¾ Sum

15. 3Sum

Medium 11245 1120 Add to List Share

Given an integer array nums, return all the triplets `[nums[i], nums[j], nums[k]]` such that `i != j`, `i != k`, and `j != k`, and `nums[i] + nums[j] + nums[k] == 0`.

Notice that the solution set must not contain duplicate triplets.

Example 1:

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

Example 2:

```
Input: nums = []
Output: []
```

Example 3:

```
Input: nums = [0]
Output: []
```

```python
class Solution:
    def threeSum(self, nums: List[int]) -> List[List[int]]:
        def twoSum(left, target):
            right = len(nums)-1
            while left < right:
                if nums[left] + nums[right] == -target:
                    rslt.append([target, nums[left], nums[right]])
                    while left < right and nums[left+1] == nums[left]:
                        left += 1
                    while left < right and nums[right-1] == nums[right]:
                        right -= 1
                    left += 1
                    right -= 1
                elif nums[left] + nums[right] < - target:
                    left += 1
                else:
                    right -= 1
            return
        nums.sort()
        if not nums or nums[-1] < 0 or nums[0] > 0:
            return []
        rslt = []
        for i, num in enumerate(nums):
            if num > 0: return rslt
            if i > 0 and num == nums[i-1]: continue
            twoSum(i+1, num)
        return rslt
```

16. 3Sum Closest

Medium 3503 182 Add to List Share

Given an array `nums` of *n* integers and an integer `target`, find three integers in `nums` such that the sum is closest to `target`. Return the sum of the three integers. You may assume that each input would have exactly one solution.

Example 1:

```
Input: nums = [-1,2,1,-4], target = 1
Output: 2
Explanation: The sum that is closest to the target is 2. (-1 + 2 + 1 = 2).
```

Constraints:

- `3 <= nums.length <= 10^3`
- `-10^3 <= nums[i] <= 10^3`
- `-10^4 <= target <= 10^4`

```python
class Solution:
    def threeSumClosest(self, nums: List[int], target: int) -> int:
        def twoSumClosest(left, curr):
            right = len(nums)-1
            while left < right:
                temp = curr + nums[left] + nums[right]
                if abs(target-temp) < abs(target-self.rslt):
                    self.rslt = temp
                if temp < target:
                    left += 1
                else:
                    right -= 1
            return
        nums.sort()
        self.rslt = float("inf")
        for i, num in enumerate(nums):
            twoSumClosest(i+1, num)
            if self.rslt == target:
                return target
        return self.rslt
```

259. 3Sum Smaller

Medium 902 95 Add to List Share

Given an array of `n` integers `nums` and an integer `target`, find the number of index triplets `i`, `j`, `k` with `0 <= i < j < k < n` that satisfy the condition `nums[i] + nums[j] + nums[k] < target`.

Example 1:

```
Input: nums = [-2,0,1,3], target = 2
Output: 2
Explanation: Because there are two triplets which sums are less than 2:
[-2,0,1]
[-2,0,3]
```

Example 2:

```
Input: nums = [], target = 0
Output: 0
```

Example 3:

```
Input: nums = [0], target = 0
Output: 0
```

```python
class Solution:
    def threeSumSmaller(self, nums: List[int], target: int) -> int:
        nums.sort()
        rslt = 0
        for i, num in enumerate(nums[:-2]):
            if num*3 >= target:
                return rslt
            left, right = i+1, len(nums)-1
            while left < right:
                if nums[left] + nums[right] + num < target:
                    rslt += right - left
                    left += 1
                else:
                    right -= 1
        return rslt
```

923. 3Sum With Multiplicity

Medium 728 127 Add to List Share

Given an integer array `arr`, and an integer `target`, return the number of tuples `i, j, k` such that `i < j < k` and `arr[i] + arr[j] + arr[k] == target`.

As the answer can be very large, return it **modulo** $10^9 + 7$.

Example 1:

```
Input: arr = [1,1,2,2,3,3,4,4,5,5], target = 8
Output: 20
Explanation:
Enumerating by the values (arr[i], arr[j], arr[k]):
(1, 2, 5) occurs 8 times;
(1, 3, 4) occurs 8 times;
(2, 2, 4) occurs 2 times;
(2, 3, 3) occurs 2 times.
```

Example 2:

```
Input: arr = [1,1,2,2,2,2], target = 5
Output: 12
Explanation:
arr[i] = 1, arr[j] = arr[k] = 2 occurs 12 times:
We choose one 1 from [1,1] in 2 ways,
and two 2s from [2,2,2,2] in 6 ways.
```

```python
class Solution(object):
    def threeSumMulti(self, A, target):
        MOD = 10**9 + 7
        count = collections.Counter(A)
        keys = sorted(count)
        self.ans = 0
        def twoSumMulti(left, x):
            right = len(keys)-1
            while left <= right:
                y, z = keys[left], keys[right]
                if y + z + x < target:
                    left += 1
                elif y + z + x > target:
                    right -= 1
                else:
                    if i < left < right:
                        self.ans += count[x] * count[y] * count[z]
                    elif i == left < right:
                        self.ans += count[x] * (count[x] - 1) // 2 * count[z]
                    elif i < left == right:
                        self.ans += count[x] * count[y] * (count[y] - 1) // 2
                    else:
                        self.ans += count[x] * (count[x] - 1) * (count[x] - 2) // 6
                    left += 1
                    right -= 1

        for i, x in enumerate(keys):
            twoSumMulti(i, x)
        return self.ans % MOD
```

18. 4Sum

Medium 3563 434 Add to List Share

Given an array `nums` of `n` integers, return *an array of all the* ***unique*** *quadruplets* `[nums[a], nums[b], nums[c], nums[d]]` such that:

- `0 <= a, b, c, d < n`
- `a`, `b`, `c`, and `d` are **distinct**.
- `nums[a] + nums[b] + nums[c] + nums[d] == target`

You may return the answer in **any order**.

Example 1:

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

Example 2:

```
Input: nums = [2,2,2,2,2], target = 8
Output: [[2,2,2,2]]
```

```python
class Solution:
    def fourSum(self, nums: List[int], target: int) -> List[List[int]]:
        nums.sort()
        return self.kSum(nums, 4, target)

    def kSum(self, nums, k, target):
        if not nums or nums[0]*k > target or nums[-1]*k < target:
            return []
        if k == 2:
            return self.twoSum(nums, target)
        rslt = []
        for i in range(len(nums)):
            if i == 0 or nums[i-1] != nums[i]:
                for curr in self.kSum(nums[i+1:], k-1, target-nums[i]):
                    rslt.append([nums[i]] + curr)
        return rslt


    def twoSum(self, nums, target):
        l, r = 0, len(nums)-1
        rslt = []
        while l < r:
            if nums[l] + nums[r] < target or (l > 0 and nums[l] == nums[l-1]):
                l += 1
            elif nums[l] + nums[r] > target or (r < len(nums)-1 and nums[r] ==
nums[r+1]):
                r -= 1
            else:
                rslt.append([nums[l], nums[r]])
                l += 1
                r -= 1
        return rslt
```