**Question 4:**

Given a string, find the length of the longest repeating subsequence, such that the two subsequences don’t have the same string character at the same position, i.e. any ith character in the two subsequences shouldn’t have the same index in the original string.

**Examples:**

**Input:** str = "abc"

**Output:** 0

There is no repeating subsequence

**Input:** str = "aab"

**Output:** 1

The two subsequence are 'a'(first) and 'a'(second).

Note that 'b' cannot be considered as part of subsequence

as it would be at same index in both.

**Input:** str = "aabb"

**Output:** 2

**Input:** str = "axxxy"

**Output:** 2

ALGORITHM:

1. STEP 1: Input: a string X of length m.
2. STEP 2: Create a 2D array dp with dimensions (m+1)x(m+1) and initialize all elements to 0.
3. STEP 3:

For i = 1 to m:

For j = 1 to m:

If X[i-1] == X[j-1] and i != j:

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

Else: dp[i][j] = max(dp[i][j-1], dp[i-1][j])

1. STEP 4: Return dp[m][m] as the length of the longest repeating subsequence.
2. STEP 5: The algorithm is on dynamic programming approach. It uses a 2D array to store the results of subproblems and fills it in a bottom-up manner. The condition i != j ensures that the same character at the same index is not counted twice in the repeating subsequence.

CODE:

*def* findLongestRepeatingSubSeq(X):

    m = len(X)

*# Create and initialize DP table*

    dp = [[0 for j in range(m+1)] for i in range(m+1)]

*# Fill dp table*

    for i in range(1, m+1):

        for j in range(1, m+1):

*# If characters match and indexes are not same*

            if X[i-1]== X[j-1] and i != j:

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

*# If characters do not match*

            else:

                dp[i][j] =max(dp[i][j-1], dp[i-1][j])

    return dp[m][m]

X=input("Enter a string:\n")

print("The length of the largest subsequence that repeats itself is: ", findLongestRepeatingSubSeq(X))

OUTPUT:

