DeepL
订阅DeepL Pro以翻译大型文件。
欲了解更多信息，请访问www.DeepL.com/pro。

项目: 序列排列

LU3IN003 -

索邦大学2022年秋季课程

摘要

这个项目的重点是一个基因组学问题：序列排列。

从生物学的角度来看，它涉及到测量两个DNA序列之间的相似性，这些序列被简单地看作是核苷酸的序列。这使得当一个新的基因组被测序时，有可能将其基因与以前测序的基因组的基因进行比较，从而确定同源性，即由于两个物种有一个共同的祖先，而这个祖先将这个基因传给了它们，即使这个基因可能随着时间的推移发生了变异（变化），也会有相似之处。

从计算机的角度来看，核苷酸序列被看作是字母表上的字。{A,T,G,C}，我们被简化为两个文本算法问题：计算任何两个词之间的编辑距离和产生一个达到这个距离的对齐。对于每一个

为了解决这些问题，首先考虑了一种天真的算法，然后是一种动态编程算法。最后，我们用分而治之的方法来提高这些算法的空间复杂性。在开篇，我们关注的是一个稍微不同的问题：局部序列对齐。

目录

1 职权范围和 评价

- *一般组织*
  - 该项目是在**同一TD组内成**对进行的。
  - 该科目由理论部分（编号的问题）和编程部分（以字母为索引的任务）组成。
  - 对于编程部分，你可以自由选择你要使用的编程语言。
  - 你必须编写一份报告，其中包括你对理论问题的回答以及你的实验研究。必须以PDF格式提交（LATEX，Libre Office Writer，或清晰的手写扫描件...）。

- *每项任务的预期*
  - 编码和对有关算法的评论
  - 在提供的实例上测试该算法
  - 实验研究该算法的性能（计算时间是实例大小的函数，解决实例的最大大小）。
  - 将这一性能与理论研究中预期的复杂性进行比较
  - 将这一性能与其他编码算法的性能进行比较

- ***评价***

该表规定，：
  - 理论部分10分（不包括第4部分，这是一个奖励）。
  - 编程部分8分（包括3分的实验）。
  - 报告和陈述的质量得2分

- *渲染图*
  - 创建一个名为nG_nameBinome1_nameBinome2的目录，其中nG是你的组号 (1：星期五下午，2：星期四上午，3：星期二上午，4：星期一下午和星期三上午），其中nomBinome1和nomBinome2是你的姓氏。
  - 把你的程序的源文件复制到这个目录。
  - 将这个目录压缩成一个tgz文件（在linux下，你可以在终端使用以下命令：tar cvzf directoryName.tgz directoryName）。
  - 将你的报告以PDF格式导出为nG_nomBinome1_nomBinome2.pdf
  - 登录Moodle，在课程页面。在页面底部的 "项目"部分，点击你的TD组的资源库，并上传你的PDF格式的报告**和**你的源代码的档案。
  - 你可以在截止日期前删除和替换你的文件。请务必给我们留下你想要的版本。

- *防御*

防守将在你的TD槽中进行，成对进行。它们持续10/15分钟。它们包括
展示你的代码，在新的实例上运行它，并回答可能与理论部分有关的问题。

- *日历*

报告和源代码应在**11月30日**星期三午夜前提交。答辩将在**12月5日**这一周进行

。

- *数学上的评论*

本主题包括一些提供澄清的数学注释，可能有助于你
以更好地理解所介绍的概念。如果没有，你可以不理会他们。

2 序列的对齐问题

在这一部分，我们首先介绍（第2.1节）文本算法的一般定义，然后是对齐和编辑距离的概念，这是本项目的核心。

正如我们在定义了路径及其成本之后，将加权图的两个顶点之间的距离定义为从一个顶点到另一个顶点的最小成本一样，我们在这里将两个词之间的编辑距离定义为这两个词的排列的最小成本，在定义了排列（2.2节）及其成本（2.3节）之后。

2.1 文本的一些一般算法定义

定义 2.1

字母表是一个有限的字符集。

让Σ是一个非空的字母表。

Σ中的一个词是Σ中的一个有限的字符序列。我们表示Σ^*，即Σ中的词的集合。

*对于$x \in \Sigma^*$，单词x的**长度**，表示为$|x|$，是x中以倍数计算的字符数。如果$|x| = n$，x中的字符将以$[1...n]$为索引，那么单词x就写成$x_1 x_2 \ldots x_n$。*

让x和y分别是Σ中长度为m和n的两个词。

*词x和y**的连接**，表示为$x\text{-}y$，就是词$x_1 x_2 \ldots x_m y_1 y_2 \ldots y_n$*

*无论字母表如何，都有一个长度为零的单字，称为**空字**，表示为ε。*

*数学上的插曲：*将x和y联系起来的操作也叫串联。

- 由于这个操作将Σ^*的两个字发送到Σ^*的一个字，所以它被称为内部操作（这个操作的结果仍然在集合内部）。形式上：$\forall (x, y) \in (\Sigma^*)^2$，$x\text{-}y \in \Sigma^*$。
- 这种操作承认一个中性元素：空词。这意味着，通过将一个词与空字，之前或之后，我们不改变这个词。形式为：$\forall x \in \Sigma^*$，$\varepsilon\text{-}x = x\text{-}\varepsilon = x$。
- 这个操作是关联性的，也就是说，将y连接到x，然后将z*连接*到所产生的词，与先将z连接到y，然后将所产生的词连接到x是一样的。形式上：$\forall (x, y, z) \in (\Sigma^*)^3$，$(x\text{-}y)\text{-}z = x\text{-}(y\text{-}z)$。

→ 在代数中，我们说$(\Sigma^*, -)$是一个自由单体来表示这个结构。

请注意，只要字母表Σ有两个以上的字母，串联操作就不是换元的，即一般来说$x\text{-}y$和$y\text{-}x$不是同一个词。

例子：集合$\Sigma = \{a, b, c\}$是一个字母表。$x = ba$, $y = aaa$, $z = babaaa$是Σ中的单词。我们有$|x|=2$，$|y|=3$和$|z|=6$。x和y的连接是$z' = baaaa$。x与自身的串联是$z'' = baba$。

2.2 对齐两个 的单词

从现在开始，我们把Σ固定为一个非空的字母表，不包含*空格*，也就是字符-。我们表示$\Sigma_- = \Sigma \cup \{-\}$，π是$\Sigma_-^*$在Σ中的函数*，它将删除所有间隙得到的子词关联到一个词。

例如：如果$\Sigma = \{a, b, c\}$，$\Sigma_- = \{a, b, c, -\}$，$\bar{x} = c\ b\text{-}a\text{-}$，$\bar{y} = a\ b\ c$和$\bar{z} = \text{-}$ 是Σ_-^*的词。$\bar{x}$的长度分别为6，3和2，那么$\pi(\bar{x}) = cba$，$\pi(\bar{y}) = abc$，$\pi(\bar{z}) = \varepsilon$。

定义2.2

设$(x, y) \in \Sigma^* \times \Sigma^*$。让$(\bar{x}, \bar{y}) \in \Sigma_-^* \times \Sigma_-^*$。

*我们说$(\bar{x}, \bar{y})$是(x, y)的**排列**，如果*

(i) $\pi(\bar{x}) = x$

(ii) $\pi(\bar{y}) = y$

(iii) $|\bar{x}| = |\bar{y}|$

(iv) $\forall i \in [1...|\bar{x}|], \bar{x}_i = \text{-}$ 或 $\bar{y}_i = \text{-}$

对齐（$\overline{x,\ y}$）的长度将被称为$\overline{x}$的长度。

在法语中，$(\bar{x}, \bar{y})$是(x, y)的排列组合，当且仅当我们通过删除$\bar{x}$(分别是$\bar{y}$)中的所有间隙得到x（分别是y），如果$\bar{x}$和$\bar{y}$的长度相同，并且它们在同一位置没有间隙。

例子：对于$\Sigma = \{A,T,G,C\}$，x=ATTGTA和y=ATCTTA，这里有几个(x, y)排列的例子。

$\bar{x}$: ATTGTA	$\bar{x}$: AT-TGTA	$\bar{x}$: ATTGT-A	$\bar{x}$: -----ATTGTA
$\bar{y}$: ATCTTA	$\bar{y}$: ATCT-TA	$\bar{y}$: AT-CTTA	$\bar{y}$: ATCTTA-----

问题1

证明如果$(\bar{x}, \bar{y})$和$(\bar{u}, \bar{v})$分别是(x, y)和(u, v)的排列，则$(\bar{x}\bar{u}, \bar{y}\bar{v})$是$(x\text{-}u, y\text{-}v)$的排列。*证明你的答案是正确的*

问题2

如果$x \in \Sigma^*$是一个长度为n的词，$y \in \Sigma^*$是一个长度为m的词，那么(x, y)的最大对齐长度是多少？*注：最高值是指达到的主要数量。因此，你必须用一个合理的例如，你提出的长度是由至少一个排列组合实现的。*

2.3 对齐和距离的成本 编辑

$(\bar{x}, \bar{y})$的对齐方式也可以看作是描述一种使用以下三种操作（称为编辑操作）将单词x转化为y的方式。

- **插入**，包括从y中插入一个字母到单词x中，由$\bar{x}$中的一个缺口来编码。
- **删除**，包括从单词x中删除一个字母，由$\bar{y}$中的一个缺口来编码。
- 替**换**，包括将x的一个字母变为y的一个字母，由$\bar{x}$和$\bar{y}$中同一位置的两个不同字母来编码。

例子：在前面的例子中，第3个排列组合表示x可以转化为y
通过：-删除第二个T[*删除*]。
- 将G转化为C[*替换*]。
- 在最后一个T之后插入一个T[*插入*]。

从这个角度来看，例子中提出的不同排列方式是不平等的。第4个排列组合编码的转换需要5个插入和5个删除，而一个插入和一个删除就足够了，如第2个排列组合中所示。因此，第4种排列方式的成本将高于第2种。但是，为了知道在两次替换（第1次对齐）和一次替换+一次插入+一次删除（第3次对齐）之间哪个更昂贵，我们需要为这些操作中的每一个设置成本。

因此，我们设定$c_{ins} \in \mathbb{R}^+$ (resp. $c_{del} \in \mathbb{R}^+$)为插入(resp. a deletion)的费用，对于任何一对不同的字母$(a, b) \in \Sigma^2$，$c_{sub}(a, b) \in \mathbb{R}^+$为用$b$*代替*$a$*的*费用(我们用$b$代替$a$)。出于实际原因，我们允许写$c_{sub}(a, b)$，即使$a = b$。
也就是说，即使用b代替a的操作相当于什么都没做，因此也不需要花费什么。所以我们摆出$\forall a \in \Sigma$，$c_{sub}(a, a) = 0$。

定义2.3

设$(\bar{x}, \bar{y})$是长度为l的$(x, y) \in \Sigma^* \times \Sigma^*$的排列。
*$(\bar{x}, \bar{y})$排列的**成本**，用$C(\bar{x}, \bar{y})$表示，定义如下。*

$$C(\bar{x}, \bar{y}) = \sum_{k=1}^{l} c(\bar{x}_k, \bar{y}_k) \quad \text{其中} \forall (a, b) \in \Sigma^2 \setminus \{(-, -)\}, c(a, b) = \begin{cases} c_{ins} & \text{如果} a = - \\ c_{del} & \text{如果} b = - \\ c_{sub}(a, b) & \text{否则} \end{cases}$$

由于只有有限数量的双词排列，我们可以谈论双词排列的最小成本，从而定义编辑距离。

定义2.4

设（x，y）$\in \Sigma^* \times \Sigma^*$。

从x到y的编辑距离是$d(x, y) = \min C(\bar{x}, \bar{y}) \mid (\bar{x}, \bar{y})$ 是(x, y)的排列。

请注意，按照定义，d不是数学意义上的距离，它甚至不是先验的对称性。另一方面，如果$c_{del} = c_{ins}$，并且如果c_{sub}是Σ上的一个距离，那么d确实是一个距离。

例子：考虑字母表$\Sigma=\{A,C,T,G\}$，成本参数如下：$c_{del} = c_{ins} = 2$，如果$\{a, b\}$是匹配对，即$\{a, b\}=\{A,T\}$或$\{a, b\}=\{G,C\}$，则$c_{sub}(a, b) = 3$，否则$c_{sub}(a, b) = 4$。

对于x =ATTGTA和y =ATCTTA，$d(x, y) = 4$，这个编辑距离特别是通过以下最小排列实现。

$\bar{x}$: A T - T G T A

$\bar{y}$: A T C T - T A

3 序列的排列算法

在"序列排列"的名称下，实际上有两个问题：计算编辑距离和产生最小成本排列。

DIST	输入：x和y是Σ中的两个词。 输出：$d(x, y)$	辽宁省	输入：x和y是Σ中的两个词。 输出：（$\bar{x}$ $\bar{y}$）一个（x，y）的排列组合。 这样，$C(\bar{x}\bar{y}) = d(x, y)$

在本节中，提出了不同的算法来解决这两个问题中的每一个。

3.1 通过 枚举的天真方法

在这一小节中，我们考虑通过列出两个词之间所有可能的对齐方式来解决这两个问题。问题3至4旨在根据两个词的长度，计算两个词之间有多少种排列方式的可能性。

问题3

给定$x \in \Sigma^*$ 一个长度为n的词，有多少个词$\bar{x}$通过添加到x中得到的确切地说是k的差距？*换句话说，有多少个词$\bar{x}$使得$|\bar{x}| = n + k$，$\pi(\bar{x}) = x$？我们将把这个值表示为二项式系数。*

问题4

我们现在试图推导出一对词（x，y）的可能排列的数量

假设n. m，长度分别为n和m。一旦在x上加了k个缺口，得到一个词$\bar{x}$ 那么在y上将会加多少个缺口？有多少种方法可以将这些空隙插入y中，要知道这样得到的$\bar{y}$中的空隙不得与a的位置相同。$\bar{x}$的差距？推导出（x，y）的可能排列的数量。*不需要简化*

但你要用这个表达式在机器上计算出$|x| = 15$和$|y| = 10$的可能排列的数量。

问题5

一个由列举两个词的所有排列组成的天真算法会有什么样的时间复杂度，以便找到这两个词之间的编辑距离？为了找到一个最小成本的排列组合？

问题6

为了找到这两个词之间的编辑距离，一个天真的算法将列举两个词的所有对齐方式，其空间复杂度（所需内存空间的数量级）是多少？

下面我们给出一个递归函数DIST_NAIF_REC的伪代码，用于列举两个词x和y的所有可能对齐方式，它的参数是:

- 的一对词（x，y），长度分别为n和m。
- 两个指数$i \in [0...n]$和$j \in [0...m]$。
- $(x_{[1..i]}, y_{[1..j]})$的排列成本c。
- 在函数被调用之前，已知(x, y)的最佳排列的*远距离*成本。

并探索所有可能的(x, y)对齐方式，假设存在成本为c的$(x_{[1..i]}, y_{[1..j]})$对齐方式，最后返回它所探索的最佳对齐方式的成本，如果它优于$dist$，否则$dist$。

为了进行这种探索，这个函数最多进行三次递归调用，因为它考虑，如果可能的话，通过替换来扩展成本为c的$(x_{[1..i]}, y_{[1..j]})$的排列。

这导致了部分排列$(x_{[1..i]} - x_{i+1}, y_{[1..j]} - y_{j+1})$,$(x_{[1..i]} --, y_{[1..j]} - y_{j+1})$或$(x_{[1..i]} - x_{i+1}, y_{[1..j]} --)$。*我们并不关心这些部分线路，我们也不想建造它们，我们只关心它们的成本。*

当这个函数被调用时，$i = |x|$和$j = |y|$，$c < dist$，它更新了值
疏*散*到c，然后代表（x，y）的对齐成本。
我们还定义了一个函数$DIST$_NAIF，它从一对(x, y)中简单地调用DIST_NAIF_REC$(x, y, 0, 0, 0, dist)$，用于初始化为$+\infty$（空集的最小值）的变量。DIST_NAIF

输入：x和y两个词
输出：$d(X, Y)$

返回 DIST_NAIF_REC$(x, y, 0, 0, 0, +\infty)$

DIST_NAIF_REC

输入：x和y两个词。
i是$[0...|x|]$中的索引，j是$[0...|y|]$中的索引。
c对准$(x_{[1..i]}, y_{[1..j]})$的费用
在此调用之前，将已知的(x, y)的最佳对齐方式的成本*去除*。
输出：在这次调用之后，对(x, y)的最佳已知对齐方式的成本进行*区分*。

如果$i = |x|$和$j = |y|$
那么如果$(c<dist)$，那么$dist \leftarrow c$
否则
如果$(i< |x|)$和$(j < |y|)$
那么 $dist \leftarrow$ DIST_NAIF_REC$(x, y, i+1, j+1, c + c_{sub}(x_{i+1}, y_{j+1}), dist)$
如果$(i< |x|)$
那么 $dist \leftarrow$ DIST_NAIF_REC$(x, y, i+1, j, c + c_{del}, dist)$
如果$(j < |y|)$
那么 $dist \leftarrow$ DIST_NAIF_REC$(x, y, i, j+1, c + c_{ins}, dist)$

返还$Dist$

- ***尸体***

本项目的实例在Instances_genom_1.tgz档案中提供，可在
模块网站。该档案包含对第3节有用的文件。

这些实例是以非常简单的文本格式给出的。在每个文件中，前两行包含两个序列各自的长度，后两行包含字符被空格隔开的序列。

```
例子。        8                    //序列的长度y
             10                    //序列长度x
  a c t g c c t g          //序列x
  c g a t t t g c a t      //该序列y
```

在档案Instances_genom_1.tgz中，同一实例的两个序列x和y的长度$n=|x|$和$m=|y|$相当接近。此外，它们是按照n长度的增加来排序的。在你的实验测试过程中，只有在以下情况下才有可能对你的研究进行参数调整

作为n*值*的一个函数。

重要说明。在项目的所有数字实验中，项目中使用的字母是相同的。
*sub*系统的成本参数将是$\Sigma = \{A,C,T,G\}$，成本参数如下：$c_{del} = c_{ins} = 2$，如果$a = b$，$c_{sub}(a, b) = 0$，如果$\{a, b\}$是匹配对，即$\{a, b\} = \{A,T\}$或$\{a, b\} = \{G,C\}$，$c_{sub}(a, b) = 4$，否则。

- ***数据结构***

虽然这个项目的编程语言是免费的，但你需要实现
的功能，使它们与它们的理论复杂性相对应。要做到这一点，遵守以下两个准则是很有用的。

1. 一个实例*的*词x和y可以用两个数组X和Y*来*编码，这两个数组定义了访问权限

快速索引（以$O(1)$为例）。它们在功能期间不会被修改。然而，能够引用（没有复制成本）一个子字$x' = x_{[i+1..j+1]}$是很有用的，所以这里指的是子数组$X' = X_{[i..j]}$，其索引0单元格是$X[i]$单元格，索引$j-i$单元格是$X[j]$。
在Python语言中可以直接提取子表，但要注意
到指数。事实上，对于一个Python数组T，数组$T' = T[i:j]$表示从i到$j-1$的单元。这种提取在C语言中也是直接可行的：对于一个数组$char * T$，如果我们考虑参考$T' = T + i$，那么$T'[0..j-i-1]$正是这个子数组。

2. 一个排列组合（$\overline{x\ y}$）可以用两个双链列表来编码，以便
有一个快速的连接函数（例如，在$O(1)$中）。请注意，根据你选择的语言，列表复制可能会非常耗费内存和CPU，一个

传递一个列表作为参数会导致整个列表被复制。在这个项目的所有函数中，完全可以对一个列表的单一副本进行操作，即使这意味着在列表中加入一个前导（或后导）元素，之后再删除这个元素。

任务A

- 对函数DIST_NAIF和DIST_NAIF_REC进行编码。
- 在编辑距离为10、8和2的Inst_0000010_44.dna、Inst_0000010_7.dna和Inst_0000010_8.dna
实例上测试你的实现是否有效。
- 为了评估这种方法的性能，请考虑你能在一分钟内解决所提供的实例，有多大的实例。
- 估计运行这个方法所需的内存消耗。在内存消耗只在程序开始时进行的情况下，这可能是

只需在执行过程中发出linux命令top即可完成（top的显示通过按空格来更新）。也有可能是要求的内存消耗太高，然后系统就会停止程序。

3.2 编程 动态

3.2.1 通过动态编程计算编辑距离

对于以下问题，考虑（x，y）$\in \Sigma^* \times \Sigma^*$ 一对各自长度的词
n和m。对于$i \in [0...n]$和$j \in [0...m]$，我们引入以下两个符号。

$$D(i,j) = d(x_{[1..i]}, y_{[1..j]}) \text{ 和 } Al(i,j) = \{(\bar{u}, \bar{v}) \mid (\bar{u}, \bar{v}) \text{ 是}(x_{[1..i]}, y_{[1..j]})\text{r}\text{的排列组合}$$

因此，D（n，m）$=d$（x，y）。

问题7

设$(\bar{u}, \bar{v})$是长度为l的$(x_{[1..i]}, y_{[1..j]})$的排列组合。如果$\bar{u} = -, \bar{u}$ 是什么？如果$\bar{v} = -$，什么是$\bar{u}$?如果 $\bar{u} /= -$，$\bar{v} /= -$，那么 $\bar{u}$ 和 $\bar{v}$ 是什么？*迅速说明理由。*

问题8

区分问题7中考虑的三种情况，用$C(\bar{u}, \bar{v})$表示$C(\bar{u}_{[1..l-1]}, \bar{v}_{[1..l-1]})$。
不期望有任何理由。

问题9

对于$i \in [1...n]$和$j \in [1...m]$，从问题7和8中推导出$D(i, j)$*的*表达式，即从条款$D(i', j')$，其中$i' \neq i, j' \neq j$和$(i', j') /= (i, j)$。*证明你的答案是正确的。*

问题10

什么是$D(0, 0)$？*证明这一点。*

问题11

对于$j \in [1...m]$，$D(0, j)$是什么？对于$i \in [1...n]$，什么是$D(i, 0)$？*说明理由。*

问题12

根据问题9、10和11的答案，给出一个名为DIST_1的迭代算法的伪代码，该算法将两个词作为输入，用D的所有值填充一个二维数组T，最后返回这两个词的编辑距离。

问题13

DIST_1算法的空间复杂性是什么？

问题14

DIST_1算法的时间复杂度是多少？请*简要说明理由。*

3.2.2 通过动态编程计算出最佳排列方式

本小节的目的是产生一个已经知道所有D值的最优排列，因为我们刚刚看到了如何计算它们。

对于$i \in [0...n]$和$j \in [0...m]$，我们在前面的符号中增加了一个最佳排列组合的符号。

$Al^*(i,j) = \{(\bar{u}, \bar{v}) \mid (\bar{u}, \bar{v})$ 是$(x_{[1..i]}, y_{[1..j]})$的排列，这样$C(\bar{u}, \bar{v}) = d(x_{[1..i]}, y_{[1..j]})\}$

问题15

让（i，j）$\in [0...n] \times [0...m]$。表明：

- 如果$j>0$，且$D(i,j)=D(i,j-1)+C_{ins}$，则$\forall (\bar{s}, \bar{t}) \in Al^*(i,j-1), (\bar{s}, \bar{t}y) \in Al^*(i,j)$
- 如果$i>0$且$D(i,j)=D(i-1,j)+C_{del}$，则$\forall (\bar{s}, \bar{t}) \in Al^*(i-1,j), (\bar{s}x, \bar{t}) \in Al^*(i,j)$
- 如果$D(i,j)=D(i-1,j-1)+C_{sub}(x_i, y_j)$，那么$\forall (\bar{s}, \bar{t}) \in Al^*(i-1,j-1), (\bar{s}x, \bar{t}y) \in Al^*(i,j)$

你可以选择只发展三种情况中的一种。

问题16

给出一个名为SOL_1的**迭代**算法的伪代码，该算法从一对词（x，y）和一个由$[0...|x|] \times [0...|m|]$索引的包含$D$值的数组$T$中，返回一个（$x$，$y$）的最小排列。

问题17

通过结合算法DIST_1和SOL_1，我们用什么时间复杂度来解决ALI问题？

问题18

通过结合DIST_1和SOL_1的算法，我们以怎样的空间复杂度来解决这个问题？
ALI？

任务B

- 编写两个函数DIST_1和SOL_1，以及一个函数PROG_DYN，该函数只接受单词x和y作为输入，并返回距离$d(x,y)$和最佳排列。测试
在几个实例上的这些功能。
- 绘制CPU时间消耗与所提供的一对实例的第一个字的大小$|x|$的函数。得到的曲线是否与理论上的复杂度相符？
你可以将自己限制在每次花费不到10分钟的实例中。
- 估计PROG_DYN对一个非常大的实例所使用的内存量。

3.3 改善 距离计算的空间复杂性

在这一小节中，我们通过略微修改DIST_1算法，提出了一种具有线性空间复杂性的编辑距离计算算法。这是动态编程算法的一个经典改进。

问题19

解释一下为什么在DIST_1算法中填充表T的第$i>0$行时，只要能访问表的第$i-1$行和第i行就足够了（对后者来说是部分填充）。

问题20

利用前一个问题的注释，给出**迭代**算法DIST_2的伪代码，该算法的规格与DIST_1相同，但具有线性空间复杂性（以$\Theta(m)$为单位）。

任务C

- 对DIST_2函数进行编码并在几个实例上进行测试。
- 绘制CPU时间消耗与所提供的一对实例的第一个字的大小$|x|$的函数。得到的曲线是否与理论上的复杂度相符？

你可以将自己限制在每次花费不到10分钟的实例中。

- 将结果与之前的DIST_1函数的结果进行比较。
- 估计DIST_2对一个非常大的实例所使用的内存量。

3.4 通过 "分而治之 "的方法改善光学对准计算的空间复杂性

在上一小节中，我们注意到只有最后的计算值对计算下面的值有用，因此改进了编辑距离计算的空间复杂性。然而，在计算SOL_1中的最佳排列时，我们使用的是最开始计算的数值，因为我们在T表中回到了第一个单元。事实上，我们使用

潜在的数组的所有单元，即对于每个单元$(i,\ j) \in [0...n]\times[0...m]$，我们可以找到一个实例，在重构a时读取该单元中写入的值

最佳排列。因此，需要一种不同的方法来减少空间的复杂性。

我们将在这里使用**分而治之**的方法。其思路是将x分解为$x^1 \cdot x^2$，将y*分解*为$y^1 \cdot y^2$，然后递归计算(s, t)为(x^1, y^1)的最优排列，(u, v)为(x^2, y^2)的最优排列，最后将它们连接起来，即返回$(s\cdot u,\ t\cdot v)$，根据问题1，这的确是一个(x, y)的排列。只有在$|x| \geq 2$和$|y| \geq 1$的情况下才会对x和y进行分割，其他情况将直接处理，不需要递归调用。以下两个问题是为了更容易处理这些基本案例。

问题21

给出一个函数mot_gaps的伪代码，给定一个自然数k，返回由k个间隙组成的词。

问题22

给出一个函数align_word_letter的伪代码，该函数给定x为长度为1的词，y为任何长度的非空词，返回(x, y)的最佳排列。

分割和征服的理念是将x分解为$x^1 \cdot x^2$，将y分解为$y^1 \cdot y^2$。然而，如果我们把x和y分别切在中间，结果的最优性就不能得到保证，因为

我们将在下一个问题中证明这一点。

问题23

考虑一个例子，Σ是拉丁字母，由26个大写字母组成，$x = BALL$ 和$y = ROUND$。将x和y在中间切开：$x^1 = BAL$，$x^2 = LON$，$y^1 = RO$，$y^2 = ND$。

我们设 $c_{del} = c_{ins} = 3$，而 $c_{sub}(a, b)$ = 0，如果$a = b$；3 如果a和b是两个不同的元音；5 如果a和b是两个不同的辅音；7 否则

给予 $(\bar{s}, \bar{t})$是(x^1, y^1)的最优排列，$(\bar{u}, \bar{v})$是(x^2, y^2)的最优排列。

证明（$\overline{su}$, $\overline{tv}$）不是（x，y）的最优排列。*不需要论证其最优性*

$(\bar{s}, \bar{t})$*和*$(\bar{u}, \bar{v})$。

为了克服这个缺点，在中间切割x后，我们将以这样的方式切割y，使之对准 x^1 与 y^1 和 x^2 与 y^2 导致（x, y）的最佳排列。为了正式确定这一点，我们引入了与给定索引i^* 相关的*切割*概念，这个术语表示当我们将x切割成i^* （$x^1 = x_{[0..i^*]}$和$x^2 = x_{[i^*+1..|x|]}$）时要切割y的索引。在实践中，我们将使用$i^* = \frac{|x|}{2}$。

定义3.1（*针对本项目的先验定义*）

让x和y分别是Σ中长度为$n>1$和$m \geq 1$的两个词。让$i^* \in [0...|x|]$。让$x^1 = x_{[0..i^*]}$和$x^2 = x_{[i^*+1...n]}$。

让$j^* \in [0...|y|]$。让$y^1 = y_{[0..j^*]}$和$y^2 = y_{[j^*+1...m]}$。

*如果存在$(\bar{s}, \bar{t})$一个(x^1, y^1)的排列，那么索引j^*就是与(x, y)的i^*相关的**截止点**。*

和$(\bar{u}, \bar{v})$是(x^2, y^2)的一个排列，这样$(\overline{su}, \overline{tv})$是$(x, y)$的最小排列。

例子：考虑字母表{A,C,T,G}，成本参数如下：$c_{del} = c_{ins} = 2$。

$c_{sub}(a, b) = 1$，如果$a \neq b$，否则为0。

对于x=AGTACGCA和y=TATGC，$d(x, y)$=7，这个编辑距离尤其是通过以下最小排列实现的。

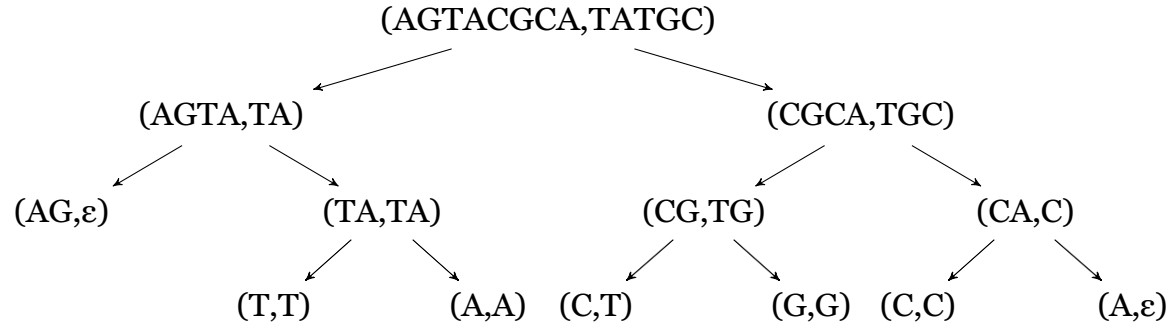
$\bar{x}$: A G T A C G C A

$\bar{y}$: - - T A T G C -

因此，与(x, y)的$i^* = 4$相关的截点是$j^* = 2$，即在之前的最佳排列中，x的前4个字母，即AGTA，与

y，即TA。

首先，我们假设有一个切割函数，从一对长度分别为$n>1$和$m \geq 1$的词（x，y）中，返回一个索引 j^*，这是一个与$i^* = \frac{|x|}{2}$ 相关的（x，y）的切割。然后，我们可以通过递归来构建（x，y）的最优排列，如上面例子中的一对（x，y）的递归调用树所示。通过从左到右串联在叶子层面上获得的排列组合，可以得到（x，y）的最佳排列组合。



问题24

假设有截断函数，请给出递归分割和征服算法的伪代码，该算法名为SOL_2，从一对字（x，y）中计算出（x，y）的最小排列。

我们现在将看到如何计算一个截止日期。我们的想法是，在计算$D(i,j)$值时，要记录切割的情况。

例子：考虑字母表$\{A,C,T,G\}$和以下费用：$c_{del} = c_{ins} = 2$, $c_{sub}(a,b) = 3$ 如果$\{a,b\}$是一个匹配对，即$\{a,b\}=\{A,T\}$或$\{a,b\}=\{G,C\}$和c_{sub} (a,b)=4，否则。

对于x=ATTGTA和y=ATCTTA，我们有$d(x,y) = 4$，这个编辑距离尤其是通过以下最小排列实现的。

$\bar{x}$: A T - T G T A

$\bar{y}$: A T C T - T A

表中给出了$i \in [0...6]$的$D(i,j)$的值。

和$j \in [0...6]$。箭头表示，对于每个单元格（i，j），哪个单元格通过公式获得了D（i，j）的值

问题9中的递归原则。

从视觉上看，如果我们在表格D上表示支票

从($|x|$，$|y|$)到(0，0)的最佳排列的最小值(粗体箭头)，确定与i^*有关的截断j^*，包括确定该路径与i^*的哪一列j^*相交。

要做到这一点，从计算指数线i^*

($i^* = \lfloor \frac{|x|}{2} \rfloor = 3$为例)，我们在一个表格中保留了

I，对于每个单元格（i，j），其所在列的索引k为

从（i，j）到（0，0）的最佳路径与直线i相交*。在对面的表格中，记录的指数显示为

第二条线。

因此，我们可以在方框（5,4）中读到，$k = 2$是与$i^* = 3$相关的一对（ATTGT,ATCT）的切割。在框（6,6）中，我们读到$k = 4$是与$i^* = 3$相关的(x, y)的截断：因此截断(x, y)返回4。

请注意，这种方法是通过只保留表的i-1和i行来实现的。

D和I，否则我们在内存消耗方面就不经济。

		0	1	2	3	4	5	6
			A	T	C	T	T	A
0		0	2	4	6	8	10	12
1	A	2	0	2	4	6	8	10
2	T	4	2	0	2	4	6	8
3	T	6	4	2	4	2	4	6
		0	1	2	3	4	5	6
4	G	8	6	4	5	4	6	8
		0	1	2	2	4	4	4
5	T	10	8	6	7	5	4	6
		0	1	2	2	2	4	4
6	A	12	10	8	9	7	6	4
		0	1	2	2	2	4	4

问题25

如上所述，请给出一个截断函数的伪代码。

问题26

截断的空间复杂性是什么？请*简要说明理由。*

问题27

SOL_2的空间复杂性是多少？请*简要说明理由。*

问题28

切断的时间复杂性是什么？请*简要说明理由。*

任务D

- 对这种"分而治之"的方法所对应的函数进行编码。
- 绘制CPU时间消耗与所提供的一对实例的第一个字的大小$|x|$的函数。*你可以将自己限制在需要少于10分钟的实例中每一个。*
- 估计运行这个方法所需的内存消耗。

问题29

我们是否通过提高空间复杂性而失去了时间复杂性？*通过实验比较*SOL_2*和*SOL_1*的时间复杂性。不需要*

理论上的证明。

4 一个扩展：局部序列排列 (Bonus)

到目前为止，我们一直试图将一个词x的整体性与一个词y的整体性对齐。如果x和y是以前确定的基因，这就说得通了：我们正试图找出这两个基因之间是否有很多进化。另一方面，如果我们要在一个完整的基因组x中寻找一个基因y的祖先，那么单词x和y的全局排列是不适合的。
事实上，如果x的长度与y的长度相比非常大，而且字母表很小（如$\{A,T,G,C\}$），我们几乎可以肯定，通过在y*中*插入尽可能多的间隙，可以使y中的每个字母与x中的相同字母对齐。这就是在对齐中所做的事情
下面。这导致了成本排列$(|x|-|y|)\,c_{del}$，即最小的
因为在$(\bar{x}, \bar{y})$的排列中，(x, y)的间隙数是可以做到的。 $\bar{y}$是
$|\bar{y}|-|y| = |\bar{x}|-|y| = |x|-|y|$。

$\bar{x}$ A T T G C T C A T T C A G T A C

$\bar{y}$: - T - - C - - A - - - G T - C

$\bar{x}$ A T T G C T C A T T C A G T A C

$\bar{y}$: - - - - - T C A G T C -

相反，一个允许少量替换以避免有许多间隙的y错位的排列方式，其成本较高，尽管它更相关。
在上面的例子中，右边的建议排列显示，*X基因组*中的一个基因（即TCATTC）可能通过一次替换而进化成了*Y*。

问题30
通过生成一些实例，通过实验发现，全局排列的成本为
(x, y)当$|y|<<|x|$(相对于$|\Sigma|$)是$(|x|-|y|)c_{del}$。

在项目的这一部分，我们想知道x和y这两个词的哪些部分或*因素是*需要对齐的。*在本节末尾不需要执行。*

定义4.1

如果一个词y是由x中的连续字母组成的，那么它就是x的一个因子。
即如果存在$(i, j) \in [1...|x|]^2$，*使得*$y = x_{[i..j]}$。

要做到这一点，第一个想法是考虑$\bar{x}$和$\bar{y}$这两个词的开头和结尾的空隙成本为零。这将很容易从第三部分提出的解决方案中实现。对于开头的空白，只需改变基本情况：$T[i][0]$和$T[0][j]$将是0。对于结尾的空白，只需取最后一行和一列的最小值：$T[i][|y|]$是将$x_{[1..i]}$与y对齐的成本，因此通过将$\bar{x}$与y完成对齐的成本是
缝隙，不需要额外费用；同理，$T[|x|][j]$也是如此。

问题31
这看起来是个好主意吗？*例如，你可能会问，正如你在问题2中提出的例子，较长的排列方式的成本会是多少。*

第二个想法是，不仅要对插入、删除和替换进行惩罚，还要对两个相同的字母的排列进行支付，我们将其称为协整。与其说是最小化C，一个正的成本函数，我们不如说是最大化S，一个函数**分数**，定义与C相同，但参数不同：$c_{ins} < 0, c_{del} < 0, c_{sub}(a, b) < 0$ if $a \neq b$
和$c_{sub}(a, a) > 0$ for $a \in \Sigma$。
这将确定一个排列组合是不相关的，它是被迫的，当它有一个得分时
负的，或者相反，如果分数是正的，或者足够大，则是有趣的。例如，下面的排列（$\bar{p}, \bar{q}$）和（$\bar{s}, \bar{t}$）具有相同的成本。然而，$(\bar{p}, \bar{q})$似乎是一种被迫的排列（没有字母匹配），而$(\bar{s}, \bar{t})$似乎是

相关的，因为它突出了一个共同因素。因此，该*费用*将那些被判定为质量非常不同的排列组合放在同等地位上。相反，*分数*反映了这种质量上的差异，因为（$\bar{p}$，$\bar{q}$）会有一个严格意义上的负分，而（$\bar{s}$，$\bar{t}$）对于选择好的参数会有一个正分（例如$c_{ins} = c_{del} = -2$，而$c_{sub}(A, A) = c_{sub}(T, T) = 5$）。

$\overline{p}$: AAAA- -
- -
$\overline{q}$: - - - TTT

$\overline{s}$ C C G A T T - - - -
$\overline{t}$: - - - A T T C G T

此外，匹配的报酬意味着最短的排列并不总是最好的，而距离的情况则不是这样。因此，具有最佳分数排列的x和y的一对因子（x'，y'）是有趣的，而（ε，ε）总是一对因子

与最佳成本对准，而且没有任何意义。在上述例子中，(s, t)的最佳因素对将是(ATT, ATT)。

因此，局部对齐问题正式定义如下。

BEST_SCORE 输入：X和Y两个词在Σ中。

输出：$S^*(x, y) = \max S(\bar{x}\bar{y})$ | $(\bar{x}\bar{y})$是(x', y') x'是x的一个因子。y'是y的一个因子

辽宁省商务厅 | 输入：x和y是Σ中的两个词。
输出：$(\bar{x}\bar{y})$一个(x, y)的局部对齐，得分$S^*(x, y)$

问题32

在第三部分的基础上，BEST_SCORE和ALI_LOC问题如何能在空间复杂度$\Theta(n\ m)$中得到解决？有什么时间上的复杂性？那么，我们是否可以通过分而治之的方法来降低空间复杂度而不降低时间复杂度呢？*希望有一个相对简短的答案，特别是什么可以调整，如何调整，相反，什么不能调整，为什么？*