Experiment1.2

Student Name: Himanshu **UID:** 20BCS7944
Branch: BE-CSE **Section/Group:** 905/A
Semester: 6th **Date of Performance:** 15/02/2023
Subject Name: Competitive Coding-II **Subject Code:** 20CSP-351

1. Aim:

To implement the concept of string manipulation.

2. Objective:

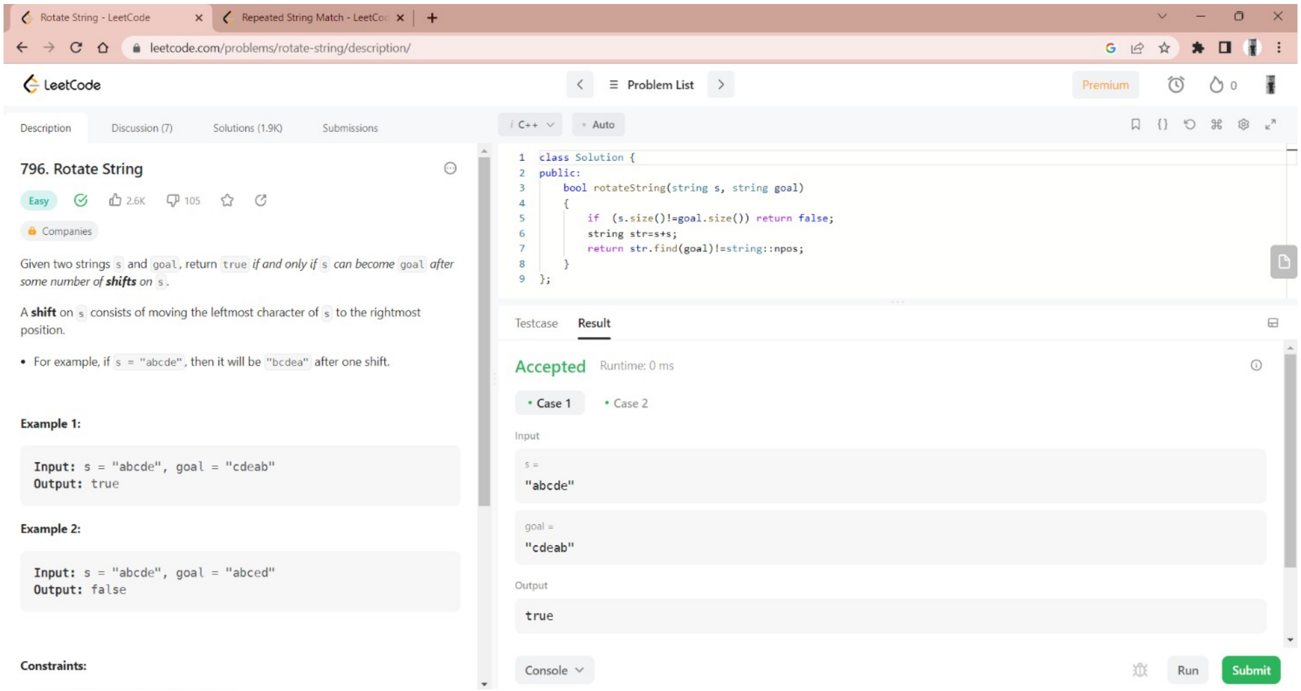
- The objective is to build problem solving capability and to learn the basic concepts of data structures.
- The implementation of rotate string which shows and brushes up the concept of strings and can be solved through various approaches.
- The implementation of repeated string matching in which the concept of npos was introduced.

3. LeetCode code and output:

- **ROTATE STRING**

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

OUTPUT:



796. Rotate String

Easy 2.6K 105

Companies

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.

- For example, if `s = "abcde"`, then it will be `"bcdea"` after one shift.

Example 1:

```
Input: s = "abcde", goal = "cdeab"
Output: true
```

Example 2:

```
Input: s = "abcde", goal = "abced"
Output: false
```

Constraints:

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

Accepted Runtime: 0 ms

Input

s =
"abcde"

goal =
"cdeab"

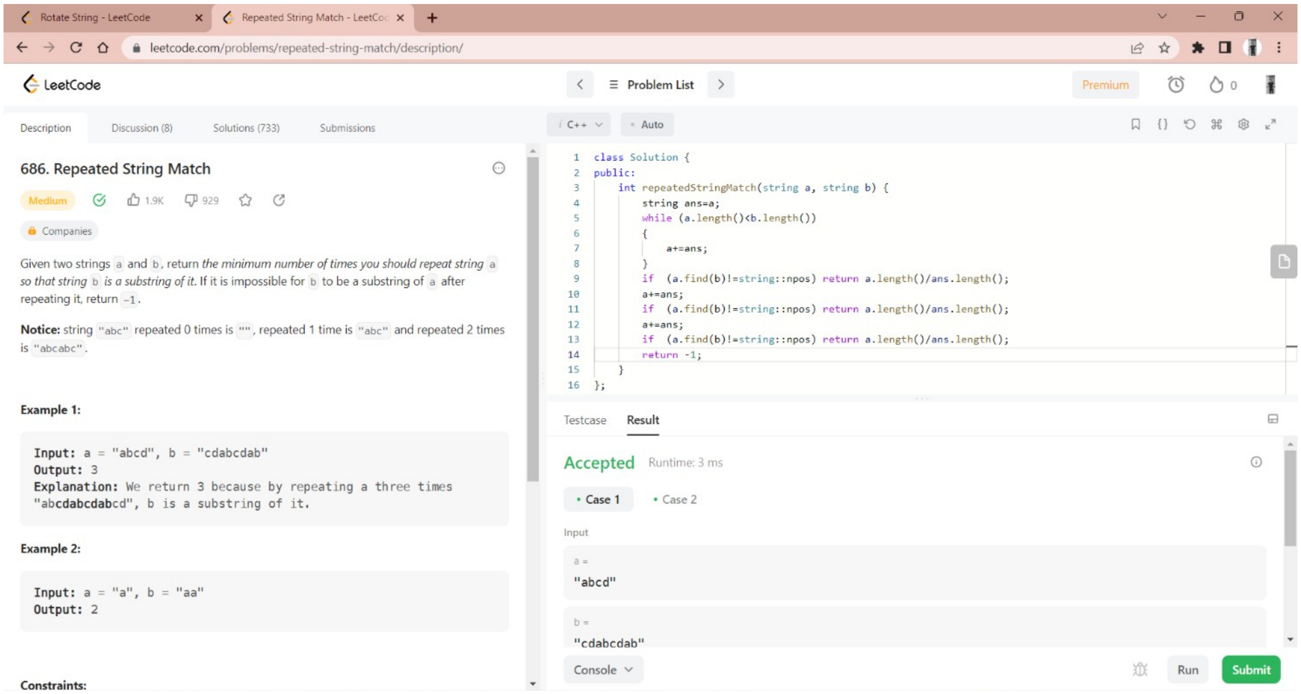
Output

true

- **REPEATED STRING MATCHING**

```
class Solution {
public:
    int repeatedStringMatch(string a, string b) {
        string ans=a;
        while (a.length()<b.length())
        {
            a+=ans;
        }
        if  (a.find(b)!=string::npos) return a.length()/ans.length();
        a+=ans;
        if  (a.find(b)!=string::npos) return a.length()/ans.length();
        a+=ans;
        if  (a.find(b)!=string::npos) return a.length()/ans.length();
        return -1;
    }
};
```

OUTPUT:



686. Repeated String Match

Given two strings `a` and `b`, return *the minimum number of times you should repeat string* `a` *so that string* `b` *is a substring of it*. If it is impossible for `b` to be a substring of `a` after repeating it, return `-1`.

Notice: string `"abc"` repeated 0 times is `""`, repeated 1 time is `"abc"` and repeated 2 times is `"abcabc"`.

Example 1:

```
Input: a = "abcd", b = "cdabcdab"
Output: 3
Explanation: We return 3 because by repeating a three times "abcdabcdabcd", b is a substring of it.
```

Example 2:

```
Input: a = "a", b = "aa"
Output: 2
```

Constraints:

```
class Solution {
public:
    int repeatedStringMatch(string a, string b) {
        string ans=a;
        while (a.length()<b.length())
        {
            a+=ans;
        }
        if  (a.find(b)!=string::npos) return a.length()/ans.length();
        a+=ans;
        if  (a.find(b)!=string::npos) return a.length()/ans.length();
        a+=ans;
        if  (a.find(b)!=string::npos) return a.length()/ans.length();
        return -1;
    }
};
```

Accepted Runtime: 3 ms

Input

a =
"abcd"

b =
"cdabcdab"