实验报告

1. 解题思路

在 Task 1 的基础上，本题的主要难点分为两部分：

1. 如何将 `long.fasta` 中的 read 片段快速匹配到参考字符串 ref 上
2. 如何在 read 片段平均 15% 的噪音干扰下，准确找到 SV 片段

这里我们参考了 Minimap2 [1] 的论文思路，根据实际情况进行了简化和调整。算法核心分为以下三个步骤：

1. 对参考字符串 ref 利用最小哈希建立索引
2. 利用 ref 的索引，将 read 片段匹配到 ref 上的对应区域
3. 比较 read 片段和 ref 上匹配到的区域，查找 SV 片段

1.1 建立索引

由于噪音和 SV 片段的存在，我们无法简单地在 ref 字符串中直接查找一个 read 子字符串片段。因此，我们需要建立索引。

如何对参考字符串 ref 建立索引？简单来说，就是将 ref 中每个长度为 k（即 `HASH_SIZE`，默认为 `15`，所有参数均可在 src/utils/config.cpp 中调整）的子字符串，利用一个哈希函数转化为一个整数。具体来说，我们利用一个 2 位二进制整数来表示一个 DNA 碱基：`00` 表示 `A`、`01` 表示 `T`、`10` 表示 `C`、`11` 表示 `G`。对于一条 DNA 链，其哈希值就是将所有碱基的二进制数表示连接起来，例如 `GCTA` 的哈希值就是 `11100100`。通过这种方式，我们可以将每个连续的 k 位子字符串转化为一个 $2k$ 位二进制数作为其哈希值 hash。我们称这样一个从 hash 到这个子字符串在 ref 中位置 $[i, i+k)$ 的映射为一个 k-mer。

对于一条长度为 N 的 DNA 链，就包含了 $N-k+1$ 个这样的 k-mer。为了减少 k-mer 的数量，以减少使用的内存空间，我们维护一个长度为 `WINDOW_SIZE`（默认为 `10`）的滑动窗口，只有滑动窗口中哈希值最小的 k-mer 才会被保存作为索引。

具体逻辑可参见 src/common/dna.cpp 中函数 `Dna::CreateIndex` 的实现。为了方便重复使用，我们提供了 `Dna::PrintIndex` 函数用于将索引导出成文件，以及 `Dna::ImportIndex` 函数用于从文件中读取索引。

1.2 字符串片段的模糊匹配

1.2.1 生成 minimizer

建立完索引后，我们就可以在每个 read 片段中遍历所有长度为 k 的子字符串，根据其哈希值查找是否有相同哈希值的 k-mer。同时，我们对于 read 片段的反向互补序列 read' 也进行同样的操作。对于每个找到的 k-mer，我们保存一个这样的结构：$\{\text{range}_{\text{ref}}, \text{key}_{\text{read}}, \text{range}_{\text{read}}\}$，我们称其为一个 minimizer。其中，$\text{range}_{\text{ref}}$ 表示 k-mer 映射到 ref 上的位置 $[i, i+k)$，key_{read} 表示 read 的编号（例如 S1_1），$\text{range}_{\text{read}}$ 表示这个子字符串在 read（或 read'）上的位置 $[j, j+k)$。同时，在每个 range 中还额外保存了一个原字符串（ref 或 read）的指针，用于之后读取及合并这个子字符串的值。在 $\text{range}_{\text{read}}$ 中还额外保存了 `mode` 字段和 `unknown` 字段，分别用于指示当前 read 的模式（是否是反向互补序列），以及是否包含一定数量的未知字符 `N`（在合并时用于提高效率，不关键）。

随后，我们根据 read 和 read' 片段上 minimizer 的数量，决定是否对 read 进行反向互补操作。即如果 read' 上的 minimizer 较多，则进行反向互补操作，反之则不进行。

具体逻辑可参见 src/common/dna.cpp 中函数 `Dna::FindOverlaps` 的实现。同时，我们提供了 `Dna::PrintOverlaps` 函数用于将 minimizer 导出成文件，以及 `Dna::ImportOverlaps` 函数用于从文件中读取 minimizer。

1.2.2 合并 minimizer

生成 minimizer 后，我们需要对它们进行过滤及合并。其中，过滤指的是将错误匹配的 minimizer 移除，合并指的是将两个 minimizer 根据其 $\text{range}_{\text{ref}}$ 的范围 $[i_1, i_1+k), [i_2, i_2+k)$ 进行合并。

具体来说，在与一个聚类合并时，对于每一个 minimizer，我们比较此次合并后 $\text{range}_{\text{ref}}$ 和 $\text{range}_{\text{read}}$ 表示范围的增量 Δ_{ref} 和 Δ_{read}。如果它们的差距不大，则将这个 minimizer 归并到当前聚类，同时此聚类的计数器加 1；反之则尝试合并到下一个聚类，如果没有可合并的聚类，则将其单独分到一个新的聚类。

合并后，新的 minimizer 的 $\text{range}_{\text{ref}}$ 为 $[\min\{i_1, i_2\}, \max\{i_1, i_2\}+k)$，$\text{key}_{\text{read}}$ 为空字符串，$\text{range}_{\text{read}}$ 为 $[0, l)$，其中 l 为合并后新生成的 read 字符串的长度，同时 $\text{range}_{\text{read}}$ 中保存的指针指向这个新字符串。

于是，我们就得到了若干 minimizer 聚类。我们将其中计数器值较小或者范围较小的 minimizer 过滤。

具体逻辑可参见 src/common/dna_overlap.cpp 中函数 `DnaOverlap::Merge` 的实现。

1.2.3 匹配 ref 链和 sv 链

由于 `long.fasta` 中同时包含了多条 sv 链的采样，我们需要从中找到与 ref 链匹配的 sv 链。这里我们根据 minimizer 在 ref 上的覆盖率，选择覆盖率最高的 sv 链与 ref 链相匹配。

在 src/utils/config.cpp 中修改 `LOG_LEVEL` 为 `DEBUG`，即可在日志 `logs/output.log` 中看到 minimizer 的覆盖率（搜索 `cover rate`）。对于本题的正式数据，我们的覆盖率分别达到了：

- `NC_017999.1`：99.05%（`S3`）
- `NC_010513.1`：99.01%（`S1`）
- `NC_014752.1`：98.07%（`S2`）

具体逻辑可参见 src/common/dna_overlap.cpp 中函数 `DnaOverlap::SelectChain` 和 `DnaOverlap::CheckCoverage` 的实现。

1.3 查找 SV 片段

最终，我们将问题化归到了类似于 Task 1 的情形。根据每个 minimizer 中保存的 $\text{range}_{\text{ref}}$ 和 $\text{range}_{\text{read}}$，我们可以得到两个需要比较的字符串。接下来复用 `Dna::FindDeltasChunk` 函数的逻辑即可。

但是，由于 Task 2 的数据含有一定量的噪声，原先对 Task 1 的数据处理方式不再适用于 Task 2，我们需要重新研究如何处理通过 `Dna::FindDeltasChunk` 函数得到的 SV。

具体来说，由于噪声的存在，SV 变得更加零散，同时我们难以区分一个 SV 是真正的 SV 还是只是噪声。我们曾经尝试过利用 SV 的间隔来判断一个 SV 是否是噪声，也尝试过魔改 Myers' Diff Algorithm 来消除部分噪声，但效果都不理想。最后经过助教的提示，我们调整了方式，使用一定范围内 SV 的密度来估计 SV 可能存在的范围。这是因为如果是纯噪声，SV 的密度大约会在 15% 左右，而对于真实的 SV，其密度往往在 50% 以上。通过观察 SV 密度的变化，就有可能判断 SV 的位置。相关逻辑参见 src/common/dna_delta.cpp 中函数 `DnaDelta::GetDensity` 的实现。

具体逻辑可参见 src/common/dna.cpp 中函数 `Dna::FindDeltasFromSegments` 的实现。

接下来就是调整参数的工作了，在配置文件 src/utils/config.cpp 中有大量可以调节的参数，其中比较重要的参数有 `SIGNAL_RATE`，`DENSITY_WINDOW_SIZE`，`DELTA_MIN_LEN`，`SNAKE_MIN_LEN`，`GAP_MIN_DIFF` 等。由于时间关系，没有很多时间用来调参了，因此最后的实验结果尚不理想。目前在 Task 2 上的累计耗时可参见页首的 wakatime 徽章。

2. 运行代码

本项目使用 C++17 实现，要求使用 gcc 版本 **≥ 9.0（不支持 8.0 及以下版本）**。本项目已于 Ubuntu 18.04.5 LTS (5.4.72-microsoft-standard-WSL2) + gcc 10.3.0 环境下通过测试，Task 2 数据生成的

`sv.bed` 文件位于 tests/test_2/sv.bed。

为了方便进行编译，本项目利用 GNU Make 编写了编译脚本。可用指令如下：

- `make`：构建项目
- `make help`：显示参数及其用法
- `make run`：完整启动算法程序
- `make index`：只创建 ref 的索引
- `make minimizer`：只生成 minimizer
- `make start`：只查找 SV 片段
- `make clean`：清除构建文件

Makefile 中默认使用 g++ 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)。

---

1. Heng Li. Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics, 34, 18, 2018: 3094–3100. ↩