# 实验报告



## 1. 解题思路

本题的本质其实就是找出两个字符串的最小编辑距离，因此很容易联想到用于找出最长公共子序列（Longest Common Subsequence, LCS）的动态规划解法，这部分详见 src/utils/utils.cpp 中函数 `FuzzyCompare` 的部分实现。

然而，朴素动态规划解法的时间复杂度和空间复杂度均为 $O(mn)$，其中 $m, n$ 表示两个字符串的长度。对于题目中 $m, n \approx 10^6$ 的规模来说，未免还是太大了。因此我的第一个思路是，将原字符串分成一个个片段，然后对每个片段逐一进行比较。这里片段的大小可以通过调整配置文件 src/utils/config.h 中的 `CHUNK_SIZE` 字段进行修改。如此，时间复杂度和空间复杂度均降低到 $O(k\max(m, n))$，其中 $k$ 表示片段的长度。当 $k$ 较小时，可以有效降低时间的开销。

当然，朴素动态规划解法还是有点慢了。本解法最后实现的核心算法是基于贪心算法的 Myers' Diff Algorithm，时间复杂度和空间复杂度均为 $O(d(m+n))$，其中 $d$ 表示两个字符串之间的编辑距离。本题中，两个字符串大体上是相同的，差别的数量最多只有 $50 \times 1000 \times 2 = 10^5$ 量级，当然实际上要比这个更小。在进行了分段优化后，期望时间复杂度和空间复杂度降低到 $O(dk)$。具体代码可参见 src/common/dna.cpp 中函数 `Dna::FindDeltasChunk` 的实现，对 Myers' Diff Algorithm 的详解可参考这篇文章 [1]。

Myers' Diff Algorithm 最终可以找到原字符串 `ref` 基础上所有的缺失片段和插入片段，接下来我们要考虑的是如何将它们转化为题目需求的 SV 类型。

### 1.1 INS - 片段插入

在 Myers' Diff Algorithm 中，我们会维护当前处理的字符所在的位置 $(x, y)$。其中 $x$ 表示字符在原字符串 `ref` 中的位置，$y$ 表示字符在修改字符串 `sv` 中的位置。我们记录节点在图中经过的路径 [2]，其中在 $x$ 轴上向右移动即表示删除字符，在 $y$ 轴上向下移动即表示插入字符，沿对角线向右下移动即表示不作修改，跳到下一字符。

因此，将所有向下移动路径的起点和终点保存下来，即可作为 INS 类型的 SV 片段。

然而，在实现中会遇到一个问题。由于很多时候插入 / 删除的片段中会包含和后续字符相同的字符，因此原 Myers' Diff Algorithm 得到的路径往往是不连续的，即新增几个字符、跳过几个字符、又新增几个字符。这不符合题目中 SV 片段长度 $l \in [50, 1000]$ 的要求。

这里我们的解决方案分为两部分：

首先，我们修改了 Myers' Diff Algorithm 中的部分逻辑。当节点沿着对角线向右下方移动结束（即跳过所有相同的字符）后，立即进行一次对 snake 长度的判断。其中，snake 表示节点沿对角线移动的距离。如果 snake 的长度小于 `SNAKE_MIN_LEN` 的值（可在配置文件中调整），即表示连续的相同字符数量没有超过设定的阈值，则将节点移回对角线的起点，本轮不进行跳过字符。如此，即可防止出现频繁的抖动，确保 SV 片段基本上是完整的片段。参见 src/common/dna.cpp:148。

其次，我们在保存 SV 片段时进行一次判定。如果在之前保存的同类型 SV 片段中，包含与当前片段**大致**重叠或临近的片段，那么我们就将这两个片段进行合并（当然，对于过长的临近片段不会进行合并）。如此，即可确保不会出现 SV 片段的碎片。参见 src/common/dna_delta.cpp 中函数 `DnaDelta::Combine` 和 `DnaMultiDelta::Combine` 的实现，重点在于 src/common/range.cpp 中函数 `FuzzyCompare` 的实现。

这样以来，我们能够基本保证 INS 类型 SV 片段的正确性。

## 1.2 DEL - 片段缺失

基本同 INS 类型，区别在于 INS 类型保存向下移动的路径，DEL 类型则保存向右移动的路径。

## 1.3 DUP - 片段串联重复

在得到所有 INS 和 DEL 类型 SV 片段的基础上，我们从中识别出剩下 3 种 SV 类型。

对于 DUP 类型，我们遍历所有的 INS 类型片段 $[\text{start}, \text{end})$，比较此片段字符串 $\text{SV}[\text{start'}...\text{end'}]$（已预先保存）与其插入位置前面的字符串 $\text{REF}[2 * \text{start - end}...\text{start}]$ 是否**大致**相同。如果是，则将此片段的类型修改为 DUP 类型。参见 src/common/dna.cpp 中函数 `Dna::FindDupDeltas` 的实现。

关于两个字符串的比较，我们进行两种判定：

1. 找出**最长公共子串**（Longest Common Substring）的长度，计算其覆盖率
2. 找出**最长公共子序列**（Longest Common Subsequence）的长度，计算其覆盖率

其中任意一个判定达到标准，即认为两个字符串大致相同，具体可参见 src/utils/utils.cpp 中函数 `FuzzyCompare` 的实现。这样可以尽可能应对 SV 片段重叠、SV 片段少量出错或未知等问题，提高算法鲁棒性。

## 1.4 INV - 染色体倒位

对于 INV 类型，我们遍历同染色体下的所有 INS 和 DEL 类型片段，找出其中**大致**相邻且大小**大致**相同的 INS 和 DEL 类型片段，将其中 INS 类型片段的位置修改为与 DEL 类型片段对齐（只是为了和要求的输出格式相符），然后对其代表的字符串进行比较。字符串的比较方式同 DUP 类型中的描述，区别在于其中一个字符串要进行反向互补操作，即先进行反转，再逐位将 `A`, `T` 互换、 `C`, `G` 互换。如果字符串基本匹配，则移除对应的 INS 和 DEL 类型片段，新增一个 INV 类型片段。参见 src/common/dna.cpp 中函数 `Dna::FindInvDeltas` 的实现。

### 1.5 TRA - 染色体间异位

TRA 类型类似于 INV 类型，我们遍历所有染色体的 INS 和 DEL 类型片段，找出每条染色体中**大致**相邻且大小**大致**相同的片段，分别放入一个 INS / DEL 缓存，然后移除。然后，我们遍历 INS 和 DEL 缓存，找出其中**大致**相邻且大小**大致**相同的 INS 和 DEL 类型片段，然后对其代表的字符串进行比较。字符串的比较方式同 DUP 类型中的描述。如果字符串基本匹配，则新增一个 TRA 类型片段。最后将未匹配的缓存中的片段放回原处。参见 src/common/dna.cpp 中函数 `Dna::FindTraDeltas` 的实现。

## 2. 运行代码

本项目使用 C++17 实现，要求使用 gcc 版本 > 7.0，或 clang 版本 > 5.0。本项目已于 macOS Catalina 10.15.7 + clang 12.0.0 (x86_64) 环境下通过测试，task 1 数据生成的 `sv.bed` 文件位于 `tests/test_1/sv.bed`。

为了方便进行编译，本项目利用 GNU Make 编写了编译脚本。执行 `make` 即可构建项目，执行 `make run` 即可启动算法程序。Makefile 中默认使用 clang++ 作为编译器，如需改动，可以在 Makefile 文件中对应修改。Windows 环境下，推荐使用 Windows Subsystem for Linux (WSL) 或 Mingw-w64。

---

1. Investigating Myers' diff algorithm: Part 1 of 2 - CodeProject ↩
2. LCS 并不唯一，我们只保存找到的第一条；实现中，需要在每次发生编辑操作（节点移动到了其他 k-line 上）时保存当前 LCS 的快照，用于在最后进行回溯，这也是空间复杂度 $O(d(m+n))$ 的实际来源。 ↩