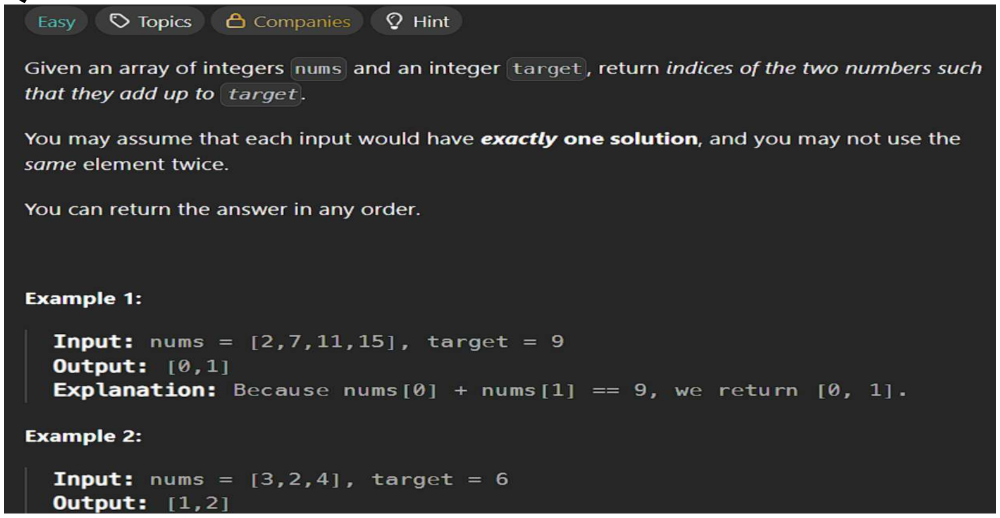
1. Two Sum

20 November 2025 22:11

Question



Easy · Topics · Companies · Hint

Given an array of integers `nums` and an integer `target`, return *indices of the two numbers such that they add up to* `target`.

You may assume that each input would have ***exactly* one solution**, and you may not use the *same* element twice.

You can return the answer in any order.

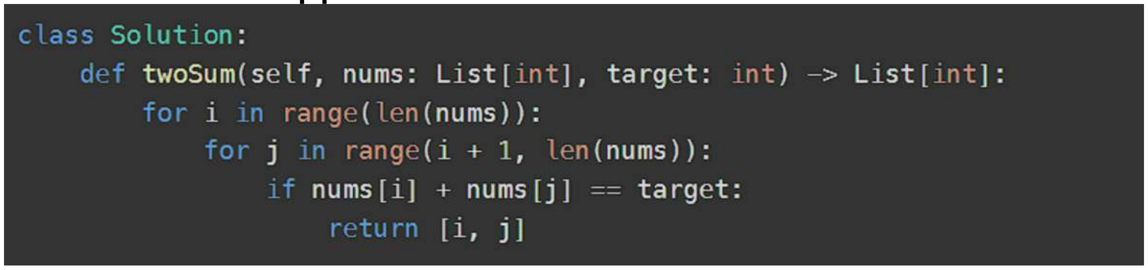
Example 1:

```
Input: nums = [2,7,11,15], target = 9
Output: [0,1]
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].
```

Example 2:

```
Input: nums = [3,2,4], target = 6
Output: [1,2]
```

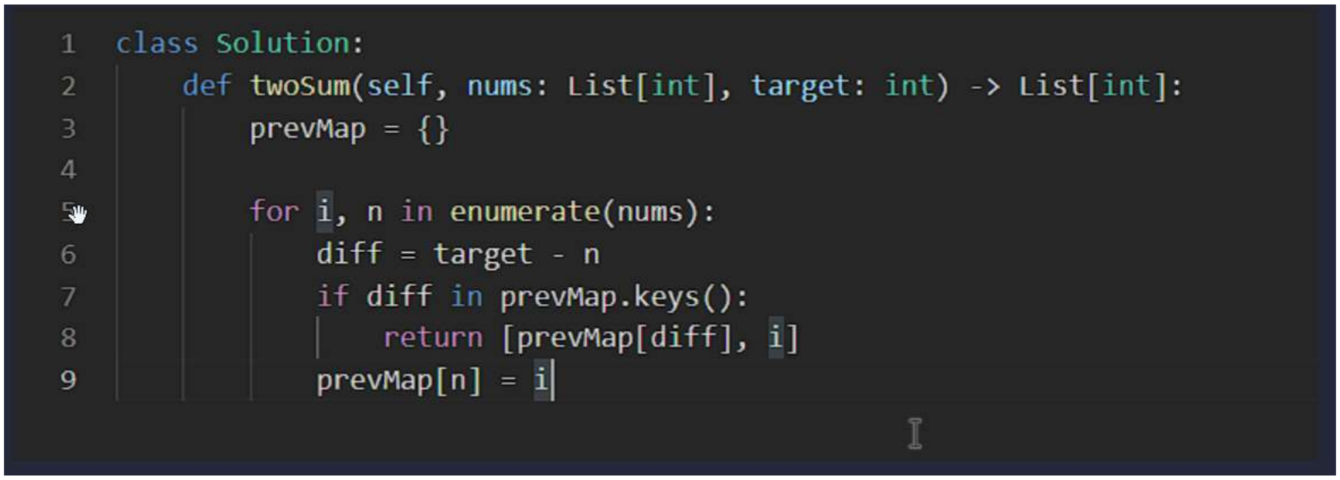
Brute Force Approach



```python
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        for i in range(len(nums)):
            for j in range(i + 1, len(nums)):
                if nums[i] + nums[j] == target:
                    return [i, j]
```

Optimized Approach with Hashmap

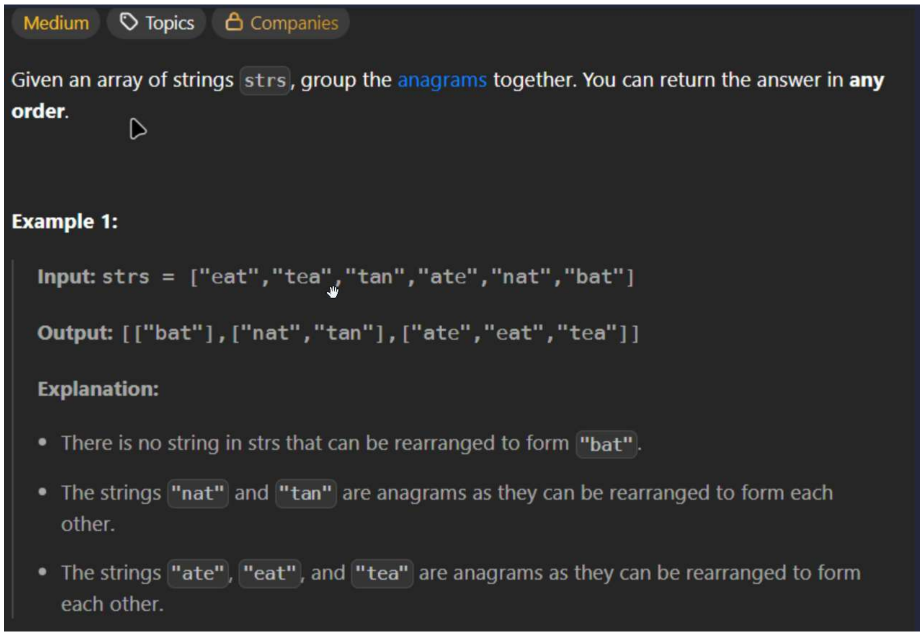
- The idea goes like this, we will have a hashmap which will contain {number : index},
- Now, we will loop through the array, we will have a difference which will be target - curr_n
- If the difference is in the hashmap.keys(), then we have found two sum. Return the index of the diff (as difference is already stored in the hashmap as key) and the index of the current number.
- If its not, just keep adding the number as key and index as value.
- Example, array = [2, 3, 4, 5] target = 5, now
    - First we will check if diff = 5 - 2 is in map ? no
    - Then, we will put map = {2: 0}.
    - Second, we will check if diff = 5 - 3 = 2 2 is in map ? Yes, the first key is 2.
    - Then, we will return the 2's index 0 and curr number 3's index 1 as [0,1] , hence we got our answer.



```python
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        prevMap = {}

        for i, n in enumerate(nums):
            diff = target - n
            if diff in prevMap.keys():
                return [prevMap[diff], i]
            prevMap[n] = i
```

Part 1 - 49. Group Anagrams

21 November 2025 13:35



Medium Topics Companies

Given an array of strings `strs`, group the anagrams together. You can return the answer in **any order**.

Example 1:

Input: `strs = ["eat","tea","tan","ate","nat","bat"]`

Output: `[["bat"],["nat","tan"],["ate","eat","tea"]]`

Explanation:

- There is no string in strs that can be rearranged to form `"bat"`.
- The strings `"nat"` and `"tan"` are anagrams as they can be rearranged to form each other.
- The strings `"ate"`, `"eat"`, and `"tea"` are anagrams as they can be rearranged to form each other.

Brute Force Approach Using Hashmap:

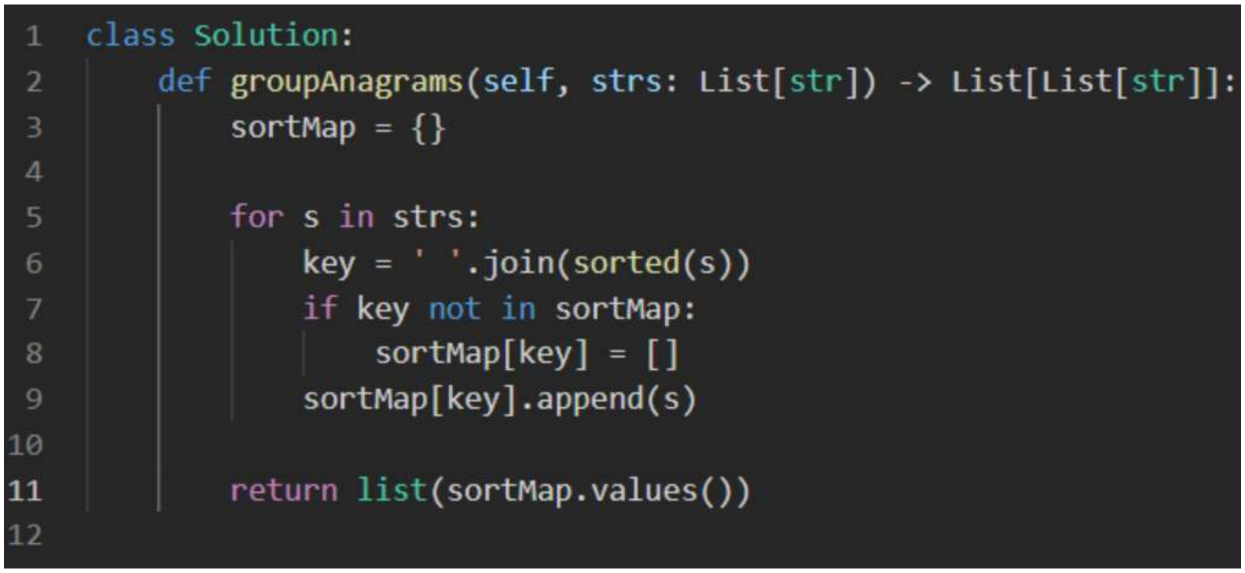
Here, naive approach is just sorting all the strings in the list, creating a hashmap and then do:

- If the sorted string isnt present in the hashmap, make the sorted string as key and create a empty list as value.
- then append the original string to the hashmap as it will anyway be correctly mapped according to the sorted string.

Now, return the values of the hashmap as a list.

Time Complexity : O(m * n log n)
Space Complexity : O(m * n)

Below you can see the approach:



```python
class Solution:
    def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
        sortMap = {}

        for s in strs:
            key = ' '.join(sorted(s))
            if key not in sortMap:
                sortMap[key] = []
            sortMap[key].append(s)

        return list(sortMap.values())

```

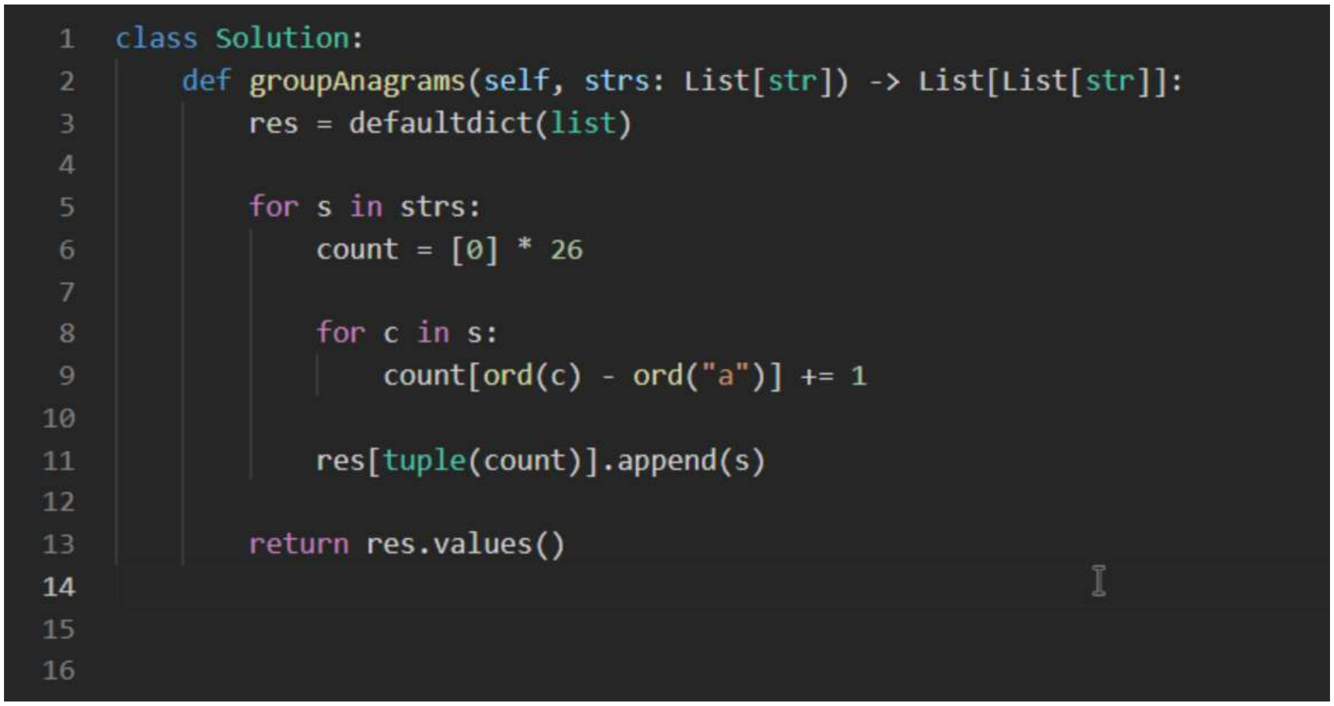
Part 2 - 49. Group Anagrams

21 November 2025 15:06

Optimized Approach using Character frequency Count

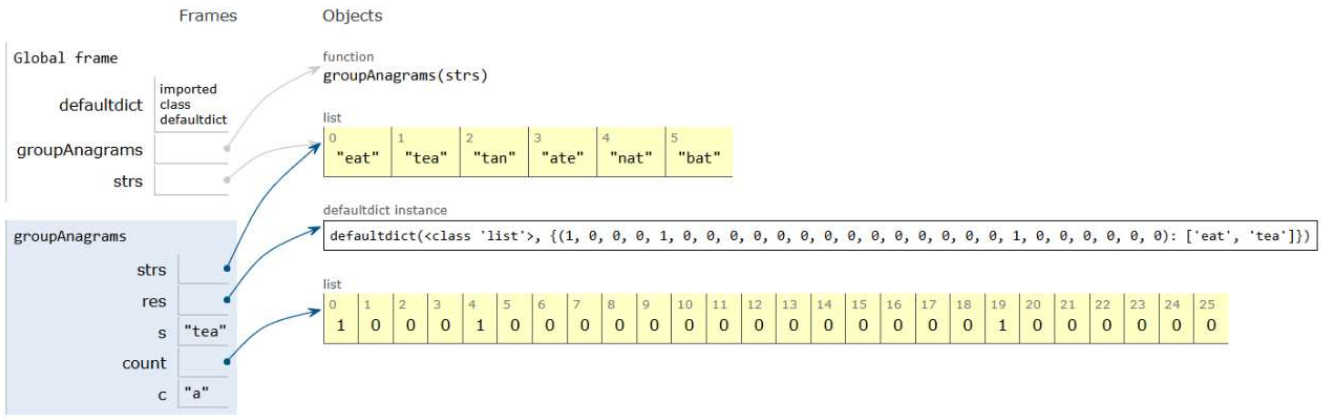
- Here, we will be getting the frequency counts of each letter in the each string, then we will map the frequency as the key, the string as the value.
- If we keep iterating, the same letters will have same frequencies which will gives us grouped anagrams.
- Time Complexity is O(m * n)
- Space Complexity is O(m * n)

Below you can see the approach



```python
class Solution:
    def groupAnagrams(self, strs: List[str]) -> List[List[str]]:
        res = defaultdict(list)

        for s in strs:
            count = [0] * 26

            for c in s:
                count[ord(c) - ord("a")] += 1

            res[tuple(count)].append(s)

        return res.values()
```

Code Approach :



0	1	2	3	4	5
"eat"	"tea"	"tan"	"ate"	"nat"	"bat"

```
defaultdict(<class 'list'>, {(1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0): ['eat', 'tea']})
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0