestamp: int) -> int:
        return sum(h for t, h in zip(self.timestamps, self.hits) if
timestamp - t < 300)
```

22. Merge Intervals

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

1 способ

```
class Solution:
    # Time: O(nlogn) / Space: O(1)
    def merge(self, arr):
```

```
        arr.sort(key = lambda x: x[0])
        m = []

        s = -10000
        max = -100000

        for i in range(len(arr)):
            a = arr[i]
            if a[0] > max:
                if i != 0:
                    m.append([s,max])
                max = a[1]
                s = a[0]
            else:
                if a[1] >= max:
                    max = a[1]

        if max != -100000 and [s, max] not in m:
            m.append([s, max])

        return m
```

2 способ

```
class Solution:
    # Time: O(nlogn) / Space: O(n)
    def merge(self, intervals):

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

        merged = []
        for interval in intervals:
            # если список объединенных интервалов пуст или текущий
            # интервал не пересекается с предыдущим, просто добавляем
его.
            if not merged or merged[-1][1] < interval[0]:
                merged.append(interval)
            else:
            # в противном случае происходит перекрытие, поэтому мы
объединяем
            # текущий и предыдущий интервалы.
                merged[-1][1] = max(merged[-1][1], interval[1])

        return merged
```

- Временная сложность: O(nlogn)

Помимо вызова сортировки, мы выполняем простое линейное сканирование списка, поэтому во время выполнения преобладает сложность сортировки O(nlogn).

- Пространственная сложность: O(logN) (или O(n))

Если мы можем сортировать интервалы на месте, нам не потребуется больше, чем постоянное дополнительное пространство, хотя сама сортировка занимает O(logn) пространства. В противном случае мы должны выделить линейное пространство для хранения копии интервалов и отсортировать ее.

23. Trapping Rain Water

Имея n неотрицательных целых чисел, представляющих карту высот, где ширина каждой полосы равна 1, вычислите, сколько воды она может собрать после дождя.

- Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
- Output: 6
- Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

```
class Solution:
    def trap(self, height: List[int]) -> int:
        left = 0
        right = len(height) - 1
        ans = 0
        left_max = 0
        right_max = 0

        while (left < right):
            if height[left] < height[right]:
                if height[left] >= left_max:
                    left_max = height[left]
                else:
                    ans += (left_max - height[left])
                left += 1
            else:
                if height[right] >= right_max:
                    right_max = height[right]
                else:
                    ans += (right_max - height[right])
                right -= 1
        return ans
```

24. Two Sum

- Дан массив целых чисел nums и целочисленную цель, вернуть индексы двух чисел так, чтобы они складывались в цель.
- Вы можете предположить, что каждый вход будет иметь ровно одно решение, и вы не можете использовать один и тот же элемент дважды.
- Вы можете вернуть ответ в любом порядке.

- Input: nums = [2,7,11,15], target = 9
- Output: [0,1]
- Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

```
# Time O(n); Space O(n)

class Solution:
    def twoSum(self, nums, target):
        hashmap = {}

        for i in range(len(nums)):
            tmp = target - nums[i]

            if tmp in hashmap:
                return [i, hashmap[tmp]]
            hashmap[nums[i]] = i
```

25. Meeting rooms ii

Дан массив интервалов времени проведения собраний, где intervals[i] = [start_i, end_i], возвращает минимальное количество требуемых конференц-залов.

- Input: [[0, 30], [5, 10], [15, 20]]
- Output: 2
- Input: [[7,10], [2,4]]
- Output: 1

```
# Time: O(nlogn)
# Space: O(N)

class Solution:
    def minMeetingRooms(self, intervals) -> int:
        n = len(intervals)
        ans = 0
        starts = []
        ends = []

        for start, end in intervals:
            starts.append(start)
```

```
            ends.append(end)

        starts.sort()
        ends.sort()

        j = 0
        for i in range(n):
            if starts[i] < ends[j]:
                ans += 1
            else:
                j += 1

        return ans
```

26. Find All Anagrams in a string

Имея две строки s и p, вернуть массив всех начальных индексов анаграмм p в s. Вы можете вернуть ответ в любом порядке.

Анаграмма — это слово или фраза, образованная путем перестановки букв другого слова или фразы, обычно с использованием всех исходных букв ровно один раз.

Алгоритм:

- Пусть res представляет список результатов, который мы хотим вернуть. Изначально res пустой.
- Если len(p) >= len(s), то мы можем вернуть [], так как не может быть анаграммы слова s в слове p
- Поскольку слова написаны строчными буквами английского алфавита, как указано в условии задачи, мы можем использовать список для заполнения частотного массива для обоих слов.
- Во-первых, выполните итерацию по слову p и заполните массив частот для p
- Возьмите скользящее окно len(p) и проведите по массиву s. Обновить список частот для слова s для каждого окна
- Для i==0 нам нужно заполнить все элементы в скользящем окне и уменьшить счетчик первого символа скользящего окна и закончить итерацию.
- Для случаев i > 0 нам нужно увеличить последний символ скользящего окна и начало итерации и уменьшить первый символ скользящего окна и конец итерации
- Если оба списка одинаковы, добавьте индекс в список результатов.
- Вернуть список результатов.

```
class Solution:
    # Time: O(|s| + |p|)
    # Space: O(1)
```

```
    def findAnagrams(self, s: str, p: str):
        cnt = Counter(p)

        ans = []
        l = 0
        for r, c in enumerate(s):
            cnt[c] -= 1
            while cnt[c] < 0:  # If number of characters `c` is more
than our expectation
                cnt[s[l]] += 1  # Slide left until cnt[c] == 0
                l += 1
            if r - l + 1 == len(p):  #  If we already filled enough
`p.length()` chars
                ans.append(l)  # Add left index `l` to our result

        return ans
```

27. Implement Rand10() using Rand7()

```
class Solution:
    def rand10(self):
        v1, v2 = rand7(), rand7()
        while v1 > 5:
            v1 = rand7()
        while v2 == 7:
            v2 = rand7()
        return v1 if (v2 <= 3) else (v1 + 5)
```

28. Check symmetric tree

```
# Time Complexity: O(N)
# Auxiliary Space: O(h) where h is the maximum height of the tree

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: Optional[TreeNode]) -> bool:
        if not root:
            return True
        return self.check_leftright(root.left,root.right)

    def check_leftright(self, node1, node2):
        if not node1 and not node2:
            return True

        elif node1 and node2 and  node1.val == node2.val:
            return self.check_leftright(node1.left,node2.right) and
self.check_leftright(node1.right,node2.left)
```

```
        else:
            return False
```

29. Longest substring without repeating characters

```
# Time O(n)
# Space O(m), m - мощность множества входных символов

class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        seen = {}
        l = 0
        output = 0
        for r in range(len(s)):
            # Если s[r] не в seen, мы можем продолжать увеличивать
размер окна, перемещая указатель вправо
            if s[r] not in seen:
                output = max(output, r-l+1)
            else:
                if seen[s[r]] < l:
                    # s[r] не находится внутри текущего окна,
                    # мы можем продолжать увеличивать окно
                    output = max(output,r-l+1)
                else:
                    # s[r] находится внутри текущего окна, нам нужно
изменить окно,
                    #     переместив левый указатель на seen[s[r]] +
1.
                    l = seen[s[r]] + 1
            seen[s[r]] = r
        return output
```

30. Validate Binary Tree

```
# Time O(n)
# Space O(n)

import math

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

        def validate(node, low=-math.inf, high=math.inf):
            # Empty trees are valid BSTs.
            if not node:
                return True
            # The current node's value must be between low and high.
            if node.val <= low or node.val >= high:
                return False

            # The left and right subtree must also be valid.
```

```
            return (validate(node.right, node.val, high) and
                    validate(node.left, low, node.val))

        return validate(root)

---------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
last)
/var/folders/1f/0qfny53s1f54hmwr3c2dpq9w0000gn/T/ipykernel_42318/27442
46509.py in <module>
      4 import math
      5
----> 6 class Solution:
      7     def isValidBST(self, root: TreeNode) -> bool:
      8

/var/folders/1f/0qfny53s1f54hmwr3c2dpq9w0000gn/T/ipykernel_42318/27442
46509.py in Solution()
      5
      6 class Solution:
----> 7     def isValidBST(self, root: TreeNode) -> bool:
      8
      9         def validate(node, low=-math.inf, high=math.inf):

NameError: name 'TreeNode' is not defined
```

```python
# Time O(n)
# Space O(n)

class Solution:
    def isValidBST(self, root: TreeNode) -> bool:
        stack, prev = [], -math.inf

        while stack or root:
            while root:
                stack.append(root)
                root = root.left
            root = stack.pop()
            # Если следующий элемент в порядке обхода
            # меньше предыдущего
            # это не BST.
            if root.val <= prev:
                return False
            prev = root.val
            root = root.right

        return True
```

31. Num of islands

Максимум m*n рекурсивных вызовов будет открыто в один момент времени, и вызовы не будут повторяться, потому что каждая ячейка помечается нулем после ее проверки.

```
# Time O(m*n), Space O(m*n)

def numIslands(self, grid):
    if not grid:
        return 0

    count = 0
    for i in range(len(grid)):
        for j in range(len(grid[0])):
            if grid[i][j] == '1':
                self.dfs(grid, i, j)
                count += 1
    return count

def dfs(self, grid, i, j):
    if i<0 or j<0 or i>=len(grid) or j>=len(grid[0]) or grid[i][j] !=
'1':
        return
    grid[i][j] = '#'
    self.dfs(grid, i+1, j)
    self.dfs(grid, i-1, j)
    self.dfs(grid, i, j+1)
    self.dfs(grid, i, j-1)
```

32. Flatten Nested List Iterator

Вам дан вложенный список целых чисел nestedList. Каждый элемент является либо целым числом, либо списком, элементы которого также могут быть целыми числами или другими списками. Реализуйте итератор, чтобы сгладить его.

```
class NestedInteger:
    def isInteger(self) -> bool:
        """
        @return True if this NestedInteger holds a single integer,
rather than a nested list.
        """
        pass
    def getInteger(self) -> int:
        """
        @return the single integer that this NestedInteger holds, if
it holds a single integer
        Return None if this NestedInteger holds a nested list
        """
```

```
        pass

    def getList(self):
        """
        @return the nested list that this NestedInteger holds, if it
holds a nested list
        Return None if this NestedInteger holds a single integer
        """
        pass

class NestedIterator:
    def __init__(self, nestedList: [NestedInteger]):
        self.data = []
        self.flatten(nestedList)

    def flatten(self, lst):
        for el in lst:
            if el.isInteger():
                self.data.append(el.getInteger())
            else:
                self.flatten(el.getList())

    def hasNext(self) -> bool:
        return len(self.data)

    def next(self) -> int:
        return self.data.pop(0)
```

33. Consecutive Characters

Мощность строки — это максимальная длина непустой подстроки, содержащей только один уникальный символ. Учитывая строку s, вернуть мощность s.

```
# Time O(N)
# Space O(1)

class Solution:
    def maxPower(self, s: str) -> int:
        count = 0
        max_count = 0
        previous = None
        for c in s:
            if c == previous:
                # if same as previous one, increase the count
                count += 1
            else:
                # else, reset the count
                previous = c
                count = 1
```

```
            max_count = max(max_count, count)
        return max_count
```

34. Interval List Intersections

Вам дано два списка закрытых интервалов, firstList и secondList, где firstList[i] = [starti, endi] и secondList[j] = [startj, endj]. Каждый список интервалов попарно не пересекается и отсортирован.

Возвращает пересечение этих двух списков интервалов.

Замкнутый интервал [a, b] (с a <= b) обозначает множество действительных чисел x с a <= x <= b.

Пересечение двух закрытых интервалов представляет собой набор действительных чисел, которые либо пусты, либо представлены в виде замкнутого интервала. Например, пересечение [1, 3] и [2, 4] равно [2, 3].

- Input: firstList = [[0,2],[5,10],[13,23],[24,25]], secondList = [[1,5],[8,12],[15,24],[25,26]]
- Output: [[1,2],[5,5],[8,10],[15,23],[24,24],[25,25]]

```
# Временная сложность: O(M+N), где M,N — длины A и B соответственно.
# Сложность по памяти: O(M+N)

class Solution:
    def intervalIntersection(self, A: List[List[int]], B:
List[List[int]]) -> List[List[int]]:
        ans = []
        i = j = 0

        while i < len(A) and j < len(B):
            # Проверим, пересекается ли A[i] с B[j].
            # lo - начальная точка пересечения
            # hi - конечная точка перекрестка
            lo = max(A[i][0], B[j][0])
            hi = min(A[i][1], B[j][1])
            if lo <= hi:
                ans.append([lo, hi])

            # Удалить интервал с наименьшей конечной точкой
            if A[i][1] < B[j][1]:
                i += 1
            else:
                j += 1

        return ans
```