

NLP 第一次编程作业

强基数学 002

吴天阳 2204210460, 马煜璇 2204220461

1 实验目的

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.
2. 修改操作代价改变最小编辑距离.

注: 最小编辑距离定义为: 假设 s_1 和 s_2 是两个给定的字符串, 考虑对 s_1 字符串进行 n 次添加字符、删除字符、替换字符三种操作, 将 s_1 变化为 s_2 , 求 n 的最小值.

例子: $s_1 = \text{aadfagha}$, $s_2 = \text{abcdefgh}$, 则可通过以下 5 次变换操作将 s_1 变为 s_2 :

- 1) 替换: **a** b d f a g h a
- 2) 添加: a b c d f a g h a
- 3) 添加: a b c d e f a g h a
- 4) 删除: a b c d e f g h a
- 5) 删除: a b c d e f g h

2 实验原理

2.1 最小编辑距离并显示编辑过程

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 s_1, s_2 为字符串, l_i 表示字符串 s_i 的长度 ($i = 1, 2$), $s_1[i]$ 表示字符串 s_1 中的第 i 个下标对应的字符.
- $s_1[i \cdots j]$ 表示截取字符串 s_1 下标从 i 到 j 的子串.
- s_1 与 s_2 完全匹配, 记为 $s_1 = s_2$, 当且仅当, s_1 与 s_2 长度相同, 且对任意下标 i 都有 $s_1[i] = s_2[i]$.

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 $s_1[1 \cdots i]$ 与子串 $s_2[1 \cdots j]$ 完全匹配所需的最小编辑次数, 根据字符串 $s_1[i]$ 与 $s_2[j]$ 是否相同可分为以下两种情况:

1. 若 $s_1[i] = s_2[j]$, 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 $s_1[i] \neq s_2[j]$, 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 $s_1[1 \cdots i] = s_2[1 \cdots j]$, 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 $s_1[i]$ 替换为 $s_2[j]$, 则只需保证 $s_1[1 \cdots i-1] = s_2[1 \cdots j-1]$ 即可, 而使得 $s_1[1 \cdots i-1] = s_2[1 \cdots j-1]$ 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: $\mathcal{O}(l_1 l_2)$.

2.2 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

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

3 实验步骤与结果分析

3.1 动态规划部分代码

对应 Luogu 原题: [P2758 编辑距离](#)

```

1  #include <iostream>
2  #include <cstring>
3  using namespace std;
4  const int N = 1e4 + 10; // 字符串最大长度
5  char s1[N], s2[N];
6  int dp[N][N]; // dp 为动态规划数组

```

```

7 signed main() {
8     memset(dp, 127, sizeof(dp)); // 初始化 dp 数组为极大值
9     cin >> s1 >> s2; // 字符串读入
10    int len1 = strlen(s1), len2 = strlen(s2);
11    for (int i = 0; i <= max(len1, len2); i++) dp[0][i] = dp[i][0] = i; //
    ↪ 初始化边界条件
12    for (int i = 1; i <= len1; i++) {
13        for (int j = 1; j <= len2; j++) {
14            if (s1[i-1] == s2[j-1]) {
15                dp[i][j] = dp[i-1][j-1];
16                continue;
17            }
18            dp[i][j] = min(dp[i-1][j-1], min(dp[i-1][j], dp[i][j-1])) + 1;
19        }
20    }
21    printf("%d\n", dp[len1][len2]);
22    system("pause");
23    return 0;
24 }

```

3.2 显示编辑路径

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）：

```

1  #include <iostream>
2  #include <cstring>
3  using namespace std;
4  const int N = 1e4 + 10; // 字符串最大长度
5  char s1[N], s2[N];
6  int dp[N][N], opt[N][N]; // dp 为动态规划数组, opt 记录每次操作值
7  // 0: none, 1: delete, 2: add, 3: change
8  string optName[] = {"none", "del", "add", "chg"}, nowString;
9  int delta = 0;
10 void dfs(int i, int j, int o) {
11     if (opt[i][j] == 0) dfs(i-1, j-1, 0);
12     else if (opt[i][j] == 1) dfs(i-1, j, 1);
13     else if (opt[i][j] == 2) dfs(i, j-1, 2);
14     else if (opt[i][j] == 3) dfs(i-1, j-1, 3);
15     if (o != 0) {

```

```

16         if (o == 1) nowString.erase(i+delta, 1), delta--;
17         else if (o == 2) nowString.insert(i+delta, 1, s2[j]), delta++;
18         else nowString[i+delta] = s2[j];
19         cout << optName[o] << ": " << nowString << '\n';
20     }
21 }
22 signed main() {
23     memset(dp, 127, sizeof(dp)); // 初始化 dp 数组为极大值
24     memset(opt, -1, sizeof(opt)); // 初始化 opt 数组为-1
25     cin >> s1 >> s2;
26     int len1 = strlen(s1), len2 = strlen(s2);
27     for (int i = 0; i <= max(len1, len2); i++) dp[0][i] = dp[i][0] = i;
28     for (int i = 1; i <= len1; i++) {
29         for (int j = 1; j <= len2; j++) {
30             if (s1[i-1] == s2[j-1]) {
31                 opt[i][j] = 0;
32                 dp[i][j] = dp[i-1][j-1];
33                 continue;
34             }
35             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
36             for (int k = 0; k < 3; k++) {
37                 if (tmp[k] < mn) {
38                     mn = tmp[k];
39                     mnId = k;
40                 }
41             }
42             dp[i][j] = mn + 1;
43             opt[i][j] = mnId + 1;
44         }
45     }
46     printf("minimal step: %d\n", dp[len1][len2]);
47     //----- 以上部分为动态规划部分 -----
48     nowString = string(s1);
49     cout << "s1: " << nowString << '\n';
50     dfs(len1, len2, 0);
51     system("pause");
52     return 0;
53 }
54
55 #if 0
56 Input:
57 aadfagha

```

```

58 abcdefgh
59
60 Output:
61 minimal step: 5
62 s1: aadfagha
63 chg: abdfagha
64 add: abcdfagha
65 add: abcdefagha
66 del: abcdefgha
67 del: abcdefgh
68 #endif

```

3.3 带权重的最小编辑

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. 完整代码请见源代码文件.

```

1 ...
2 int w[3]; // w[3] 分别对应删除, 添加, 替换的权重
3 ...
4 signed main() {
5 ...
6     cin >> w[0] >> w[1] >> w[2]; // 读入权重
7 ...
8     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
9     for (int i = 1; i <= len1; i++) {
10         for (int j = 1; j <= len2; j++) {
11             ...
12             for (int k = 0; k < 3; k++) {
13                 if (tmp[k] < mn) {
14                     mn = tmp[k] + w[k]; // 对第 k 个操作加上权重
15                     mnId = k;
16                 }
17             }
18             dp[i][j] = mn;
19             opt[i][j] = mnId + 1;
20         }
21     }
22     printf("minimal weight: %d\n", dp[len1][len2]);
23     //----- 以上部分为动态规划部分 -----
24     ... // 递归部分完全一致

```

```
25     return 0;
26 }
27
28 #if 0
29 Input:
30 aadfagha
31 abcdefgh
32 1 2 1
33
34 Output:
35 minimal weight: 6
36 s1: aadfagha
37 add: abadfagha
38 chg: abcdfagha
39 chg: abcdeagha
40 chg: abcdefgha
41 del: abcdefgh
42 #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。

4 结论与讨论

源代码：全部源代码均上传至[GitHub 仓库](#)，以下为三个代码文件具体内容：

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](#)

缺点：由于 Windows 下 C++ 只能英文读入，所以无法处理中文字符串。但可通过类似思路用 Python 实现即可。