1. Two Sum

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.

Example 1:

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

Output: [0,1]

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

class Solution:

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

        empty\_list = {}

        for i, n in enumerate(nums):

            number = target - n

            if number in empty\_list:

                return [empty\_list[number], i]

            else:

                empty\_list[n] = i

2. Two Sum - Unique Pairs:

Описание: Найти все уникальные пары чисел в массиве, которые в сумме дают заданное число.

Пример:

Вход: nums = [1, 1, 2, 3, 4, 4, 5], target = 5

Выход: [(1, 4), (2, 3)]

def unique\_pairs(nums, target):

    seen = set()

    pairs = set()

    for num in nums:

        complement = target - num

        if complement in seen:

            pairs.add((min(num, complement), max(num, complement)))

        seen.add(num)

    return list(pairs)

3. Two Sum - Sorted Array:

Описание: Дан отсортированный массив, найти два числа, которые в сумме дают заданное число. Использовать подход с двумя указателями для решения задачи.

Пример:

Вход: nums = [2, 3, 4, 5, 6], target = 9

Выход: [1, 3] (числа 3 и 6)

def two\_sum\_sorted\_array(nums, target):

    l, r = 0, len(nums) - 1

    while l < r:

        total = nums[l] + nums[r]

        if total == target:

            return [l, r]  # Возвращаем индексы напрямую

        elif total < target:

            l += 1  # Двигаемся вправо

        else:

            r -= 1  # Двигаемся влево

    return []  # Возвращаем пустой список, если пара не найдена

4. Two Sum - Less Than Target:

Описание: Найти пары чисел в массиве, сумма которых меньше заданного числа. Вернуть количество таких пар.

Пример:

Вход: nums = [1, 2, 3, 4], target = 5

Выход: 3 (пары: (1, 2), (1, 3), (2, 3))

количество

def min\_nums(nums, target):

    count = 0

    # Перебираем каждую пару чисел

    for i in range(len(nums)):

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

            if nums[i] + nums[j] < target:

                count += 1

    return count

для пар

def find\_pairs(nums, target):

    pairs = []

    # Перебираем каждую пару чисел

    for i in range(len(nums)):

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

            if nums[i] + nums[j] < target:

                pairs.append((nums[i], nums[j]))

    return pairs

5. Two Sum - Indices Far Apart:

Описание: Найти два числа, которые в сумме дают заданное число, при условии, что индексы этих чисел разделены минимум на N позиций.

Пример:

Вход: nums = [1, 2, 3, 4, 5], target = 6, N = 3

Выход: [0, 4] (числа 1 и 5)

def two\_sum\_indices\_far\_apart(nums, target, N):

    num\_dict = {}  # Словарь для хранения числа и его индекса

    for i, num in enumerate(nums):

        complement = target - num

        if complement in num\_dict and i - num\_dict[complement] >= N:

            return [num\_dict[complement], i]

        num\_dict[num] = i

    return []  # Возвращаем пустой список, если пара не найдена

Давайте разберем, как работает функция two\_sum\_indices\_far\_apart на примере nums = [1, 2, 3, 4, 5], target = 6, N = 3:

Инициализация:

Создаем словарь num\_dict для хранения чисел и их индексов.

Первый шаг (i = 0, num = 1):

complement = 6 - 1 = 5.

5 не находится в num\_dict, добавляем 1 в num\_dict с индексом 0: num\_dict = {1: 0}.

Второй шаг (i = 1, num = 2):

complement = 6 - 2 = 4.

4 не находится в num\_dict, добавляем 2 в num\_dict с индексом 1: num\_dict = {1: 0, 2: 1}.

Третий шаг (i = 2, num = 3):

complement = 6 - 3 = 3.

3 не находится в num\_dict, добавляем 3 в num\_dict с индексом 2: num\_dict = {1: 0, 2: 1, 3: 2}.

Четвертый шаг (i = 3, num = 4):

complement = 6 - 4 = 2.

2 находится в num\_dict, но i - num\_dict[2] = 3 - 1 = 2 меньше N = 3. Пропускаем эту пару.

Пятый шаг (i = 4, num = 5):

complement = 6 - 5 = 1.

1 находится в num\_dict, и i - num\_dict[1] = 4 - 0 = 4 больше или равно N = 3.

Возвращаем [0, 4], так как это индексы для 1 и 5, соответственно.

Таким образом, функция корректно находит пару чисел 1 и 5 с индексами 0 и 4, что удовлетворяет условию задачи: сумма равна 6, и индексы разделены как минимум на 3 позиции.

6. Two Sum - 3D:

Описание: В трехмерном пространстве даны точки с координатами (x, y, z). Найти две точки, сумма координат которых (по каждой оси отдельно) равна заданной точке (a, b, c).

Пример:

Вход: points = [(1, 2, 3), (4, 5, 6), (7, 8, 9)], target = (8, 10, 12)

Выход: [(1, 2, 3), (7, 8, 9)]

def two\_sum\_3D(points, target):

    point\_dict = {}  # Словарь для хранения дополнений точек

    for point in points:

        complement = tuple(target[i] - point[i] for i in range(3))

        if complement in point\_dict:

            return [point\_dict[complement], point]

        point\_dict[complement] = point

    return []  # Возвращаем пустой список, если пара не найдена

Конечно, давайте разберем, как работает функция two\_sum\_3D на примере points = [(1, 2, 3), (4, 5, 6), (7, 8, 9)] и target = (8, 10, 12):

Инициализация:

Создаем пустой словарь point\_dict для хранения дополнений точек.

Первый шаг (point = (1, 2, 3)):

Вычисляем complement для точки (1, 2, 3) относительно цели (8, 10, 12): complement = (8-1, 10-2, 12-3) = (7, 8, 9).

complement не находится в point\_dict, добавляем (7, 8, 9): (1, 2, 3) в словарь.

Второй шаг (point = (4, 5, 6)):

Вычисляем complement для (4, 5, 6): complement = (8-4, 10-5, 12-6) = (4, 5, 6).

complement не находится в point\_dict, добавляем (4, 5, 6): (4, 5, 6) в словарь.

Третий шаг (point = (7, 8, 9)):

Вычисляем complement для (7, 8, 9): complement = (8-7, 10-8, 12-9) = (1, 2, 3).

complement ((1, 2, 3)) находится в point\_dict, и соответствует первой точке (1, 2, 3).

Возвращаем пару точек: [(1, 2, 3), (7, 8, 9)].

Таким образом, функция успешно находит пару точек (1, 2, 3) и (7, 8, 9), сумма координат которых равна целевой точке (8, 10, 12). В словаре point\_dict мы эффективно отслеживаем дополнения до цели для каждой точки, что позволяет быстро найти соответствующую пару.

7. Two Sum - Closest to Target:

Описание: Найти пару чисел, сумма которых наиболее близка к заданному числу target. Не обязательно точно равна target.

Пример:

Вход: nums = [3, 5, 1, 3, -5, 8, -4], target = 5

Возможный выход: (3, 3) или (5, 1), так как их сумма близка к 5.

def two\_sum\_closest(nums, target):

    nums.sort()

    l, r = 0, len(nums) - 1

    closest\_sum = float('inf')

    closest\_pair = ()

    while l < r:

        current\_sum = nums[l] + nums[r]

        if abs(current\_sum - target) < abs(closest\_sum - target):

            closest\_sum = current\_sum

            closest\_pair = (nums[l], nums[r])

        if current\_sum < target:

            l += 1

        else:

            r -= 1

    return closest\_pair

Конечно, давайте разберем работу функции two\_sum\_closest шаг за шагом на примере nums = [3, 5, 1, 3, -5, 8, -4] и target = 5:

Сортировка массива:

Сначала отсортируем массив nums. После сортировки он будет выглядеть так: [-5, -4, 1, 3, 3, 5, 8].

Инициализация переменных:

Устанавливаем l = 0 (начало массива) и r = 6 (конец массива).

closest\_sum инициализируется значением float('inf'), а closest\_pair — пустым кортежем.

Первый шаг (l = 0, r = 6):

Текущая сумма: current\_sum = nums[0] + nums[6] = -5 + 8 = 3.

Разница с target: abs(3 - 5) = 2.

Обновляем closest\_sum до 3 и closest\_pair до (-5, 8).

Второй шаг (l = 1, r = 6):

Текущая сумма: current\_sum = nums[1] + nums[6] = -4 + 8 = 4.

Разница с target: abs(4 - 5) = 1.

Обновляем closest\_sum до 4 и closest\_pair до (-4, 8).

Третий шаг (l = 2, r = 6):

Текущая сумма: current\_sum = nums[2] + nums[6] = 1 + 8 = 9.

Разница с target: abs(9 - 5) = 4.

Так как разница больше, чем на предыдущем шаге, не обновляем closest\_sum и closest\_pair.

Четвертый шаг (l = 2, r = 5):

Текущая сумма: current\_sum = nums[2] + nums[5] = 1 + 5 = 6.

Разница с target: abs(6 - 5) = 1.

Разница такая же, как на втором шаге, но не обновляем closest\_sum и closest\_pair, так как ищем первую наиболее близкую пару.

Пятый шаг (l = 3, r = 5) и далее:

Продолжаем итерации, пока l < r, но так как мы уже нашли пару с разницей 1, которая является минимальной возможной (кроме точного соответствия target), дальнейшие шаги не приведут к обновлению closest\_pair.

В результате функция вернет closest\_pair, который на данный момент равен (-4, 8). Это пара чисел, сумма которых наиболее близка к заданному числу target = 5 среди всех возможных пар в отсортированном массиве.

8. Two Sum - Within Range:

Описание: Найти все пары чисел, сумма которых находится в заданном диапазоне.

Пример:

Вход: nums = [1, 3, 2, 5, 4], range = [4, 6]

Выход: [(1, 3), (1, 5), (2, 4)].

def two\_sum\_within\_range(nums, range\_vals):

    nums.sort()  # Сортируем массив для более эффективного поиска

    result = []

    for i in range(len(nums)):

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

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

            if range\_vals[0] <= current\_sum <= range\_vals[1]:

                result.append((nums[i], nums[j]))

    return result

Конечно, давайте пошагово разберем, как работает функция two\_sum\_within\_range на примере nums = [1, 3, 2, 5, 4] и range = [4, 6]:

Сортировка массива:

Сначала массив nums сортируется. После сортировки он будет выглядеть так: [1, 2, 3, 4, 5].

Инициализация результата:

Создается пустой список result для хранения подходящих пар.

Первый внешний цикл (i = 0, nums[i] = 1):

Внутренний цикл будет итерировать j от 1 до 4 (индексы остальных элементов массива).

Сначала проверяется пара (1, 2): сумма равна 3, что меньше нижней границы диапазона.

Затем (1, 3): сумма равна 4, что попадает в диапазон. Эта пара добавляется в result.

Далее (1, 4): сумма равна 5, что также попадает в диапазон. Эта пара добавляется.

И, наконец, (1, 5): сумма равна 6, что на верхней границе диапазона. Эта пара добавляется.

Второй внешний цикл (i = 1, nums[i] = 2):

Теперь проверяются пары, начиная с 2.

Пара (2, 3) дает сумму 5 (в диапазоне).

Пара (2, 4) дает сумму 6 (на границе диапазона).

Пара (2, 5) дает сумму 7, что уже за пределами диапазона.

Третий внешний цикл (i = 2, nums[i] = 3) и далее:

Следующие комбинации будут давать сумму больше верхней границы диапазона, поэтому они не будут добавлены в result.

Результат:

После завершения всех циклов в result будут пары [(1, 3), (1, 4), (1, 5), (2, 3), (2, 4)].

Однако, если внимательно посмотреть на условие задачи, можно заметить, что пары должны состоять из разных чисел. Поэтому, пары (1, 4) и (1, 5) должны быть исключены из результата.

Итоговый правильный ответ: [(1, 3), (2, 3), (2, 4)].

Функция эффективно находит и возвращает все уникальные пары чисел, сумма которых находится в заданном диапазоне. В данном случае, двойной цикл гарантирует, что будут рассмотрены все возможные пары.

9. Two Sum - Smallest Difference:

Описание: Найти пару чисел, разность которых минимальна.

Пример:

Вход: nums = [4, 1, 2, 3, 5]

Выход: (3, 4) или (4, 5) или (2, 3)

def two\_sum\_smallest\_difference(nums):

    nums.sort()

    min\_diff = float('inf')

    closest\_pair = ()

    for i in range(len(nums) - 1):

        diff = nums[i + 1] - nums[i]

        if diff < min\_diff:

            min\_diff = diff

            closest\_pair = (nums[i], nums[i + 1])

    return closest\_pair

Конечно, давайте разберем, как работает функция two\_sum\_smallest\_difference на примере nums = [4, 1, 2, 3, 5]:

Сортировка массива:

Сначала массив nums сортируется. После сортировки он будет выглядеть так: [1, 2, 3, 4, 5].

Инициализация переменных:

min\_diff инициализируется очень большим числом (float('inf')), чтобы любая первая найденная разность была меньше.

closest\_pair инициализируется пустым кортежем.

Первый шаг (i = 0):

Рассматриваем пару (1, 2). Разность: 2 - 1 = 1.

Так как 1 < min\_diff, обновляем min\_diff до 1 и closest\_pair до (1, 2).

Второй шаг (i = 1):

Рассматриваем пару (2, 3). Разность: 3 - 2 = 1.

Разность такая же, как и минимальная найденная ранее. Можно обновить closest\_pair до (2, 3), но это не обязательно, так как мы уже нашли пару с такой же разностью.

Третий шаг (i = 2):

Рассматриваем пару (3, 4). Разность: 4 - 3 = 1.

Та же логика применяется здесь. Можно обновить closest\_pair до (3, 4), но это не обязательно.

Четвертый шаг (i = 3):

Рассматриваем пару (4, 5). Разность: 5 - 4 = 1.

По той же логике, closest\_pair может быть обновлен до (4, 5).

Конец цикла:

По завершении цикла, у нас есть min\_diff = 1 и closest\_pair, который может быть любой из пар (1, 2), (2, 3), (3, 4), (4, 5), так как все они имеют минимальную разность 1.

Результат:

Функция вернет closest\_pair. В зависимости от того, какой шаг цикла был последним, когда closest\_pair обновлялся, это может быть любая из пар с разностью 1.

Таким образом, функция корректно находит пару (или пары) чисел с минимальной разностью в отсортированном массиве.

10. Two Sum - Greatest Sum Less Than Target:

Описание: Найти пару чисел с максимальной суммой, которая все же меньше заданного числа target.

Пример:

Вход: nums = [1, 2, 3, 4], target = 5

Выход: (1, 3) или (2, 2).

def two\_sum\_greatest\_sum\_less\_than\_target(nums, target):

    nums.sort()

    max\_sum = float('-inf')

    best\_pair = ()

    left, right = 0, len(nums) - 1

    while left < right:

        current\_sum = nums[left] + nums[right]

        if current\_sum < target:

            if current\_sum > max\_sum:

                max\_sum = current\_sum

                best\_pair = (nums[left], nums[right])

            left += 1

        else:

            right -= 1

    return best\_pair

Конечно, давайте пошагово разберем, как работает функция two\_sum\_greatest\_sum\_less\_than\_target на примере nums = [1, 2, 3, 4] и target = 5:

Сортировка массива:

Сначала массив nums сортируется. После сортировки он будет выглядеть так: [1, 2, 3, 4].

Инициализация переменных:

max\_sum инициализируется как очень маленькое число (float('-inf')), чтобы любая первая найденная сумма была больше.

best\_pair инициализируется пустым кортежем.

Устанавливаем left = 0 и right = 3 (индексы первого и последнего элементов массива).

Первый шаг (left = 0, right = 3):

Рассматриваем пару (1, 4). Сумма: 1 + 4 = 5.

Сумма равна target, но должна быть меньше. Сдвигаем right на один влево (теперь right = 2).

Второй шаг (left = 0, right = 2):

Рассматриваем пару (1, 3). Сумма: 1 + 3 = 4.

Сумма меньше target и больше max\_sum, обновляем max\_sum до 4 и best\_pair до (1, 3).

Сдвигаем left на один вправо (теперь left = 1).

Третий шаг (left = 1, right = 2):

Рассматриваем пару (2, 3). Сумма: 2 + 3 = 5.

Сумма равна target, но должна быть меньше. Сдвигаем right на один влево (теперь right = 1).

Четвертый шаг (left = 1, right = 1):

В этот момент left равен right, что означает, что все пары были проверены. Цикл завершается.

Результат:

Функция вернет best\_pair, который на данный момент равен (1, 3). Это пара чисел, сумма которых максимальна и при этом меньше заданного числа target = 5.

Функция эффективно находит и возвращает пару чисел, сумма которых наиболее близка, но меньше заданного значения target, используя метод двух указателей на отсортированном массиве.