寻找样本DNA序列中的重复片段

23300240009 佟铭洋

算法

采用图论算法。显然我们需要用的是最短路算法（而不是最长路）。原因是：最长路算法的时间复杂度太烂了。

我们逆向思维一下，最大化得分就是最小化代价。这样我们可以考虑用最短路算法来实现。

设 ref 为参考序列，$query$ 为查询序列。另外预处理一个 ref 的互补（但不反向）序列 ref_{rev}。 ref 和 $query$ 的长度分别为 n 和 m。下标从 1 开始。

我们认为有以下的点：

- 第 A 类：$NodeA(i,j)$ 表示正向匹配完 $ref[i]$ 与 $query[j]$ 的状态。
- 第 B 类：$NodeB(i,j)$ 表示反向匹配完 $ref[i]$ 与 $query[j]$ 的状态。
- 第 C 类：$NodeC(k)$ 表示某一匹配段以 $query[k]$ 结尾的状态。

对于 A 类点为起点的边：

- $NodeA(i,j) \to NodeA(i+1,j+1)$，当 $ref[i+1]$==$query[j+1]$ 边权为 0，否则边权为 1。
- $NodeA(i,j) \to NodeA(i+1,j)$，边权为 1。
- $NodeA(i,j) \to NodeA(i,j+1)$，边权为 1。
- $NodeA(i,j) \to NodeC(j)$，边权为 1。

对于 B 类点为起点的边：

- $NodeB(i,j) \to NodeB(i-1,j+1)$，当 $ref_{rev}[i-1]$==$query[j+1]$ 边权为 0，否则边权为 1。
- $NodeB(i,j) \to NodeB(i-1,j)$，边权为 1。
- $NodeB(i,j) \to NodeB(i,j+1)$，边权为 1。
- $NodeB(i,j) \to NodeC(j)$，边权为 1。

对于 C 类点为起点的边：

- $NodeC(j) \to NodeA(i,j)$，边权为 0。
- $NodeC(j) \to NodeB(i,j)$，边权为 0。

因此，总代价就是 $NodeC(0)$ 到 $NodeA(k,m)$ 的最短路径。

复杂度分析

初步探索，多了一个 $\log$ 的复杂度

其实这个算法是一个最短路径算法。我们可以用 Dijkstra 算法来实现。考虑到点的个数有 $O(nm)$ 个。其中：

- A 类点有 $\Theta(nm)$ 个，每个 A 类点有至多 4 条边，这是 $O(nm)$。

- B 类点有 $\Theta(nm)$ 个，每个 B 类点有至多 4 条边，这是 $O(nm)$。
- C 类点有 $\Theta(m)$ 个，每个 C 类点有至多 n 条边，这是 $O(nm)$。

故点数为 $O(nm)$，边数为 $O(nm)$。

- 使用 Dijkstra 算法，时间复杂度为 $O((nm)\log(nm))$。

这个时间复杂度并不满足不差于平方量级的要求，我们需要进一步优化。

去掉 log 的复杂度

这个 $\log(nm)$ 的复杂度是由于优先队列中最差情况可能会有 $O(nm)$ 个元素。问题来自于优先队列本身。

考虑到 Dijkstra 算法的一个特例：对于 0-1 边权的图，这可以退化为 BFS 算法。我们可以使用双端队列来实现，由于队列的复杂度为 $O(1)$，所以我们可以将复杂度降到 $O(nm)$。

- 使用双端队列，时间复杂度为 $O(nm)$。

伪代码：

Algorithm 1 ProcessOfNode

function PROCESS($Node, queue$)
  $i \leftarrow Node.i$
  $j \leftarrow Node.j$
  if $Node.type == A$ **then**
    if $i+1 \le |ref|$ **and** $j+1 \le |query|$ **then**
      $weight \leftarrow 0, if(reference[i+1] == query[j+1]), else, 1$
      if $dis[NodeA(i+1,j+1)] > dis[Node] + weight$ **then**
        $dis[NodeA(i+1,j+1)] \leftarrow dis[Node] + weight$
        if $weight == 0$ **then**
          $queue.front \leftarrow (NodeA(i+1,j+1))$
        else
          $queue.back \leftarrow (NodeA(i+1,j+1))$
        end if
      end if
    end if
    if $i+1 \le |ref|$ **then**
      if $dis[NodeA(i+1,j)] > dis[Node] + 1$ **then**
        $dis[NodeA(i+1,j)] \leftarrow dis[Node] + 1$
        $queue.back \leftarrow (NodeA(i+1,j))$
      end if
    end if
    if $j+1 \le |query|$ **then**
      if $dis[NodeA(i,j+1)] > dis[Node] + 1$ **then**
        $dis[NodeA(i,j+1)] \leftarrow dis[Node] + 1$
        $queue.back \leftarrow (NodeA(i,j+1))$
      end if
    end if
  else if $Node.type == B$ **then**
    if $i-1 \ge 0$ **and** $j+1 \le |query|$ **then**
      $weight \leftarrow 0, if(reference_{rev}[i-1] == query[j+1]), else, 1$
      if $dis[NodeB(i-1,j+1)] > dis[Node] + weight$ **then**
        $dis[NodeB(i-1,j+1)] \leftarrow dis[Node] + weight$
        if $weight == 0$ **then**
          $queue.front \leftarrow (NodeB(i-1,j+1))$
        else
          $queue.back \leftarrow (NodeB(i-1,j+1))$
        end if

```
            end if
        end if
        if i − 1 ≥ 0 then
            if dis[NodeB(i − 1, j)] > dis[Node] + 1 then
                dis[NodeB(i − 1, j)] ← dis[Node] + 1
                queue.back ← (NodeB(i − 1, j))
            end if
        end if
        if j + 1 ≤ |query| then
            if dis[NodeB(i, j + 1)] > dis[Node] + 1 then
                dis[NodeB(i, j + 1)] ← dis[Node] + 1
                queue.back ← (NodeB(i, j + 1))
            end if
        end if
    else if Node.type==C then
        for i ← 0 to |ref| do
            if dis[NodeA(i, j)] > dis[Node] then
                dis[NodeA(i, j)] ← dis[Node]
                queue.front ← (NodeA(i, j))
            end if
            if dis[NodeB(i, j)] > dis[Node] then
                dis[NodeB(i, j)] ← dis[Node]
                queue.front ← (NodeB(i, j))
            end if
        end for
    end if
end function
```

Algorithm 2 ProcessQueue

```
function ProcessQueue(queue)
    while queue is not empty do
        Node ← queue.front
        if Node.j == m then
            return
        end if
        queue.pop()
        if visited[Node] then
            continue
        end if
        visited[Node] ← true
        Process(Node, queue)
    end while
end function
```

结果

最终，可以在第一个样例拿到 29.813k（距离基线 7），在第二个样例拿到 2080（距离基线 10）。

考虑达不到基线的原因是没有考虑长度为 30 以下的片段被忽略的问题。

程序

- `main2.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**

常数比较大，而且需要遍历所有的点，大概需要的内存为 $200 \times n \times m\text{Bytes}$，例如对样例 1 这大概需要 190GB。

对内存占用做了一些优化，可以去掉一些点，大约节省了 40 的内存，但会导致时间加倍：

- `main2_mem.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**