Sum of Four Numbers

Contents

1. Design
2. Result

1. Design

1.fourSum_enumeration() —— enumeration sum function

The function uses the set() data type to store the four-tuple that meets the question conditions.

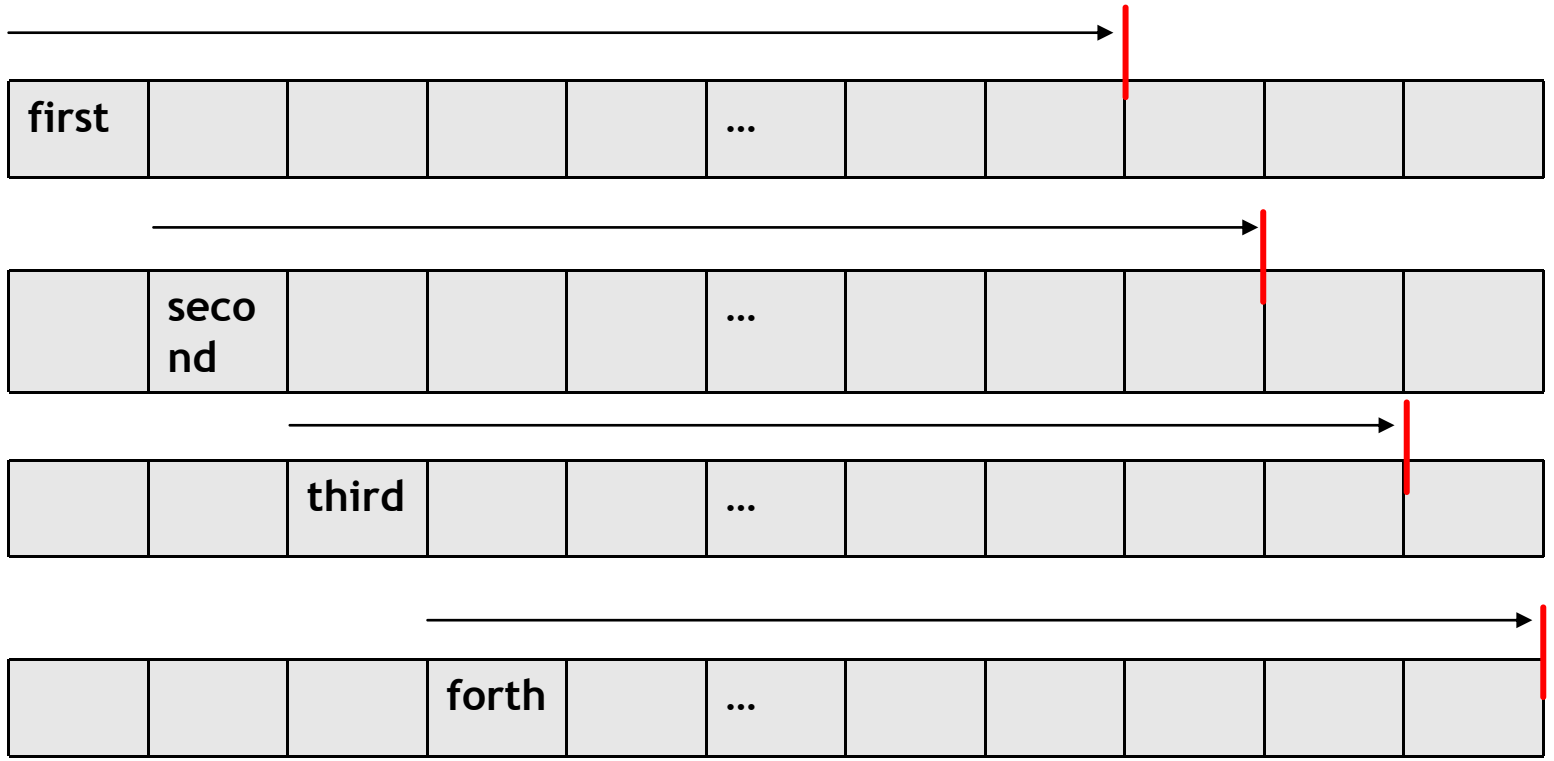
The enumeration method traverses the array, so four for loops are used. Each loop determines one element and accumulates it one by one.

Sum up and store those that meet the conditions into the set.

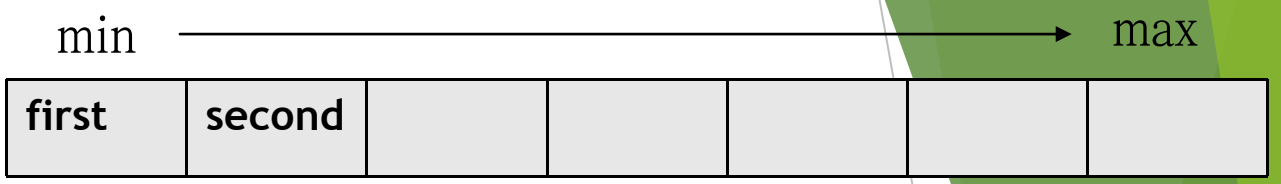
```
n = len(nums)
for first in range(0, n-3):
    for second in range(first+1, n-2):
        for third in range(second+1, n-1):
            for forth in range(third+1, n):
                if nums[first] + nums[second] + nums[third] + nums[forth] == target:
                    res.add((nums[first], nums[second], nums[third], nums[forth]))
```

1. Design

1.fourSum_enumeration() —— enumeration sum function



1. Design



2.fourSum_pointer() —— Double pointer sum function

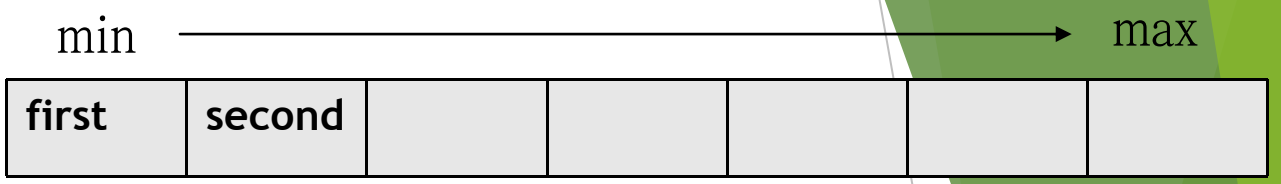
(1) This function first uses the sort() function to rearrange the array into an ascending array.

(2) Enter the first level of for loop and take out the numbers one by one as the first element of the quadruple. An if condition is used here to determine pruning.

(3) Enter the second level for loop and take the number as the second element of the quadruple. The pruning conditions are the same as the previous step.

(4) Use double pointers to get the remaining numbers and sum them up to determine whether the conditions are met. A while loop is used here to remove duplicates. The function first uses the sort() function to rearrange the array into an ascending array.

1. Design



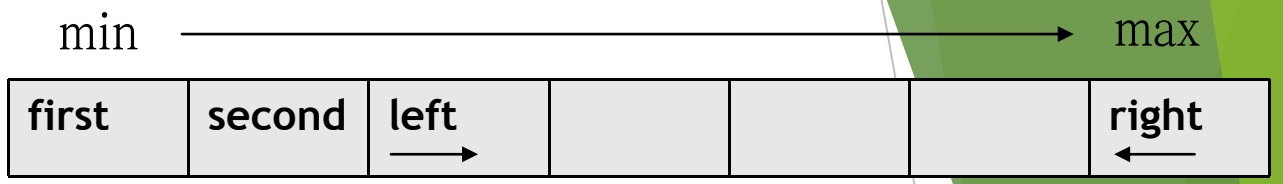
2.fourSum_pointer() —— Double pointer sum function

```
for first in range(0, n - 3):
    if nums[first] + nums[first + 1] + nums[first + 2] + nums[first + 3] > target:
        break
    if nums[first] == nums[first - 1]:
        continue
    if nums[first] + nums[n - 3] + nums[n - 2] + nums[n - 1] < target:
        continue
    for second in range(first + 1, n - 2):
        if nums[first] + nums[second] + nums[second + 1] + nums[second + 2] > target:
            break
        if nums[second] == nums[second - 1]:
            continue
        if nums[first] + nums[second] + nums[n - 2] + nums[n - 1] < target:
            continue
```

exit loop

Enter the next cycle

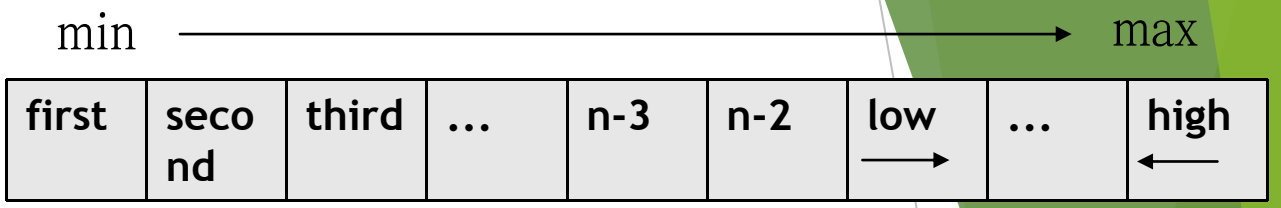
1. Design



2.fourSum_pointer() —— Double pointer sum function

```
left = second + 1
right = n - 1
while left < right:
    sum = nums[first] + nums[second] + nums[left] + nums[right]
    if sum > target:
        right -= 1
    elif sum < target:
        left += 1
    elif sum == target:
        res.append([nums[first], nums[second], nums[left], nums[right]])
        left += 1
        right -= 1
        while left < right and nums[left] == nums[left - 1]:
            left += 1
        while left < right and nums[right] == nums[right + 1]:
            right -= 1
```

1. Design



3.nSum(n, nums, target, pos) —— DFS

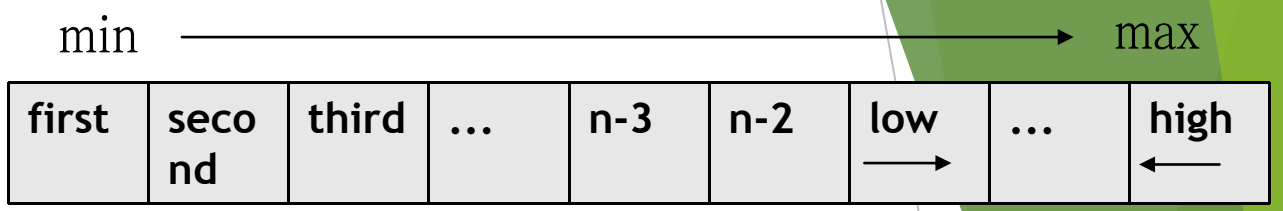
(1) This function first uses the sort() function to rearrange the array into an ascending array.

(2) When it is judged that the number of summed elements is not equal to 2, enter the next recursion, reduce the number of summed elements by one, reduce the target value by the number corresponding to the starting position, and add one to the subscript of the starting element.

(3) The element corresponding to the starting position is added to the list, and the while loop is used to remove duplicates.

(4) When it is judged that the number of summed elements is equal to 2, the next subscript of the starting element is regarded as low, and the subscript of the last digit of the array is regarded as high.

1. Design



3.nSum(n, nums, target, pos) —— DFS

```
i = pos
while i < size:
    subs = nSum(n - 1, nums, target - nums[i], i + 1)
    for elem in subs:
        elem.append(nums[i])
        res.append(elem)
    while i < size - 1 and nums[i] == nums[i + 1]:
        i = i + 1
    i += 1
```

```
low = pos
high = size - 1
while low < high:
    sum = nums[low] + nums[high]
    left = nums[low]
    right = nums[high]
    if sum < target:
        low += 1
    elif sum > target:
        high -= 1
    else:
        res.append([left, right])  # 以list形式加到res中
        while low < high and nums[low] == left:
            low += 1
        while low < high and nums[high] == right:
            high -= 1
```

1. Design

4.fourSum(nums, target) —— Backtracking

(1) Use if-else conditions to determine whether the correct solution is found or perform pruning operations.

(2) The following if and elif conditions determine whether the data is small or large to determine whether to enter the next round of loop or exit the loop.

(3) The else part calls the search() function again. The first call means not to select the number nums[i], and the second call means to select this number. The Notselected[] list is used to record unselected numbers and is also convenient for pruning.

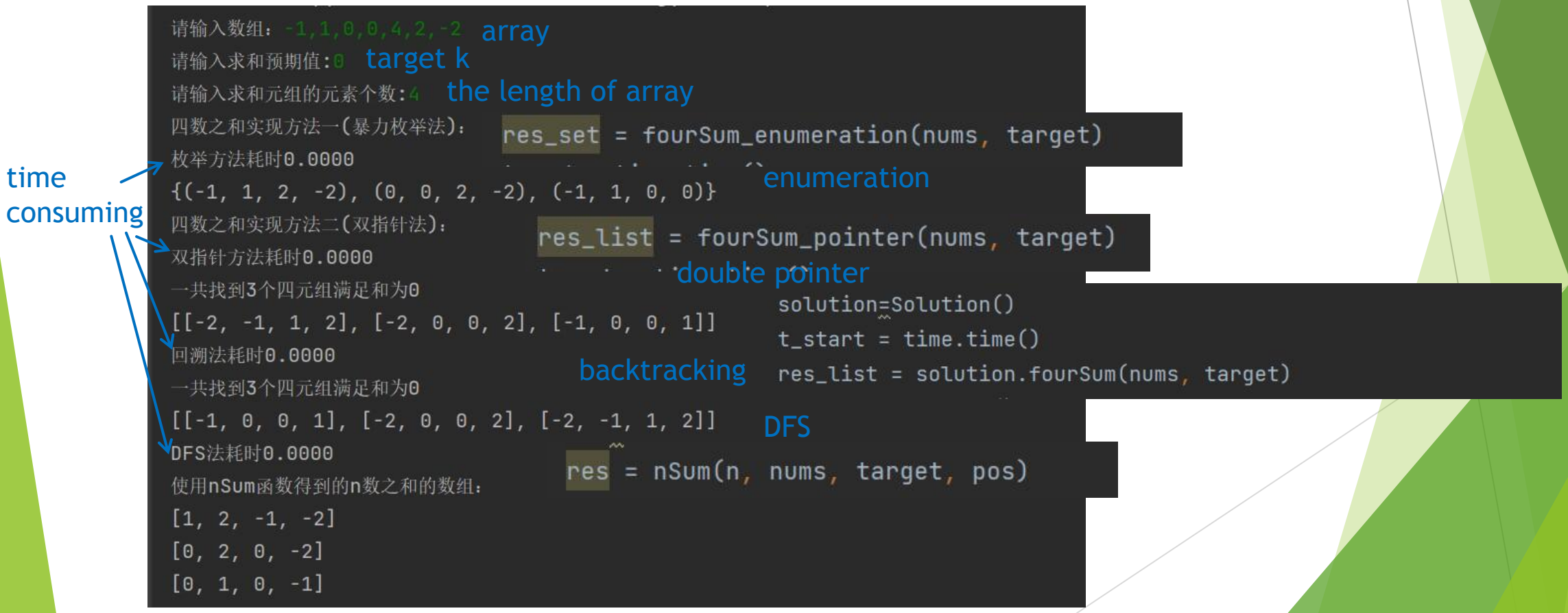
```
def Search(i, target, oneSolution, notSelected):
    if target == 0 and len(oneSolution) == 4:
        output.append(oneSolution)
        return
    elif len(oneSolution) > 4 or i >= len(nums):
        return

    if target - nums[i] - (3 - len(oneSolution)) * nums[-1] > 0 or nums[i] in notSelected:
        Search(i + 1, target, oneSolution, notSelected)
    elif target - (4 - len(oneSolution)) * nums[i] < 0:
        return
    else:
        Search(i + 1, target, oneSolution, notSelected + [nums[i]])
        Search(i + 1, target - nums[i], oneSolution + [nums[i]], notSelected)
```

2.Result

1. Function execution with small amount of data

The user manually enters the array, the target value and the number of summed elements, and puts the parameters into the function to calculate the result.



2.Result

2. Large data volume function operation

Compare the running time of the enumeration method, double pointer method and backtracking method when nums is an array with a large number of elements:

enumeration

double pointer

backtracking

DFS

```
枚举方法耗时38.4011
双指针方法耗时0.0020
回溯法耗时0.0120
DFS方法耗时0.0030
符合条件的元组：5261
```

Tuples that meet the conditions