Solution Review: A Sublist with a Sum of 0

This review provides a detailed analysis of the solution to the A Sublist with a Sum of 0 Challenge.

We'll cover the following

Solution: Iterative Hashing

```python
def find_sub_zero(my_list):
    # Use hash table to store the cumulative sum as a key and
    # the element as the value till which the sum has been calculated
    # Traverse the list and return true if either
    # elem == 0 or sum == 0 or hash table already contains the sum
    # If you completely traverse the list and haven't found any
    # of the above three conditions, then simply return false
    ht = dict()
    total_sum = 0
    # Traverse through the given list
    for elem in my_list:
        total_sum += elem
        if elem is 0 or total_sum is 0 or ht.get(total_sum) is not None:
            return True
        ht[total_sum] = elem
    return False


my_list = [6, 4, -7, 3, 12, 9]

print(find_sub_zero(my_list))
print(find_sub_zero(my_list))

```

The naive solution would be to iterate the list in a nested loop, summing each element with all the elements succeeding it.

A hash table makes things much simpler.

We basically have to check for 3 conditions:

- If `0` exists in the list
- If the sum becomes zero in the iteration
- If the sum reverts back to a value which was already a key in the hash table. This means that there was a sublist that has a sum of zero making the overall sum to go back to a previous value.

Any of these three conditions confirms the existence of a sublist that sums up to be zero.

Time Complexity

As always, a linear iteration over **n** elements means that the algorithm's time complexity is *O(n)*.

← Back

Challenge 6: A Sublist with a Sum of 0

Challenge 7: Word Formation Using a ...

☑ Mark as Completed

Report an Issue