南开大学
Nankai University

南　　开　　大　　学

计算机学院和网络空间安全学院

算法导论课程作业

2022 年 6 月 6 日

目录

一、 问题描述

2019 年，新冠肺炎疫情爆发，引起新冠肺炎疫情是一种冠状病毒。病毒的遗传物质储存在 RNA 中，RNA 是单链结构，稳定性差，变异很快，从而使新型冠状病毒产生了许多变种。通过设计检测算法，输入一个病毒的多条 RNA 和新冠肺炎病毒多条 RNA 进行比对，比较其相似性，判断是否为同源病毒或者变异品种。

RNA 分子是由四种核苷酸组成的长链，这四种核苷酸分别是 A（腺嘌呤）、G（鸟嘌呤）、C（胞嘧啶）、U（尿嘧啶）。习惯上用一个字符集为 {A,G,C,U} 的字符串来表示一个 RNA 分子序列。一条 RNA 分子的碱基个数有几百到几万个不等，冠状病毒作为最大的一类病毒，一条 RNA 分子的碱基个数约有十几万个。

一个病毒 RNA 条数变化很大，有的病毒只有几条 RNA，有的病毒有几百条 RNA 分子。在进行病毒 RNA 比对时，不仅需要对来自不同病毒的每条 RNA 一一比对，找到最佳比对得分，也需要对来自两个病毒的多条 RNA 进行一一匹配，获取最优的匹配方式，使整体匹配得分最高。

本题抽象为输入两个病毒的 RNA 个数和每一条 RNA 的碱基序列，进行相似性比较。如：有两个不同病毒 A 和 B，A 病毒有 3 条 RNA 序列，分别为：①ACUUCG ②GUGAUAGACAC ③AUGGAGA；B 病毒有 5 条 RNA 序列,分别为：①ACUAG ②GUGAUAGACAC ③AUCGUGUG ④CCGUGGA ⑤CUAUGUG。在进行相似性比对时，需要对 A 病毒和 B 病毒的多条 RNA 进行匹配，如 A 病毒的 RNA①和 B 病毒的 RNA①进行匹配，这两条 RNA 匹配的得分由序列比对这两条 RNA 的碱基序列的方式得出。需要寻找一个最优的匹配方式，使匹配的总得分最高，由这个得分代表两个病毒的相似度。

在上面例子中，显然 A 病毒的 RNA①和 B 病毒的 RNA①、A 病毒的 RNA②和 B 病毒的 RNA②、A 病毒的 RNA③和 B 病毒的 RNA③、B 病毒的 RNA④和 B 病毒的 RNA⑤没有 A 病毒的 RNA 进行配对，这样的匹配方式会使匹配的总得分最高。

二、 设计算法

本问题算法设计包括两部分，第一部分解决两条 RNA 分子匹配时的最佳匹配方式和得分；第二部分解决两个病毒的两组 RNA 分子的最佳匹配方式和得分，带权值的匹配问题中的权值由第一部分的算法（两条 RNA 分子匹配时的得分）获得。

(一) 两条 RNA 匹配得分

对于两条 RNA，将其相似性定义为最优碱基对应方式的寻找，这个比对将依照下述标准进行。假定有两条 RNA 序列 X 和 Y 进行比对，定义参数 δ 为空隙罚分。如果 Y 中的一个碱基和 X 中的一个空隙匹配，则将空隙罚分添加到总得分中；定义参数 s_{ij} 为匹配代价，X 中的一个碱基和 Y 中的一个的碱基匹配，不同的匹配得分不同，将匹配代价添加到总得分中。

在对 RNA 进行比对时，使用 BLAST 矩阵进行赋分，获得更好的比对效果。即被比对的两个核苷酸相同时得分为 +5，不同时得分为-4，空隙罚分为-3。

	A	U	C	G
A	+5	-4	-4	-4
U	-4	+5	-4	-4
C	-4	-4	+5	-4
G	-4	-4	-4	+5

图 1: BLAST 赋分矩阵

1. 使用动态规划算法解决问题

动态规划算法是把一个复杂的大问题切分为若干个小问题，而后通过逐一找到小问题的最优解，最终解决大问题的最优解。用 F(i,j) 表示 $x_1 \cdots x_i$（X 序列第 1 位到第 i 位）$y_1 \cdots y_j$（Y 序列第 1 位到第 j 位）。s_{ij} 表示碱基 A 和碱基 B 对应的得分，δ 为空隙罚分，则有以下递推式成立

$$F(0,0)=0$$

$$F(i,j)=\max\begin{cases}F(i-1,j-1)+s(x_i,y_j)\\F(i-1,j)+\delta\\F(i,j-1)+\delta\end{cases}$$

图 2: 递推式

Algorithm 1: 动态规划算法进行两条 RNA 序列比对

```
// k 为空隙罚分，X、Y 为进行比对的两个 RNA 序列
Input: k, X, Y
// 一个 m×n 大小的二维数组,m 和 n 分别为 X,Y 两条碱基序列的长度
Output: Array[m][n]
for i = 0 → m − 1 do
    A[i][0] ← i × k
for j = 0 → n − 1 do
    A[0][j] ← j × k
for i = 1 → m − 1 do
    for j = 1 → n − 1 do
        A[i][j] ← max(A[i − 1][j − 1] + s(x_i, y_j), A[i − 1][j] + k, A[i][j − 1] + k)
```

2. 使用分治策略优化空间复杂度

一条 RNA 约由 150 左右个碱基组成，相当于一个 150 字节的字符串，在存储中如果使用数组，约需要 150×150 大的二维数组。很显然，这样的空间代价过大，很难满足我们的需求，所以考虑将问题划分成几次递归调用，一次计算所需要的空间可以被下一次计算重用。

因为最终只需要获取两条 RNA 序列比对的值，并不需要获取最优的匹配方式，所以只需要获取数组中 A[m-1][n-1] 的值；定义的递推式也只需要 A 数组中的前一行和当前行的信息，所以可以将数组 A 压缩成 m×2 的数组，在迭代时，新数组的第一行存储 A 数组前一行的信息，第二行存储 A 数组当前行的信息。

Algorithm 2: 分治策略优化后进行两条 RNA 序列比对

```
// k 为空隙罚分，X、Y 为进行比对的两个 RNA 序列
Input: k
// 一个 m×2 大小的二维数组
Output: Array[m][2]
for i = 0 → m − 1 do
    A[i][0] ← i × k
for i = 1 → m − 1 do
    A[1][0] ← i × k for j = 1 → n − 1 do
        A[1][j] ← max(A[0][j − 1] + s(x_i, y_j), A[0][j] + k, A[1][j − 1] + k)
    for j = 1 → n − 1 do
        A[0][j] = A[1][j]
return A[0][n − 1]
```

(二) 两组病毒的 RNA 寻找最佳匹配

在这个问题中，我们需要对两组 RNA 根据其不同的匹配得分进行配对，使其达到最大的得分。

1. 使用贪心算法解决问题

A 病毒的 RNA 组有 m 条 RNA，B 病毒的 RNA 组有 n 条 RNA。因为需要对两组 RNA 分子一一比对，所以需要使用二重循环遍历。在对 A 病毒 RNA 组进行遍历的过程中，记录 A 病毒 RNA 组中一条 RNA 和 B 病毒 RNA 组中所有 RNA 匹配比对的最大值，标记出匹配最大值的 RNA 在 B 病毒 RNA 组中的位置；继续对 A 病毒的 RNA 组进行遍历，寻找和 B 病毒 RNA 组中所有未被标记 RNA 匹配比对的最大值。遍历 A 病毒的 RNA 组，将获取的匹配最大值依次累加，结果为通过一次贪心算法获取的匹配最大值。

但是，通过一次贪心算法是无法获得真正的最大匹配值，因为如果在一次取 A 病毒的 RNA 组中 i 位置的 RNA 和 B 病毒 RNA 组所有病毒比对过程中，获得 B 病毒 RNA 组 j 位置的 RNA 和 A 组 i 位置的 RNA 有一个最大的匹配值，那么下次对 A 病毒的 RNA 组中 k 位置 RNA 和 B 病毒 RNA 组所有病毒比对过程中，即使 B 病毒 RNA 组 j 位置的 RNA 和 A 组 k 位置的匹配值非常优秀（即匹配值非常大），也不能将 k 位置 RNA 和 j 位置 RNA 两者匹配，因为 j 位置的 RNA 已经匹配完成。

那么，继续对贪心算法进行改进，考虑进行多次贪心算法是否能获得最大匹配值。改变 A 病毒的 RNA 组遍历的次序，进行 n 次变化，按照 $[1 \to n-1]$,$[2 \to n-1, 1]$,$[3 \to n-1, 1 \to 2]$ 这样的次序进行 n 次贪心算法的序列比对，获得匹配值最大的一次匹配。

上述算法看上去获取的值是最大匹配值，但下面的例子可以证明，通过上述贪心算法获取的最大匹配值，在某些情况下，仍然不是最大匹配的值。

Algorithm 3: 贪心算法进行两组病毒的 RNA 寻找最大权匹配

```
// 存储 A,B 病毒 RNA 组的数组
Input: p1[m], p2[n]
Output: 最大匹配值
MaxAll ← −∞;
for t = 0 → n − 1 do
    MaxAllTemp ← 0;
    p2[i].judge ← 0 ;
    for i = 0 → m − 1 do
        MaxTemp ← −∞;
        for j = t → n − 1, 0 → t − 1 do
            if p2[j].judge == 0 then
                temp ← SequenceAlignment(p1[i], p2[j].data);
                if temp > MaxTemp then
                    MaxTemp ← temp, index ← j;
        MaxAllTemp ← MaxAllTemp + MaxTemp, p2[index].judge ← 1;
    MaxAll ← max(MaxAll, MaxAllTemp);
```

如图，两个各含有 5 条 RNA 的病毒进行相似性比对，已将其每两条 RNA 进行匹配的得分存储中一个 5×5 二维数组中

$$\begin{bmatrix} 128 & 105 & 131 & 115 & 123 \\ 142 & 136 & 132 & 125 & 129 \\ 105 & 120 & 105 & 96 & 146 \\ 127 & 109 & 108 & 122 & 90 \\ 139 & 140 & 163 & 122 & 130 \end{bmatrix}$$

图 3: 5×5 的数量 RNA 比对

按照上述算法, 获取的最大匹配为每一列取相对的最大值：142+140+131+122+146=681。

但实际上，存在更优的匹配为:128+136+163+122+146=695，按照顺序对每列取相对最大值并不能找到最佳匹配，所以改进后贪心算法也并不能求出两组病毒的 RNA 寻找最大权匹配。

2. 将问题转化为二分匹配，使用最大流设计算法

对于具有二部划分 (V_1, V_2) 的加权完全二分图,其中 $V_1 = \{x_1, x_2, x_3, ..., x_m\}$,$V_2 = \{y_1, y_2, y_3, ..., y_n\}$，边 $< x_i, y_j >$ 具有权值 $W_{i,j}$。该带权二分图中一个总权值最大的最大匹配，称之为最佳匹配。

用一个二维数组存储两组 RNA 每对匹配得分。可以将这个二维数组的不同行不同列的最大值匹配转化成一个带权的二分图的最佳匹配。如图4，两个病毒的 RNA 都为三条，使用一个 3×3 的数组存储两组 RNA 匹配得分，最终需要求出数组中不同行不同列的最大值之和，可以转化为右边的左右各有三个点的二分图，每两个点连接后产生不同的权值，最终获得最佳匹配。

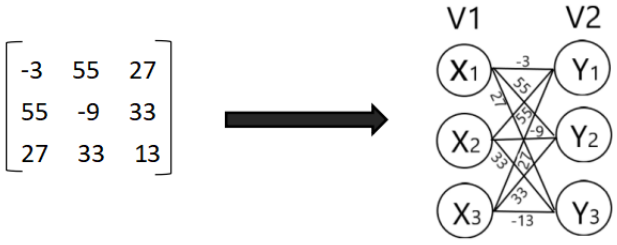

图 4: 矩阵转化成二分图

首先需要了解几个概念:

- **增广路径**

若 P 是图 G 中一条连通两个未匹配顶点的路径，并且即已匹配和待匹配的边在 P 上交替出现，则称 P 为相对于 M 的一条增广路径。增广路径有如下特性：1. 有奇数条边、2. 路径上的点一定是一个在 X 边，一个在 Y 边，交错出现、3. 整条路径上没有重复的点、4. 起点和终点都是目前还没有配对的点，其他的点都已经出现在匹配子图中、5. 路径上的所有第奇数条边都是目前还没有进入目前的匹配子图的边，而所有第偶数条边都已经进入目前的匹配子图。奇数边比偶数边多一条边、6. 于是当把所有第奇数条边都加到匹配子图并把条偶数条边都删除，匹配数增加了 1。

- **匈牙利算法**

匈牙利算法是一种寻找二部图最大匹配的最大流算法，该算法的核心就是寻找增广路径，如果寻找到一条增广路径，由于增广路径的性质，将增广路径偶数边替换为奇数边，会使匹配数加 1，直到找到最大匹配。

Algorithm 4: 获取最大匹配的匈牙利算法

```
// 存储两点是否可以配对的信息
Input: Array[m][n]
// 存储配对方式的数组
Output: match[m]
Def FindPath(i, vb[n]):
    for j = 1 → n do
        if Array[i][i] == true & vb[j] == false then
            vb[j] → 1;
            if match[j] == −1 & FindPath(match[j], vb) then
                match[j] = i;
                return true;
    return flase;
for i = 1 → m do
    memeset(vb, −1, sizeof(vb));
    FindPath(i, vb);
```

因为匈牙利算法只能计算不带权值的最大匹配，下面，尝试在匈牙利算法的基础上进行改进，实现加权二部图的最佳匹配。

建立数组 $lx[i], ly[j]$，满足 $lx[i] + ly[j] \geq w[i, j]$(w[i,j] 是 A 病毒第 i 条 RNA 和 B 病毒第 j 条 RNA 的匹配得分)，$lx[i], ly[j]$ 称为可行顶标。把所有点和满足 $lx[i] + ly[j] = w[i, j]$ 的边组合起来，形成的一个子图，称为可行子图。如果能找到一个顶标的填数方案，使得可行子图有最大匹配，那么这个匹配就是答案。

正确性证明：任何一种顶标方案下，所有完美匹配边权和必然小于等于 $\sum lx[i] + \sum ly[j]$。而根据相等子图的定义，若它有完备匹配，这个匹配的边权和等于顶标和。所以不可能有别的匹配的边权和大于这个匹配。

算法过程：初始化 $lx[i] = \max(w(i, j)), ly[j] = 0$；对 X 集合里的点遍历，寻找增广路径；如果没找到增广路满足 $lx[i] + ly[j] = w[i, j]$，就调整顶标，让更多的相等边（满足 $lx[i] + ly[j] = w[i, j]$ 的边）加入到树上，并且保持顶标的合法性。

给树中的所有属于 X 集的点每个顶标减一个小量 delta，同时给在树中的属于 Y 集的点加一个 delta。保证了原来树里的边还在树里，不在树里的边现在树里的可能性变大。

$delta = \min(lx[i] + ly[j] - w(i, j)), i \in X \wedge i \in tree, j \in Y \wedge j \notin tree$

这样原树中的边两端点一个 +delta 一个-delta，和不变，还在树中；X 在树中 Y 不在树中边只有 X 减了 delta，并且 delta 是这种情况里面能减量的最小值，保证剪完之后还满足顶标合法性。

再设置一个 slack[m] 数组，每次开始找增广路时初始化为无穷大。在寻找增广路的过程中，检查边 $< i, j >$ 时，如果它不在相等子图中，则让 slack[j] 变成原值与的较小值。这样，在修改顶标时，取所有不在交错树中的 y 顶点的 slack 值中的最小值作为 d 值即可。但还要注意一点：修改顶标后，要把所有的不在交错树中的 y 顶点的 slack 值都减去 d。

伪代码附在最后一页。

三、 分析算法

(一) 两条 RNA 匹配得分的复杂度分析

1. 时间复杂度

假设进行比对的两条 RNA 分别有 m 和 n 个碱基，其时间复杂度为 O(mn)。因为算法设计中使用了二重循环遍历，$i \in (0\ m)$,$j \in (0\ n)$，所以时间复杂度为 O(mn)。

随机输入不同序列、不同长度的 RNA 序列，测量每次两条 RNA 匹配的运行时间如图所示（横纵坐标表示进行匹配的 RNA 序列的碱基个数）

RNA 碱基个数	100	150	200	250
100	0.0020s	0.0035s	0.0040s	0.0055s
150	0.0030s	0.0045s	0.0065s	0.0090s
200	0.0045s	0.0060s	0.0085s	0.0095s
250	0.0060s	0.0090s	0.0095s	0.0115s

图 5: 实际运行时间测试

观察测量结果发现，随着 RNA 序列碱基个数增加，匹配需要的时间不断增加，并且结果符合 O（mn）的时间复杂度。如：当两条 RNA 的碱基个数从 100 变为 200 时，匹配时间从 0.0020s

变为 0.0085s 秒，约为原来的四倍，符合 O（mn）的时间复杂度规律。

2. 空间复杂度

两条 RNA 进行比对的空间复杂度为 O（m），因为在匹配过程中只需要申请一个大小为 $m \times 2$ 的数组。

(二) 两组病毒的 RNA 寻找最佳匹配

两组 RNA 的数目分别为 m,n。

1. 贪心算法

贪心算法的时间复杂度为 O(n^2m)(m,n 分别为两组 RNA 的条数)，因为算法进行了三重嵌套循环，所以复杂度为每次遍历整个 $m \times n$ 所有 RNA，进行 n 次。

空间复杂度为 O(m+n)，在贪心算法中，最大值由每次遍历比较得出，并没有将 RNA 比对的值单独存在数组中，所以，需要的空间就为存储两组 RNA 开辟的空间。

2. 最大流算法

将其转化为二分图，使用最大流解决问题的算法的时间复杂度 $O(n^3)$，因为需要找 O(n) 次增广路，每次增广最多需要修改 O(n) 次顶标，每次修改顶标时复杂度为 O(n)。所以，时间复杂度总计为 $O(n^3)$。

通过实验测量每组 RNA 变化时，系统所用的总时间变化。

RNA 条数	10	50	100
10	0.207s	1.033s	2.070s
50	1.045s	5.294s	10.445s
100	2.167s	11.365s	23.610s

图 6: RNA 碱基个数都设置为 100 时，进行的测试

可以发现，在一组 RNA 数目不变时，另一组 RNA 数目倍增，所用时间倍增；所以时间复杂度其实并没有达到 $O(n^3)$，而是介于 $O(n^2)$ 和 $O(n^3)$ 之间。

空间复杂度为 O(mn)，因为需要开辟一个二维数组，存储两组 RNA 一一匹配的值。

四、 总结分析

在本次算法设计中，应用了动态规划、分治策略、贪心算法、最大流等知识进行设计分析，解决了不同个体病毒的 RNA 的相似性比对分析问题。

RNA 比对的算法在生物信息学中有重要的应用，在对两条 RNA 进行比对时，除了动态规划算法之外，还可以使用点阵序列比较、词或 K 串方法进行。

简单介绍一下点阵序列比较的算法：点阵序列比较就是把两个序列写在矩阵两边，然后用 1 标出对应残基相等的位置，如果两条序列完全相等，那么在对角线处可以看到连续的 1，如果有局部可以匹配，也会出现对角线连线。Word size 是点阵序列的唯一参数，表示最少有几个碱基的数目相同，如果 word size 为 2 表示当对角线出现 2 个得分时即可算比对成功。相比于动态规划算法，点阵序列比较的算法更加直观，且不需要引入空隙罚分 δ。

Algorithm 5: 获取最佳匹配的算法

Input: $Array[m][n]$

Output: $match[m]$

```
1  la[i = 0 → m − 1] ← max(w[i][j = 0 → n − 1]);
2  lb[j = 0 → n] ← 0 ;
3  for x = 0 → m − 1 do
4      slack[i = 0 → m − 1] ← −∞, fa[i = 0 → m − 1] ← −1;
5      visx[i = 0 → m − 1] ← false, visy[j = 0 → n − 1] ← false;
6      con = 1, leaf = −1;
7      while true do
8          if con == 1 then
9              leaf ← FindPath, fir = 0;
10         else
11             for i = 0 → m − 1 do
12                 if slack[i] == 0 then
13                     slack[i] = −∞, leaf = FindPath;
14             if leaf>0 then
15                 break;
16         if leaf > 0 then
17             p = leaf;
18             while p>0 do
19                 match[p − m] = fa[p], p = fa[fa[p]];;
20                 break;
21         else
22             delta = −∞;
23             for i = 0 → m − 1 do
24                 if visx[i] == 1 & delta > slack[i] then
25                     delta = slack[i];
26             for i → m − 1 do
27                 if visx[i] == 1 then
28                     la[i]− = delta, slack[i] = −delta;
29             for j → n − 1 do
30                 if visy[j] == 1 then
31                     lb[i]+ = delta;
```