**★★★☆☆**

**題組：Problem Set Archive** **with Online Judge**

**題號：10069: Distinct Subsequences**

**解題者：鄔曜陽**

**解題日期：2023年4月20日**

**題意：**

本題輸入會給多組case，每一組case有兩個字串X、Z。需要在母字串X中找出字串Z出現的最大次數。

**題意範例：**

X = “babgbag”, Z = “bag”

**babgbag babgbag babgbag babgbag babgbag**

output:5

X = “rabbbit”, Z = “rabbit”

**rabbbit rabbbit rabbbit**

output : 3

**解法：**

用dp[i][j]代表字串Z的前j+1個字元(0~j)在字串X的前i+1個字元(0~i)中出現過幾次，可以簡單歸納成兩種情況：

 1. X[i] != Z[j] ⇒ dp[i][j] = dp[i-1][j]

 2. X[i] == Z[j] ⇒ dp[i][j] = dp[i-1][j] + dp[i-1][j-1]

* **X[i] == Z[j]**

例1：X = “babgbag” , Z = “bag”

babgba**g**  ba**g** ⇒ **X[i] == Z[j]**

* 1. 我們選用X的最後一位的g，作為匹配Z的字元。這時由於g已經被選中，X只需要再從babgba找出Z剩下的字元: ba出現的次數，也就是dp[i-1][j-1]，就能計算出情形1的可能次數。
	2. 我們不選用X的最後一位的g，作為匹配Z的字元。這時需要找出babgba中bag出現的次數，也就是dp[i-1][j]。

結合以上兩種情形(選中X的最後一位與不選X的最後一位)，結果相加後，就是Z在X中出現的次數 ⇒ dp[i][j] = dp[i-1][j] + dp[i-1][j-1]。

* **X[i] != Z[j]**

例2：X = “babgba” , Z = “bag”

babgb**a**  ba**g** ⇒ **X[i] != Z[j]**

因最後一位不相同，Z的組成不可能使用到X的最後一位，所以X的最後一位與對Z沒有任何影響。這時可以去除X的最後一位，也就是

dp[i][j] = dp[i-1][j]。

得出以上的兩行公式後，就可以使用遞迴來運算。

**解法範例：**

無

**討論：**

1. 本題使用遞迴來運算，需要使用動態規劃，將計算過的答案儲存在陣列dp中，以減少重複計算。
2. output有可能大於2200，不能用int，需要使用字串實作。

**程式：**

#include <iostream>

#include <string>

#include <cstring>

#include <algorithm>

#include <cmath>

using namespace std;

string add(string, string);

string cal(int , int , bool\*\*, string\*\*, string, string);

int main()

{

 string \*\*dp = new string\* [10005]; //dp[i][j] means how many times Z[0~j] appears in X[0~i]

 for(int i=0;i<10005;i++)

 dp[i] = new string[105];

 bool \*\*solve = new bool\* [10005]; //record completed dp element

 for(int i=0;i<10005;i++)

 solve[i] = new bool[105];

 for(int i=0;i<10005;i++)

 for(int j=0;j<105;j++)

 solve[i][j] = false;

 int testcase;

 string s1, s2;

 cin >> testcase;

 for(int n=0; n<testcase; n++)

 {

 cin >> s1;

 cin >> s2;

 string ans = cal(s1.length()-1, s2.length()-1, solve, dp, s1, s2);

 cout << ans << endl;

 for(int i=0;i<10005;i++)

 for(int j=0;j<105;j++)

 solve[i][j] = false;

 for(int i=0;i<10005;i++)

 for(int j=0;j<105;j++)

 dp[i][j] = "";

 }

 return 0;

}

string add(string a, string b) //add two strings that store numbers

{

 bool carryout = 0;

 if(a.length() < b.length())

 {

 string temp = a; a = b; b = temp;

 }

 string c = a;

 int len1 = a.length();

 int len2 = b.length();

 for(int i=0; i<len2 ; i++)

 {

 int ch1 = (int)a[len1-1-i] - 48;

 int ch2 = (int)b[len2-1-i] - 48;

 int tmp1 = ch1 + ch2 + (int)carryout;

 carryout = tmp1 / 10;

 c[len1-1-i] = (char)(tmp1 - (int)carryout \* 10 + 48);

 }

 for(int i=len1-len2-1; i>=0; i--)

 {

 int tmp2 = (int)c[i]-48 + carryout;

 carryout = tmp2/10;

 c[i] = (char)(tmp2 - (int)carryout \* 10 + 48);

 }

 if(carryout==1)

 c = "1" + c;

 return c;

}

string cal(int m, int n, bool \*\*solve, string \*\*dp, string s1, string s2) //recursive

{

 string number;

 if(m<0||n<0)

 return 0;

 if(solve[m][n]==true)

 return dp[m][n];

 if(n==0) //Z only has 1 character. Calculate how many z are in X[0]~X[i]

 {

 number = "0";

 for(int i=0;i<=m;i++)

 {

 if(s2[0] == s1[i])

 number = add(number, "1");

 }

 return number;

 }

 if(m == n) //same length. return 0(same) or 1(different)

 {

 for(int i=0;i<=m;i++)

 {

 if(s1[i]!=s2[i])

 {

 dp[m][n] = "0";

 solve[m][n] = true;

 return "0";

 }

 }

 dp[m][n] = "1";

 solve[m][n] = true;

 return "1";

 }

 else if(s1[m] != s2[n]) //dp[i][j] = dp[i-1][j]

 number = cal(m-1, n, solve, dp, s1, s2);

 else if(s1[m] == s2[n]) //dp[i][j] = dp[i-1][j] + dp[i-1][j-1]

 {

 string cal1 = cal(m-1, n, solve, dp, s1, s2);

 string cal2 = cal(m-1, n-1, solve, dp, s1, s2);

 number = add(cal1, cal2);

 }

 dp[m][n] = number; //store in dp

 solve[m][n] = true; //record completed dp

 return number;

}