

Experiment – 1.2

Student Name: Avinash Jena

UID: 20BCS2690

Branch: CSE

Section/Group: 707/B

Subject Name: Competitive Coding II

Subject Code: 20CSP-351

1. Aim: 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.

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

Example 1:

Input: *s* = "abcde", *goal* = "cdeab"

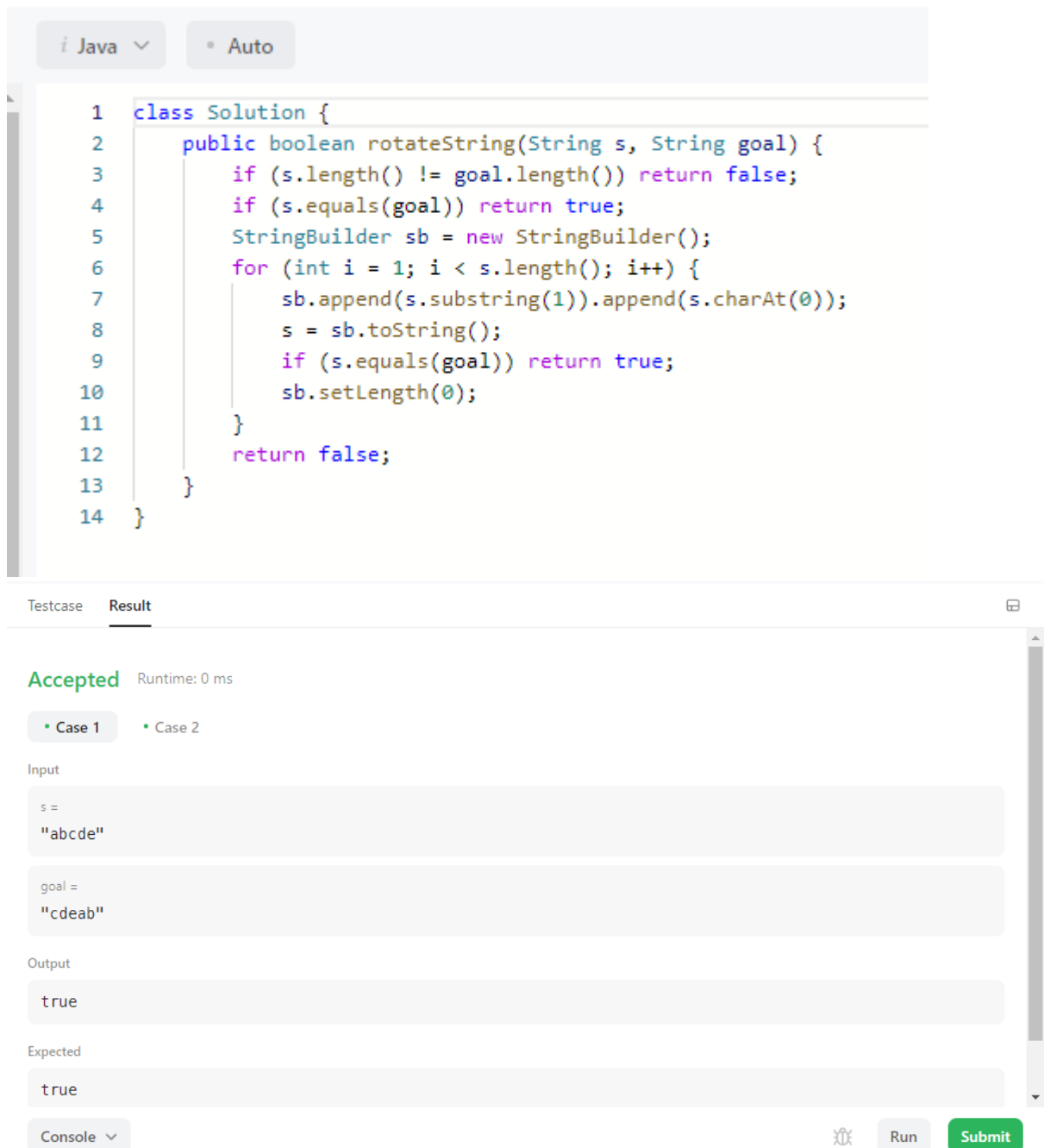
Output: true

3. Code:

```
class Solution {
    public boolean rotateString(String s, String goal) {
        if (s.length() != goal.length()) return false;
        if (s.equals(goal)) return true;
        StringBuilder sb = new StringBuilder();
        for (int i = 1; i < s.length(); i++) {
            sb.append(s.substring(1)).append(s.charAt(0));
            s = sb.toString();
            if (s.equals(goal)) return true;
        }
    }
}
```

```
sb.setLength(0);  
    }  
    return false;  
}  
}
```

4. Output:



The screenshot displays a Java IDE with a code editor and a test runner interface. The code editor shows a Java class named `Solution` with a method `rotateString` that checks if a string `s` can be rotated to match a `goal`. The method uses a `StringBuilder` to rotate the string by one position and checks for equality with the goal. The test runner interface shows the test case "Case 1" as "Accepted" with a runtime of 0 ms. The input values are `s = "abcde"` and `goal = "cdeab"`, and the output is `true`, which matches the expected result.

```
1 class Solution {  
2     public boolean rotateString(String s, String goal) {  
3         if (s.length() != goal.length()) return false;  
4         if (s.equals(goal)) return true;  
5         StringBuilder sb = new StringBuilder();  
6         for (int i = 1; i < s.length(); i++) {  
7             sb.append(s.substring(1)).append(s.charAt(0));  
8             s = sb.toString();  
9             if (s.equals(goal)) return true;  
10            sb.setLength(0);  
11        }  
12        return false;  
13    }  
14 }
```

Testcase Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

s =
"abcde"

goal =
"cdeab"

Output

true

Expected

true

Console Run Submit

5. Aim: Repeated String Match

6. Objective:

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.

7. Code:

```
class Solution {
    public int repeatedStringMatch(String A, String B) {
        if(A==null || B==null)
            return -1;
        int[] pref = longPrefSub(B);
        StringBuilder temp = new StringBuilder();
        int count = 0;
        while(temp.length()<B.length()){
            temp.append(A);
            count++;
        }
        if(isFound(temp.toString(), B, pref))
            return count;
        if(isFound(temp.append(A).toString(), B, pref))
            return count+1;
        return -1;
    }

    private int[] longPrefSub(String word) {
```

```
int[] pattern = new int[word.length()];
Arrays.fill(pattern, -1);
int i = 1, j = 0;
while (i < word.length()) {
    if (word.charAt(i) == word.charAt(j)) {
        pattern[i] = j;
        i++;
        j++;
    } else if (j > 0) {
        j = pattern[j - 1] + 1;
    } else {
        i++;
    }
}
return pattern;
}

private boolean isFound(String string, String substring, int[] pattern) {
    int i = 0, j = 0;
    while (i + substring.length() - j <= string.length()) {
        if (string.charAt(i) == substring.charAt(j)) {
            if (j == substring.length() - 1) {
                return true;
            }
            i++;
            j++;
        } else if (j > 0) {
            j = pattern[j - 1] + 1;
        } else {
            i++;
        }
    }
    return false;
}
}
```

8. Output:

```
Java Auto
1 class Solution {
2     public int repeatedStringMatch(String A, String B) {
3         if(A==null || B==null)
4             return -1;
5         int[] pref = longPrefSub(B);
6         StringBuilder temp = new StringBuilder();
7         int count = 0;
8         while(temp.length()<B.length()){
9             temp.append(A);
10            count++;
11        }
12        if(isFound(temp.toString(), B, pref))
13            return count;
14        if(isFound(temp.append(A).toString(), B, pref))
15            return count+1;
16        return -1;
17    }
18
19    private int[] longPrefSub(String word) {
20        int[] pattern = new int[word.length()];
21        Arrays.fill(pattern, -1);
22        int i = 1, j = 0;
23        while (i < word.length()) {
24            if (word.charAt(i) == word.charAt(j)) {
25                pattern[i] = j;
26                i++;
27                j++;
28            } else if (j > 0) {
29                j = pattern[j - 1] + 1;
30            } else {
```

Testcase	Result
	Accepted Runtime: 0 ms



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Testcase **Result**



Accepted Runtime: 0 ms

• Case 1

• Case 2

Input

a =

"abcd"

b =

"cdabcdab"

Output

3

Expected

3

[Contribute a testcase](#)

Console ▾



Run

Submit