

寻找样本 DNA 序列中的重复片段

实验文档

俞舜云 24300240045

2025 年 3 月 27 日

1 算法伪代码

本实验以动态规划算法为核心，通过以下几个核心步骤完成序列比对与重复区域识别：

1.1 预处理阶段

```
1 Procedure PREP_ANALYZE(reference, query, kmer_size)
2     // 初始化长度参数
3     refeLength  $\leftarrow$  |reference| - kmer_size + 1
4     querLength  $\leftarrow$  |query| - kmer_size + 1
5
6     // 计算哈希值
7     refeHash  $\leftarrow$  ROLLING_HASH(reference)
8     querHash  $\leftarrow$  ROLLING_HASH(query)
9     refeRevHash  $\leftarrow$  ROLLING_HASH(REVERSE_COMPLEMENT(reference))
10
11    // 初始化动态规划表
12    for i  $\leftarrow$  0 to querLength - 1 do
13        for j  $\leftarrow$  0 to refeLength - 1 do
14            Align[i][j]  $\leftarrow$  (MIN_INF, -1, 0)
15        end for
16    end for
```

```

17
18     Align[0][0] ← (0, -1, 1)
19
20     // 初始化查询序列对齐数组
21     for i ← 0 to querLength - 1 do
22         querAlign[i] ← (MIN_INF, -1, -1)
23     end for
24
25     querAlign[0] ← (0, 0, -1)
26
27     // 初始化路径记录结构
28     for i ← 0 to querLength - 1 do
29         pointRoute[i] ← (0, -1, 0)
30     end for
31 end Procedure

1 Procedure ANALYZE_ROUTE(reference, query)
2     // 主循环, 遍历每个查询序列位置
3     for i ← 1 to querLength - 1 do
4         for j ← 0 to refeLength - 1 do
5             // 考虑正向匹配
6             if IS_EQUAL(i, j) then
7                 // 新开序列得分
8                 Align[i][j] ← (querAlign[i-1].maxScore +
9                             maintainScore + newScore,
10                             querAlign[i-1].maxScoreIndex,
11                             1)
12
13             // 继续延续得分
14             if j - 1 ≥ 0 and IS_EQUAL(i-1, j-1) then
15                 if Align[i][j].maxScore < Align[i-1][j-1].
16                     maxScore +
17                         maintainScore +
18                         Align[i-1][j-1].
19                             continuousCount

```

```

*
continuousScore
then
16         Align[i][j] ← (Align[i-1][j-1].
            maxScore +
17             maintainScore +
18             Align[i-1][j-1].
                continuousCount *
                continuousScore,
19             j-1, Align[i-1][j-1].
                continuousCount +
                1)
20     end if
21 end if
22
23 // 更新当前查询位置的最佳匹配
24 if querAlign[i].maxScore < Align[i][j].
    maxScore then
25     querAlign[i] ← (Align[i][j].maxScore, j,
        Align[i][j].prevIndex)
26 end if
27
28 // 考虑反向互补匹配
29 else if IS_MATCH(i, j) then
30     // 类似正向匹配的处理，但参数和方向不同
31     // 略...
32 end if
33 end for
34 end for
35 end Procedure

1 Procedure ANALYZE_SEQUENCE()
2     // 回溯得到最优路径
3     h ← querLength - 1
4     l ← refeLength - 1

```

```

5   while  $h \geq 0$  and  $l \geq 0$  do
6       Route[h][l]  $\leftarrow$  Align[h][l].maxScore
7       pointRoute[h]  $\leftarrow$  (Align[h][l].maxScore, l, Align[h][l].
           prevIndex)
8       l  $\leftarrow$  Align[h][l].prevIndex
9       h  $\leftarrow$  h - 1
10  end while
11
12  // 提取连续子串
13  head  $\leftarrow$  0
14  while head < querLength do
15      tail  $\leftarrow$  head + 1
16      // 寻找连续序列的终点
17      while tail < querLength and |Align[tail][pointRoute[
           tail].maxScoreIndex].continuousCount|  $\neq$  1 do
18          tail  $\leftarrow$  tail + 1
19      end while
20
21      // 创建重复序列片段
22      if IS_MATCH(head, pointRoute[head].maxScoreIndex) then
23          segment  $\leftarrow$  CREATE_SEGMENT(query[head:tail-1],
24                                          pointRoute[head].
25                                              maxScoreIndex,
26                                              tail-head, 1, true, head,
27                                              tail-1)
28          segments.APPEND(segment)
29      else
30          // 处理非匹配情况
31          // 略...
32      end if
33  end while
34 end Procedure

```

```
1      Procedure LINEAR_SEARCH(A, v)
2          for i ← 1 to |A| do
3              if A[i] = v then
4                  return i
5              end if
6          end for
7          return NIL
8      end Procedure
```

2 时空复杂度分析

假设参考序列长度为 n ，查询序列长度为 m ，分析各阶段的时空复杂度：

2.1 时间复杂度

(1) 预处理阶段：

- 滚动哈希计算： $O(n^2)$ 和 $O(m^2)$ ，用于计算所有子串的哈希值
- 初始化动态规划表： $O(nm)$ ，需要初始化 $m \times n$ 大小的表格

(2) 序列比对阶段：

- 主循环： $O(nm)$ ，需要遍历查询序列的每个位置和参考序列的每个位置
- 每个位置的计算： $O(1)$ ，包括正向和反向匹配计算

(3) 序列分析与提取阶段：

- 回溯路径： $O(m)$ ，最多需要回溯查询序列的长度
- 提取连续子串： $O(m)$ ，最多遍历整个查询序列

综合分析，算法的总时间复杂度为 $O(n^2 + m^2 + nm)$ ，当查询序列和参考序列长度相近时，时间复杂度约为 $O(n^2)$ 。

2.2 空间复杂度

(1) 哈希值存储：

- 参考序列哈希: $O(n^2)$
- 查询序列哈希: $O(m^2)$

(2) 动态规划表:

- Align 矩阵: $O(nm)$
- querAlign 和 pointRoute 数组: $O(m)$
- Route 矩阵: $O(nm)$

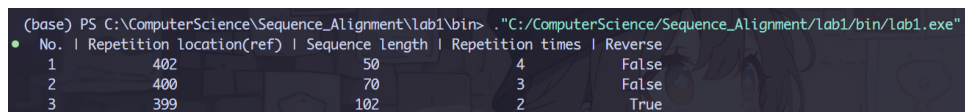
(3) 结果存储:

- segments 数组: $O(k)$, 其中 k 是识别出的序列段数量

综合分析, 算法的总空间复杂度为 $O(n^2 + m^2 + nm)$, 主要受哈希表和动态规划表的大小影响。

3 运行结果

对于样例 reference、query, 得到运行结果如下



```
(base) PS C:\ComputerScience\Sequence_Alignment\lab1\bin> . "C:/ComputerScience/Sequence_Alignment/lab1/bin/lab1.exe"
```

No.	Repetition location(ref)	Sequence length	Repetition times	Reverse
1	402	50	4	False
2	400	70	3	False
3	399	102	2	True

图 1: 终端输出

No.	Repetition location(ref)	Sequence length	Repetition times	Reverse
1	402	50	4	False
2	400	70	3	False
3	399	102	2	True

图 2: 终端输出 text

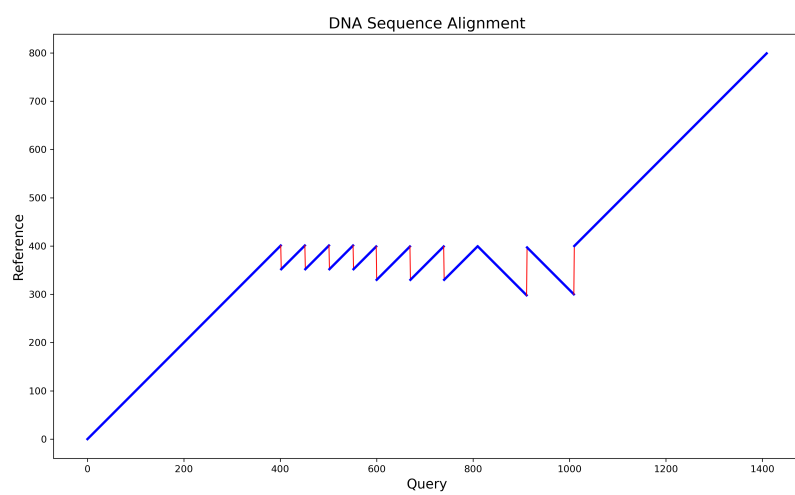


图 3: 模拟比对演示