good evening. hope its all good. i tried my best.

Путь в бинарном дереве — это последовательность узлов, в которой каждая пара соседних узлов в последовательности имеет соединяющее их ребро . Узел может появиться в последовательности не более одного раза . Обратите внимание, что путь не обязательно должен проходить через корень.

Сумма путей пути — это сумма значений узлов в пути.

Учитывая rootдвоичное дерево, верните максимальную сумму путей любого непустого пути .

Пример 1: exx1.jpg

Входные данные: root = [1,2,3] Выходные данные: 6 Объяснение: Оптимальный путь — 2 -> 1 -> 3 с суммой путей 2 + 1 + 3 = 6. Пример 2:

exx2.jpg Ввод: root = [-10,9,20,null,null,15,7] Выход: 42 Объяснение: Оптимальный путь — 15 -> 20 -> 7 с суммой путей 15 + 20 + 7 = 42.

Ограничения:

Количество узлов в дереве находится в диапазоне .[1, 3 \* 104] -1000 <= Node.val <= 1000

class TreeNode:  
 def \_\_init\_\_(self, value=0, left=None, right=None):  
 self.value = value  
 self.left = left  
 self.right = right  
  
def maxPathSum(root):  
 def maxSum(node):  
 nonlocal max\_sum  
 if not node:  
 return 0  
 left\_sum = max(maxSum(node.left), 0)  
 right\_sum = max(maxSum(node.right), 0)  
 max\_sum = max(max\_sum, left\_sum + right\_sum + node.value)  
 return max(left\_sum, right\_sum) + node.value  
  
 max\_sum = float('-inf')  
 maxSum(root)  
 return max\_sum  
  
root1 = TreeNode(1, TreeNode(2), TreeNode(3))  
root2 = TreeNode(-10, TreeNode(9), TreeNode(20, TreeNode(15), TreeNode(7)))  
  
print(maxPathSum(root1))  
print(maxPathSum(root2))

6  
42

ЗАДАЧА 2

Вам дан целочисленный массив numsи два целых числа indexDiffи valueDiff.

Найдите пару индексов (i, j)такую, что:

i != j, abs(i - j) <= indexDiff. abs(nums[i] - nums[j]) <= valueDiff, и Возврат, trueесли такая пара существует или falseнет .

Пример 1:

Ввод:

nums = [1,2,3,1], indexDiff = 3, valueDiff = 0  
 Выход: true  
 Объяснение: Мы можем выбрать (i, j) = (0, 3).

Мы удовлетворяем трем условиям:

i != j --> 0 != 3  
  
abs(i - j) <= indexDiff --> abs(0 - 3) <= 3  
abs(nums[i] - nums[j]) <= valueDiff --> abs(1 - 1) <= 0

Пример 2:

Ввод: nums = [1,5,9,1,5,9], indexDiff = 2, valueDiff = 3  
  
 Вывод: false

Объяснение: перепробовав все возможные пары (i, j), мы не можем удовлетворить три условия, поэтому мы возвращаем ложь.

Ограничения:

2 <= nums.length <= 10^5  
  
-10^9 <= nums[i] <= 10^9  
  
1 <= indexDiff <= nums.length  
  
0 <= valueDiff <= 10^9

def containsNearbyAlmostDuplicate(nums, indexDiff, valueDiff):  
 if len(nums) <= 1:  
 return False  
  
 for i in range(len(nums)):  
 for j in range(i+1, min(i+indexDiff+1, len(nums))):  
 if abs(nums[i] - nums[j]) <= valueDiff:  
 return True  
 return False  
  
nums1 = [1, 2, 3, 1]  
indexDiff1 = 3  
valueDiff1 = 0  
print(containsNearbyAlmostDuplicate(nums1, indexDiff1, valueDiff1))  
  
nums2 = [1, 5, 9, 1, 5, 9]  
indexDiff2 = 2  
valueDiff2 = 3  
print(containsNearbyAlmostDuplicate(nums2, indexDiff2, valueDiff2))

True  
False

ЗАДАЧА 3

Компании

Медиана — это среднее значение в упорядоченном целочисленном списке . Если размер списка четный, среднего значения нет, а медиана — это среднее значение двух средних значений.

Например, для arr = [2,3,4]медиана равна 3.

Например, для arr = [2,3]медиана равна (2 + 3) / 2 = 2.5.

Реализуйте класс MedianFinder:

MedianFinder() инициализирует MedianFinderобъект.

void addNum(int num)добавляет целое число numиз потока данных в структуру данных.

double findMedian()возвращает медиану всех элементов на данный момент. Принимаются ответы в пределах фактического ответа.10-5

Пример 1:

Вход

["MedianFinder", "addNum", "addNum", "findMedian", "addNum", "findMedian"]  
[[], [1], [2], [], [3], []]  
Выход  
[ноль, ноль, ноль, 1,5, ноль, 2,0]

Объяснение

MedianFinder medianFinder = новый MedianFinder();  
medianFinder.addNum(1); //объем = [1]  
medianFinder.addNum(2); //объем = [1, 2]  
medianFinder.findMedian(); // возвращаем 1,5 (т.е. (1 + 2)/2)  
medianFinder.addNum(3); // Арр[1, 2, 3]  
medianFinder.findMedian(); // возвращаем 2.0

Ограничения:

-105 <= num <= 105

Перед вызовом в структуре данных будет хотя бы один элемент findMedian. В большинстве случаев вызовы будут осуществляться на адреса и ``` .5 \* 10^4 addNum findMedian

``` Следовать за:

Если все целые числа из потока находятся в диапазоне [0, 100], как бы вы оптимизировали свое решение? Если 99%бы все целые числа из потока находились в диапазоне [0, 100], как бы вы оптимизировали свое решение?

import heapq  
  
class MedianFinder:  
  
 def \_\_init\_\_(self):  
 self.max\_heap = []  
 self.min\_heap = []  
  
 def addNum(self, num: int) -> None:  
 if not self.max\_heap or num <= -self.max\_heap[0]:  
 heapq.heappush(self.max\_heap, -num)  
 else:  
 heapq.heappush(self.min\_heap, num)  
  
 if len(self.max\_heap) > len(self.min\_heap) + 1:  
 heapq.heappush(self.min\_heap, -heapq.heappop(self.max\_heap))  
 elif len(self.min\_heap) > len(self.max\_heap):  
 heapq.heappush(self.max\_heap, -heapq.heappop(self.min\_heap))  
  
 def findMedian(self) -> float:  
 if len(self.max\_heap) == len(self.min\_heap):  
 return (-self.max\_heap[0] + self.min\_heap[0]) / 2  
 else:  
 return -self.max\_heap[0]

# код создает класс MedianFinder, в котором используются две приоритетные очереди из стандартной библиотеки Python. Метод addNum добавляет число в структуру данных, а метод findMedian возвращает медиану текущего набора чисел. Чтобы оптимизировать решение для случая, когда все целые числа из потока находятся в диапазоне от 0 до 100, можно рассмотреть использование более простой структуры данных для хранения значений, такой как список или массив. Для случая, когда 99% чисел находятся в диапазоне от 0 до 100, можно рассмотреть использование дополнительной структуры данных для подсчета чисел в этом диапазоне и уменьшить сложность операций с числами вне этого диапазона.

Задача 4 Компании Загадка с n ферзями — это задача о том, как разместить nферзей на n x nшахматной доске так, чтобы никакие два ферзя не атаковали друг друга.

Учитывая целое число n, верните все различные решения головоломки с n ферзями . Вы можете вернуть ответ в любом порядке .

Каждое решение содержит отдельную конфигурацию доски для размещения n ферзей, где 'Q'и '.'оба обозначают ферзя и пустое место соответственно.

Пример 1:

изображение.png Вход: n = 4

Выход: [[".Q..","...Q","Q...","..Q."],["..Q.","Q.. .","...Q",".Q.."]]  
 Пояснение: Существует два различных решения головоломки с четырьмя ферзями, как показано выше.

Пример 2:

Вход: n = 1  
 Выход: [["Q"]]

Ограничения:

1 <= n <= 9

def solveNQueens(n):  
 def could\_place(row, col):  
 return not (cols[col] + left\_diag[row - col] + right\_diag[row + col])  
  
 def place\_queen(row, col):  
 queens\_pos.append((row, col))  
 cols[col] = 1  
 left\_diag[row - col] = 1  
 right\_diag[row + col] = 1  
  
 def remove\_queen(row, col):  
 queens\_pos.pop()  
 cols[col] = 0  
 left\_diag[row - col] = 0  
 right\_diag[row + col] = 0  
  
 def add\_solution():  
 sol = []  
 for \_, col in queens\_pos:  
 sol.append('.'\*col + 'Q' + '.'\*(n - col - 1))  
 solutions.append(sol)  
  
 def backtrack(row):  
 for col in range(n):  
 if could\_place(row, col):  
 place\_queen(row, col)  
 if row + 1 == n:  
 add\_solution()  
 else:  
 backtrack(row + 1)  
 remove\_queen(row, col)  
  
 cols = [0] \* n  
 left\_diag = [0] \* (2 \* n - 1)  
 right\_diag = [0] \* (2 \* n - 1)  
 queens\_pos = []  
 solutions = []  
  
 backtrack(0)  
 return solutions  
  
print(solveNQueens(4))  
print(solveNQueens(1))

[['.Q..', '...Q', 'Q...', '..Q.'], ['..Q.', 'Q...', '...Q', '.Q..']]  
[['Q']]

#Задача 5

Задача 5: Учитывая rows x cols двоичный файл matrix, заполненный символами 0's и 1's, найдите самый большой прямоугольник, содержащий только 1's, и верните его площадь .

Пример 1:

изображение.png Ввод:

матрица = [["1","0","1","0","0"],["1","0","1","1","1"],[ "1","1","1","1","1"],["1","0","0","1","0"]]

Выход:

6. Пояснение: Максимальное прямоугольник показан на рисунке выше.

Пример 2:

Вход: матрица = [["0"]]  
 Выход: 0  
Пример 3:

Входные данные: матрица = [["1"]]  
 Выходные данные: 1

Ограничения:

rows == matrix.length  
cols == matrix[i].length  
1 <= row, cols <= 200  
matrix[i][j]есть '0'или '1'.

def maximalRectangle(matrix):  
 if not matrix:  
 return 0  
  
 heights = [0] \* len(matrix[0])  
 max\_area = 0  
  
 def max\_area\_in\_histogram(heights):  
 stack = [-1]  
 area = 0  
 for i in range(len(heights)):  
 while stack[-1] != -1 and heights[i] <= heights[stack[-1]]:  
 h = heights[stack.pop()]  
 w = i - stack[-1] - 1  
 area = max(area, h \* w)  
 stack.append(i)  
 while stack[-1] != -1:  
 h = heights[stack.pop()]  
 w = len(heights) - stack[-1] - 1  
 area = max(area, h \* w)  
 return area  
  
 for row in matrix:  
 for i, val in enumerate(row):  
 heights[i] = heights[i] + 1 if val == "1" else 0  
 max\_area = max(max\_area, max\_area\_in\_histogram(heights))  
 return max\_area  
  
matrix1 = [["1","0","1","0","0"],["1","0","1","1","1"],["1","1","1","1","1"],["1","0","0","1","0"]]  
print(maximalRectangle(matrix1))  
  
matrix2 = [["0"]]  
print(maximalRectangle(matrix2))  
  
matrix3 = [["1"]]  
print(maximalRectangle(matrix3))

6  
0  
1

#Задача 6

Вам дан массив prices, в котором prices[i]указана цена данной акции на текущий день.ith

Найдите максимальную прибыль, которую вы можете получить. Вы можете совершить не более двух транзакций .

Примечание. Вы не можете совершать несколько транзакций одновременно (т. е. вы должны продать акции, прежде чем купить их снова).

Пример 1:

Входные данные: цены = [3,3,5,0,0,3,1,4]  
 Выходные данные: 6  
 Объяснение: Покупайте в день 4 (цена = 0) и продавайте в день 6 (цена = 3), прибыль = 3- 0 = 3.  
Затем купите в день 7 (цена = 1) и продайте в день 8 (цена = 4), прибыль = 4-1 = 3.

Пример 2:

Входные данные: цены = [1,2,3,4,5]  
 Выходные данные: 4  
 Объяснение: Покупайте в день 1 (цена = 1) и продавайте в день 5 (цена = 5), прибыль = 5-1 = 4.  
Обратите внимание, что вы не можете купить в первый день, купить во второй день и продать позже, поскольку вы совершаете несколько транзакций одновременно. Вы должны продать, прежде чем покупать снова.

Пример 3:

Входные данные: цены = [7,6,4,3,1]  
 Выходные данные: 0  
 Объяснение: В этом случае транзакция не выполняется, т.е. максимальная прибыль = 0.

Ограничения:

1 <= prices.length <= 105  
0 <= prices[i] <= 105

def maxProfit(prices):  
 if not prices:  
 return 0  
  
 n = len(prices)  
 max\_profit\_one = [0] \* n  
 max\_profit\_two = [0] \* n  
  
 min\_price = prices[0]  
 for i in range(1, n):  
 min\_price = min(min\_price, prices[i])  
 max\_profit\_one[i] = max(max\_profit\_one[i-1], prices[i] - min\_price)  
  
 max\_price = prices[n-1]  
 for i in range(n-2, -1, -1):  
 max\_price = max(max\_price, prices[i])  
 max\_profit\_two[i] = max(max\_profit\_two[i+1], max\_price - prices[i])  
  
 max\_profit = max\_profit\_two[0]  
 for i in range(1, n):  
 max\_profit = max(max\_profit, max\_profit\_one[i-1] + max\_profit\_two[i])  
  
 return max\_profit  
  
prices1 = [3, 3, 5, 0, 0, 3, 1, 4]  
print(maxProfit(prices1))  
  
prices2 = [1, 2, 3, 4, 5]  
print(maxProfit(prices2))  
  
prices3 = [7, 6, 4, 3, 1]  
print(maxProfit(prices3))

6  
4  
0