NLP 第一次编程作业

题目 1. (计算最小编辑距离) 实现英文最小字符串的编辑距离，计算最小编辑路径并显示结果.

注：最小编辑距离定义为：假设 s1 和 s2 是两个给定的字符串，考虑对 s1 字符串进行 n 次**添加字符**、**删除字符**、**替换字符**三种操作，将 s1 变化为 s2，求 n 的最小值.
例子：s1 = aadfagha， s2 = abcdefgh，则可通过以下 5 次变换操作将 s1 变为 s2：

1) 替换：a**b**dfagha
2) 添加：ab**c**dfagha
3) 添加：abcd**e**fagha
4) 删除：abcdefgha
5) 删除：abcdefgh

解答. 该问题为经典动态规划 (dp) 问题，通过设定状态以及状态转移方程即可求解，先进行以下定义：

- 设 $s1, s2$ 为字符串，l_i 表示字符串 s_i 的长度 $(i = 1, 2)$，$s1[i]$ 表示字符串 $s1$ 中的第 i 个下标对应的字符.
- $s1[i \cdots j]$ 表示截取字符串 $s1$ 下标从 i 到 j 的子串.
- $s1$ 与 $s2$ 完全匹配，记为 $s1 = s2$，当且仅当，$s1$ 与 $s2$ 长度相同，且对任意下标 i 都有 $s1[i] == s2[i]$.

设数组 $dp[i][j]$ 为动态规划数组,赋予其中元素以下含义:$dp[i][j]$ 表示将子串 $s1[1 \cdots i]$ 与子串 $s2[1 \cdots j]$ 完全匹配所需的最小编辑次数，根据字符串 $s1[i]$ 与 $s2[j]$ 是否相同可分为以下两种情况：

1. 若 $s1[i] = s2[j]$，则说明当前字符串末端字符相同，无需编辑修改，于是最小编辑次数直接从 $dp[i-1][j-1]$ 转移得到.
2. 若 $s1[i] \neq s2[j]$，则说明当前字符串末端字符不同，需要通过以上三种修改方式中的一种，使得 $s1[1 \cdots i] = s2[1 \cdots j]$，且编辑次数最小. 于是可从三处得到转移：
   (a) $dp[i-1][j-1]+1$：表示将 $s1[i]$ 替换为 $s2[j]$，则只需保证 $s1[1 \cdots i-1] = s2[1 \cdots j-1]$ 即可，而使得 $s1[1 \cdots i-1] = s2[1 \cdots j-1]$ 相同所需的最小步数即为 $dp[i-1][j-1]$，再加上当前所需的一步，则是转移方程结果 $dp[i-1][j-1]+1$. （剩余两种推导类似）
   (b) $dp[i-1][j]+1$：表示将 $s1[i]$ 删除，则只需保证 $s1[1 \cdots i-1] = s2[1 \cdots j]$.
   (c) $dp[i][j-1]+1$：表示将 $s1[i]$ 后面添加字符 $s2[j]$，则只需保证 $s1[1 \cdots i] = s2[1 \cdots j-1]$.

综上，得到转移方程为

$$dp[i][j] = \begin{cases} 0, & s1[i] = s2[j], \\ \min\{dp[i-1][i-1], dp[i-1][j], dp[i][j-1]\} + 1, & s1[i] \neq s2[j]. \end{cases}$$

动态规划数组初始条件为 $dp[i][0] = dp[0][i] = i,\ (i = 1, 2, \cdots, \max\{l_1, l_2\})$，其他值全部初始化为最大值 INF，其中 l_i 表示字符串 s_i 的长度.

这样初始化原因是：$dp[i][0]$ 表示其中一个串的长度为 0，为使得两个串相同，必定需要删除或添加 i 个字符；其他值初始化为最大值，是因为动态规划是求解最小化问题.

总时间复杂度：$\mathcal{O}(l_1 l_2)$.

动态规划部分代码：Luogu 题目：P2758 编辑距离

```cpp
#include <iostream>
#include <cstring>
using namespace std;
const int N = 1e4 + 10;  // 字符串最大长度
char s1[N], s2[N];
int dp[N][N];  // dp 为动态规划数组
signed main() {
    memset(dp, 127, sizeof(dp));  // 初始化 dp 数组为极大值
    cin >> s1 >> s2;  // 字符串读入
    int len1 = strlen(s1), len2 = strlen(s2);
    for (int i = 0; i <= max(len1, len2); i++) dp[0][i] = dp[i][0] = i;  // 初始化边界条件
    for (int i = 1; i <= len1; i++) {
        for (int j = 1; j <= len2; j++) {
            if (s1[i-1] == s2[j-1]) {
                dp[i][j] = dp[i-1][j-1];
                continue;
            }
            dp[i][j] = min(dp[i-1][j-1], min(dp[i-1][j], dp[i][j-1])) + 1;
        }
    }
    printf("%d\n", dp[len1][len2]);
    system("pause");
    return 0;
}
```

由于题目还要求输出编辑路径，也就是动态规划每次转移路径，可通过记录每次 $dp[i][j]$ 从哪个位置转移得到的，再从最后一个节点 $dp[l_1][l_2]$ 递归回去，即可得到最优路径上的每次操作，但是这样结果是倒置的，所以需要在回溯过程中进行输出即可. 具体实现请见代码（结果中 del 表示删除字符，add 表示添加字符，chg 表示替换字符）:

```cpp
#include <iostream>
#include <cstring>
using namespace std;
const int N = 1e4 + 10;  // 字符串最大长度
```

```cpp
char s1[N], s2[N];
int dp[N][N], opt[N][N];  // dp 为动态规划数组, opt 记录每次操作值
// 0: none, 1: delete, 2: add, 3: change
string optName[] = {"none", "del", "add", "chg"}, nowString;
int delta = 0;
void dfs(int i, int j, int o) {
    if (opt[i][j] == 0) dfs(i-1, j-1, 0);
    else if (opt[i][j] == 1) dfs(i-1, j, 1);
    else if (opt[i][j] == 2) dfs(i, j-1, 2);
    else if (opt[i][j] == 3) dfs(i-1, j-1, 3);
    if (o != 0) {
        if (o == 1) nowString.erase(i+delta, 1), delta--;
        else if (o == 2) nowString.insert(i+delta, 1, s2[j]), delta++;
        else nowString[i+delta] = s2[j];
        cout << optName[o] << ": " << nowString << '\n';
    }
}
signed main() {
    memset(dp, 127, sizeof(dp));  // 初始化 dp 数组为极大值
    memset(opt, -1, sizeof(opt));  // 初始化 opt 数组为-1
    cin >> s1 >> s2;
    int len1 = strlen(s1), len2 = strlen(s2);
    for (int i = 0; i <= max(len1, len2); i++) dp[0][i] = dp[i][0] = i;
    for (int i = 1; i <= len1; i++) {
        for (int j = 1; j <= len2; j++) {
            if (s1[i-1] == s2[j-1]) {
                opt[i][j] = 0;
                dp[i][j] = dp[i-1][j-1];
                continue;
            }
            int tmp[] = {dp[i-1][j], dp[i][j-1], dp[i-1][j-1]}, mn = 1e9,
↪   mnId;  // mn 记录当前最小值, mnId 记录从哪个位置转移得到的
            for (int k = 0; k < 3; k++) {
                if (tmp[k] < mn) {
                    mn = tmp[k];
                    mnId = k;
                }
            }
            dp[i][j] = mn + 1;
            opt[i][j] = mnId + 1;
        }
    }
    printf("minimal step: %d\n", dp[len1][len2]);
```

```
    //---------- 以上部分为动态规划部分 ----------
    nowString = string(s1);
    cout << "s1:  " << nowString << '\n';
    dfs(len1, len2, 0);
    system("pause");
    return 0;
}

#if 0
Input:
aadfagha
abcdefgh

Output:
minimal step: 5
s1:  aadfagha
chg: abdfagha
add: abcdfagha
add: abcdefagha
del: abcdefgha
del: abcdefgh
#endif
```

题目 2. (修改操作代价改变最小编辑距离) 记删除，添加，替换的权重分别为 w_1, w_2, w_3，于是只需要将初始化部分和 dp 数组转移处的权重进行调整即可：

- 初始化部分：$dp[i][0] = w_1 \cdot i$ 表示需要删除掉 $s1$ 中 i 个字符，则权重为 $w_1 \cdot i$，类似的，$dp[0][j] = w_2 \cdot j$，表示需要添加 j 个字符.
- 转移方程

$$dp[i][j] = \begin{cases} 0, & s1[i] = s2[j], \\ \min \begin{cases} dp[i-1][j] + w_1, \\ dp[i][j-1] + w_2, \\ dp[i-1][j-1] + w_3 \end{cases}, & s1[i] \neq s2[j]. \end{cases}$$

仅在上一题代码上稍加修改即可，下面仅列出修改的重要部分.

```
...
int w[3];  // w[3] 分别对应删除, 添加, 替换的权重
...
signed main() {
...
    cin >> w[0] >> w[1] >> w[2];  // 读入权重
```

```
...
    for (int i = 0; i <= max(len1, len2); i++) dp[0][i] = i * w[1],
↪   dp[i][0] = i * w[0];  // 带权重的初始化
    for (int i = 1; i <= len1; i++) {
        for (int j = 1; j <= len2; j++) {
            ...
            for (int k = 0; k < 3; k++) {
                if (tmp[k] < mn) {
                    mn = tmp[k] + w[k];  // 对第 k 个操作加上权重
                    mnId = k;
                }
            }
            dp[i][j] = mn;
            opt[i][j] = mnId + 1;
        }
    }
    printf("minimal weight: %d\n", dp[len1][len2]);
    //---------- 以上部分为动态规划部分 ----------
    ...  //  递归部分完全一致
    return 0;
}

#if 0
Input:
aadfagha
abcdefgh
1 2 1

Output:
minimal weight: 6
s1:  aadfagha
add: abadfagha
chg: abcdfagha
chg: abcdeagha
chg: abcdefgha
del: abcdefgh
#endif
```

可以看到，提高了加入字符操作所具有的权重，最优路径从原来两次添加降为一次添加，并增加两次替换操作.

源代码：

1. 纯动态规划解决问题：`calc_distance_dp.cpp`

2. 显示编辑路径：calc_distance_complete.cpp
3. 带权重且显示编辑路径：calc_distance_complete_weight.cpp