242. 有效的字母异位词

给定两个字符串 s 和 t ，编写一个函数来判断 t 是否是 s 的 字母异位词. s 和 t 仅包含小写字母

示例 1:

输入: s = "anagram", t = "nagaram"

输出: true

示例 2:

输入: s = "rat", t = "car"

输出: false

进阶: 如果输入字符串包含 unicode 字符怎么办？你能否调整你的解法来应对这种情况?

In [ ]:
```python
# 尝试1,  没考虑"aacc" "ccac" 这种情况

class Solution:
    def isAnagram(self, s: str, t: str) -> bool:

        if len(s) != len(t):
            return False

        else:
            letters_1 = set()
            letters_2 = set()
            for letter in s:
                letters_1.add(letter)

            for letter in t:
                letters_2.add(letter)


            if letters_1 == letters_2:
                return True
            else:
                return False
```

In [1]:
```python
# 尝试2, 将字母统计为两个dict

from collections import defaultdict

class Solution:
    def isAnagram(self, s: str, t: str) -> bool:



        def count_letters(word):
            count_dict = ç
            for letter in word:
                count_dict[letter] += 1
            return count_dict

        if len(s) != len(t):
            return False

        else:
            if count_letters(s) == count_letters(t):
                return True
            else:
                return False

# 正确
```

defaultDict用法： defaultdict(int)

另外两种方法：

1. 用ascii序号结合列表的index作为key存储。 ascii序号是 ord(a)
2. 用counter

In [ ]:
```python
class Solution(object):
    def isAnagram(self, s: str, t: str) -> bool:
        from collections import Counter
        a_count = Counter(s)
        b_count = Counter(t)
        return a_count == b_count

class Solution(object):
    def isAnagram(self, s: str, t: str) -> bool:
        from collections import Counter
        a_count = Counter(s)
        b_count = Counter(t)
        return a_count == b_count
```

Counter 是 dict 字典的子类，Counter 拥有类似字典的 key 键和 value 值，只不过 Counter 中的键为待计数的元素，而 value 值为对应元素出现的次数 count，为了方便介绍统一使用元素和 count 计数来表示。虽然 Counter 中的 count 表示的是计数，但是 Counter 允许 count 的值为 0 或者负值。

349. 两个数组的交集

问题描述

给定两个数组 `nums1` 和 `nums2`，返回它们的 **交集**。
输出结果中的每个元素一定是 **唯一** 的，且可以 **不考虑输出结果的顺序**。

示例

示例 1

输入：
`nums1 = [1,2,2,1]`
`nums2 = [2,2]`

输出：
`[2]`

示例 2

输入：
`nums1 = [4,9,5]`
`nums2 = [9,4,9,8,4]`

输出：
`[9,4]`
解释：
`[4,9]` 也是可通过的。

提示

1. 每个数组的长度不超过 1000。
2. 数组中的值范围为 -10^9 到 10^9。

In [ ]:
```python
from typing import List

def intersection(self, nums1: List[int], nums2: List[int]) -> List[int]:
    len1 = len(nums1)
    len2 = len(nums2)
    result = set()

    # 不需要交换
    # if len2>len1:
    #     temp = nums1
    #     nums1 = nums2
    #     nums2 = temp

    for num in nums1:
        if num in nums2:
            result.add(num)

    return list(result)
```

In [7]:
```python
# 更高效的方法
# 创建set的时候直接转换
class Solution:
    def intersection(self, nums1: list[int], nums2: list[int]) -> list[int]:
        set1 = set(nums1)
        set2 = set()
        for v in nums2:
            if v in set1:
```

```
                set2.add(v)
        return list(set2)

# 更更高效的方法，把长度短的转化为set

class Solution:
    def intersection(self, nums1: List[int], nums2: List[int]) -> List[int]:
        # 把长度大的数组转换为 hash set，记为 fewerNums，另一方记为 moreNums
        if len(nums1) > len(nums2):
            moreNums = set(nums1)
            fewerNums = nums2
        else:
            moreNums = set(nums2)
            fewerNums = nums1

        # 从 moreNums 中向 ans 中添加仅在 fewerNums 中存在的数字
        ans = set()
        for v in moreNums:
            if v in fewerNums:
                ans.add(v)

        # 转换为列表再返回
        return list(ans)
```

In [6]:
```
# 一行
class Solution:
    def intersection(self, nums1: list[int], nums2: list[int]) -> list[int]:
        return list(set(nums1) & set(nums2))
```

In [ ]:
```

```

202. 快乐数

问题描述

编写一个算法来判断一个数 `n` 是否是**快乐数**。

快乐数的定义：

- 对于一个正整数，每次将该数替换为它每一位数字的平方和。
- 重复这个过程，直到：
  - 结果变为 `1`，则该数为**快乐数**。
  - 或者陷入**无限循环**，无法变到 `1`，则该数不是快乐数。

如果 `n` 是**快乐数**，返回 `true`；否则返回 `false`。

示例

示例 1

输入：
`n = 19`
输出：
`true`

解释：

```
1² + 9² = 82
8² + 2² = 68
6² + 8² = 100
1² + 0² + 0² = 1
```

示例 2

输入：
`n = 2`
输出：
`false`

解释：无限循环，无法变为 `1`。

In [ ]:
```

```

In [10]:
```
n = 1000
print(set(str(n)))
```

```
{'0', '1'}
```

In [ ]:
```
# 第一次尝试，递归不对
class Solution:
```

```
    def isHappy(self, n: int) -> bool:

        num_list = list(str(n))

        if set(num_list) & set(str(n)) != []:
            return False

        next_n = 0
        for num in num_list:
            next_n += num ** 2

        return self.isHappy(next_n)
```

In [12]:
```
# 第二次尝试，用set记录访问过的数字，通过

class Solution:
    def isHappy(self, n: int) -> bool:
        visited = set()

        def get_next_n(n):
            return sum(int(num) ** 2 for num in str(n))

        while n not in visited:
            if n == 1:
                return True
            else:
                visited.add(n)
                n = get_next_n(n)

        return False
```

In [ ]:
```
# 第三次尝试，逻辑反过来，也对

class Solution:
    def isHappy(self, n: int) -> bool:
        visited = set()

        def get_next_n(n):
            return sum(int(num) ** 2 for num in str(n))

        while n != 1:
            if n in visited:
                return False
            visited.add(n)
            n = get_next_n(n)

        return True
```

1. 两数之和

问题描述

给定一个整数数组 `nums` 和一个整数目标值 `target`，请你在该数组中找出**和为目标值 `target` 的那两个整数**，并返回它们的数组下标。

- 你可以假设每种输入只会对应一个答案。
- 你**不能使用**同一个元素两次。
- 你可以按**任意顺序**返回答案。

示例

示例 1

输入:
`nums = [2,7,11,15]`
`target = 9`

输出:
`[0,1]`

解释:
因为 `nums[0] + nums[1] == 9`，返回 `[0, 1]`。

示例 2

输入:
`nums = [3,2,4]`
`target = 6`

输出:
`[1,2]`

示例 3

输入：
`nums = [3,3]`
`target = 6`

输出：
`[0,1]`

进阶

你能想出一个时间复杂度小于 (O(n^2)) 的算法吗?

In [ ]:
```python
# 尝试1: 双指针遍历
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        slow = 0
        fast = 1
        while slow != fast:
            while fast != len(nums):
                sum = nums[slow] + nums[fast]
                if sum == target:
                    return [slow, fast]
                else:
                    fast += 1
            slow += 1
            fast = slow + 1
        return []
```

In [ ]:
```python
# 尝试2: hashmap
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        visited = dict()

        for index, value in enumerate(nums):
            margin = target - value
            if margin in visited:
                return [index, visited[margin]]
            else:
                visited[value] = index #注意这里是value
        return []
```