
基于折叠式散列映射的 K-mer Index 方法

指导老师：董志清 学生：吴佩洁 史伟 何航宇

重庆交通大学

摘 要

本文是关于 DNA 序列的 K-mer Index 的问题，通过设计了一种基于折叠式散列映射的数据索引方法，并用 C++ 编程语言设计了索引程序，实现了对固定 k 值的目标碱基片的快速查找与定位。

针对问题一，本文首先对题目所给文件进行了一个预处理，即将文件中的碱基序列转化为进制数。然后对每个 k-mer 按顺向等量划分进行折叠式运算，再对运算结果进行地址映射，得到了一个有序的地址存储单元。在地址存储单元中，把折叠法计算所得数值作为哈希表的行信息，相同行的序列地址信息通过新的数组存储原则进行记录。最后，将所要查询的 k-mer 通过同样的处理，即可得到相应的位置信息，从而实现了对目标碱基序列的 DNA 序列编号以及相应序列出现位置的查询。

针对问题二，本文通过采用逆向匹配的方式对所得的表行所有信息进行筛选，能够达到算法快速高效的目的。又因为折叠系数直接影响内存占用量，因此，通过分析实验数据，确定合适的折叠系数，将会使内存占用量尽量小。

针对问题三和问题四，本文通过分析程序语句，运用数据结构中计算复杂度的方法，得出以下结论：建立索引的时间复杂度为 $O(kn)$ ，空间复杂度为 $O(n)$ ；使用索引查询的时间复杂度为 $O(n)$ ，空间复杂度为 $O(1)$ 。

针对问题五，在内存限制为 8G 的情况下，本文所设计的数据索引方法能支持的最大 k 值为 100。其对应的数据查询效率最高，建立索引的时间约为 2.14 秒，使用索引时间约为 0.021 秒。

针对问题六，本文通过建立灰色层次分析的模型，对题目所给四项指标的重要性进行了客观的权重计算，通过大体分析锁定折叠系数的范围，进而对每个 k 值下的折叠系数候选数作出综合评价，得到每个 k 值所对应的最优折叠系数。

根据问题六得出的结论，我们运用程序算法，对特定的 k 值自动匹配最优执行效率的折叠系数。当用户键入一个 k 值时，程序接收到指令之后会自动匹配相应的最佳折叠系数进行 DNA 序列的查找定位工作，从而使得本数据索引方法更加的高效、完善。

关键词：折叠系数；折叠式散列函数；地址映射；逆向匹配；灰色层次分析法

1 问题重述

这个问题来自 DNA 序列的 k-mer index 问题。

给定一个 DNA 序列，这个序列只含有 4 个字母 ATCG，如 $S = \text{“CTGTACTGTAT”}$ 。给定一个整数值 k ，从 S 的第一个位置开始，取一连续 k 个字母的短串，称之为 k-mer（如 $k=5$ ，则此短串为 CTGTA），然后从 S 的第二个位置，取另一 k-mer（如 $k=5$ ，则此短串为 TGTAC），这样直至 S 的末端，就得一个集合，包含全部 k-mer。如对序列 S 来说，所有 5-mer 为

$\{\text{CTGTA, TGTAC, GTACT, TACTG, ACTGT, TGTAT}\}$

通常这些 k-mer 需一种数据索引方法，可被后面的操作快速访问。例如，对 5-mer 来说，当查询 CTGTA，通过这种数据索引方法，可返回其在 DNA 序列 S 中的位置为 $\{1, 6\}$ 。

问题：

现在以文件形式给定 100 万个 DNA 序列，序列编号为 1-1000000，每个基因序列长度为 100。

(1) 要求对给定 k ，给出并实现一种数据索引方法，可返回任意一个 k-mer 所在的 DNA 序列编号和相应序列中出现的位置。每次建立索引，只需支持一个 k 值即可，不需要支持全部 k 值。

(2) 要求索引一旦建立，查询速度尽量快，所用内存尽量小。

(3) 给出建立索引所用的计算复杂度，和空间复杂度分析。

(4) 给出使用索引查询的计算复杂度，和空间复杂度分析。

(5) 假设内存限制为 8G，分析所设计索引方法所能支持的最大 k 值和相应数据查询效率。

(6) 按重要性由高到低排列，将依据以下几点，来评价索引方法性能：

- 索引查询速度；
- 索引内存使用；
- 8G 内存下，所能支持的 k 值范围；
- 建立索引时间。

2 问题分析

2.1 问题的研究背景

本题是有关 DNA 序列的 k-mer index 的问题，即 DNA 序列比对问题。

随着近些年来测序技术的飞速发展，人类获得了海量的生物序列数据，亟需通过有效的计算手段进行分析和处理^[1]。而在众多的生物序列分析与处理问题中，生物序列数据的 k-mer 的快速索引是一种非常关键和重要的技术。如何加快 DNA 序列比对速度对满足当今生物信息学的需求与发展有着相当重大的意义。因此，随着 DNA 序列比对速度的提高，我们将可以从海量的生物信息中发现重要的信息，从而使人类的生活和健康产生质的飞跃。

2.2 问题的分析

2.2.1 问题一、二的分析

本题要求对给定的 k 值，给出并实现一种数据索引方法，可返回任意一个 k-mer 所在具体位置，即其所在 DNA 序列编号和相应序列中出现的位置。该问题的关键在于：如何能够使得建立和使用数据索引方法的查询时间尽可能的少，占用内存尽可能的少，在同时考虑时间复杂度和空间复杂度的情况下，找到最合适的索引方法。

基于本题的问题要求以及 DNA 序列只有四种碱基（即 ATCG）的生物序列特征，本文设计了一种基于折叠式散列映射的数据索引方法。首先对信息文件预处理，之后对碱基进制转换得到的序列数（ $A \rightarrow 0$ ， $T \rightarrow 1$ ， $C \rightarrow 2$ ， $G \rightarrow 3$ ），然后采用折叠式的散列函数（哈希函数），对 k-mer 依次映射到对应的地址单元进行存放相应的位置信息。建立索引后，程序对后面输入查找的具体的 k-mer，按上述方法同理映射，对索引到的表行所有信息相应的 k-mer 采用逆向匹配的方式筛选得出正确的 k-mer 位置信息。通过此种建立索引的方法，实现了对固定 k 值的任一碱基片段的精确与快速的查找与定位。

2.2.2 问题三、四的分析

本题要求分析建立索引和使用索引分别对应的计算复杂度和空间复杂度。其中，计算复杂度相当于时间复杂度，是指执行算法所需要的计算工作量，通常与算法的语句执行次数（时间频度）有关系。而空间复杂度是指在计算机内执行时所需存储空间的度量。而算

法的空间复杂度通常与算法程序所占的空间、输入的初始数据所占的存储空间、算法执行过程中所需要的额外空间有关。

2.2.3 问题五、六的分析

针对问题五，因为本文采用的是折叠式散列映射，所以在内存限制为 8G 的情况下，本文设计的程序支持的 k 值可以远大于 100，其相应的查询效率和折叠系数呈现正相关。

针对问题六，本文通过建立灰色层次分析的模型，对题目所给四项指标的重要性进行了客观的权重计算，通过首先大体分析确定折叠系数的合适范围，进而具有代表性的折叠系数在不同的 k 值下作出综合评价，根据得出不同折叠系数在不同 k 值得出的评价值，从而得出在不同 k 值范围内所对应的最优折叠系数的结论，最后根据其结论通过程序实现自动反馈。即当用户键入需求的 k 值时，程序接收到需求之后会自动匹配相应的最佳折叠系数进行 DNA 序列的查找定位工作，从而使得本数据索引方法更加的高效、完善。

2.3 算法流程图

本文的算法流程如图 1 所示（请见下一页）。

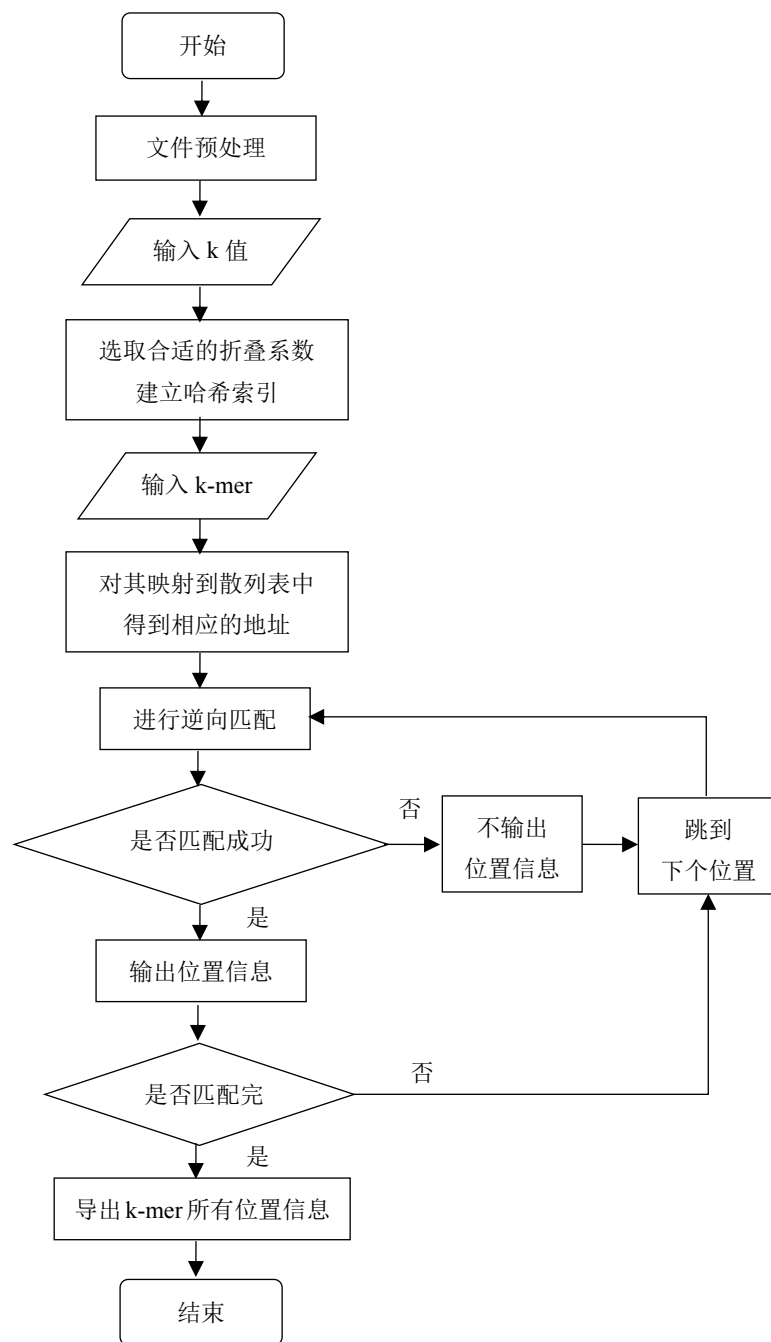


图 1 算法的流程示意图

3 变量说明

变量	变量说明
折叠系数 (Fold Degree, FD)	把模式串依次划分为相等片段的片段长度
指针变量 DNA	用于存储一亿个 DNA 的基因序列
字符数组 prn (模式串字符)	用于存储键入的模式串字符
Conver 函数 (转换函数)	用于将每一次指针指向的 k-mer 基因串转换为折叠式的编号
二维指针 hash	用二维指针创建哈希表, 用于存储在建立索引之时得到的 k-mer 的位置信息
文件流 f (抽象文件)	用文件流来联系 DNA 序列文件
指针 number (哈希行中的元素)	用来存储哈希表中每一行的元素个数
变量 start, finish (clock_t 类型)	用于计算建立索引以及索引的执行时间
顺向 (逆向) 存储	用于存放位置信息的数组在散列表某一行是依次从左向右 (从右向左) 的方式进行存储
H	建立的哈希表的行数
L	建立的哈希表的列数
L'	实际建立的哈希表的列数
B	均值遍历数
E	额外内存空间占用量

4 算法分析

由于本文将通过基于折叠式散列映射的方法,来解决该类 DNA 序列 k-mer index 问题,因此我们将根据我们建立索引和使用索引的具体实现步骤来介绍这种实现方法。建立索引和使用索引主要由以下五个部分组成:进制转换、散列表的建立、折叠式散列函数、数组存储方式、逆向匹配。

3.1 进制转换

题目中所给出的文件是由 100 万个 DNA 序列组成的，并且每个 DNA 序列都有 100 个碱基序列，因此，总共有 1 亿个碱基序列。面对数量如此巨大的生物 DNA 遗传信息，我们需要想办法将这些生物数据全部导入计算机，并且尽可能地减少数据在计算机所占的存储空间，从而为后面建立索引提供一个很好的铺垫，提高建立索引的速度，减少不必要的时间消耗。

令碱基对 A->0, T->1, C->2, G->3, 将四个碱基转换为进制序列数。例如，文本串 S=“CTGTACTGTAT”，经过该进制转换的方法，转化为了 S’=“21310213101”。这样便于后面折叠算法的计算，加快了计算速度，提高了建立索引的速度，节省了建立索引的时间损耗。

3.2 散列表的建立

通过后面基于灰色层次分析模型得到的对不同 k 下最优折叠系数的大体确立，这里具体取折叠系数为 5 来说明。当折叠系数为 5，根据 3.3.2 节“折叠法的原理及应用”，那么该散列表共设置 10^5 行，即 0~9999，这个范围便涵盖了所有 k-mer 的可能折叠结果。根据题意一共有 100w 个序列，每个序列为 100，那么便有 $(100-k+1) \times 100$ 个 k-mer，因为每个 k-mer 的映射结果具有随机性，所以若将其平均分到每一行，则每一行大概有 10~1000 个相应的数组信息，但是事实上这样的均匀分布可能性不大，所以此处的存储进行了一个灵活方式的改动，改变了以往的信息缺失及可能的大量空间的未加利用，见 3.4.2 节“数组的存储原则”。为了避免空间的浪费，这里散列表具体列数的确定方法是根据给定 k 计算出平均分配到每一行冲突值的数量，对得到的数取整加 3 作为列数选取标准，此中除了进一，剩下的两列便是用来存放散列表的行首和行末指标数，表示该行数。类似的，折叠系数为 3 和 6 下散列表的确立同理。

3.3 折叠式散列函数

3.3.1 单向散列函数

基于本题的 k-mer index 问题，最合适的建立索引的方法就是使用单向散列函数（又称哈希函数），建立哈希表（Hash Table）。所谓哈希表，也就是一个大的数组，这个数组的容量根据程序的要求来定义。每一个哈希值通过运算对应到数组中的一个位置，这样，只

要比较这个字符串的哈希值对应的位置，就可以得到最后想要的结果，因此，使用哈希表的速度是最快的。

单向散列函数（哈希函数），它可以将任意长度的消息散列为固定长度的哈希值，哈希值也被称为消息摘要、数字指纹等，因此哈希函数能够赋予每个消息唯一的“指纹”。除此之外，哈希函数在数据库查询技术中，应用也相当广泛。它是在查找表中记录的关键字和存储位置之间，建立一个确定的函数关系，即将关键字为 k 的记录存储在 $H(k)$ 的位置上， $H(k)$ 即是哈希函数。在查找时，若表中存在关键字和给定 k 相等的记录，则必定在对应的存储位置上，不需进行比较便可直接存取相应的记录，使得查找速度相当快^[3]。

定义 1-1：散列函数（哈希函数）是一个映射。

$$h: \{0,1\}^n \rightarrow \{0,1\}^n \quad \text{式 (1-1)}$$

其中， $\{0,1\}^n$ 表示任意长度的比特串集合， $\{0,1\}^n$ 表示长度为 n 的比特串集合。消息 $x \in \{0,1\}^n$ 的像 $h(x)$ 称为 x 的哈希值。

从散列函数（即哈希函数）的定义中可以看出，哈希函数其实是一个压缩函数，因此可以实现对任意长度消息的简化处理以及对消息特征的记录。

在本题中，我们可以把每个长度为 k 的 k -mer 看做是一个长度为 k 的信息集合。因为每个 DNA 序列中有 100 个碱基对，而题目中定义的 k -mer 为从第一个位置连续取长度为 k 的碱基序列，因此每个 DNA 序列中一共有 $(100 - k + 1)$ 个 k -mer，即有这么多个长度为 k 的信息集合。因此，我们便需要建立一个合适的单向散列函数（哈希函数），使得每个 k -mer 对应一个特定的哈希值。然后根据哈希表的建立方法，以每个 k -mer 的哈希值作为哈希表的表头，通过构建适用于此问题的哈希表，来实现对无序、没有规律的 k -mer 集合的空间存储问题，转化为有序、有规律的 k -mer 集合的空间存储问题，从而提高后面使用索引的速度，大幅度减少使用索引的时间。

因此，哈希算法的主要优点：查找速度快捷、直接、简单。对所给的大量数据的碱基序列先运用其碱基种类较少的特点，将其赋予特定的值，然后在 100 万行碱基序列中以单向散列函数（哈希函数）的方法对碱基序列进行地址映射，于是就得到了一个有序的碱基片段的地址存储单元，从而可以方便高效的进行查询。

3.3.2 折叠法的原理及运用

通常来讲，哈希函数的设定非常灵活，通常，我们都会尽量构建一个均匀的哈希函数。其中，一个均匀的哈希函数是指对于任意的一个关键字，经哈希函数得到的任何一个映像

地址的概率是相等的。因此，在本题中为了尽量减少后面将会遇到的哈希冲突的次数，建立一个均匀合适的哈希函数显得至关重要。而比较常用的哈希函数的构造方法有以下几种：直接定址法，数字分析法，平方取中法，除留余数法，折叠法。

因为，由于我们的 DNA 序列对比主要是用在对关键的生物遗传信息在 DNA 中的具体位置的索引，因此人们所要查询的 k-mer 的 k 值一般比较大，大约集中在 10~100 的长度的特定碱基序列的查找与定位，而长度为 2 或 3 等较小长度的查询模式串（如：“AA”、“AT”等极短碱基对）在实际操作与使用中几乎使用得很少。所以，针对这种关键字的位数比较长，即查询的模式串碱基序列 k 值比较大时，综合分析后本文采用折叠法构建哈希函数更加合适。折叠法适用于关键字位数较多，而且关键字中每一位上数字分布大致均匀的情况。这里本文采用折叠法中的移位叠加法。即将分割后的每一部分的最低位对齐，然后相加。

基于以上对 DNA 序列的实际问题的考虑与分析，本文给出具体适合 k-mer index 问题的折叠方案，首先已经对碱基对进行了进制转换，然后依次将每个 k-mer 对应的进制序列数，划分为长度相同的几个部分（这里的长度表示值本文将其命名为折叠系数），然后将这几部分相加后舍去进位作为哈希值，将此 k-mer 的位置信息以数组形式存储（即 DNA 序列编号以及所在序列的位置，后面将对数组信息压缩做出分析）。另外，如果遇到 k-mer 被划分后，剩有部分位数折叠系数的片段（例如 k=16 时，划分后会剩余一个长度为 1 的小片段），在这里我们将对这个小片段进行忽略不计。

例如：对 k=20 的文本子串碱基对的进制序列 $P' = "01232312211321112313"$ 运用折叠法进行处理，将其划分为长度为 5 的小片段，并且对其累加求和，即有 $12313+13211+31221+01232=57977$ ，得到位数超过指定的 FD，则剔除高位，所以该模式串碱基经过叠加法处理后对所得到的哈希值为 57977。

3.3.3 折叠式散列值的优化求法

假设一段序列为 30122 11230 12132 33213....

若 k=19,FD=5，那么第一段的 k-mer 就是 30122 11230 12132 3321

其折叠和便是：

$$\begin{array}{r} 30122 \\ 11230 \\ +12132 \\ \hline \end{array}$$

$$=53484$$

易知，第二段 k-mer 的折叠和便是：

$$\begin{array}{r} 01221 \\ 12301 \\ +21323 \\ \hline =34845 \end{array}$$

对于原序列，设字母 a, b, c, d, e, f, g...分别表示每连续 FD 片段的开头和结尾数字，即 a=3,b=2,c=1,d=0,e=1,f=2,g=3；设 A, B, C, D, E, F 分别表示图中折叠和计算的片段，即 A=30122, B=11230, C=12132, D=01221, E=12301, F=21323；用 M, N 分别表示两个折叠和；

易得：

$$\begin{aligned} N &= (A - a * 100^{FD-1}) * 10 + c + (B - c * 10^{FD-1}) * 10 + e + (C - e * 10^{FD-1}) * 10 + g \\ &= M * 10 - (a + c + e) * 10^{FD} + c + e + g \end{aligned} \quad (\text{式 1-2})$$

由式 1-2 可知，(a+c+e)为 M 的高位数，所以 $M * 10 - (a + c + e) * 10^{FD}$ 表示去了高位，低位补零。而 (c+e+g) 表示该段 k-mer 除了第一个 FD 片段后剩下的每一 FD 片段的对齐数，从高位到低位依次加取。

所以针对本题的折叠法建立索引可以通过该方法达到优化。

对于给定的 k，先测得其是否为 FD 倍数，不足则添加，使之达成整数倍。

针对上面第一段的 k-mer，因为不是 FD 的倍数，所以通过末尾补添，即 3。然后通过第一段 k-mer 的折叠和去高位补低位的方法，可以一次得到后面连续五个 k-mer 的折叠和。随后便刷新片段，即去掉第一个 FD 片段，末尾添加一个 FD 片段。用同样的方法去高位补除第一个 FD 片段后剩下几个片段的对齐数。

3.3.4 折叠系数的大体选择

假定哈希表行数令为 H，根据行数计算所得列数令为 L，均值遍历数令为 B，实际列数令为 L'，额外内存占用量令为 E，则可分析得到以下公式：

$$H = (3 * [k / FD] + 1)^{FD} \quad (\text{式 1-3})$$

$$L = (101 - k) * 100w / H \quad (\text{式 1-4})$$

$$B = \begin{cases} L, & L \text{ 为整数} \\ [L]+1, & L \text{ 为小数} \end{cases} \quad (\text{式 1-5})$$

$$L' = B + 2 \quad (\text{式 1-6})$$

$$E = (L' - L) * H \quad (\text{式 1-7})$$

(其中, $[k / FD] \leq 3$)

由式 1-3 和式 1-4 可得: 若 k 值偏大的时候, 对于一些偏小的 FD 来说, 是可以折叠得到 0~9 所有可能数, 但是对于 k 值偏小的时候, 即使在合理的最小 FD 下, 也不可能得到 0~9 所有可能数。比如, 当 k=10, FD=5, 这里的 k-mer 就只能折叠成两部分的加和, 得到的折叠和的可能性最大也是到 6, 而不可能出现 7、8、9 这三个数, 所以此处相应的表行就是 7^{FD} 种可能性, 所以 k 值的大小和所选用的折叠系数 FD 影响着哈希表的具体建立。

至于在给定 k 下, 采用哪个折叠系数 FD 最为合适, 则需要根据情况具体分析。大体上来讲, 筛选的原则有二个: 一是计算所得列数足够小; 二是同等遍历下, 额外内存使用量尽量小。

根据给定的 k 和折叠系数 FD, 根据式 1-3 和式 1-4 可计算出对应哈希表的行数和列数。得到的列数若为整数, 则取该列数, 若得到的列数为小数, 则需对列数取整加一, 这样得到的数可当做是基于每一种 k-mer 下的均值遍历数。

题目所给的是 $100w \times 100$ 这样庞大的数据量, 四种碱基的出现概率近乎相同, 因此, 在这样的大数据情况下, 对一个目标 k-mer 的索引查询时间可以相当于该 k-mer 对应的哈希值下的遍历数。

对于给定的 k 下, 不同折叠系数 FD 对应的行数和列数的乘积是一个定值, 即哈希表的建立框架, 这个定值等同于相同的内存使用量, 但是对于具体得到的列数既有整数, 也有小数, 而事实上列数的分配并不能分配小数的列数, 因此, 只能对其作取整加一的处理, 此时让其成为整数列的部分即属于额外内存的使用量。根据前面的 3.2 节的“散列表的建立”, 对于不同散列表下, 都要额外加上两列存储空间用于存放行首和行末指标数, 所以此时的给定 k 值下, 不同折叠系数 FD 的内存占用量的不同主要来源于额外内存空间使用, 因此, 通过比较额外内存空间使用量便可以得到在给定 k 值情况下, 不同折叠系数 FD 的

内存占用量的差别。

本文通过大体筛选的两项指标（即计算所得列数、额外内存使用量）对给定 k 值下，众多的折叠系数 FD 可以进行一个初步的筛选，最后对得到的候选数，采用灰色层次分析模型，得到给定 k 下最优折叠系数 FD 的综合评价结果，在本文后面的“算法的评价”部分有详细的介绍。

例如，当 $k=22$ 时，所有的折叠系数对应的行数、列数、均值遍历数和额外使用量的实验数据请见附录。根据上述的两个筛选原则，便可得到以下被筛选出来合适的折叠系数 FD 的候选数（即 7、8、9、10、12、13、14，一共 7 个候选数），如表 1 所示。

表 1 $k=22$ 时的折叠系数大致筛选值

折叠系数	行数	列数	均值遍历数	额外内存使用量
7	1×10^7	7.90	8	2.1×10^7
8	5.76×10^6	13.70	14	2.32×10^7
9	4.04×10^7	1.96	2	8.24×10^7
10	2.82×10^8	0.28	1	7.68×10^8
12	1.68×10^7	4.71	5	3.84×10^7
13	6.71×10^7	1.18	2	1.89×10^8
14	2.68×10^8	0.29	1	7.26×10^8

3.4 数组存储方式

3.4.1 数据压缩方法

在现有的相关研究中，对文本串中依次读取的连续长度为 k 的 k -mer 进行散列映射后的存储方式一般采用的是二维数组，通过二维数组对每个读取的 k -mer 的具体位置进行记录，二维数组的行标为所在的 DNA 序列编号，二维数组的列表为相应序列出现的位置。然而，这种用二维数组来存储每个 k -mer 的信息位置，十分消耗计算机的内存空间，因为本题所给的 DNA 序列众多，依次生成的长度为 k 的 k -mer 数量更多。如果存储位置信息所占的内存空间过大，会直接导致建立索引的时间过长，不符合题意。

因此,我们提出了一种新的数据压缩方式,即用一维数组代替二维数据对相应的 **k-mer** 的位置信息进行记录与存储。在该方法中,我们定义用一个数来表示每个 **k-mer** 的位置信息,整数部分为所在 DNA 序列编号,小数部分为相应序列出现的位置。通过这种数据压缩的方法,可以有效节省接近一般的计算机内存空间,使建立索引的时间减少一半,符合题目要求。

3.4.2 数组的存储原则

在散列表的初始的建立下,大体上其列数是固定的,但是不可避免存在冲突频繁进而导致列数在具体到某一行下的信息溢出损失的情况。即在本文中就可能发生多数不同的 **k-mer** 下折叠和相同进而存储到同一地址,为了避免信息损失,我们调整了以往固定的存储方式,使其更加灵活,保证了在数量上正确的位置信息导出的完整性。

具体的存储原则如下:当 **k-mer** 映射到相应的地址,

- a.此时该行若未滿,则逆向存储;
- b.若该行已滿,则看该行末指标,根据行末指标数来决定改数组存向哪一行,存储方式为顺向存储;
- c.若遇到 b 情况下将要存储到一个已滿的行内,处理方式为将该行最后一个信息下放下一行顺向存储,原行数组右移,将第一个位置留放给将要存储的数组。
- d.当存放数组到已滿行内,行动结束后要判定该行最后一个位置信息相应的 **k-mer** 是否属于该行,若属于,行末指标不变;若不属于,则行末指标+1。
- e.查询时通过映射到相应的表行,查看行末指标数,来决定去哪一行遍历匹配。

3.5 逆向匹配

在前面的建立索引阶段,我们已经将转化为进制序列数的文本串划分成了多个相同长度的子串,从而进行折叠式散列,得到相应的散列值,以此来存入其位置信息到相应表行内。同理,在进行模式串的查找匹配阶段时,同样对模式串进行同样的折叠式散列来索引到相应的表行,并对该行所有位置信息相应的 **k-mer** 进行匹配,从而筛选得到正确的位置信息导出。这里采用什么样的匹配方法带来的效率往往也是不同的。

基于本题的折叠思想的划分方式,在不足 FD 的倍数的 **k-mer** 下的折叠,也就是在最后一部分不累加的前提下,将模式串和所有相应的 **k-mer** 进行逆向匹配要比顺向匹配筛选快的多,因为通过折叠式散列函数得到的散列值与忽略掉的最后一部分是无关的,换句话

说，当模式串与相应的 k-mer 从最后一个序列数依次往前匹配的时候，两个被忽略掉的部分进行匹配往往存在较大差异，所以在很大程度上匹配第一个序列数便可以否定一个不相符的 k-mer，相比较顺向匹配，效率显然大大提高。如果不匹配，则放弃该子串，读取该行下一个位置信息，匹配成功则导出其相应的位置信息。

在查询的这一环节中，我们通过逆向匹配的方法能够有效减少匹配的次数，从而提高使用索引的效率，大幅度减少查询时间。

5 程序运行结果

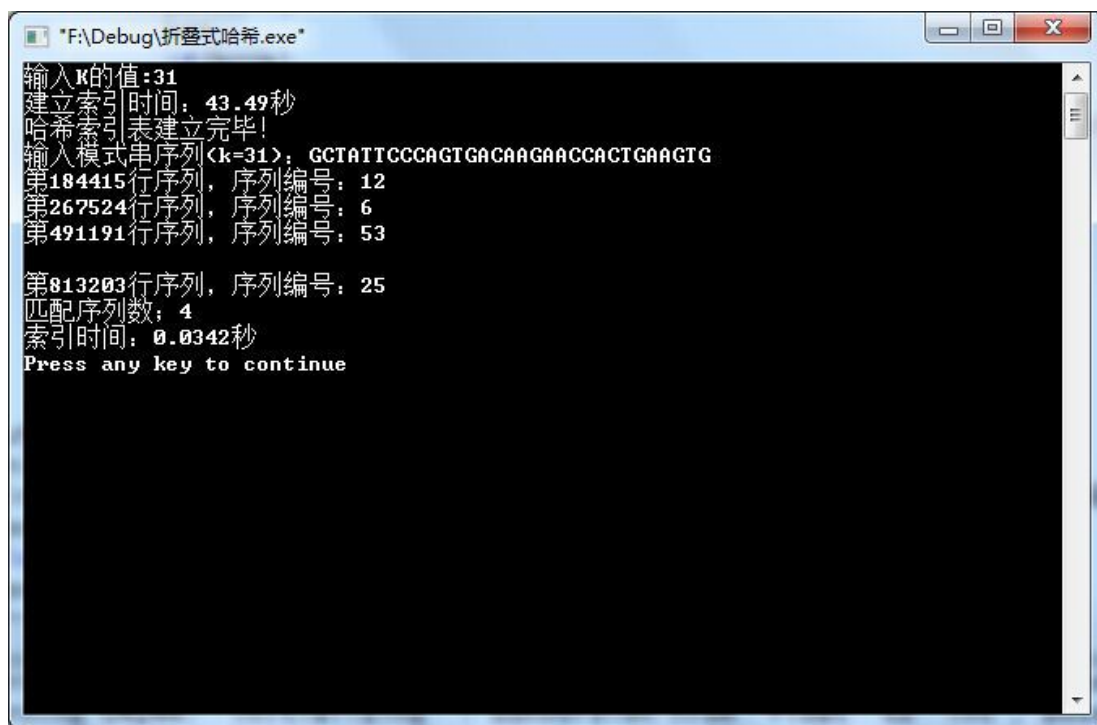
程序运行的电脑配置如下：

- (1) 处理器：Intel (R) Core (TM) i7-3632QM CPU @ 2.20GHz 2.20GHz
- (2) 安装内存 (RAM) 8.00GB (7.58GB 可用)
- (3) 系统类型：64 位操作系统

```
"D:\Debug\折叠式哈希.exe"
输入K的值:10
正在建立索引.....
建立索引时间: 20.26秒
哈希索引表建立完毕!
输入模式串序列 (k=10): GCTATTCCCA
第365550行序列, 序列编号: 6
第389829行序列, 序列编号: 31
第482604行序列, 序列编号: 3
第491191行序列, 序列编号: 53
第496848行序列, 序列编号: 68
第588672行序列, 序列编号: 37
第813203行序列, 序列编号: 25
第872103行序列, 序列编号: 56
第874274行序列, 序列编号: 87
第979982行序列, 序列编号: 12

匹配序列数:10
索引时间: 0.0372秒
Press any key to continue
```

图 2 程序运行结果 (1)



```
*F:\Debug\折叠式哈希.exe*
输入k的值:31
建立索引时间: 43.49秒
哈希索引表建立完毕!
输入模式串序列(k=31): GCTATTCCCAGTGACAAGAACCCTGAAGTG
第184415行序列, 序列编号: 12
第267524行序列, 序列编号: 6
第491191行序列, 序列编号: 53
第813203行序列, 序列编号: 25
匹配序列数: 4
索引时间: 0.0342秒
Press any key to continue
```

图 3 程序运行结果 (2)

结果一:

如图 6 的程序运行截图, 我们可以看出当输入的 k 值为 10 时, 目标模式串为“GCTATTCCCA”, 索引返回的对应 k-mer 有 10 个, 即:

第 365550 行序列, 序列位置为 6、第 389829 行序列, 序列位置为 31、第 482604 行序列, 序列位置为 3、第 491191 行序列, 序列位置为 53、第 496848 行序列, 序列位置为 68、第 588672 行序列, 序列位置为 37、第 813203 行序列, 序列位置为 25、第 872103 行序列, 序列位置为 56、第 874274 行序列, 序列位置为 87, 第 979982 行序列, 序列位置为 12。

该目标模式串的建立索引时间为 20.26 秒, 使用索引时间为 0.0372 秒。

结果二:

如图 7 所示的程序运行截图, 我们可以看出当输入的 k 值为 31 时, 目标模式串为“GCTATTCCCAGTGACAAGAACCCTGAAGTG”, 索引返回的对应的 k-mer 有 4 个, 即:

第 184415 行序列, 序列位置为 12、第 267524 行序列, 序列位置为 6、第 491191 行序列, 序列位置为 53。

该目标模式串的建立索引时间为 43.49 秒, 使用索引时间为 0.0342 秒。

6 复杂度分析

6.1 建立索引复杂度

6.1.1 时间复杂度分析

(1) 进制转换 $O(n)$

(2) 折叠加和 $O(kn)$

(3) 查找散列表头, 最好 $O(1)$, 平均 $O(\log_2 n)$, 最差 $O(n)$

(4) 数组信息存放, 最好 $O(1)$, 平均 $O(\log_2 n)$, 最差 $O(n)$

综上所述, 时间复杂度最好: $O(n) + O(kn) + O(\log_2 n) + O(\log_2 n) = O(kn)$

6.1.2 空间复杂度分析

(1) 进制转换 $O(n)$

(2) 累加 $O(n/k)$

(3) 散列表 $O(n)$

综上所述, 空间复杂度最好: $O(n) + O(n/k) + O(n) = O(n)$

6.2 使用索引复杂度

6.2.1 时间复杂度分析

(1) 进制转换 $O(1)$

(2) 查找头, 最好 $O(1)$, 平均 $O(\log_2 n)$ 最差 $O(n)$

(3) 冲突匹配 $O(n)$

综上所述, 时间复杂度最好: $O(1) + O(\log_2 n) + O(n) = O(n)$

6.2.2 空间复杂度分析

折叠算法, 查找的空间复杂度为 $O(1)$

7 算法的评价

7.1 灰色层次分析法的简述

灰色层次分析法是基于灰色关联分析法和层次分析法的基础上提出的一种新的评价方法, 是运用灰色关联分析的一种改进版的层次分析法。相较于传统的层次分析法, 灰色

层次分析法综合了灰色关联分析法和层次分析法的优点，是定性分析与定量分析相结合的客观评价法，既可以很好地解决评价指标难以准确量化和统计的问题，又可以尽量减少人为主观因素的影响，使最终的评价结果更加接近客观事实。

结合本题第六问的题目要求，需要我们按重要性的高低排列的顺序（即索引查询速度、索引内存使用、8G 内存下所支持的 k 值范围以及建立索引的时间）来评价本论文中的索引方法的性能。如果只是单纯地使用传统的层次分析法来评价我们自己设计的算法，就会显得主观因素对评价结果有很大的影响，即仅依靠专家做出的评价结果，会产生不可避免的人的主观选择、判断偏好对结果的很大的影响，引起评价结果跟客观实际有很大的差距。另外，由于小组能收集到的专家评价结果有限，因此我们得到的数据呈现出一种“数据小，信息不完全”的特点。根据这种特点，本小组提出了使用灰色层次分析法的较客观的评价方法，尽可能地减少因人为主观因素对结果的影响，最大程度上客观地对我们设计的算法按照四个指标进行一个综合评价。

7.2 灰色层次分析法的思路

由层次分析法构建指标体系的层次结构关系图，依据判断矩阵计算出指标的权重，并得到被评价对象的评价值。由于层次分析法中对原始数据的处理使用极值法，可知，若取矩阵中每列的最优值构成一虚拟的被评价对象，则在层次分析法中，该虚拟评价对象的评价值为 1，那么各被评价对象的评价值可看作是与最优值的近似程度，这样构造的虚拟对象与灰色关联分析中的理想对象是等价的。由关联系数计算各评价对象对于理想对象的关联度时，考虑指标本身权重的影响，结合层次分析法中得到的指标权重，得到一组新的关联度，进而评价各对象^[5]。

7.3 灰色层次分析法的步骤

基于上述构思，灰色层次分析法的指标体系评价模型如下：

Step1:根据层次分析法的结构图构建了一个由 n 个指标， m 个被评价对象组成的单层指标体系， m 个被评价对象的原始指标数据矩阵记为 $R = (r_{ij})_{m \times n}$ 。

$$R = \begin{pmatrix} r_{11} & \cdots & r_{1n} \\ \vdots & \ddots & \vdots \\ r_{m1} & \cdots & r_{mn} \end{pmatrix}$$

Step2:对 R 矩阵中的原始数据进行极值化处理，得到矩阵 $S = (s_{ij})_{m \times n}$ 。

$$S = \begin{pmatrix} s_{11} & \cdots & s_{1n} \\ \vdots & \ddots & \vdots \\ s_{m1} & \cdots & s_{mn} \end{pmatrix}$$

Step3:对指标的重要性进行两两比较，由专家的评分构造判断矩阵 $A = (a_{ij})_{m \times n}$ ：

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$$

Step4:计算A的最大特征值和最大特征值对应的特征向量，并将特征向量做归一化处理；

Step5:对判断矩阵A进行一致性检验，若A满足给定的一致性标准，则由上一步求出的归一化后的特征向量记为指标权重，记为 $w_j, j=1,2,\dots,n$ ；反之，若A不满足给定的一致性标准，则需要对A 进行修改，重复前两步，直到A满足给的一致性标准为止；

Step6:计算 $x_i = \sum_{j=1}^n s_{ij} * w_j (i=1,2,\dots,m)$

$$\text{有 } \rho = \frac{1}{n} \sum_{i=1}^n x_i ;$$

Step7:确定理想对象 R_0 ，记 $R_0 = (r_{01}, r_{02}, \dots, r_{0n})$ 其中， $r_{0j} (j=1,2,\dots,n)$ 为第j个指标在所有待评对象中的最优值：

$$r_{0j} \begin{cases} \max_i r_{ij} , & \text{其中第 } j \text{ 个指标为收益性指标} \\ \min_i r_{ij} , & \text{其中第 } j \text{ 个指标为成本性指标} \end{cases}$$

Step8:记 $S = (s_{i1}, s_{i2}, \dots, s_{in}), i=1,2,\dots,m$ 。将 $S_0 = (1 \ 1 \ \cdots \ 1)$ 作为参考数列，计算 S_i 的第j个指标的关联系数 $\beta_i(j), i=1,2,\dots,m$ ：

$$\beta_i(j) = \frac{\min_i \min_j |s_{0j} - s_{ij}| + \rho \max_i \max_j |s_{0j} - s_{ij}|}{|s_{0j} - s_{ij}| + \rho \max_i \max_j |s_{0j} - s_{ij}|}$$

通过如上计算，得到关联系数矩阵 β ：

$$\beta = \begin{pmatrix} \beta_1(1) & \dots & \beta_1(n) \\ \vdots & \ddots & \vdots \\ \beta_m(1) & \dots & \beta_m(n) \end{pmatrix}$$

$$X = \beta * \omega$$

Step9:计算集成的评价值

其中， $\omega = (\omega_j)_{n \times 1}$ 是由前三步得出的n个评价指标的权重矩阵，满足 $\sum_{j=1}^n \omega_j = 1$ ，

$X = (x_i)_{m \times 1}$ 为m个被评价对象的集成评价结果矩阵，则第i个被评价对象的评价结果为：

$$x_i = (\beta_i(1), \beta_i(2), \dots, \beta_i(n)) \cdot \begin{pmatrix} \omega_1 \\ \omega_2 \\ \vdots \\ \omega_n \end{pmatrix}$$

x_i 越大，可知第i个被评价对象与理想对象越接近，表明第i个被评价对象越优，由此可排出各评价对象的优劣，从而得到分析评价结果。

7.4 运用灰色层次法对本算法进行综合评价

本题第六问的解决思路就是：运用灰色层次分析法的方法对每个 k 值对应的大致筛选的折叠系数 FD 的各个方案，按照四项指标（即索引查询时间、运行的内存占用、k 值的取值范围以及建立索引的时间）进行综合评价，得到最终的评价结果。

具体步骤如下：

步骤一：运用层次分析法的结构图建立起本题所对应的目标层、准则层和方案层，如图 4 所示（这里以 k=22 为例，方案层一共有 7 个）：

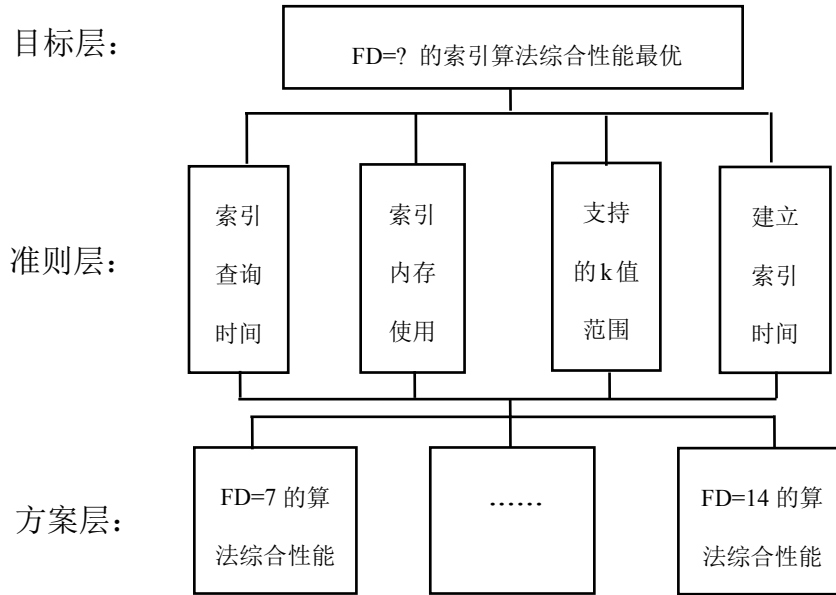


图 4 层次分析法的结构示意图

步骤二：按照本题第六问中所规定的重要性大小的顺序，对这四个指标进行两两比较，得出评价矩阵 A：

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} = \begin{bmatrix} 1 & 3 & 5 & 7 \\ 1/3 & 1 & 3 & 5 \\ 1/5 & 1/3 & 1 & 3 \\ 1/7 & 1/5 & 1/3 & 1 \end{bmatrix}$$

步骤三：运用 Matlab 软件计算出评价矩阵 A 的最大特征值为 $\lambda = 4.119$ ，最大特征值对应的特征向量为 $\vec{n} = (0.558, 0.263, 0.121, 0.058)^T$ 。

步骤四：对矩阵 A 进行一致性检验。

表 2 随机一致性指标 RI											
n	1	2	3	4	5	6	7	8	9	10	11
RI	0	0	0.58	0.90	1.12	1.24	1.32	1.41	1.45	1.49	1.51

因为本题的评价矩阵的 $n=4$ ，因此， $RI=0.90$

$$CI = \frac{\lambda - n}{n - 1} = \frac{4.119 - 4}{4 - 1} = 0.0397$$

$$\text{随机一致性比率: } CR = \frac{CI}{RI} = \frac{0.0397}{0.9} = 0.0441 < 0.1$$

因此，以上针对本题所构建的评价矩阵符合一致性要求，由其计算所得的特征向量 n 可作为四个指标对应的权向量，即 $\omega_j = (0.558, 0.263, 0.121, 0.058)^T$ 。

步骤五：由前面的 3.3.4 节的“折叠系数的大致选择”可知，当 $k=22$ 时，经过筛选后得到的折叠系数 FD 的候选数有 7、8、9、10、12、13、14，一共 7 个候选数，另外，由于索引查询时间等同于均值遍历数，索引内存使用等同于额外空间占用量。这样的指标替换的好处就是能得到具有代表性的数值。

因此，我们现在的四项评价算法的指标可以等同于均值遍历数、额外空间占用量、支持 k 值的范围以及建立索引的时间。

步骤六：当 $k=22$ 时，它对应四项指标的实验数据，如表 2 所示(数据统一保留 3 位小数，并且 C 值表示一个常数)。

表 3 $k=22$ 时对应的四项指标的实验数据				
折叠系数	均值遍历数	额外空间占用量	k 值范围	检索时间
7	8	2.10×10^7	C	28.98
8	14	1.32×10^7	C	28.23
9	2	8.24×10^7	C	26.78
10	1	7.68×10^8	C	25.76
12	5	3.84×10^7	C	24.98
13	2	1.89×10^8	C	26.34
14	1	7.26×10^8	C	27.22

步骤七：当 $k=22$ 时，它对应的原始矩阵 R 如下：

$$R = \begin{pmatrix} 8 & 2.10 \times 10^7 & C & 28.98 \\ 14 & 1.32 \times 10^7 & C & 28.23 \\ 2 & 8.24 \times 10^7 & C & 26.78 \\ 1 & 7.68 \times 10^8 & C & 25.76 \\ 5 & 3.84 \times 10^7 & C & 24.98 \\ 2 & 1.89 \times 10^8 & C & 26.34 \\ 1 & 7.26 \times 10^8 & C & 27.22 \end{pmatrix}$$

步骤八：经过 Matlab 软件的计算，得到极值化后的矩阵 S 如下：

$$S = \begin{pmatrix} 0.125 & 0.630 & 1 & 0.862 \\ 0.071 & 1 & 1 & 0.885 \\ 0.5 & 0.161 & 1 & 0.933 \\ 1 & 0.017 & 1 & 0.970 \\ 0.2 & 0.344 & 1 & 1 \\ 0.5 & 0.070 & 1 & 0.948 \\ 1 & 0.018 & 1 & 0.918 \end{pmatrix}$$

步骤九：确定参数 ρ 和理想对象 $R_0 = (1, 1.32 \times 10^7, C, 24.98)$ 按照关联系数的计算公式，得到相应的关联系数矩阵 β 如下：

$$\beta = \begin{pmatrix} 0.360 & 0.571 & 1 & 0.781 \\ 0.346 & 1 & 1 & 0.810 \\ 0.496 & 0.369 & 1 & 0.880 \\ 1 & 0.333 & 1 & 0.942 \\ 0.381 & 0.428 & 1 & 1 \\ 0.496 & 0.346 & 1 & 0.905 \\ 1 & 0.334 & 1 & 0.857 \end{pmatrix}$$

步骤十：将该关联系数矩阵 β 与权重矩阵 $\omega_j = (0.558, 0.263, 0.121, 0.058)^T$ 相乘，得到

最终的灰色层次分析的评价值矩阵 X 如下:

$$X = (x_1, x_2, x_3, x_4, x_5, x_6, x_7)^T = (0.517, 0.624, 0.546, 0.821, 0.504, 0.541, 0.816)^T$$

结论: 当 $k=22$ 时, 折叠系数为 10 的评价值为 0.821, 评价值最高, 因此, 当 $k=22$ 时的最优折叠系数为 10。

步骤十一: 根据每个给定的 k 值下, 大体筛选出来的折叠系数 FD 的候选数所对应的四项指标的实验数据 (具体数据表格请见附录), 按照 $k=22$ 时的灰色层次评价模型一样的步骤, 得出每个给定的 k 值下的最优折叠系数。

得到的最终评价结果, 如表 3 所示。

表 4 不同 k 值下的最优折叠系数

k 值	1~14	15~19	20~26	27~100
最优折叠系数	k	14	10	8

结论: 本文通过运用灰色层次分析法, 对不同 k 值下的不同的折叠系数, 根据四项指标进行综合评价, 得到了每个 k 值对应的最优折叠系数。

首先, 根据 3.3.4 节的“折叠系数的大致选择”中的两项筛选原则, 本文首先得到了每个 k 值折叠系数 FD 的候选数, 然后本文通过用灰色层次分析法对每个 k 值下的折叠系数的几个候选数进行综合评价, 最终得到评价结果: 当 $k=1\sim14$ 时, 最优折叠系数为 k 值; 当 $k=15\sim19$ 时, 最优折叠系数为 14; 当 $k=20\sim26$ 时, 最优折叠系数为 10; 当 $k=27\sim100$ 时, 最优折叠系数为 8。

8 算法的优点

(1) 对题目所给的 FA 格式文件进行了预处理, 即设计了一个 C 语言程序, 将原 FA 格式文件中没有用的信息进行删除, 输出了一个只有 DNA 序列碱基对的文本。通过事先对文件的预处理, 使建立索引的速度大大提高, 节省了不必要的查找时间。

(2) 采用散列函数 (哈希函数) 建立索引, 将无序的碱基对转化为有序的哈希值, 使查询速度最快, 查询目标碱基对更加快捷、直接、简单。

(3) 采用折叠法对 k -mer 进行划分, 累加求和得出相应的哈希值, 通过这种方法构造了一个分布较均匀的散列函数 (哈希函数), 极大地提高了算法的效率。

(4) 采用新式的数组存储方式解决了哈希表使用过程中产生的因冲突过多导致的信息损失的问题, 提高了查询数量完整性。

(5) 用一维数组代替传统的二维数组存放 k-mer 的位置信息, 节省了接近一半的内存占用量。

(6) 用逆向匹配代替传统的顺向匹配, 提高了字符串匹配成功的效率, 让查询时间缩短得更多。

(7) 该算法是由 C++ 语言编写的, 使用 C++ 这款软件可以比 java、C# 等编程语言在执行处理方面速度更快, 效率更高, 尤其是在处理诸如这类问题的大数据时, C++ 语言编程更有优势, 提高了查询速度与效率。

(8) 采用灰色层次分析模型, 尽可能地减少因人为主观因素对结果的影响, 最大程度上客观地给出了四个指标下的一个综合评价。

(9) 通过得出的结论进而反馈改进程序, 得到不同 k 下最优折叠系数的选取。

参考文献

- [1] 张鑫鑫 《生物序列数据 K-mer 频次统计与可视化研究》 第一期 2014 年.
- [2] 李志敏 《哈希函数设计与分析》 第一期 2009 年.
- [3] 贾丹 《基于哈希函数的数据库查询技术的研究》 第 32 卷第二期 2012 年 4 月.
- [4] 杨矫云 《大规模生物序列分析的高性能算法和模型》 第一期 2014 年 5 月.
- [5] 费智聪 《熵权层次分析法与灰色层次分析法研究》 2009 年 5 月

附件：

一、程序源代码

```
#include<iostream.h>
#include<fstream.h>
#include<malloc.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

int conver(int *a,int k);//转换函数声明
const int FD=5;    //折叠式系数为 5

void main()
{
    fstream f;                //联系着 DNA 序列文件
    clock_t start,finish;      //用 start, finish 两个时间变量来记录程序运行的时间
    char *DNA=(char *)malloc(100000000*sizeof(char));//存储一亿个 DNA 序列
    char *head=NULL,*point=NULL,ch,prn[102];
    head=DNA;
    point=DNA;
    int ID=0,k,seg=1,i=0,n=0, h=0,s=0;;                //定义 K 值, seg 存储 DNA 序列的数
字信息, ID 存储每个序列的编号
    printf("输入 K 的值:");cin>>k;
    printf("正在建立索引.....\n");
    long *number=(long *)malloc(100000*sizeof(long));
    for(int g=0;g<100000;g++)
    {
        *(number+g)=0;
    }
    int *temp=(int *)malloc(k*sizeof(int));
```

```
float **hash;
hash=new float*[100000];
for(int j=0;j<100000;j++)
{
    hash[j] = new float[3000];
}

f.open("F:\\dna.txt",ios::in|ios::binary);//导入 DNA 序列文件
f.get(ch);      *(DNA+i)=ch;

while(!f.eof())
{
    *(DNA+i)=ch;
    i++;
    f.get(ch);
}

start=clock();          //记录程序开始时间
while(seg<=1000000)     //从此处起开始建立折叠式哈希索引表
{
    while(ID<=100-k)
    {
        for(int j=0;j<k;j++)
        {
            switch(*(point+j))
            {
                case 'A':*(temp+j)=0;break;
                case 'T':*(temp+j)=1;break;
                case 'C':*(temp+j)=2;break;
                case 'G':*(temp+j)=3;break;
                default: break;
            }
        }
    }
}
```

```

        }
    }
    int con=conver(temp,k);
    long count=(*(number+con));
    ((* (hash+con))+count)=(float)((float)seg+((float)ID/100));
    (*(number+con))++;
    point++;
    ID++;
}
ID=0;
seg++;
head=(head+100);
point=head;
}
finish=clock();
printf("建立索引时间: %.2f 秒\n", (double)(finish - start) / CLOCKS_PER_SEC);
printf("哈希索引表建立完毕! \n");
printf("输入模式串序列(k=%d): ",k);cin>>prn; //键入模式串
start=clock();
for(j=0;j<k;j++)
{
    switch(prn[j])
    {
        case 'A':*(temp+j)=0;break;
        case 'T':*(temp+j)=1;break;
        case 'C':*(temp+j)=2;break;
        case 'G':*(temp+j)=3;break;
        default: break;
    }
}

```

```

    }
    int fina=conver(temp,k);
    if(*(number+fina)==0)
    {
        cout<<"无匹配结果! ";
    }
    else
    {
        for(i=0;i<*(number+fina);i++)
        {
            int final_seg=(*(hash+fina)+i);
            int final_ID=(((*(hash+fina)+i)-(float)final_seg)*(float)100);
            point=DNA+((final_seg-1)*100+final_ID);
            while((*point)==prn[h])
            {
                point++;
                h++;
                if(h==k)
                {
                    cout<<" 第 "<<final_seg<<" 行 序 列 ,  "<<" 序 列 编 号 :
"<<final_ID<<endl;//输出与模式串相同的 K-mer 的位置信息
                    s++;    //匹配序列数 (即跟模式串匹配的 K-mer 的个数)
                    break;
                }
            }
            h=0;
        }
    }
    cout<<endl;

```

```

    cout<<"匹配序列数:"<<s<<endl;
    finish=clock();
    printf("索引时间: %.4f 秒\n", ((double)(finish - start) / CLOCKS_PER_SEC));
    //下面是清除程序占用空间
    free(DNA);
    delete hash;
    free(number);
    free(temp);

}
/*

```

此函数为转换函数，建立在折叠系数（FD）为 5 的基础上，将每一次 K-mer 都转换为折叠式的编号，再导出编号

```

*/
int conver(int *temp,int k)
{
    int sum=0,num=k/5;
    for(int i=0;i<num;i++)
    {
        for(int j=0;j<5;j++)
        {
            switch(j)
            {
                case 0:sum+=((*temp+(i*5+j)))*10000;break;
                case 1:sum+=((*temp+(i*5+j)))*1000;break;
                case 2:sum+=((*temp+(i*5+j)))*100;break;
                case 3:sum+=((*temp+(i*5+j)))*10;break;
                case 4:sum+=(*temp+(i*5+j));break;
                default:break;
            }
        }
    }
}

```

```

    }
}
}
if(sum>=100000)
    sum=sum%100000;
return sum;
}

```

/*定义哈希链表，用于储存哈希表中的位置信息

```

struct hashlink{
    float message;
    struct hashlink *next;
}
*/

```

二、当 k=22 时，所有折叠系数 FD 的实验数据（用于 FD 的大致筛选）

折叠系数	行数	列数	均值遍历数	额外内存占用量
1	1×10	7.90×10^6	7.90×10^6	20
2	1×10^2	7.90×10^5	7.90×10^5	2×10^2
3	1×10^3	7.90×10^4	7.90×10^4	2×10^3
4	1×10^4	7.90×10^3	7.90×10^3	2×10^4
5	1×10^5	7.90×10^2	7.90×10^2	2×10^5
6	1×10^6	7.90×10	7.90×10	2×10^6
7	1×10^7	7.90	8	2.10×10^7
8	5.76×10^6	13.70	14	1.32×10^7
9	4.04×10^7	1.96	2	8.24×10^7
10	2.82×10^8	0.28	1	7.68×10^8
11	1.98×10^9	0.04	1	5.85×10^9
12	1.68×10^7	4.71	5	3.84×10^7

13	6.71×10^7	1.18	2	1.89×10^8
14	2.68×10^8	0.29	1	7.26×10^8
15	1.74×10^9	0.07	1	3.14×10^9
16	4.29×10^9	0.02	1	1.28×10^{10}
17	1.72×10^{10}	0.01	1	5.15×10^{10}
18	6.87×10^{10}	0.00	1	2.06×10^{11}
19	2.75×10^{11}	0.00	1	8.25×10^{11}
20	1.10×10^{12}	0.00	1	3.30×10^{12}
21	4.40×10^{12}	0.00	1	1.32×10^{13}
22	1.76×10^{13}	0.00	1	5.28×10^{13}

三、给定 k 值下不同折叠系数的实验数据及评价

指标 K 值	折叠系数	均值遍历数	额外内存 占用量	K 值范围	建索时间	评价值
K=14	7	106	1.94×10^6	C	18.65	0.449
	8	1328	1.63×10^5	C	19.45	0.615
	9	232	5.56×10^5	C	18.68	0.464
	10	83	2.13×10^6	C	18.01	0.451
	11	21	9.47×10^6	C	17.98	0.452
	12	6	4.72×10^7	C	17.56	0.471
	13	2	1.81×10^8	C	17.02	0.543
	14	1	7.18×10^8	C	16.76	0.824
K=15	7	105	2.12×10^6	C	20.12	0.445
	8	1313	1.80×10^5	C	21.32	0.611
	9	329	7.70×10^5	C	20.56	0.454
	10	83	3.13×10^6	C	19.34	0.446
	11	21	1.05×10^7	C	18.76	0.454
	12	6	4.82×10^7	C	18.17	0.47

	13	2	1.82×10^8	C	17.12	0.545
	14	1	7.19×10^8	C	17.43	0.822
K=19	7	100	2.00×10^6	C	27.92	0.615
	8	15	1.60×10^7	C	25.01	0.462
	9	3	1.20×10^8	C	24.23	0.503
	10	79	2.93×10^6	C	23.56	0.824
	11	20	1.03×10^7	C	25.76	0.463
	12	5	3.54×10^7	C	24.12	0.481
	13	2	1.86×10^8	C	25.03	0.539
	14	1	7.23×10^8	C	23.89	0.823
K=20	7	99	2.18×10^6	C	27.92	0.615
	8	15	1.70×10^7	C	25.01	0.462
	9	3	1.21×10^8	C	24.23	0.503
	10	1	7.66×10^8	C	23.56	0.824
	11	20	1.13×10^7	C	25.76	0.463
	12	5	3.64×10^7	C	24.12	0.481
	13	2	1.87×10^8	C	25.03	0.539
	14	1	7.24×10^8	C	23.89	0.823
K=26	7	8	2.50×10^7	C	31.23	0.631
	8	1	2.25×10^8	C	30.45	0.822
	9	2	8.64×10^7	C	29.13	0.556
	10	1	7.72×10^8	C	28.34	0.824
	14	1	7.30×10^8	C	29.12	0.821
K=28	7	8	2.70×10^7	C	34.82	0.636
	8	1	2.27×10^8	C	32.21	0.831
	10	1	7.74×10^8	C	32.18	0.826
	15	1	3.15×10^9	C	33.87	0.819
	5	10	2×10^5	C	2.45	0.641

K=100	6	1	$2 \cdot 10^6$	C	3.87	0.794
	7	1	$2.9 \cdot 10^7$	C	5.76	0.794
	8	1	$2.99 \cdot 10^8$	C	5.32	0.806