DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Experiment 4(A)

Student Name: Mohammad Salman **UID:** 22BCS12474
Branch: CSE **Section/Group:** IOT 624/B
Semester: 6th **Date of Performance:** 11 Feb 2025
Subject Name: Advanced Programming Lab-2 **Subject Code:** 22CSP-351

1. **Title:** Rotate String

2. **Objective:** Given two strings s and goal, return true if and only if s can become goal after some number of shifts on s. A shift on s consists of moving the leftmost character of s to the rightmost position.

3. **Algorithm**

   1) **Check Lengths:**
   - If s and goal have different lengths, return false immediately.

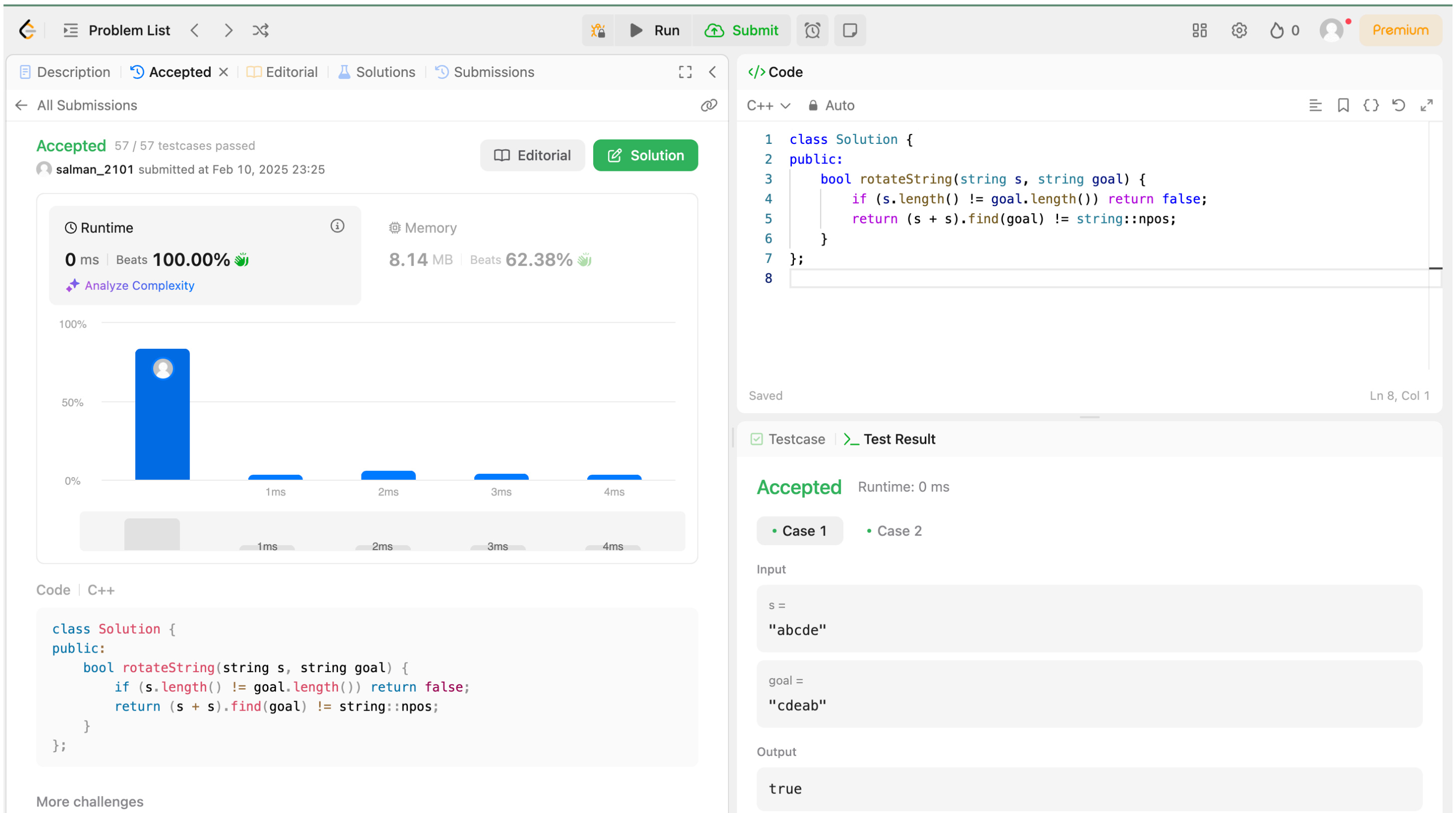
   2) **Concatenate the Original String:**
   - Create a new string concatenated by appending s to itself (s + s).

   3) **Check for Rotation:**
   - If goal is a substring of concatenated, return true.
   - Otherwise, return false.

4. **Implementation/Code:**

```
class Solution {
public:
    bool rotateString(string s, string goal)
     {
        if (s.length() != goal.length())
        return false;
        return (s + s).find(goal) != string::npos;
    }
};
```

5. Output:



6. Time Complexity : O(n)

7. Space Complexity: O(n)

8. Learning Outcomes

a) Understand how to check if one string is a substring of another using built-in functions.
b) Understand how to check if two strings can be rotations by comparing their lengths.
c) Analyze why the approach runs in **O(N)** time and requires **O(N)** space.

Experiment 4(B)

1. **Title:** Find the Index of the First Occurrence in a String

2. **Objective:** Given two strings needle and haystack, return the index of the first occurrence of needle in haystack, or -1 if needle is not part of haystack.

3. **Algorithm:**

   a) **Get Lengths:**
   - Store the lengths of haystack (n) and needle (m).

   b) **Edge Case Check:**
   - If needle is empty (m == 0), return 0 immediately.

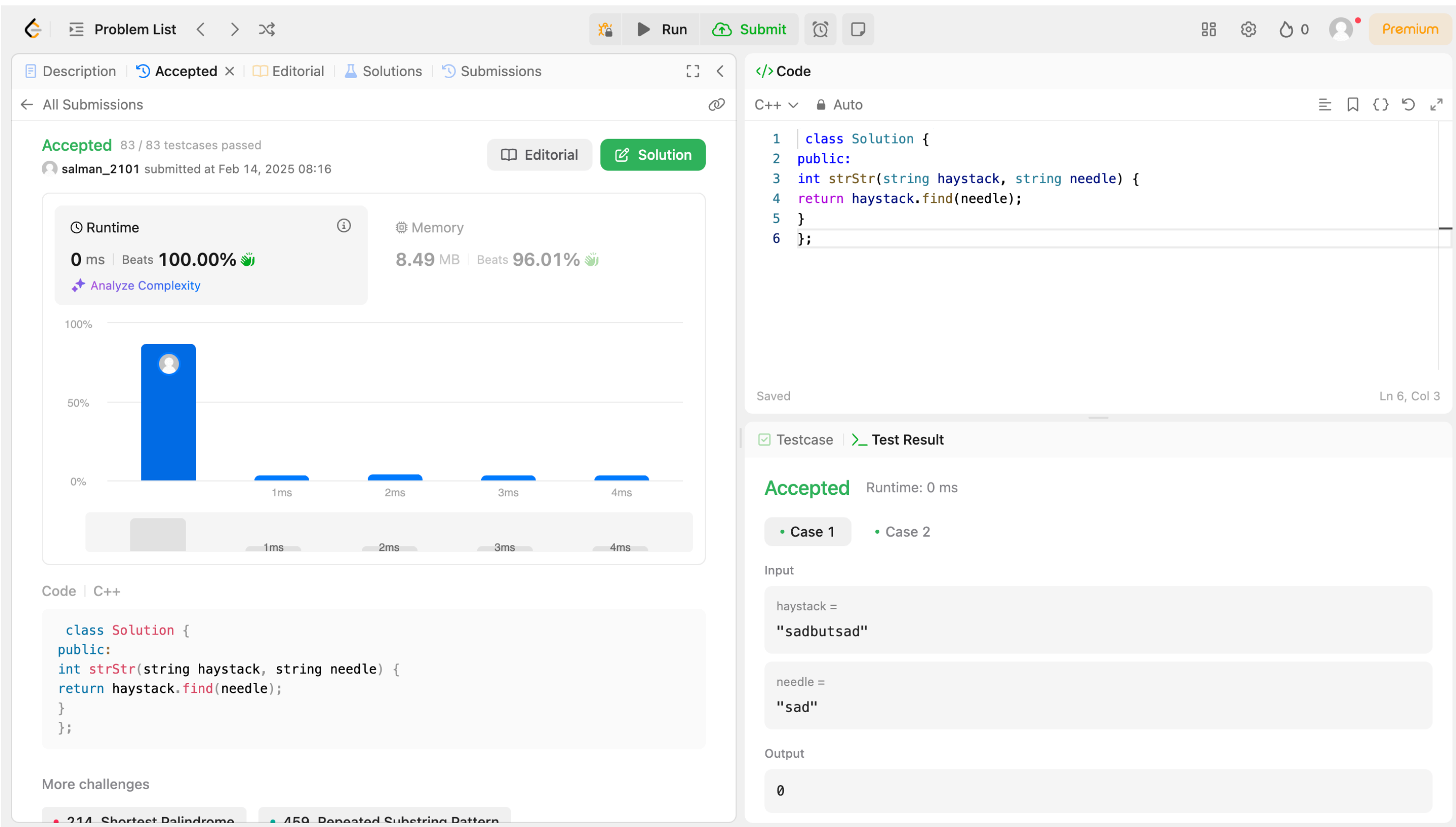
   c) **Loop Through haystack:**
   - Iterate from index 0 to n - m.
   - Extract the substring of length m from haystack.
   - Compare it with needle.
   - If they match, return the starting index.

   d) **Return -1 if Not Found:**
   - If the loop completes without a match, return -1.

4. **Implementation/Code**

```
class Solution
{
public:
int strStr(string haystack, string needle)
{
return haystack.find(needle);
}
};
```

5. Output:



6. Time Complexity: O(n*m)

7. Space Complexity: O(1)

8. Learning Outcomes:

- Understand how to handle cases where `needle` is empty or longer than `haystack`.
- Recognize that substring comparison in a loop leads to **O(N * M)** complexity.
- Learn how to extract substrings and compare them efficiently.