南京工業大學
NANJING TECH UNIVERSITY

ICPC Template Manual

作者: 贺梦杰

September 24, 2019

Contents

0.1 最长公共子序列

给定 S1 和 S2 串，要求一个序列，同时是 S1 和 S2 的子序列。

0.1.1 CCPC 2019 秦皇岛 C.Sakura Reset

0.1.1.1 题目描述

给定 A、B 串及其长度 n、m($1 \le n, m \le 5000$)，要求 A 的子序列 a 和 B 的子序列 b，使得 a 的长度大于 b 的长度或 a、b 长度相等且字符串 a 大于字符串 b。

0.1.1.2 解决方案

整体上，要分类讨论：1.a 的长度大于 b 的长度；2. 长度相等且字符串 a 大于字符串 b

对于分类 1：
即要分别考虑 A 和 B 的各个长度的本质不同的子序列个数。答案就是枚举 A 的每个长度，其个数乘上 B 的小于该长度的自序列个数。
如何统计呢？首先要理解贪心匹配，即如何判断 t 是否是 s 的子序列？根据贪心匹配，对于元素 t_1 我们就是要找它在 s 中出现的**第一个**位置 p_1 （$t_1 = s_{p_1}$），接下来在 s 的 p_1 后面找 t_2 并令其为 p_2，以此类推。
根据上面的规则，对于一个串 X，我们令 $f(i)$ 表示以 $X[i]$ 为结尾的**与之前的不重复**的且**本质不同**的子序列个数。则转移方程

$$f(i) = \sum_{j=last[i]}^{i-1} f(j)$$

其中 $last[i]$ 表示 $X[i]$ 上一次出现的位置，若第一次出现 $last[i] = 0$。
为什么是从 $last[i]$ 开始累加呢？因为以下标小于 $last[i]$ 的元素为结尾的子序列后面再加上当前的 $X[i]$ 得出的新的子序列，和之前加上 $last[i]$ 处的 $X[last[i]]$ 组成的子序列重复了。通过**一维前缀和**可以优化至 $O(n)$。回到现在的问题，我们要求每个长度上的本质不同的子序列个数，只要加上一维长度维度即可 ($f(i,j)$)。时间 $O(n^2)$。

对于分类 2：
要求长度相等，且按字符串的方式比较 a 大于 b。即要求 a 和 b 有一段长度大于等于 0 的相同前缀，在第一个不相等的位置 a 的元素大于 b 的元素，后面只要有长度相等的后缀即可。
令 $g(i,j)$ 表示以 $A[i]$ 结尾的 a 和以 $B[j]$ 结尾的 b **与之前的不重复**的且**本质不同**的公共子序列对数。
若 $A[i] \ne b[i]$，显然 $g(i,j) = 0$；若 $A[i] = B[j]$，

$$g(i,j) = \sum_{u=last[i]}^{i-1} \sum_{v=last[j]}^{j-1} g(u,v)$$

再考虑后缀，令 $h(i,j)$ 为从后向前以 $A[i]$ 结尾的 a 和以 $B[j]$ 结尾的 b **与之前的不重复**的且**本质不同**的长度相等子序列对数。后缀的求法有两种，可以和求 f 向类似，但是在算答案的时候回非常复杂，也可以和求 g 类似，只不过是倒着求，并且不要求 $A[i] = B[j]$。最后枚举 i、j，若 $A[i] > B[j]$，则

$$ans+ = (\sum_{u=1}^{i-1} \sum_{v=1}^{j-1} g(u,v)) * h(i,j)$$

以上都可以通过**二维前缀和或二维后缀和**优化至 $O(N^2)$。

注意：上面两个前缀和的意义是不一样的。f 的前缀和是一维的，它只对下标 i 求前缀和，不对长度 j 求；而下面 g 的前缀和以及 h 的后缀和是二维的，即 i,j 都要求。所以还要注意两种在边界上的初始化是不一样的。

0.1.1.3 代码

```cpp
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;
const ll N = 5e3 + 10, P = 998244353;

ll len[2];          // 两串的长度
ll S[2][N];         // A串和B串
```

```
ll f[2][N][N];      // A串和B串的dp, f[0][i][j]:以A[i]为结尾且长度为j的本质不同的子序列个数
ll fpre[2][N][N];  // 对f的**i**求前缀和, 并不是二维前缀和
ll last[2][N];      // A串和B串, A[i]或B[i]上一次出现的位置
ll pos[2][110];     // pos[i], 值为i的元素上一次出现的位置
ll g[N][N];         // 二维矩阵, g[i][j]表示以A[i]结尾的子序列和以B[i]结尾的子序列为公共子序列的方案数
ll gpre[N][N];      // g的二维前缀和
ll h[N][N];         // 二维矩阵, h[i][j]表示 从后向前 以A[i]结尾的子序列和以B[i]结尾的子序列长度相等但本质不同的后缀对个数
ll hsuf[N][N];      // h的二维后缀和

inline bool isOk(ll r, ll c) {
    return r >= 0 && c >= 0;
}

inline ll presum(ll r1, ll c1, ll r2, ll c2) {
    ll v1 = 0, v2 = 0, v3 = 0, v4 = 0;
    if (isOk(r2, c2))
        v1 = gpre[r2][c2];
    if (isOk(r1 - 1, c2))
        v2 = gpre[r1 - 1][c2];
    if (isOk(r2, c1 - 1))
        v3 = gpre[r2][c1 - 1];
    if (isOk(r1 - 1, c1 - 1))
        v4 = gpre[r1 - 1][c1 - 1];
    return v1 - v2 - v3 + v4;
}

inline ll sufsum(ll r1, ll c1, ll r2, ll c2) {
    return hsuf[r2][c2] - hsuf[r1 + 1][c2] - hsuf[r2][c1 + 1] + hsuf[r1 + 1][c1 + 1];
}

int main() {
    ios::sync_with_stdio(0);
    cin.tie(0);

    ll i, j, k, ans = 0;

    cin >> len[0] >> len[1];

    for (k = 0; k <= 1; k++)
        for (i = 1; i <= len[k]; i++)
            cin >> S[k][i];

    // 预处理last
    for (k = 0; k <= 1; k++)
        for (i = 1; i <= len[k]; i++)
            last[k][i] = pos[k][S[k][i]],
            pos[k][S[k][i]] = i;

    // 对于长度不同的情况------------------------------------------
    // 首先求f和fpre
    f[0][0][0] = f[1][0][0] = fpre[0][0][0] = fpre[1][0][0] = 1;
    for (k = 0; k <= 1; k++) {
        // 初始化边界
        for (i = 1; i <= len[k]; i++)
            fpre[k][i][0] = 1;
        for (i = 1; i <= len[k]; i++)
            for (j = 1; j <= i; j++)
                f[k][i][j] = (fpre[k][i - 1][j - 1] - fpre[k][last[k][i]][j - 1] + f[k][last[k][i]][j - 1]) % P,
                fpre[k][i][j] = (fpre[k][i - 1][j] + f[k][i][j]) % P;
    }
    // 计算长度不同的答案
    ll t = 0;
    for (i = 1; i <= len[0]; i++)
        ans = (ans + fpre[0][len[0]][i] * t % P) % P,
```

```
        t += fpre[1][len[1]][i];

    // 对于长度相同的情况------------------------------------------
    // 首先求g和gpre
    g[0][0] = gpre[0][0] = 1;
    // 初始化边界
    for (i = 1; i <= len[0]; i++)
        gpre[i][0] = 1;
    for (i = 1; i <= len[1]; i++)
        gpre[0][i] = 1;
    for (i = 1; i <= len[0]; i++)
        for (j = 1; j <= len[1]; j++)
            g[i][j] = (S[0][i] == S[1][j] ? presum(last[0][i], last[1][j], i - 1, j - 1) : 0),
            gpre[i][j] = gpre[i - 1][j] + gpre[i][j - 1] - gpre[i - 1][j - 1] + g[i][j];
    // 再求从后向前以i为结尾的长度为j的本质不同的子序列个数，求法类似之前求的f，只是倒过来了
    // 在此之前，先预处理last
    for (k = 0; k <= 1; k++)
        for (i = 1; i <= 100; i++)
            pos[k][i] = len[k] + 1;
    for (k = 0; k <= 1; k++)
        for (i = len[k]; i >= 1; i--)
            last[k][i] = pos[k][S[k][i]],
            pos[k][S[k][i]] = i;
    // 开始求
    hsuf[len[0] + 1][len[1] + 1] = h[len[0] + 1][len[1] + 1] = 1;
    // 初始化边界
    for (i = 1; i <= len[1]; i++)
        hsuf[len[0] + 1][i] = 1;
    for (i = 1; i <= len[0]; i++)
        hsuf[i][len[1] + 1] = 1;
    for (i = len[0]; i >= 1; i--)
        for (j = len[1]; j >= 1; j--)
            h[i][j] = sufsum(last[0][i], last[1][j], i + 1, j + 1),
            hsuf[i][j] = hsuf[i + 1][j] + hsuf[i][j + 1] - hsuf[i + 1][j + 1] + h[i][j];
    // 计算长度相等时的答案
    for (i = 1; i <= len[0]; i++)
        for (j = 1; j <= len[1]; j++)
            if (S[0][i] > S[1][j])
                ans = (ans + gpre[i - 1][j - 1] * h[i][j]) % P;

    cout << ans << endl;

    return 0;
}
```