

# Assignment Two Programming 说明文档

## 1 分析

### • 1.1 问题重述

一串 $N$ 位密码，每一位都可以在一个0-9九个数字组成的循环圈中转动，也可理解为每一位数字进行模10加减法，每一步可对连续的1至3位以一个方向（向上或向下）转动一格。

例如：

567890 -> 567901 （向上转动最后三位）  
000000 -> 000900 （向下转动第四位）

输入：两个同等长度的字符串，分别代表初始状态和目标状态（密码）

输出：从初始状态到目标状态所需的最少步数

### • 1.2 参数分析

密码的长度 $N$ 限制在1000以下，即  $N \leq 1000$ 。

## 2 算法设计

### • 2.1 算法思路

假设 $A=\{a_1, a_2, a_3, \dots\}$ 是一最优的转动步骤，则显然更改 $A$ 中序列的顺序不影响 $A$ 的最优性。要获取最优解 $A$ ，可以先进行 $A$ 中靠前位置的转动，当靠前位置处理完成后，后续转动对其无影响，因此可以不用考虑。

通过这样的无序有序化，先将  $len$  位长的初始串 $s_1$ 中的第一位调整到目标数字（密码 $s_2$ 对应位的数字），再考虑除去第一位之外剩下的  $len - 1$  位长的新串 $s_1'$ ，接着将 $s_1'$ 的第一位（即 $s_1$ 的第二位）调整到目标数字，得到  $len - 2$  位长的 $s_1'' \dots$ 以此类推，经过  $len$  次调整，可将初始状态 $s_1$ 转化成目标状态 $s_2$ 。

$dp[i][j][k]$  表示：前  $i$  位已经完全匹配、第  $i + 1$  位已加  $j$ 、第  $i + 2$  位已加  $k$ （减去  $x$  等价于加上  $10-x$ ）时所用的最少操作步数。

初始： $dp[0][0][0] = 0$ ，即没有任何旋转。

终止： $dp[len][0][0]$ ，即所有位都被旋转到目标数字的所用最小步数。（ $len$ 表示密码长度）

$dp[i]$  意味着前  $i$  位已经调整好，需要考虑调整第  $i + 1$  位的情况，那么怎么由  $dp[i]$  转移到  $dp[i + 1]$  呢？转移的方向分为两种，即第  $i + 1$  位有向上和向下两个方向去到达目标数字，分别讨论这两种情况，接着再枚举第  $i + 2$  位和第  $i + 3$  位在对应方向下允许的转动量，然后对  $dp[i + 1]$  进行更新。

需要注意的是，在每一次转动中有下列三种情况：

1. 第  $i + 1$  位单独改变
2. 第  $i + 1, i + 2$  位同时改变
3. 第  $i + 1, i + 2, i + 3$  位同时改变

因此对于每一次调整（若干次转动），如果第  $i + 1, i + 2, i + 3$  位分别变化了  $a, b, c$ ，则必定有  $|c| \leq |b| \leq |a|$ 。

## • 2.2 实现

```
#include <iostream>
using namespace std;

const int MAX_N = 1010;
const int inf = 100000000;

int main()
{
    int dp[MAX_N][10][10]; // 前i个已经完全匹配、第i+1个已经加了j、第i+2位已经加了
    k（减去 x 等价于加上 10-x）时所用的最少次数
    char s1[MAX_N], s2[MAX_N];
    cin >> s1 >> s2;
    int len = strlen(s1);
    int up, dw; // 第 i+1 位需要往上加、减的步数

    // 初始化
    for (int i = 0; i <= len; i++) {
        for (int j = 0; j < 10; j++) {
            for (int k = 0; k < 10; k++) {
                dp[i][j][k] = inf;
            }
        }
    }
    dp[0][0][0] = 0; // 最初时（0个匹配）步数为0

    for (int i = 0; i < len; i++) {
        for (int j = 0; j < 10; j++) {
            up = (s2[i] - s1[i] - j + 20) % 10;
            dw = (10 - up) % 10;
            // 第 i+1 位调整好后，枚举第 i+2、i+3 位可以加或减的值
            for (int k = 0; k < 10; k++) {
                for (int x1 = 0; x1 <= up; x1++) { // 枚举第 i+2 位可以加
                    的值 - x1
                    int t = (k + x1) % 10; // 调整第 i+2 位已加的值
                    for (int y1 = 0; y1 <= x1; y1++) { // 枚举第 i+3 位可以加
                        的值 - y1
                        dp[i + 1][t][y1] = min(dp[i + 1][t][y1], dp[i][j][k]
                        + up);
                    }
                }
                for (int x2 = 0; x2 <= dw; x2++) { // 枚举第 i+2 位可以减
                    的值 - x2
                    int t = (k - x2 + 10) % 10; // 调整第 i+2 位已加的值
```

```

        for (int y2 = 0; y2 <= x2; y2++) { // 枚举第 i+3 位可以减
            的值 - y2
            dp[i + 1][t][(10 - y2) % 10] = min(dp[i + 1][t][(10
            - y2) % 10], dp[i][j][k] + dw);
        }
    }
}

cout << dp[len][0][0] << endl;

return 0;
}

```

## • 2.3 复杂度分析

时间复杂度	(额外) 空间复杂度
$O(10^4 \times N)$	$O(1)$

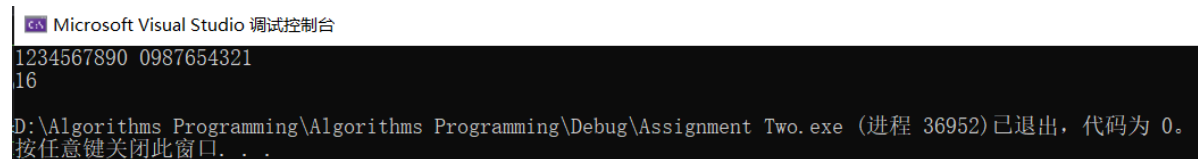
## 3 测试

### case 1:

Input : 1234567890 0987654321

Output : 16

screenshots of running results :



```

Microsoft Visual Studio 调试控制台
1234567890 0987654321
16
D:\Algorithms Programming\Algorithms Programming\Debug\Assignment Two.exe (进程 36952) 已退出，代码为 0。
按任意键关闭此窗口。 . . .

```

### case 2:

Input : 15102342 15123094

Output : 8

screenshots of running results :

15102342 15123094

8

D:\Algorithms Programming\Algorithms Programming\Debug\Assignment Two.exe (进程 46916) 已退出，代码为 0。  
按任意键关闭此窗口. . .