**★★★★★**

**題組：Contest Volumes 110 with Online Judge**

**題號：11081: Strings**

**解題者：楊貽婷**

**解題日期：2025年5月1日**

**題意：**

輸入三個長度60以內的字串(s1, s2, s3)，要求出有多少個組合能使前兩個字串的substring之組合組成第三個字串，最終輸出需要%10007。

**題意範例：**

2

abc abc abc 8

abbcd bccde abcde 18

**解法：**

用動態規劃計算。三維陣列dp[i][j][k]表示，分別從 s1 當中選取 前 i 個元素和從 s2 當中選取前 j 個元素，此時能組合出 s3 的前 k 個的所有可能。

現在假設我們已經選好 s3 的前 k-1 個元素，那麼第 k 個元素可以來自 s1 或 s2 字串，因此用 dp1 跟 dp2 個別對兩種情況討論，之後將兩種情況相加即為此時的 dp[i][j][k] 的解。以下為針對個別字串的詳細動態規劃方程式，其意義為針對目前字串第n個元素，該元素可分為選擇與不選兩種情況，將兩種情況的結果相加即為目前所累計的結果。

$$dp1\left[i\right]\left[j\right]\left[k\right]=dp1\left[i-1\right]\left[j\right]\left[k\right] if s1\left[i\right] !=s3[k]$$

$$dp1\left[i\right]\left[j\right]\left[k\right]=dp1\left[i-1\right]\left[j\right]\left[k\right]+dp\left[i-1\right]\left[j\right]\left[k-1\right] otherwise$$

$$dp2\left[i\right]\left[j\right]\left[k\right]=dp2\left[i\right]\left[j-1\right]\left[k\right] if s2\left[j\right] !=s3[k]$$

$$dp2\left[i\right]\left[j\right]\left[k\right]=dp2\left[i\right]\left[j-1\right]\left[k\right]+dp\left[i\right]\left[j-1\right]\left[k-1\right] otherwise$$

**解法範例：**

ab ab ab 🡺 4

先思考若 k=0背後的意義是什麼，發現當k=0，s3可以當作是空字串，代表s1跟s2可以不選任何元素，此種情況只有一種即是都不選擇，因此dp1[i][j][0]=dp2[i][j][0]=dp[i][j][0] = 1，此種即為動態規劃演算法的base case，有了base case以及動態規劃方程式，接著窮舉演算法的過程進行解釋:

k=0:

dp:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **j=0** | **j=1** | **j=2** |
| **i=0** | 1 | 1 | 1 |
| **i=1** | 1 | 1 | 1 |
| **i=2** | 1 | 1 | 1 |

k=1:

以dp1[1][1][1]為例，分為兩種情況，第一種為s1中的第一個元素不選，即有dp1[0][1][1]種可能，另一種為選擇第一個元素，有dp[0][1][0]可能，兩者相加即為dp1[1][1][1]的值。

dp1:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **j=0** | **j=1** | **j=2** |
| **i=0** | 0 | 0 | 0 |
| **i=1** | 1 | 1 | 1 |
| **i=2** | 1 | 1 | 1 |

dp2:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **j=0** | **j=1** | **j=2** |
| **i=0** | 0 | 1 | 1 |
| **i=1** | 0 | 1 | 1 |
| **i=2** | 0 | 1 | 1 |

dp:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **j=0** | **j=1** | **j=2** |
| **i=0** | 0 | 1 | 1 |
| **i=1** | 1 | 2 | 2 |
| **i=2** | 1 | 2 | 2 |

k=2:

dp1:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **j=0** | **j=1** | **j=2** |
| **i=0** | 0 | 0 | 0 |
| **i=1** | 0 | 0 | 0 |
| **i=2** | 1 | 2 | 2 |

dp2:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **j=0** | **j=1** | **j=2** |
| **i=0** | 0 | 0 | 1 |
| **i=1** | 0 | 0 | 2 |
| **i=2** | 0 | 0 | 2 |

dp:

|  |  |  |  |
| --- | --- | --- | --- |
|  | **j=0** | **j=1** | **j=2** |
| **i=0** | 0 | 0 | 1 |
| **i=1** | 0 | 0 | 2 |
| **i=2** | 1 | 2 | 4 |

演算法結束輸出dp[s1.length][s2.length][s3.length] % 10007即為結果，代表此演算法已經檢查完所有可能的s1跟s2的substring組合組出s3。

**討論：**

(1) (a + b) mod n = a mod n + b mod n

計算dp1跟dp2以及dp的過程中，實際上所儲存的數字是經過模餘運算的，這樣做的好處為，進行動態規劃演算法時，數字常常越加越大，透過此舉可以避免加到overflow導致數字異常。

(2) 演算法為 O($s\_{1}.length\*s\_{2}.length$ $s\_{3}.length$)

因為動態規劃演算法所使用的是三維陣列，至少需要跑三層迴圈，而跑迴圈的次數由輸入字串的長度決定。

(3) 可以用一個三維DP陣列做，但套公式會有重複計算的問題。

範例: addc bcc adc 🡺 6

k=0,，dp:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **j=0** | **j=1** | **j=2** | **j=3** |
| **i=0** | 1 | 1 | 1 | 1 |
| **i=1** | 1 | 1 | 1 | 1 |
| **i=2** | 1 | 1 | 1 | 1 |
| **i=3** | 1 | 1 | 1 | 1 |
| **i=4** | 1 | 1 | 1 | 1 |

k=1，dp:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **j=0** | **j=1** | **j=2** | **j=3** |
| **i=0** | 0 | 0 | 0 | 0 |
| **i=1** | dp[0][0][0]=1 | dp[0][1][0] =1 | dp[0][2][0] =1 | dp[0][3][0]=1 |
| **i=2** | dp[0][0][0]=1 | dp[0][1][0] =1 | dp[0][2][0] =1 | dp[0][3][0]=1 |
| **i=3** | dp[0][0][0]=1 | dp[0][1][0] =1 | dp[0][2][0] =1 | dp[0][3][0]=1 |
| **i=4** | dp[0][0][0]=1 | dp[0][1][0] =1 | dp[0][2][0] =1 | dp[0][3][0]=1 |

k=2, dp:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **j=0** | **j=1** | **j=2** | **j=3** |
| **i=0** | 0 | 0 | 0 | 0 |
| **i=1** | 0 | 0 | 0 | 0 |
| **i=2** | dp[1][0][1]=1 | dp[1][1][1]=1 | dp[1][2][1]=1 | dp[1][3][1]=1 |
| **i=3** | dp[1][0][1]+ dp[2][0][1] =2 | dp[1][1][1]+ dp[2][1][1]=2 | dp[1][2][1]+ dp[2][2][1]=2 | dp[1][3][1]+ dp[2][3][1]=2 |
| **i=4** | dp[1][0][1]+ dp[2][0][1] =2 | dp[1][1][1]+ dp[2][1][1]=2 | dp[1][2][1]+ dp[2][2][1]=2 | dp[1][3][1]+ dp[2][3][1]=2 |

k=3，dp:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **j=0** | **j=1** | **j=2** | **j=3** |
| **i=0** | 0 | 0 | dp[0][1][2]=0 | dp[0][1][2]+ dp[0][2][2]=0 |
| **i=1** | 0 | 0 | dp[1][1][2]=0 | dp[1][1][2]+ dp[1][2][2]=0 |
| **i=2** | 0 | 0 | dp[2][1][2]=1 | dp[2][1][2]+ dp[2][2][2]=2 |
| **i=3** | 0 | 0 | dp[3][1][2]=2 | dp[3][1][2]+ dp[3][2][2]=4 |
| **i=4** | dp[3][0][2]=2 | dp[3][1][2]=2 | dp[3][2][2]+ dp[4][1][2]=4 | dp[3][3][2]+ dp[4][1][2]+ dp[4][2][2]=6 |

觀察到如果固定 j，對i往下遍歷，會發現對不同的j值，組成結構都類似，而且符合剛剛的dp1公式。甚至如果固定 i，對j往右遍歷，會發現對不同的i值，組成結構都類似，而且符合剛剛的dp2公式。但是在i=4時，s1[i] = ‘c’，要組成s3的時候，s1[i]為可選選項，因此用紅字標記表達選擇s1[i]的情況。但是在i=4時，s1[i] = ‘c，要組成s3的時候，s1[i]為可選選項，因此用紅字標記表達選擇s1[i]的情況，另外以紫字標示 s2[j] 被選擇的情況，因此若只使用一個DP陣列會出現重複計算的情況。

**程式：**

#include <bits/stdc++.h>

using namespace std;

#define MAX 70

#define mod 10007

int res\_dp[MAX][MAX][MAX], dp1[MAX][MAX][MAX], dp2[MAX][MAX][MAX];

int main(){

 int cases;

 cin >> cases;

 for(int c=0; c<cases; c++){

 string s1, s2, s3;

 cin >> s1 >> s2 >> s3;

 // initialize

 memset(res\_dp, 0, sizeof(res\_dp));

 memset(dp1, 0, sizeof(dp1));

 memset(dp2, 0, sizeof(dp2));

 // get string length (range of dp array)

 int n1 = s1.length(), n2 = s2.length(), n3 = s3.length();

 // base case: no elements was selected -> 1

 for (int i=0; i<=n1; i++){

 for (int j=0; j<=n2; j++){

 res\_dp[i][j][0] = 1;

 dp1[i][j][0] = 1;

 dp2[i][j][0] = 1;

 }

 }

 for (int k=1; k<=n3; k++){

 for (int i=0; i<=n1; i++){

 for (int j=0; j<=n2; j++){

 if (i){

 // when i-1th+jth element satisfies the condition, store it

 dp1[i][j][k] = dp1[i-1][j][k];

 // select the ith element from k-1 substring

 dp1[i][j][k] += ((s1[i-1] == s3[k-1]) ? res\_dp[i-1][j][k-1] : 0);

 dp1[i][j][k] %= mod;

 }

 if (j){

 // when ith+j-1th element satisfies the condition, store it

 dp2[i][j][k] = dp2[i][j-1][k];

 // select the jth element from k-1 substring

 dp2[i][j][k] += ((s2[j-1] == s3[k-1]) ? res\_dp[i][j-1][k-1] : 0);

 dp2[i][j][k] %= mod;

 }

 // plus both dp1 and dp2

 res\_dp[i][j][k] = (dp1[i][j][k] + dp2[i][j][k]) % mod;

 }

 }

 }

 // output the result

 cout << res\_dp[n1][n2][n3] << endl;

 }

 return 0;

}