[Click on the logo to find **Problem Statement**]

Intuition

This code implements the `strStr` function, which searches for the first occurrence of a `needle` string within a `haystack` string. It first checks if the `needle` is an empty string, returning 0 if so. Then, it iterates through the `haystack` string character by character. Within this iteration, it checks if the substring of `haystack` starting at the current index matches the `needle` string. If a mismatch is found at any point, it breaks out of the inner loop. If the entire `needle` string is matched, it returns the starting index of the match. If no match is found, it returns -1. This algorithm employs a nested loop, with the outer loop traversing the `haystack` and the inner loop comparing characters with the `needle` .

Approach

1. First, it checks if the `needle` string is empty. If it is, it immediately returns 0 because an empty string is always found at index 0 within any string.
2. Then, it iterates through each character in the `haystack` string using a for loop. The range of the loop is determined such that it ensures the substring starting from the current index is at least as long as the `needle` string, ensuring no out-of-bounds access.
3. Within this outer loop, there is another loop that iterates through each character in the `needle` string. It compares each character of the `haystack` substring starting from the current index with the corresponding character of the `needle` string.
4. If a mismatch is encountered at any point during the comparison, the inner loop breaks, and the outer loop moves to the next character in the `haystack` .
5. If the inner loop completes without any mismatches, it means that the entire `needle` string is found starting from the current index in the `haystack` . In this case, it returns the starting index of the match.
6. If the outer loop completes without finding any matches, it means the `needle` string is not present in the `haystack` , so it returns -1.

Complexity

- Time complexity: O(n * m)
- Space complexity: O(1)

Code

```python
class Solution:
    def strStr(self, haystack: str, needle: str) -> int:
        if needle == "":
            return 0

        for i in range(len(haystack) + 1 - len(needle)):
            for j in range(len(needle)):
                if haystack[i + j] != needle[j]:
                    break
                if j == len(needle) - 1:
                    return i
        return -1
```

If you want to see more solutions to coding problems, you can visit:

GitHub