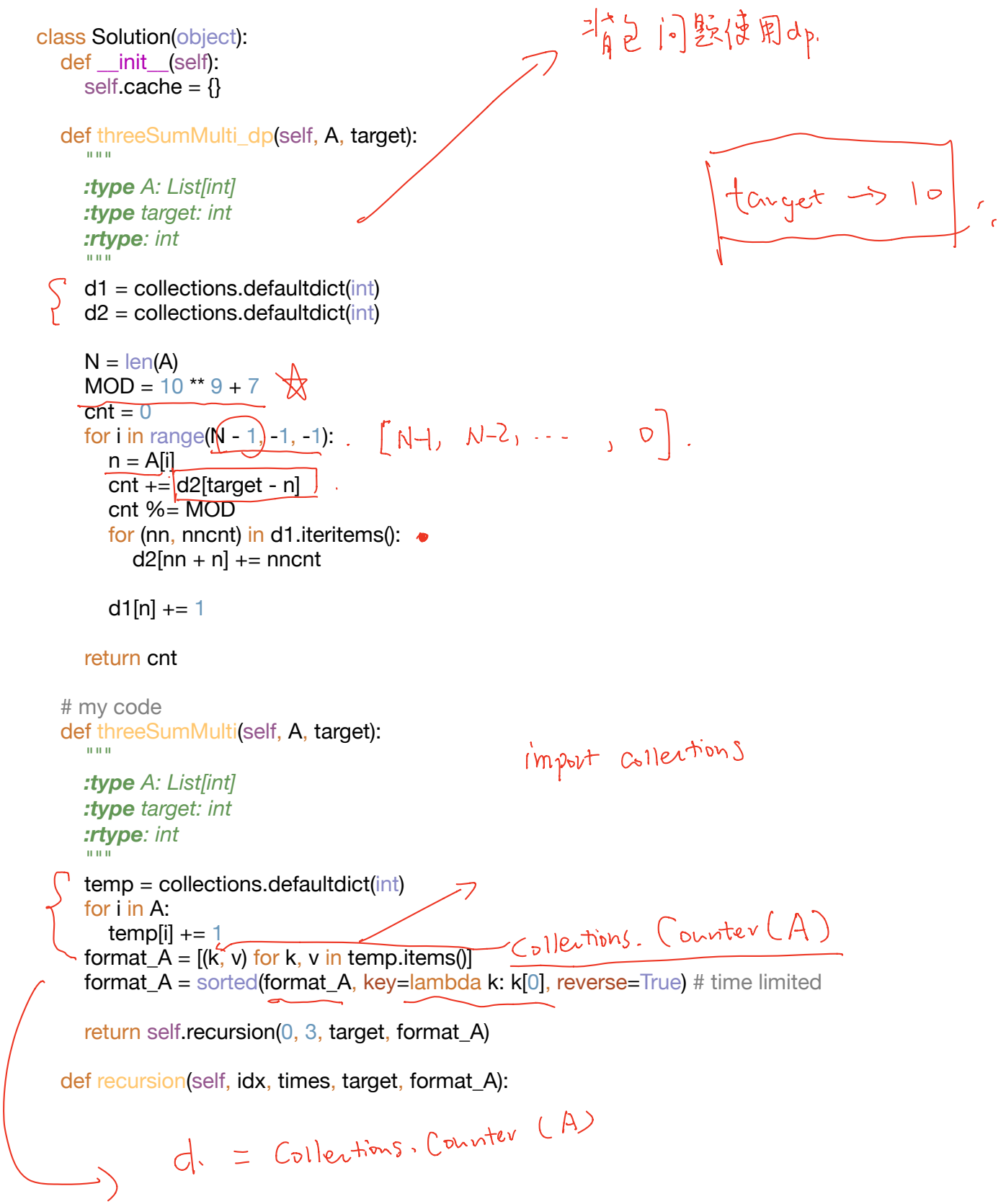
923

```
import collections
import math
from collections import Counter


class Solution(object):
    def __init__(self):
        self.cache = {}

    def threeSumMulti_dp(self, A, target):
        """
        :type A: List[int]
        :type target: int
        :rtype: int
        """
        d1 = collections.defaultdict(int)
        d2 = collections.defaultdict(int)

        N = len(A)
        MOD = 10 ** 9 + 7
        cnt = 0
        for i in range(N - 1, -1, -1):
            n = A[i]
            cnt += d2[target - n]
            cnt %= MOD
            for (nn, nncnt) in d1.iteritems():
                d2[nn + n] += nncnt

            d1[n] += 1

        return cnt

    # my code
    def threeSumMulti(self, A, target):
        """
        :type A: List[int]
        :type target: int
        :rtype: int
        """
        temp = collections.defaultdict(int)
        for i in A:
            temp[i] += 1
        format_A = [(k, v) for k, v in temp.items()]
        format_A = sorted(format_A, key=lambda k: k[0], reverse=True) # time limited

        return self.recursion(0, 3, target, format_A)

    def recursion(self, idx, times, target, format_A):
```



```
        key = (idx, times, target)
        if key in self.cache:
            return self.cache[key]

        if target < 0:
            self.cache[key] = 0
            return 0

        if times == 0:
            if target == 0:
                self.cache[key] = 1
                return 1
            else:
                self.cache[key] = 0
                return 0

        if idx == len(format_A):  # problem 1
            self.cache[key] = 0
            return 0

        # 0
        n0 = self.recursion(idx + 1, times, target, format_A)
        # 1
        n1 = format_A[idx][1] * self.recursion(idx + 1, times - 1, target - format_A[idx][0],
format_A)
        # 2
        if format_A[idx][1] >= 2:
            n2 = format_A[idx][1] * (format_A[idx][1] - 1) * 0.5 * self.recursion(idx + 1, times - 2,
                                                                  target - format_A[idx][0] * 2,
                                                                  format_A)
        else:
            n2 = 0
        # 3
        if format_A[idx][1] >= 3 and times == 3 and format_A[idx][0] * 3 == target:
            n3 = format_A[idx][1] * (format_A[idx][1] - 1) * (format_A[idx][1] - 2) / 6
        else:
            n3 = 0
        mx = int(n0 + n1 + n2 + n3)
        self.cache[key] = mx
        return mx % int(math.pow(10, 9) + 7)
    # my code

    def threeSumMulti_permutation(self, A, target):
        """
        :type A: List[int]
        :type target: int
        :rtype: int
        """
        d = Counter(A)
        choice = sorted(list(d.keys()))
```

self.cache 这种写法真的好吗，该题中不会重复的，完全不需要cache.

合并一起

合并　ret = 1 if target==0 else 0.

⇒ 做题时该解题思路考清楚

直接12，不然 *0.5 会变成小数.

合并

合并

→ 直接 10**9+7.

```
        def fact(start, end):
            if start == end:
                return start
            else:
                return start * fact(start - 1, end)

        def permulate(idx1, idx2, idx3):
            require = Counter([choice[idx1], choice[idx2], choice[idx3]])
            out = 1
            for num, count in require.items():
                if d[num] < count:
                    return 0
                else:
                    out *= fact(d[num], d[num] - count + 1) / fact(count, 1)
            return out
            # main solution

        res = 0
        for idx1 in range(len(choice)):
            idx2, idx3 = idx1, len(choice) - 1
            temp_target = target - choice[idx1]
            while idx2 <= idx3:
                if choice[idx2] + choice[idx3] == temp_target:
                    # permutate
                    res += permulate(idx1, idx2, idx3)
                    idx2 += 1
                    idx3 -= 1
                elif choice[idx2] + choice[idx3] > temp_target:
                    idx3 -= 1
                else:
                    idx2 += 1
        return int(res) % (10 ** 9 + 7)


if __name__ == "__main__":
    obj = Solution()
    print obj.threeSumMulti([1,1,2,2,2,2], 5)
    print obj.threeSumMulti_dp([1,1,2,2,2,2], 5)
    print obj.threeSumMulti_permutation([1,1,2,2,2,2], 5)
```

923

new version

```python
import collections


class Solution(object):
    def threeSumMulti(self, A, target):
        """
        :type A: List[int]
        :type target: int
        :rtype: int
        """
        temp = collections.defaultdict(int)
        for i in A:
            temp[i] += 1

        return self.recursion(0, 3, target, sorted([(k, v) for k, v in temp.items()], key=lambda m: k[0], reverse=True))

    def recursion(self, idx, times, target, format_a):
        if target < 0 or times == 0 or idx > len(format_a) - 1:
            return 1 if target == 0 and times == 0 else 0
        key, num = format_a[idx][0], format_a[idx][1]
        n0 = self.recursion(idx + 1, times, target, format_a)
        n1 = num * self.recursion(idx + 1, times - 1, target - format_a[idx][0], format_a)
        n2 = 0 if num <= 1 else num * (num - 1) / 2 * self.recursion(idx + 1, times - 2, target - key * 2, format_a)
        n3 = num * (num - 1) * (num - 2) / 6 if num >= 3 and times == 3 and key * 3 == target else 0
        return int(n0 + n1 + n2 + n3) % int(10 ** 9 + 7)


if __name__ == "__main__":
    obj = Solution()
    print obj.threeSumMulti([1,1,2,2,2,2], 5)
```

未必非要一次性写成这样

这步将所有可能全放一起.

idx永远增加

不需要.