滚动数组优化

主讲人：邓哲也

滚动数组优化

比如有一个两维的 dp 数组 f[i][j]。

其中 f[i][j] 是从 f[i - 1][k] 转移过来的。

并且在算 f[i][*] 之前 f[i-1][*] 已经全部算好了。

那么这个时候就可以把第一维的空间优化掉。

因为 f[i-1][*] 更新完 f[i][*] 后就没有用了。

滚动数组优化

可以用两个数组 f 和 g

```
for (int i = 1;i <= n;i ++){
	memcpy(g, f, sizeof(f)); // 把上一个状态的 f 数组存进 g
	memset(f, 0, sizeof(f)); // 初始化 f
	// 用 g 来更新 f
}
```

滚动数组优化

可以用二维数组 f[2][N];

```
for (int i = 1;i <= n;i ++){
    int p = i & 1;
    memset(f[p], 0, sizeof(f[p])); // 初始化 f[p]
    // 用 f[p^1] 来更新 f[p]
}
```

NOIP 2015 Day2 子串

有两个仅包含小写英文字母的字符串 A 和 B。

现在要从字符串 A 中取出 k 个互不重叠的非空子串，然后把这 k 个子串按照其在字符串 A 中出现的顺序依次连接起来得到一个新的字符串。请问有多少种方案可以使得这个新串与字符串 B 相等？

注意：子串取出的位置不同也认为是不同的方案。

$1 \leqslant |A| \leqslant 1000$， $1 \leqslant k \leqslant |B| \leqslant 200$

样例输入：（答案：7）

6 3 2

aabaab

aab

NOIP 2015 Day2 子串

设计状态：

f[i][j][k][p] 表示从 A[1..i] 中选出了 k 个子串，拼出了 B[1..j]。p ∈ {0, 1}，1 表示 A[i] 在第 k 个子串中，0 表示 A[i] 不在第 k 个子串中。

那么我们就看 A[i] 和 B[j + 1] 是否相等，讨论转移。

NOIP 2015 Day2 子串

如果 A[i] = B[j + 1]

那么可以开启一个新子串，变成 k + 1 个子串。

f[i][j + 1][k + 1][1] += f[i - 1][j][k][0]

f[i][j + 1][k + 1][1] += f[i - 1][j][k][1]

也可以接在上一个子串后面，仍然是 k 个子串。

f[i][j + 1][k][1] += f[i - 1][j][k][1]

NOIP 2015 Day2 子串

如果 $A[i] \neq B[j + 1]$

那么这个时候不能开启新的子串，也不能把 A[i] 接在上一个子串后面。

$f[i][j][k][0] \mathrel{+}= f[i - 1][j][k][0]$

$f[i][j][k][0] \mathrel{+}= f[i - 1][j][k][1]$

NOIP 2015 Day2 子串

可以发现状态数是 O(NMK) 的，每次转移 O(1)。

因此时间复杂度是 O(NMK)

空间上第一维可以用滚动数组优化掉，因为我们是先算出

f[i - 1] 的所有值再去更新 f[i] 的。

因此空间复杂度是 O(MK)

NOIP 2015 Day2 子串

```
int f[2][201][201][2];

f[0][0][0][0] = 1;
for(int i = 1;i <= n;i ++){
    int p = i & 1;
    memset(f[p], 0, sizeof(f[p]));
    for(int j = 0;j <= m;j ++){
        for(int l = 0;l <= k;l ++){
            if (f[p^1][j][l][0] || f[p^1][j][l][1]){
                if(A[i] == B[j+1]){
                    add(f[p][j + 1][l][1], f[p^1][j][l][1]);
                    add(f[p][j + 1][l + 1][1], f[p^1][j][l][0]);
                    add(f[p][j + 1][l + 1][1], f[p^1][j][l][1]);
                }
```

NOIP 2015 Day2 子串

```
                add(f[p][j][1][0], f[p^1][j][1][1]);
                add(f[p][j][1][0], f[p^1][j][1][0]);
            }
        }
    }
}
printf("%d\n", (f[n&1][m][k][0] + f[n&1][m][k][1]) % mo);
```

下节课再见