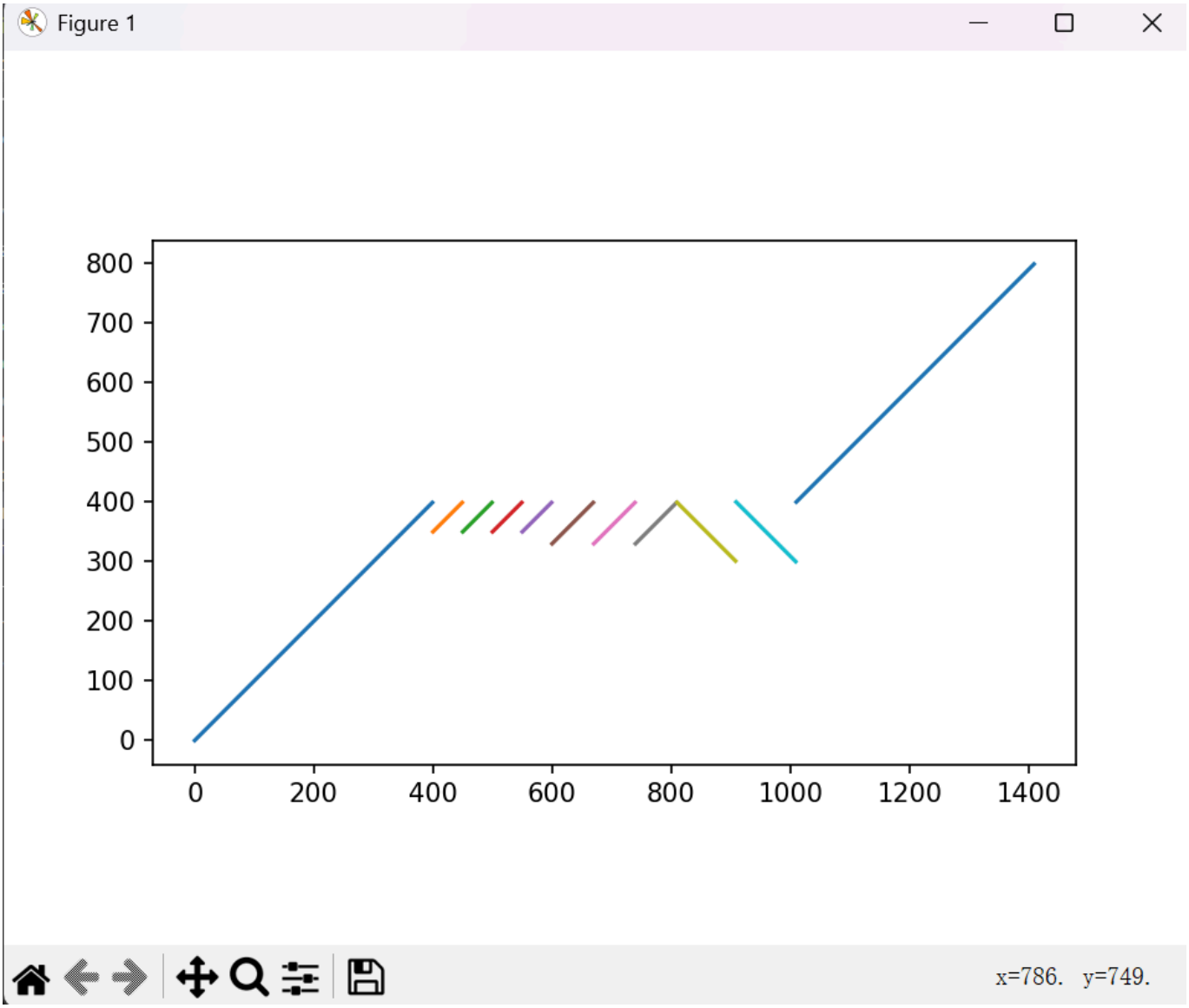
寻找样本DNA序列中的重复片段

23300240009 佟铭洋

对于下发的样例的输出

```
|-------------------------------------------------|
|                   Statistics                    |
|-------------------------------------------------|
| Pos in Ref | Pattern | Length | Inverted | Count |
|-------------------------------------------------|
| 400 -  799 | CT...AT |    400 |    False |     1 |
| 301 -  399 | AG...CG |     99 |     True |     1 |
| 330 -  399 | TA...CG |     70 |    False |     3 |
| 300 -  400 | CA...CC |    101 |     True |     1 |
| 350 -  399 | CT...CG |     50 |    False |     4 |
|   0 -  399 | CT...CG |    400 |    False |     1 |
|-------------------------------------------------|
```



思路

以下约定：s 表示 reference 串，t 表示 query 串。$s[i]$ 表示 reference 串的第 i 个字符（从 0 开始），$t[i:j]$ 表示 query 串的第 i 到第 j 个字符（包括首尾）。

s 和 t 的长度分别为 n 和 m。

使用哈希表保存所有 reference 串的子串（以及逆转后的子串）。

使用 $f[k]$ 表示从 $t[k]$ 到 $t[n]$ （注意不是 $n-1$）的**最少跳转次数**（保证平均每个片段最长）。初始化 $f[k]$ 为无穷大，$f[n]$ 为 0。

对于 query 串的每个子串 $t[i:j]$，寻找其是否在 reference 中出现过。若出现过则尝试更新 $f[j]$。

重点

遍历 query 串每个子串的顺序非常重要。其实这里存在两种思路：从前往后扫和从后往前扫，理论上效果是一致的。此处使用从后往前扫。即先遍历所有 query 的后缀，然后是去掉最后一个字符的后缀，然后是去掉最后两个字符的后缀，以此类推。

精细化处理

- 保存是从哪里跳的，简单记录即可，以下不再介绍。
- $\Theta(1)$ 求子串哈希，以下将介绍。
- 美观输出，与算法无关，以下不再介绍。

程序

- `final.cpp` **需要使用 C++20 编译，因为使用了 std::unordered_map::contains 这个语法糖，若没有对应环境请改成std::unordered_map::find != std::unordered_map::end**
- `plot.py` 用于画图。

伪代码

Algorithm 1 PreHash

```
procedure PreHash(s, l, r, ht)
    h ← 0
    for i ← l to r − 1 do
        h ← h × 131 + s[i]
        ht[i] ← h
    end for
end procedure
```

Algorithm 2 PreHashInv

```
procedure PreHashInv(l, r, s, ht)
    n ← Length(s)
    for i ← 0 to n − 1 do
        reference_rc[i] ← s[n − i − 1]
    end for
    reference_rc[n] ← NULL
    for i ← 0 to n − 1 do
        if reference_rc[i] =′ A′ then
            reference_rc[i] ←′ T′
        else if reference_rc[i] =′ T′ then
```

```
            reference_rc[i] ←′ A′
        else if reference_rc[i] =′ C′ then
            reference_rc[i] ←′ G′
        else
            reference_rc[i] ←′ C′
        end if
    end for
    Hash(0, n, reference_rc, ht)
end procedure
```

Algorithm 3 PartialHash

```
function PartialHash(l, r, ht)
    if l = 0 then
        return ht[r]
    end if
    return ht[r] − ht[l − 1] × pow131[r − l + 1]
end function
function PartialHashInv(l, r, n, htinv)
    return PartialHash(n − r − 1, n − l − 1, htinv)
end function
```

Algorithm 4 PartialHash

```
procedure ComputeF(query_len, hash_query, substrings, f, f_from)
    Memset(f, ∞, SizeOf(f))
    f[query_len] ← 0
    for i ← query_len − 1 to 0 do
        for j ← i to query_len − 1 do
            h ← PartialHash(i, j, hash_query)
            if h ∈ substrings then
                jump_from ← j + 1
                if f[i] > 1 + f[jump_from] then
                    f[i] ← 1 + f[jump_from]
                    begin ← substrings[h].begin
                    inverted ← substrings[h].inverted
                    f_from[i] ← {jump_from, begin, inverted}
                end if
            end if
        end for
    end for
end procedure
```

复杂度分析

时间复杂度

- 预处理幂 $\Theta(n+m)$。
- 预处理哈希 $\Theta(n+m)$。
- 预处理所有 reference 子串哈希 $\Theta(n^2)$。
- 预处理所有 reference 逆转子串哈希 $\Theta(n^2)$。
- 计算 f 数组 $\Theta(m^2)$。
- 回溯 $O(m)$。

总时间复杂度为 $\Theta(n^2+m^2)$。