<https://leetcode.ca/all/1062.html>

Given a string S, find out the length of the longest repeating substring(s). Return 0 if no repeating substring exists.

Example 1:

Input: "abcd"

Output: 0

Explanation: There is no repeating substring.

Example 2:

Input: "abbaba"

Output: 2

Explanation: The longest repeating substrings are "ab" and "ba", each of which occurs twice.

Example 3:

Input: "aabcaabdaab"

Output: 3

Explanation: The longest repeating substring is "aab", which occurs 3 times.

Example 4:

Input: "aaaaa"

Output: 4

Explanation: The longest repeating substring is "aaaa", which occurs twice.

Note:

* The string S consists of only lowercase English letters from 'a' - 'z'.
* 1 <= S.length <= 1500

**Attempt 1: 2023-05-26**

**Solution 1: DP (10 min)**

class Solution {

public int longestRepeatingSubstring(String s) {

int n = s.length();

int result = 0;

// Let dp[i][j] denotes in string s, up to index i,

// and up to index j < i, the number of common suffix.

// dp[i][j] := # of repeating chars of s[0..i) and s[0..j) when j < i

int[][] dp = new int[n + 1][n + 1];

for(int i = 1; i <= n; i++) {

for(int j = 1; j < i; j++) {

if(s.charAt(i - 1) == s.charAt(j - 1)) {

dp[i][j] = dp[i - 1][j - 1] + 1;

result = Math.max(result, dp[i][j]);

}

}

}

return result;

}

}

Time Complexity : O(N^2)

Space Complexity : O(N^2)

**Refer to**

<https://www.cnblogs.com/Dylan-Java-NYC/p/11986893.html>

**Let dp[i][j] denotes in string s, up to index i, and up to index j < i, the number of common suffix.**

**When j and i are pointing to same char, dp[i][j] = dp[i-1][j-1]+1.**

**Maintain the maximum.**

e.g "abbaba"

row tag with i, col tag with j, only when j < i and s[i - 1] = s[j - 1]

will do dp[i][j] = dp[i - 1][j - 1] + 1

\* a b b a b a

\* 0 0 0 0 0 0 0 i=0

a 0 0 0 0 0 0 0 i=1

b 0 0 0 0 0 0 0 i=2

b 0 0 1 0 0 0 0 i=3

a 0 1 0 0 0 0 0 i=4

b 0 0 2 1 0 0 0 i=5

a 0 1 0 0 2 0 0 i=6

j=0,1,2,3,4,5,6

**Implementation:**

class Solution {

public int longestRepeatingSubstring(String S) {

if(S == null || S.length() == 0){

return 0;

}

int n = S.length();

int res = 0;

int [][] dp = new int[n+1][n+1];

for(int i = 1; i<=n; i++){

for(int j = 1; j<i; j++){

if(S.charAt(i-1) == S.charAt(j-1)){

dp[i][j] = dp[i-1][j-1]+1;

res = Math.max(res, dp[i][j]);

}

}

}

return res;

}

}

Time Complexity: O(n^2). n = S.length().

Space: O(n^2).