



Experiment 1.4

Student Name: Harshit Raj

Branch: CSE

Semester: 6th

Subject Name: CC

UID: 20BCS9266

Section/Group: 608/"A"

Date of Performance: 2/03/23

Subject Code: 20CSP-351

Aim: <https://leetcode.com/problems/word-pattern/>

WORD PATTERN

Given a pattern and a string s, find if s follows the same pattern.

Here follow means a full match, such that there is a bijection between a letter in pattern and a non-empty word in s.

Example 1:

```
Input: pattern = "abba", s = "dog cat cat dog"
Output: true
```

Example 2:

```
Input: pattern = "abba", s = "dog cat cat fish"
Output: false
```

CODE:

```
class Solution { public boolean wordPattern(String
    pattern, String s) {
        Map<Character,String> m=new HashMap<>();
        String words[]=s.split(" ");

        if(pattern.length()!=words.length) return false;

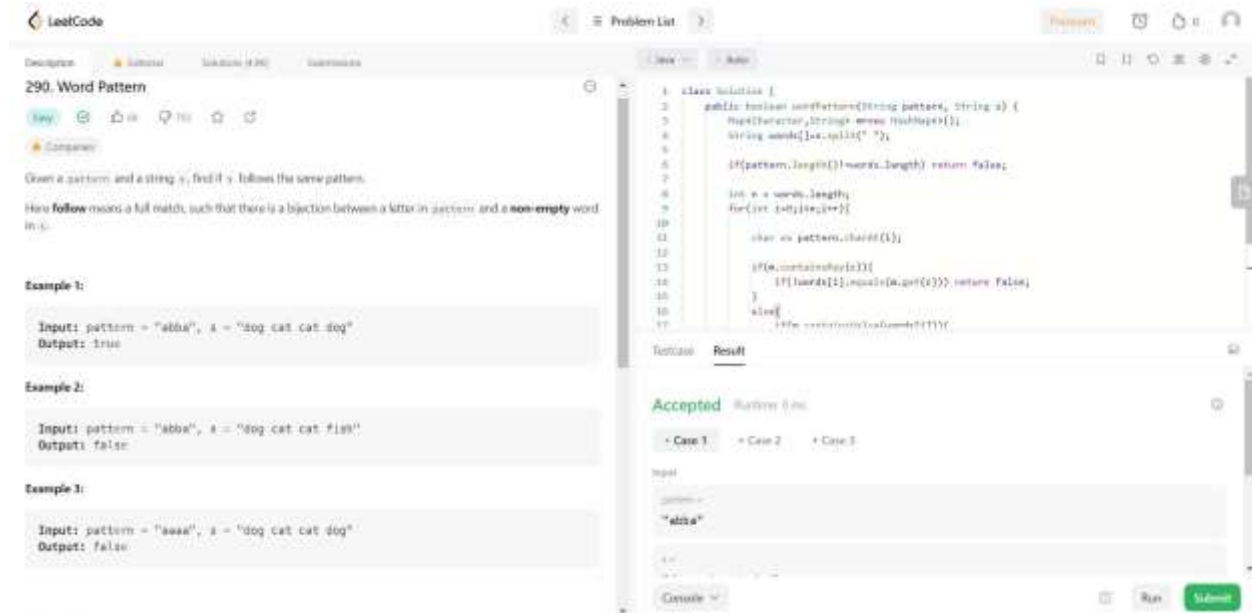
        int n = words.length;
        for(int i=0;i<n;i++){ char
            c= pattern.charAt(i);
```

```

        if(m.containsKey(c)){
            if(!words[i].equals(m.get(c))) return false;
        }
        else{
            if(m.containsValue(words[i])){
                return false;
            }
            m.put(c, words[i]);
        }
    } return true;
}
}

```

OUTPUT:



The screenshot displays the LeetCode interface for the '290. Word Pattern' problem. On the left, the problem description states: 'Given a pattern and a string s, find if s follows the same pattern. Here follow means a full match, such that there is a bijection between a letter in pattern and a non-empty word in s.' It includes three examples: Example 1 (Input: pattern = "abba", s = "dog cat cat dog", Output: true), Example 2 (Input: pattern = "abba", s = "dog cat cat fish", Output: false), and Example 3 (Input: pattern = "aaaa", s = "dog cat cat dog", Output: false). On the right, a Java solution is shown, which has been accepted. The solution uses a Map to store the mapping between pattern characters and words, and a Set to ensure each character maps to a unique word. The code is as follows:

```

1 class Solution {
2     public boolean wordPattern(String pattern, String s) {
3         Map<Character, String> memo = new HashMap<>();
4         String words[] = s.split(" ");
5
6         if(pattern.length() != words.length) return false;
7
8         int n = words.length;
9         for(int i=0; i<n; i++){
10
11             char ch = pattern.charAt(i);
12
13             if(memo.containsKey(ch)){
14                 if(words[i].equals(memo.get(ch))) return false;
15             }
16             else{
17                 memo.put(ch, words[i]);
18             }
19         }
20         return true;
21     }
22 }

```

The 'Result' section shows 'Accepted' with a runtime of 0 ms. Below, the 'Testcase' section shows 'Case 1' with input 'pattern = "abba", s = "dog cat cat dog"' and output 'true'. At the bottom, there are buttons for 'Run' and 'Submit'.



AIM: <https://leetcode.com/problems/longest-duplicate-substring/>

LONGEST DUPLICATE SUBSTRING

Given a string *s*, consider all duplicated substrings: (contiguous) substrings of *s* that occur 2 or more times. The occurrences may overlap.

Return any duplicated substring that has the longest possible length. If *s* does not have a duplicated substring, the answer is "".

Example 1:

Input: *s* = "banana"
Output: "ana"

Example 2:

Input: *s* = "abcd"
Output: ""

CODE:

```
class Solution {
    public String longestDupSubstring(String S) {
        int n = S.length();
        int[] nums = new int[n]; for
        (int i = 0; i < n; i++) {
            nums[i] = (int)S.charAt(i) - (int)'a';
        }
        int a = 26;
        long modulus = (long)Math.pow(2, 32);

        int left = 1, right = n; int L; while (left
        <= right) { L = left + (right - left) / 2; if
        (search(L, a, modulus, n, nums) == -1) {
            right = L - 1;
        } else { left =
            L + 1;
        }
    }

    int start = search(left - 1, a, modulus, n, nums);
    return S.substring(start, start + left - 1); }
    public int search(int L, int a, long modulus, int n, int[] nums) {
        long h = 0; for (int i = 0; i
        < L; i++) {
```

```

        h = (h * a + nums[i]) % modulus;
    }

    HashSet<Long> seen = new HashSet();
    seen.add(h); long aL = 1; for
    (int i = 1; i <= L; i++) {
        aL = (aL * a) % modulus;
    }

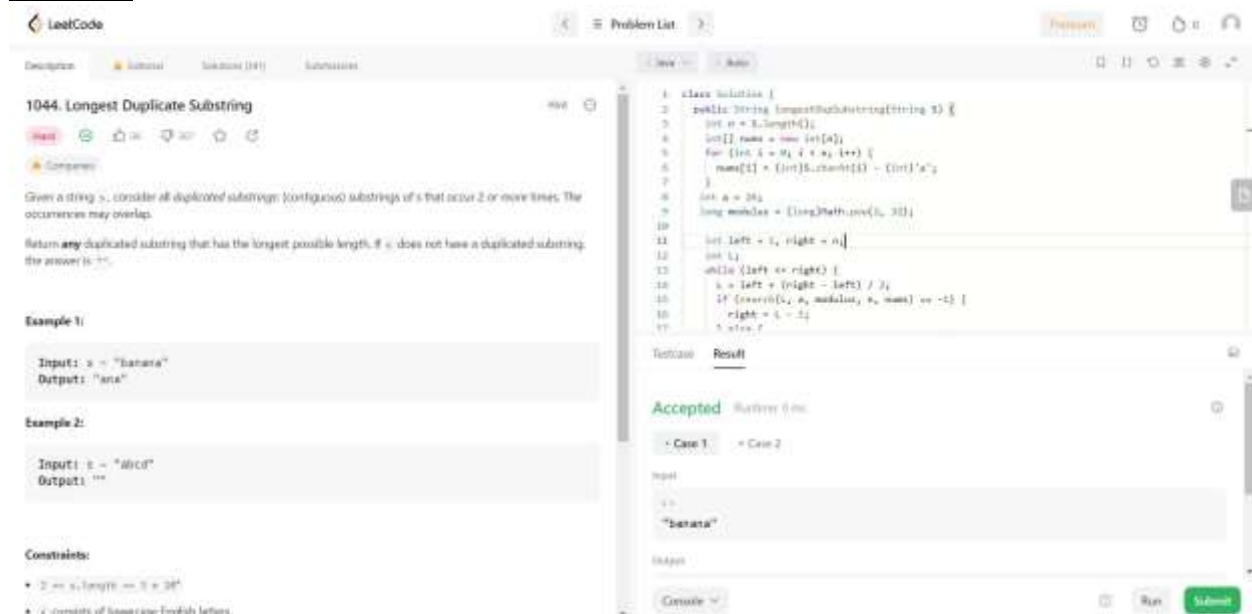
    for (int start = 1; start < n - L + 1; start++) {
        h = (h * a - nums[start - 1] * aL % modulus + modulus) % modulus;
        h = (h + nums[start + L - 1]) % modulus;

        if (seen.contains(h)) {
            return start;
        } else {
            seen.add(h);
        }
    }

    return -1;
}
}

```

OUTPUT:



The screenshot shows the LeetCode interface for problem 1044, "Longest Duplicate Substring". The problem description states: "Given a string s, consider all duplicated substrings (contiguous substrings of s that occur 2 or more times. The occurrences may overlap). Return any duplicated substring that has the longest possible length. If s does not have a duplicated substring, the answer is ""." Examples provided are: Example 1: Input: s = "banana", Output: "ana"; Example 2: Input: s = "abcd", Output: "".

The Java solution code is as follows:

```

class Solution {
    public String longestDupSubstring(String S) {
        int n = S.length();
        int[] nums = new int[n];
        for (int i = 0; i < n; i++) {
            nums[i] = (int)S.charAt(i) - (int)'a';
        }
        int a = 26;
        long modulus = (long)Math.pow(2, 32);

        int left = 1, right = n;
        int L;
        while (left <= right) {
            L = (left + right) / 2;
            if (search(L, modulus, nums) >= 1) {
                right = L - 1;
            } else {
                left = L + 1;
            }
        }
        return S.substring(0, L);
    }
}

```

The solution is marked as "Accepted" with a runtime of 0 ms. The test case shows input "banana" and output "banana".