454. 四数相加 II

问题描述

给定四个整数数组 `nums1`、`nums2`、`nums3` 和 `nums4`，每个数组的长度都是 `n`。请计算有多少个元组 `(i, j, k, l)` 满足以下条件：

1. (0 \leq i, j, k, l < n)
2. (nums1[i] + nums2[j] + nums3[k] + nums4[l] == 0)

示例

示例 1

输入：

```
nums1 = [1, 2]
nums2 = [-2, -1]
nums3 = [-1, 2]
nums4 = [0, 2]
```

输出：

```
2
```

解释： 两个满足条件的元组为：

1. ((0, 0, 0, 1)) -> (nums1[0] + nums2[0] + nums3[0] + nums4[1] = 1 + (-2) + (-1) + 2 = 0)
2. ((1, 1, 0, 0)) -> (nums1[1] + nums2[1] + nums3[0] + nums4[0] = 2 + (-1) + (-1) + 0 = 0)

示例 2

输入：

```
nums1 = [0]
nums2 = [0]
nums3 = [0]
nums4 = [0]
```

输出：

```
1
```

In [ ]:
```python
# 尝试1:

from collections import defaultdict

class Solution:
    def fourSumCount(self, nums1: List[int], nums2: List[int], nums3: List[int], nums4: List[int]) -> int:
        pair1 = defaultdict(int)
        pair2 = defaultdict(int)

        # divide and counqur

        for i, num1 in enumerate(nums1):
            for j, num2 in enumerate(nums2):
                key = num1 + num2 # this is actually the sum of num1 and num2, but as key
                pair1[key] += 1

        for num3 in nums3:
            for num4 in nums4:
                key = num3 + num4
                pair2[key] += 1

    # 在 Python 中，直接访问不存在的键（例如 pair2[-key]）会抛出 KeyError，即使使用 defaultdict。
    # 更安全的方式是用 get 方法：pair2.get(-key, 0)，如果键不存在，返回默认值 0。

        count = 0
        for key, _ in pair1.items(): # 注意dict访问用这个方法
            # if pair2.get(-key,0) != 0:
            if -key in pair2:
                count += pair1[key] * pair2[-key]

        return count
```

In [ ]:
```python
# 尝试2: 优化，减少第三个循环

from collections import defaultdict

class Solution:
    def fourSumCount(self, nums1: List[int], nums2: List[int], nums3: List[int], nums4: List[int]) -> int:
        pair1 = defaultdict(int)

        # divide and counqur

        for num1 in nums1:
```

```
            for num2 in nums2:
                key = num1 + num2 # this is actually the sum of num1 and num2, but as key
                pair1[key] += 1

        count = 0
        for num3 in nums3:
            for num4 in nums4:
                key = num3 + num4
                if -key in pair1:
                    count += pair1[-key]

        return count

    # 在 Python 中，直接访问不存在的键（例如 pair2[-key]）会抛出 KeyError，即使使用 defaultdict。
    # 更安全的方式是用 get 方法：pair2.get(-key, 0)，如果键不存在，返回默认值 0。

    # 优化点：
    # 之前用的是：
    # if pair1.get(-key,0) != 0
    # if -key in pair1:

    # in的性能远好于get
```

383. 赎金信

问题描述

给你两个字符串：`ransomNote` 和 `magazine`，判断 `ransomNote` 能不能由 `magazine` 里面的字符构成。

如果可以，返回 `true`；否则返回 `false`。

`magazine` 中的每个字符只能在 `ransomNote` 中使用一次。

示例

示例 1

输入：

```
ransomNote = "a"
magazine = "b"
```
输出：

```
false
```

示例 2

输入：

```
ransomNote = "aa"
magazine = "ab"
```
输出：

```
false
```

示例 3

输入：

```
ransomNote = "aa"
magazine = "aab"
```
输出：

```
true
```

提示

1. (1 \leq ransomNote.length, magazine.length \leq 10^5)
2. `ransomNote` 和 `magazine` 由小写英文字母组成。

In [ ]:
```
class Solution:
    def canConstruct(self, ransomNote: str, magazine: str) -> bool:
        magazine_set = set(magazine)
        ransomNote_set = set(ransomNote)
        if magazine_set | ransomNote_set != magazine_set:
            return False
        else:
            return True
```

```
# 理解错题意了，magazine 中的每个字符只能在 ransomNote 中使用一次
```

> > > x – y # Difference 差集

set(['a', 'c', 'e'])

> > > x | y # Union 并集

set(['a', 'c', 'b', 'e', 'd', 'y', 'x', 'z'])

> > > x & y # Intersection 交集

set(['b', 'd'])

> > > x ^ y # Symmetric difference (XOR) 补集

In [ ]:
```python
class Solution:
    def canConstruct(self, ransomNote: str, magazine: str) -> bool:

        ransomNote_dict = {}
        magazine_dict = {}

        for letter in magazine:
            if letter in magazine_dict:
                magazine_dict[letter] += 1
            else:
                magazine_dict[letter] = 1

        for letter in ransomNote:
            if letter not in magazine_dict:
                return False
            else:
                magazine_dict[letter] -= 1
                if magazine_dict[letter] < 0:
                    return False

        return True
```

In [ ]:
```python
# 其他版本:
class Solution:
    def canConstruct(self, ransomNote: str, magazine: str) -> bool:
        counts = {}
        for c in magazine:
            counts[c] = counts.get(c, 0) + 1
        for c in ransomNote:
            if c not in counts or counts[c] == 0:
                return False
            counts[c] -= 1
        return True

#  (版本四) 使用Counter

from collections import Counter

class Solution:
    def canConstruct(self, ransomNote: str, magazine: str) -> bool:
        return not Counter(ransomNote) - Counter(magazine)

#  (版本五) 使用count

class Solution:
    def canConstruct(self, ransomNote: str, magazine: str) -> bool:
        return all(ransomNote.count(c) <= magazine.count(c) for c in set(ransomNote))

# (版本六) 使用count(简单易懂)
class Solution:
    def canConstruct(self, ransomNote: str, magazine: str) -> bool:
        for char in ransomNote:
            if char in magazine and ransomNote.count(char) <= magazine.count(char):
                continue
            else:
                return False
        return True
```

15. 三数之和

https://leetcode.cn/problems/3sum/solutions/284681/san-shu-zhi-he-by-leetcode-solution/

问题描述

给你一个整数数组 `nums` ，判断是否存在三元组 `[nums[i], nums[j], nums[k]]` 满足以下条件：

1. (i \neq j)，(i \neq k)，且 (j \neq k)。
2. (nums[i] + nums[j] + nums[k] == 0)。

请返回所有和为 0 且不重复的三元组。

注意：答案中不可以包含重复的三元组。

示例

示例 1

输入：

```
nums = [-1, 0, 1, 2, -1, -4]
```
输出：

```
[[-1, -1, 2], [-1, 0, 1]]
```
解释：

- (nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0)
- (nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0)
- (nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0)

不同的三元组是 `[-1, 0, 1]` 和 `[-1, -1, 2]`。

示例 2

输入：

```
nums = [0, 1, 1]
```
输出：

```
[]
```
解释： 唯一可能的三元组和不为 0。

示例 3

输入：

```
nums = [0, 0, 0]
```
输出：

```
[[0, 0, 0]]
```
解释： 唯一可能的三元组和为 0。

提示

- (3 \leq nums.length \leq 3000)
- (-10^5 \leq nums[i] \leq 10^5)

第一次尝试， 两个问题：

- 索引匹配问题：

在第一部分构造 record2 时，你将 nums[slicer:] 的枚举结果赋值给 j，但这里的 j 是相对于 nums[slicer:] 的索引，而不是 nums 的全局索引。因此，i != j 的判断是错误的。

- 重复三元组的问题：

即使使用 set 存储结果，当前代码依然可能出现重复三元组的情况，因为三元组 (nums[i], nums[j], nums[k]) 是直接用数值进行存储，缺乏排序来确保唯一性。

In [3]:
```python
from typing import List
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        # key is (i,j,k)
        # i != j, i != k, j != k

        record2 = dict()
        result = set()

        for i, num1 in enumerate(nums):
            slicer = i + 1
            for j, num2 in enumerate(nums[slicer:]):
                if i != j:
                    if num1+num2 in record2:
                        record2[num1+num2].append([i,j])
                    else:
                        record2[num1+num2] = [[i,j]]

        for k, num3 in enumerate(nums):
            if -num3 in record2:
```

```
                for pair in record2[-num3]:
                    i, j  = pair
                    if i != k and j != k:
                        result.add((nums[i],nums[j],nums[k]))

        return list(result)
```

In [4]:
```
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        # key is (i,j,k)
        # i != j, i != k, j != k

        record = dict()
        result = set()

        for i, num1 in enumerate(nums):
            slicer = i + 1
            for j, num2 in enumerate(nums[slicer:], start = slicer): # 修正1: j从slicer开始
                if i != j:
                    if num1+num2 in record:
                        record[num1+num2].append([i,j])
                    else:
                        record[num1+num2] = [[i,j]]

        for k, num3 in enumerate(nums):
            if -num3 in record:
                for pair in record[-num3]:
                    i, j  = pair
                    if i != k and j != k:
                        triplet = tuple(sorted((nums[i], nums[j], nums[k])))  # 修正2: 排序tuple, 确保唯一性
                        result.add(triplet)

        return list(result)

### 结果正确, 但超出时间限制
```

In [ ]:
```
# 答案: 双指针法

class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        result = []
        nums.sort()

        for i in range(len(nums)):
            # 如果第一个元素已经大于0, 不需要进一步检查
            if nums[i] > 0:
                return result

            # 跳过相同的元素以避免重复
            if i > 0 and nums[i] == nums[i - 1]:
                continue

            left = i + 1
            right = len(nums) - 1

            while right > left:
                sum_ = nums[i] + nums[left] + nums[right]

                if sum_ < 0:
                    left += 1
                elif sum_ > 0:
                    right -= 1
                else:
                    result.append([nums[i], nums[left], nums[right]])

                    # 跳过相同的元素以避免重复
                    while right > left and nums[right] == nums[right - 1]:
                        right -= 1
                    while right > left and nums[left] == nums[left + 1]:
                        left += 1

                    right -= 1
                    left += 1

        return result
```

In [ ]:
```
# 答案: 字典法
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        result = []
        nums.sort()
        # 找出a + b + c = 0
        # a = nums[i], b = nums[j], c = -(a + b)
        for i in range(len(nums)):
            # 排序之后如果第一个元素已经大于零, 那么不可能凑成三元组
            if nums[i] > 0:
                break
            if i > 0 and nums[i] == nums[i - 1]: #三元组元素a去重
                continue
            d = {}
            # d 是用来记录b 也就是num[j] 的, 如果新的b在d中, 就跳过
            for j in range(i + 1, len(nums)):
                if j > i + 2 and nums[j] == nums[j-1] == nums[j-2]: # 三元组元素b去重
```

```
                continue
            c = 0 - (nums[i] + nums[j])
            if c in d:
                result.append([nums[i], nums[j], c])
                d.pop(c) # 三元组元素c去重
            else:
                d[nums[j]] = j
        return result
```

18. 四数之和 （难，回来复习）

问题描述

给你一个由 n 个整数组成的数组 `nums` ，和一个目标值 `target` 。请你找出并返回满足下述全部条件且不重复的四元组 ([nums[a], nums[b], nums[c], nums[d]]) （若两个四元组元素一一对应，则认为两个四元组重复）：

- (0 \leq a, b, c, d < n)
- (a, b, c) 和 (d) 互不相同
- (nums[a] + nums[b] + nums[c] + nums[d] = target)

你可以按**任意顺序**返回答案。

示例

示例 1

输入：

```
nums = [1, 0, -1, 0, -2, 2]
target = 0
```
输出：

```
[[-2, -1, 1, 2], [-2, 0, 0, 2], [-1, 0, 0, 1]]
```

示例 2

输入：

```
nums = [2, 2, 2, 2, 2]
target = 8
```
输出：

```
[[2, 2, 2, 2]]
```

提示

- (4 \leq n \leq 200)
- (-10^9 \leq nums[i] \leq 10^9)
- (-10^9 \leq target \leq 10^9)

In [ ]:
```
# 用字典，遍历三个数，然后对最后一个数用dict

class Solution(object):
    def fourSum(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: List[List[int]]
        """
        # 创建一个字典来存储输入列表中每个数字的频率
        freq = {}
        for num in nums:
            freq[num] = freq.get(num, 0) + 1

        # 创建一个集合来存储最终答案，并遍历4个数字的所有唯一组合
        ans = set()
        for i in range(len(nums)):
            for j in range(i + 1, len(nums)):
                for k in range(j + 1, len(nums)):
                    val = target - (nums[i] + nums[j] + nums[k])
                    if val in freq:
                        # 确保没有重复
                        count = (nums[i] == val) + (nums[j] == val) + (nums[k] == val)
                        if freq[val] > count:
                            ans.add(tuple(sorted([nums[i], nums[j], nums[k], val])))

        return [list(x) for x in ans]

# 可以过测试，但是时间比较慢
```

In [ ]:

```python
# 双指针法

class Solution:
    def fourSum(self, nums: List[int], target: int) -> List[List[int]]:
        nums.sort()
        n = len(nums)
        result = []
        for i in range(n):
            if nums[i] > target and nums[i] > 0 and target > 0:# 剪枝（可省）
                break
            if i > 0 and nums[i] == nums[i-1]:# 去重
                continue
            for j in range(i+1, n):
                if nums[i] + nums[j] > target and target > 0: #剪枝（可省）
                    break
                if j > i+1 and nums[j] == nums[j-1]: # 去重
                    continue
                left, right = j+1, n-1
                while left < right:
                    s = nums[i] + nums[j] + nums[left] + nums[right]
                    if s == target:
                        result.append([nums[i], nums[j], nums[left], nums[right]])
                        while left < right and nums[left] == nums[left+1]:
                            left += 1
                        while left < right and nums[right] == nums[right-1]:
                            right -= 1
                        left += 1
                        right -= 1
                    elif s < target:
                        left += 1
                    else:
                        right -= 1
        return result
```

In [ ]: