子数组和为k问题探讨

原创 顾梦雨 LOA算法学习笔记 2021-01-13 16:35

01 问题一

给一个正整数数组和整数k，请查看数组中是否有**两个数字相加为k**，您的算法的时间复杂度应**优于**O(n^2)

1.1 问题分析

考虑给定数组中是否有两个数字之和为k，我们可以暴力解决，两次for循环，但是显然不满足时间复杂度优于O(n^2)。考虑到哈希表的查找效率为O(1)，我们可以利用查找哈希表的时间复杂度为O(1)，把数组的值先存储在一个哈希表里面，然后考察target-nums[i]是否在哈希表中，得出结果。

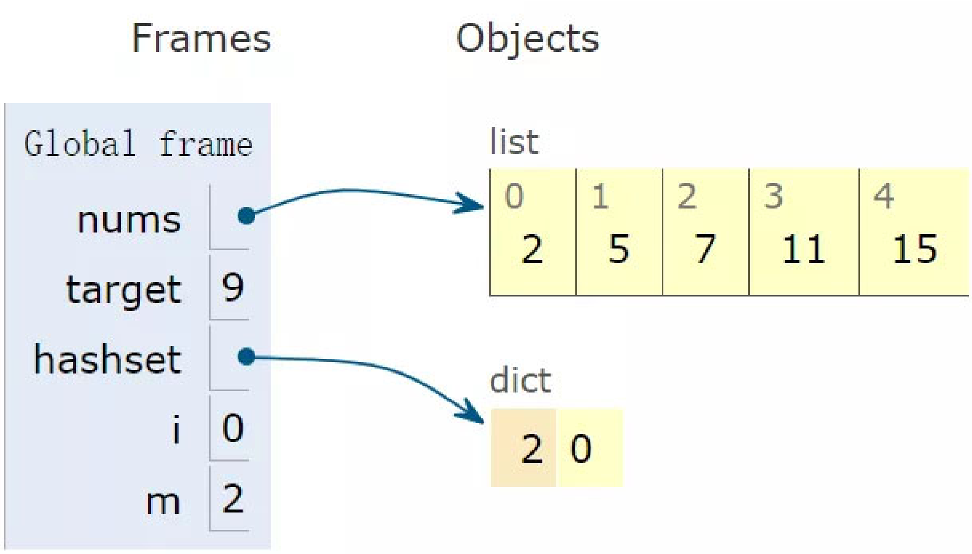
1.2 代码实现

```
if __name__ == '__main__':
    nums=[2, 5, 7, 11, 15]
    target=9
    hashset = {}
    for i, m in enumerate(nums):
        if target - m not in hashset:
            hashset[m] = i
        else:
            print((hashset[target - m] , i ))
```

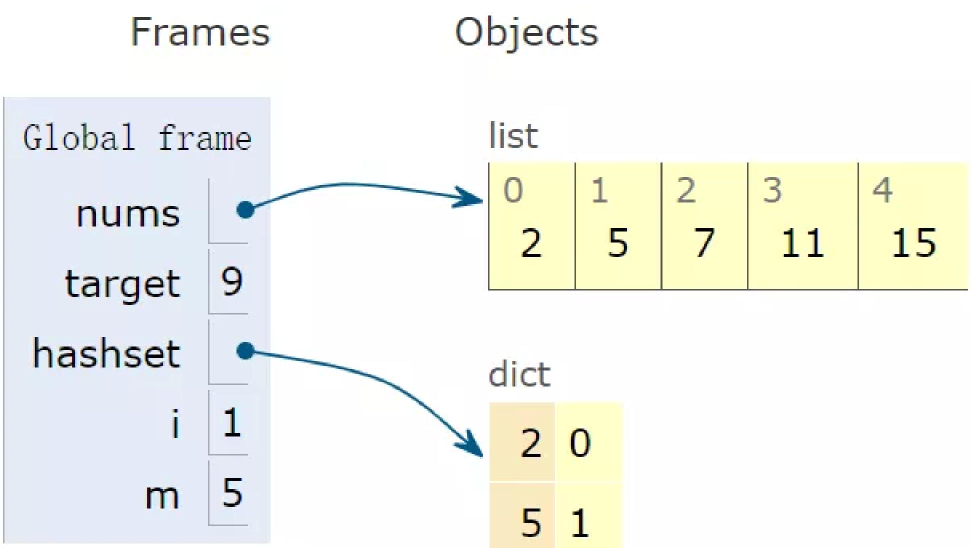
输出：(0, 2)
时间复杂度：从程序中容易分析得到时间复杂度为O(n)，仅由一个for循环贡献。

1.3 正确性分析

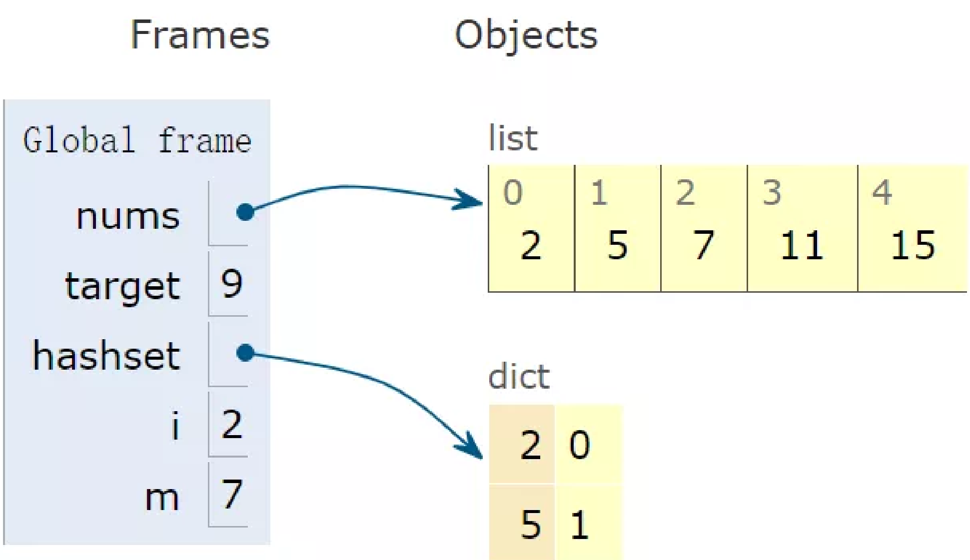
Step1：扫描数组的第1个元素



Step2：扫描数组的第2个元素



Step3：扫描数组的第3个元素



当扫描到数组的第三个元素的时候，满足条件输出(0, 2)。

02 问题二

给一个正整数数组和整数k，请查看数组中是否存在**和等于k的连续子数组**，如果存在请计算和为k的子数组的个数，您的算法的时间复杂度应优于O(n^2)

2.1 问题分析

针对此问题我们很容易想到依次枚举所有子数组，然后判断子数组之和是否等于k。但是显然时间复杂度超过O(n^2)。有了上一道题两数之和是否k的基础，我们考虑能否利用哈希表进行对子数组之和进行存储，从而减小时间复杂度。用presum(i)表示前i个数的和sum(i,j)来表示所有子数组的和，用sum依次存储a[0]+ a[1]+ a[i]的子数组之和。preSum(j)-k =preSum(i-1)则存在这么一个和为k的子数组。如果存在则hashset[sum - k]的数值加1。初始化sum(0)的个数为1。

2.2 代码实现

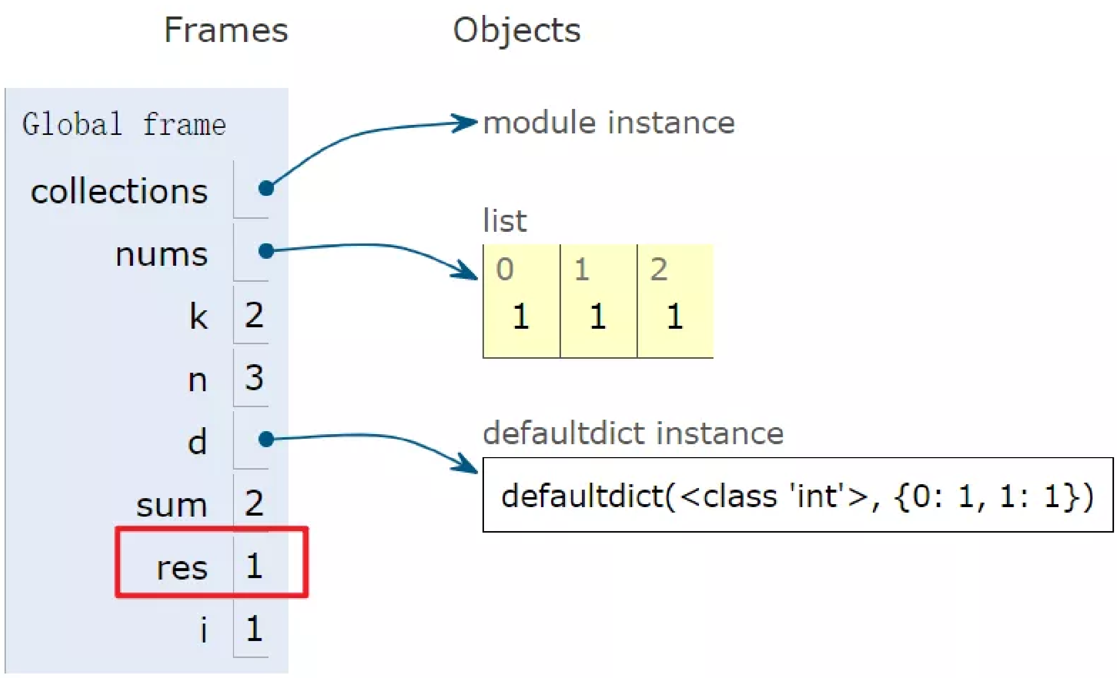
```python
if __name__ == '__main__':
     nums = [1,1,1,1]
     k=3
     n = len(nums)
     hashset = collections.defaultdict(int)
     d[0] = 1
     sum = 0
     res = 0
     for i in range(n):
         sum += nums[i]
         if sum - k in hashset:
             res += hashset[sum - k]
         hashset[sum] += 1
     print(res)
```

2.3 正确性分析

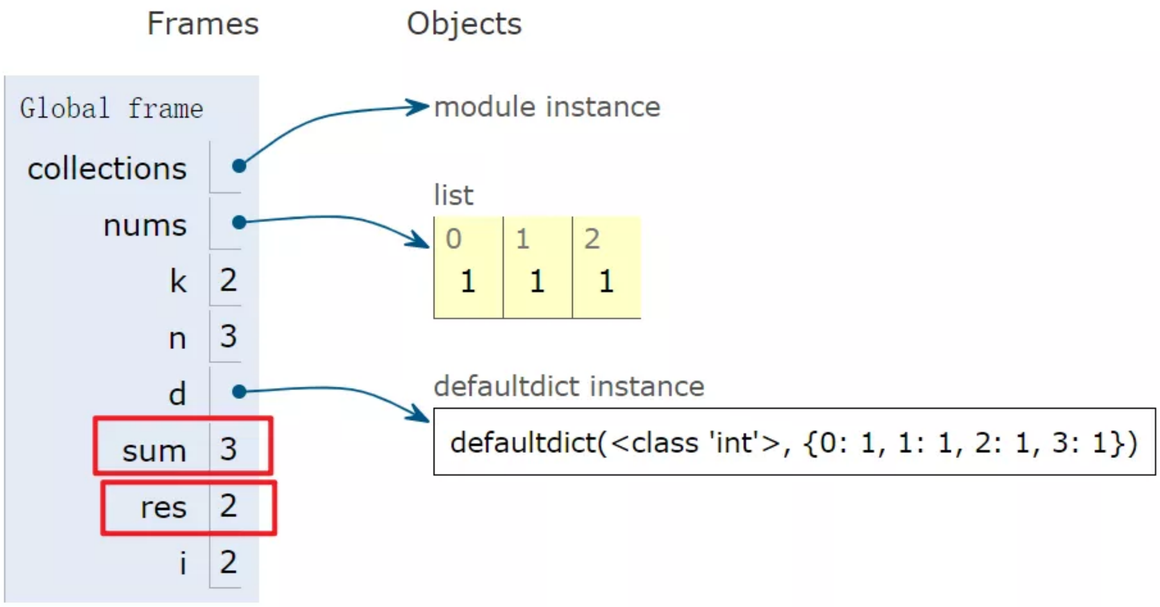
其中最不容易理解的是为什么如果hashset[sum - k]存在哈希表中即可。这是因为哈希表中的key存的是前n项和，value存的是这个和的子数组的个数，sum(i) = a[0] + a[1] + + a[i]，sum(j) = a[0] + a[1] + + a[j]。当sum(i,j)=sum(j)-sum(i)= a[i + 1] + a[i + 2] + + a[j] = k，所以我们只需要知道前面有几个sum - k的个数，就是所求的结果。

运行过程：

Setp1：扫描到第2个数字的时候，sum为2，sum-k=0，又hashset[sum-k]=1,故res为1



Setp2：扫描到第3个数字的时候，sum为3，sum-k=1，又hashset[sum-k]=1,故res=res+1，res=2。



时间复杂度：从程序中容易分析得到时间复杂度为O(n)，仅由一个for循环贡献。

03 问题三

给一个正整数数组和整数k，请查看数组中是否存在**和等于k的子数组（不一定连续）**，如果存在请计算和为k的子数组的个数。

3.1 问题分析

此题可以看作是0-1背包问题的一种扩展。当考察第i个数字的时候我们可以考虑加入a[i]是否超过当前和，若超过则不能加入，dp[i][sum]=dp[i-1][sum],若此时a[i]<=sum,则dp[i][sum]=dp[i-1][sum]+ dp[i-1][sum-a[i]]。
我们得到递归方程为：

$$\mathrm{dp}[i][j]=\begin{cases} 1, & j=0 \\ dp[i-1][sum]+dp[i-1][sum-a[i]], & a[i]<=sum \\ dp[i-1][sum], & otherwise \end{cases}$$

3.2 代码实现

```
if __name__ == '__main__':
    #5 15
    #5 5 10 2 3
    #输出4
    n, sum = map(int, input().split())
    list = [0] + list(map(int, input().split()))
    dp = [[1 if i == 0 else 0 for i in range(sum+1)] for  j in range(n + 1)]#初始化
    for i in range(1, n + 1):
        for j in range(1, sum + 1):
            if list[i] <= j:
                dp[i][j] = dp[i-1][j]+dp[i-1][j-list[i]]
            else:
                dp[i][j] = dp[i-1][j]
```

```
    print(dp[n][sum])
```

未优化：**时间复杂度**为O(n*sum), **空间复杂度**O(n*sum)
使用滚动数组优化的方案为：

```
if __name__ == '__main__':
    # 5 15
    # 5 5 10 2 3
    # 输出4
    n, sum = [int(i) for i in input().split()]
    array = list(map(int, input().split()))
    dp = [0 for i in range(sum + 1)]
    dp[0] = 1
    for i in range(n):
        j = sum
        while j >= array[i]:
            dp[j] += dp[j - array[i]]
            j -= 1
    print(dp[sum])
```

时间复杂度O(n*sum), **空间复杂度**(sum)
正确性分析参照0-1背包问题。