DNA 序列的 k-mer index 问题

摘　要：本文研究的是 DNA 序列的 k-mer index 问题。首先通过基于 K-mean 聚类和二分界值法的基数排序对 DNA 序列进行排序并利用二分法查找。然后分别计算时间开销和空间开销，并计算出相关指标对索引性能评价。最后考虑到多次建立索引会消耗过多的时间，我们采用了 SA-IS 算法建立有序后缀数组并利用二分法进行查找。该算法只需建立一次索引，即可实现对所有长度的序列进行查找，并结合了 MATLAB 的 GUI 功能，设计出了索引搜索的软件。该软件可以根据输入的任意 k 值，输出建立索引和查询所用的时间和结果。

关键字：基数排序　模糊综合评价　GUI 软件设计　后缀数组

1.问题重述

1.1 问题背景

给定一个 DNA 序列，这个系列只含有 4 个字母 ATCG，如 *S* = “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

{CTGTA，TGTAC，GTACT，TACTG，ACTGT，TGTAT}

1.2 问题相关信息

信息 1：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}。

信息 2：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。

信息 3：评价索引方法性能的指标：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。

1.3 需要解决的问题

问题一：

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。

问题二：

1.给出建立索引所用的计算复杂度，和空间复杂度分析。

2.给出使用索引查询的计算复杂度，和空间复杂度分析。

3.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。

问题三：

1.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。

2.模型假设与符号说明

2.1 模型假设

假设 1：k-mer 序列长度小于 3 时，不作统计。

假设 2：进行对比实验时，磁盘的磁头定位时间 t_F 都相同。

假设 3：访问文件缓存区时，失效率恒为 0.2。

2.2 符号说明

符号	符号的含义
$d_{ij}^{(t)}$	表示第i个样本到第 j 个聚类中心的距离
r_{nk}	表示为 1 表示 $\vec{X}_n \in \omega_k$；为 0 表示 $\vec{X}_n \notin \omega_k$
$\vec{m}_k$	表示表示类心
k_i^i	表示序列值 c_i 的第i个关键字
SP_s	表示存取一维小矩阵数据块所需空间
N	表示索引项
m	表示小矩阵长度
SP	表示整个空间开销
v_k	表示索引查询速度
s_k	表示 DNA 序列数字化矩阵的元素个数
τ_k	表示索引查询所用时间
Σ	表示字符集
S	表示字符串
LMS	表示 SA 后缀中的一个后缀类型

3.问题分析

本文研究的是 DNA 序列的 k-mer index 问题。首先根据文件中给定的 100 万个 DNA 序列，建立 DNA 序列矩阵，将其数值化，采用基于基数排序法的二分法查找建立索引。然后建立时间开销和空间开销模型针对所建立的索引分析时间复杂度和空间复杂度，并可以通过模糊综合判断模型对所建立的索引的性能进行评价。最后，可以采用 SA-IS 算法，针对多次查询，只需建立一次索引。

3.1 问题一的分析

题目要求对给定 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，并使得其查询速度尽量快，所用内存尽量小。首先根据首先根据文件中给定的 100 万个 DNA 序列，建立 DNA 序列矩阵。通过一一对应原则将 DNA 序列转换为数字形式，然后依次把去 k 个单位长度，视为四进制的数，转化为十进制，得到 DNA 矩阵序列数值矩阵 ，并对矩阵进行降维处理，成为一维矩阵。然后对矩阵进行 K-mean 聚类，将所有相同值的矩阵元素聚为一类，消除重复元素排序带来的时间、空间占用；由聚类类别构成完全不同的元素值的新矩阵，对新矩阵进行二分界值法，将很大的一维新矩阵用二分界值法分解为很多个长度较短的，依次连接的一维小矩阵，最后用基数排序对每个一维小矩阵进行排序，然后依次连接起来，完成对整个序列的排序，并利用二分法对排序后的序列查找出 DNA 序列编号和相应序列中出现的位置。通过比较排序和不排序两种情况下所消耗的，时间和所占用的空间大小从而可以得到最佳索引的方法。

3.2 问题二的分析

题目要求给出建立及查询索引所用的计算复杂度，和空间复杂度分析。并在内存限制为 8G 的假设前提下，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。对于时间复杂度的分析，需分别对建立索引和索引查询的时间复杂进行分析：建立索引时需要考虑降维处理、K-mean 聚类、二分界值法及基数排序四个方面；索引查询时需要考虑存取节点数据块时间和在每个节点数据块中进行二分法查找时间两个方面，存取节点数据块的时间包括磁头定位时间和传送数据块所需的时间，而在每个节点数据块中进行二分法查找时间包括在一个节点中进行二分查找的时间和每个节点数据块中进行二分法查找时间。对于空间复杂度，需考虑建立索引的空间复杂度和索引查询的空间复杂度，建立索引时同样需考虑建立索引时需要考虑降维处理、K-mean 聚类、二分界值法及基数排序四个方面；索引查询时主要考虑存取节点数据块所需空间和每个节点数据块中进行二分法查找的顺序查找空间。通过 MATLAB 编程可以分别求得建立索引和索引查

询时所花销的时间和所占用的空间及在内存限制为 8G 的假设前提下求得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。

3.3 问题的拓展分析

在问题一数据处理的基础上，考虑到多次建立索引会消耗过多的时间，所以可以采用基于 SA-IS 排序构造后缀数组的排序方法。该算法的运算过程采用了 Induced-Sorting 与递归的排序思想。算法定义了一种新的后缀类型——*LMS* 类型后缀，并证明了通过排序好的 *LMS* 类型后缀可以推导出所有 *SA* 数组中后缀的正确位置。*SA-IS* 算法分为三部分：第一部分，问题域缩减，选择抽样字符串 *LMS* 类型后缀；第二部分，利用 Induced-Sorting 的排序思想递归排序抽样字符串；第三部分，利用排序好的抽样字符串再次利用 Induced-Sorting 过程排序剩余部分 *SA* 元素。通过该排序算法，在查找任意长度的 DNA 序列时，只需建立一次索引方可查询所有长度的序列。

4.数据分析与处理

4.1DNA 序列的数值化

(1)DNA 序列矩阵的建立

将 DNA 序列作为矩阵元素建立 DNA 序列矩阵,即：

$$D=\begin{bmatrix} ATCGT\cdots GACGT \\ \cdots \quad \cdots \quad \cdots \\ TAGCA\cdots GTCAG \end{bmatrix}$$

(2)DNA 序列的数值化规则

设置 DNA 序列数值化规则如下所示：

$$A\rightarrow 0 \qquad C\rightarrow 2$$
$$T\rightarrow 1 \qquad G\rightarrow 3$$

经碱基替换之后，得到 DNA 序列数值矩阵。

(3)DNA 序列数值化结果

根据所设置的 DNA 序列数值化规则，通过 MATLAB 编程，得到数值化后的 DNA 序列如下表 1 所示：

表 1：DNA 序列数值化结果表

1	1	2	3	3	3	3	3	…	1	1	2	1	2	2	2	2	3	3	2
1	3	0	0	1	3	3	3	…	1	1	2	1	0	3	1	1	1	0	3
0	3	1	3	1	0	3	3	…	0	3	1	0	0	2	2	1	2	3	3
3	0	1	2	0	0	2	2	…	2	0	0	2	2	0	2	3	2	1	0

…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
0	1	3	0	3	1	0	2	…	2	2	1	0	3	3	2	1	0	0	3
1	2	0	2	2	0	0	1	…	1	3	0	0	3	1	0	2	1	3	0
2	3	2	3	1	3	3	1	…	0	2	1	1	3	3	1	3	3	3	3
2	3	3	1	2	0	2	0	…	3	2	3	1	1	1	2	3	0	2	0
2	2	0	0	2	0	2	2	…	1	2	3	0	3	0	2	0	0	3	1

(4)建立 k-mer 序列集矩阵。

以 k-mer 为单位，在数字序列中依次取单位作为新矩阵的单位元素。即：

$$N=\begin{bmatrix} 01232\cdots 23131 \\ 13201\cdots 31202 \\ \cdots \quad \cdots \quad \cdots \\ 10322\cdots 13032 \end{bmatrix} \xrightarrow{k-mer\text{单位化}} \begin{bmatrix} k_1^1,k_1^2,\cdots,k_1^n \\ k_1^1,k_1^1,\cdots,k_2^n \\ \cdots \quad \cdots \quad \cdots \\ k_n^1,k_n^2,\cdots,k_n^n \end{bmatrix}$$

(5)计算 k-mer 单位序列的值。

将每一个 k-mer 单位序列视为一个 4 进制的数，计算其值得大小，得到新的 k-mer 序列值矩阵。即：

$$F=\begin{bmatrix} a_1^1,a_1^2,\cdots,a_1^n \\ a_2^1,a_2^2,\cdots,a_2^n \\ \cdots \quad \cdots \quad \cdots \\ a_n^1,a_n^2,\cdots,a_n^n \end{bmatrix}$$

4.2DNA 序列中 ATCG 中各字母的频数

根据附件文本中的 DNA 序列文本数据，通过 MATLAB 编程，计算出了四种碱基各自对应的频率，其具体结果如下图 1 所示：

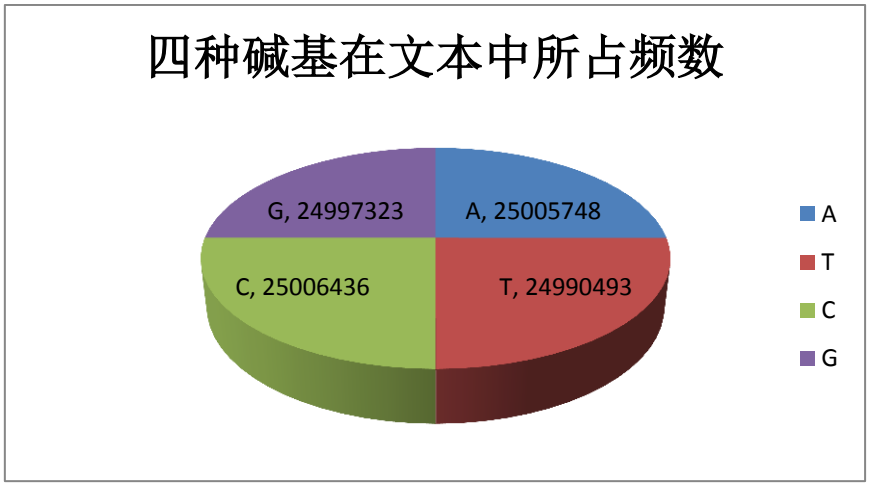


图 1：四种碱基所占频数

由图 1 可知，在 DNA 序列文本中，四种碱基 A、T、C、G 的频率基本相等，

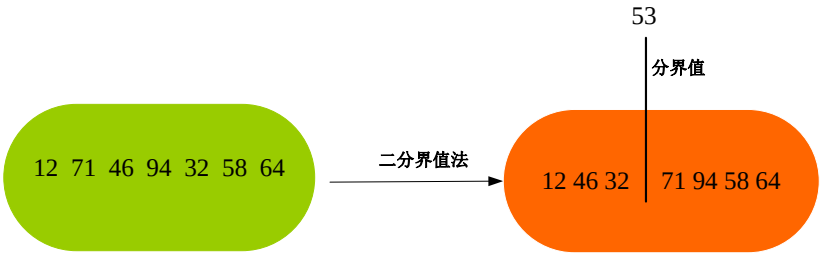
占 25%左右，由此推测，四种碱基随机排列，随机出现，符合一般生物 DNA 序列碱基的出现规律。

5.基于 K-mean 聚类和二分界值法的基数排序算法

5.1 模型的准备

二分界值法：

利用最大值与最小值的平均值，将未排序数列初略分为小的一部分和大的一部分，如下图所示：



5.2 模型的建立

建模思想：首先对矩阵进行降维处理，成为一维矩阵，再对矩阵进行 K-mean 聚类，将所有相同值的矩阵元素聚为一类，消除重复元素排序带来的时间、空间占用；由聚类类别构成完全不同的元素值的新矩阵，对新矩阵进行二分界值法，将很长很大的一维新矩阵用二分界值法分解为很多个长度较短的，依次连接的一维小矩阵，再用基数排序对每个一维小矩阵进行排序，然后依次连接起来，完成对整个序列的排序。

1.对 k-mer 序列值矩阵进行简单降维处理

对 k-mer 序列值矩阵进行行连接，将第 $i+1$ 行连接至第 i 行后面，依次连接，最终将 k-mer 序列值矩阵转化为一维矩阵，即：

$$G=\left[a_1^1,a_1^2,\cdots,a_1^n,a_2^1,a_2^2,\cdots,a_2^n,\cdots,a_n^1,a_n^2,\cdots,a_n^n\right]$$

2. K-mean 聚类

对一维 k-mer 序列值矩阵进行 K-mean 聚类，直至所有的矩阵元素都有一个类别。

（1）选取 K 个样本作为初始聚类中心；

$$\bar{Z}_1^{(0)},\ \bar{Z}_2^{(0)},...,\bar{Z}_k^{(0)},\mathrm{t}=0$$

（2）样本数据的分类。 计算样本到每个聚类中心的距离，归于距离最小的样本类。

$$d_{il}^{(t)}=\min\left[d_{ij}^{(t)}\right],\ i=1,2,...,N,\quad \bar{X}_i\in\omega_l^{(t+1)}$$

（3）计算每类样本的中心。

$$\vec{Z}_j^{(t+1)}=\frac{1}{n_j^{t+1}}\sum_{x_i\in\omega_j^{(t+1)}}\bar{X}_i,\ \ j=1,2,...,\mathrm{k}$$

（4）目标函数的确定及算法停止条件。 目标函数使得类内距离为零，类间距离尽可能大。常用的是类内距离度量准则。即：

$$J=\sum_{k=1}^{K}\sum_{n=1}^{N}r_{nk}\left\|\vec{X}_n-\vec{m}_k\right\|^2$$

$$r_{nk}=\begin{cases}1 & \vec{X}_n\in\ \omega_k\\ 0 & \vec{X}_n\notin\ \omega_k\end{cases}$$

其中，$\vec{m}_k$ 为类心，$\vec{m}_k=\dfrac{\sum_n r_{nk}\vec{X}_n}{\sum_n r_{nk}}$

3 组成新矩阵

提取每一类的元素值，组成新的一维矩阵，即：

$$H=[b_1,b_2,b_3,\cdots,b_n]$$

4.二分界分解。

对一维矩阵进行二分界值分解，分解成数个小的一维矩阵。

（1）计算 k-mer 序列值中最大值和最小值；

（2）对一维矩阵用二分法，利用平均值为分界值，将排序矩阵分解为两个矩阵；

（3）再用分解成的两个矩阵各自的分界值把其又分别分解成两个较小的矩阵；

（4）重复上述二分界值法，直至分解的矩阵里的元素较少为止；

5.基数排序。

对二分界值法分解后的最少元素矩阵排序。$I=[c_1,c_2,c_3,\cdots,c_n]$进行 MSD 基数排序。

（1）将 I 矩阵视为一个序列，有 n 个记录的序列$\{c_1,c_2,c_3,\cdots,c_n\}$,每个记录 c_i 的关健字 k_i 是一个 d 位 r 进制数,

即设 $c_i=(k_i^1,k_i^2,\cdots,k_i^d)\ 0\le k_i^j\le r-1,1\le j\le d$

其中关键字的第 1 位称为最高位 ,第 d 位称为最低位，k_i^1 称为最高位关键字，k_i^d 称为最低位关键字；

（2）按最高位关键字对记录进行排序 ,得到按最高位关键字排好序的若干记录的子序列 ,每一子序列中记录的最高位关键字相同,第一个子序列中记录的最高位关键字最小, 最后一个子序列中记录的最高位关键字最大；

（3）重复上述操作，分别对这若干子序列中的每一子序列中的记录按关键

字 k_i^2 进行排序,将其再分为若干子序列,这时每一子序列中记录关键字 k_i^1, k_i^2 都相同；

（4）依次按照关键字进行操作，直至排序到最低位 k_i^d

（5）将所有子序列依次连接在一起，就得到排好序的序列；

6.对所有最少元素矩阵进行 MSD 基数排序，依次连接，得到所需的排序矩阵。

5.3 模型的求解

通过 MATLAB 编程，分别求得排序和不排序的所建立索引所消耗的时间和所占用的空间如下表 2 所示：

表 2：排序与不排序的结果对比

排序索引查询				不排序索引查询			
建立时间	查询时间	运行内存	索引内存	建立时间	查询时间	运行内存	索引内存
14.1191s	0.0002s	1.306G	1.089G	3.6017s	0.3775s	0. 497G	0. 428G

5.4 排序与不排序索引下结果分析

5.4.1 对排序情况下的结果分析

1.根据所建立的索引，求得不同 K 值下所占用的时间, 并画出建立索引及利用索引查询所占用的时间曲线图，如下图 3 所示：

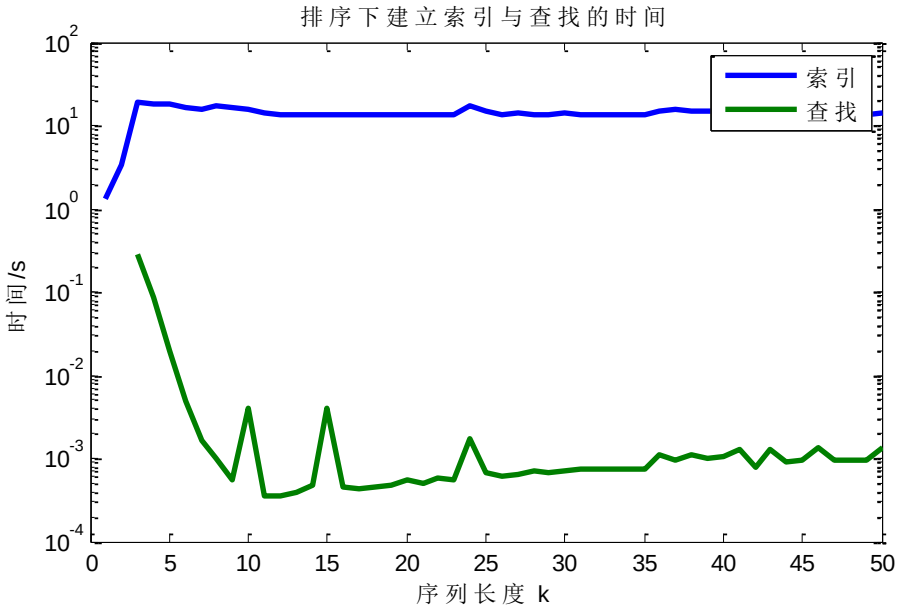


图 3：排序索引不同 K 值下所占用的时间

由图 3 可知，采用排序方法建立索引的时间约 14.1191s，且其索引建立时间在 K 值大于 3 以后比较稳定，整体波动不大；利用排序索引的查询时间很短，平均查询时间仅为 0.0002175s，效率很高。

5.4.2 排序与不排序索引下的结果对比分析

根据所建立的索引，求得排序和不排序两种情况下不同 K 值下所占用的时间，并画出建立索引及利用索引查询所占用的时间曲线图，如下图 5 所示：

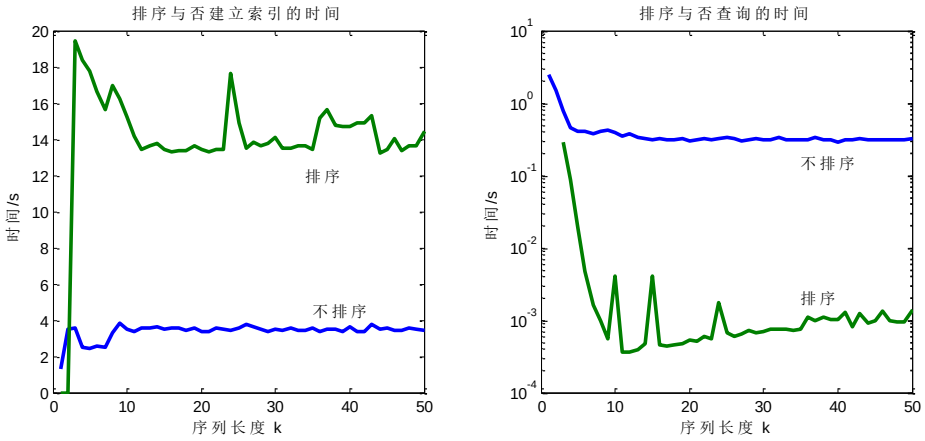


图 5：排序和不排序的结果对比

由图 5 可知：对于建立索引时间比较，排序建立索引消耗的时间高于不排序建立索引所消耗的时间，相差约 10s；对于索引查询时间的比较，排序索引查询消耗时间是不排序索引查询时间相差很大，排序索引查询时间是不排序索引查询时间的 0.11%。由于评价算法时，优先考虑查询索引时间且建立索引时间最后考虑，由此可以看出，在一定条件下，排序索引算法比不排序索引算法好。

6 算法的时间复杂度和空间复杂度的计算

6.1.1 时间开销的计算

通过 MATLAB 编程，求得随机搜索中存取节点数据块时间和在每个节点数据块中进行二分法查找时间及整个时间复杂度的结果如下表 3 所示：

表 3：时间开销模型的结果

建立索引时间	索引查找时间	时间复杂度
14.5314s	0.0002s	$O(n\log_2 n)$

6.1.2 空间开销的计算

通过 MATLAB 编程，求得随机搜索中存取节点数据块空间和在每个节点数据块中进行二分法查找空间及整个空间复杂度的优化后结果如下表 4 所示：

表 4：空间开销模型的结果

运行内存	索引内存	空间复杂度
1.306G	1.089G	$O(n)$

6.2 结果分析

6.2.1 对时间复杂度结果的分析

根据基于聚类的空间索引将 DNA 序列数据进行排序，与普遍的排序算法时间复杂度分析做对比，如下表 5 所示：

表 5：几种不同的时间复杂度结果

排序方式	冒泡排序	快速排序	插入排序	堆排序
时间复杂度	$O(n^2)$	$O(n\log_2 n)$	$O(n^2)$	$O(n^2)$

基于聚类的空间索引的时间复杂度平均情况为:$O(n\log_2 n)$,是一个以对数函数，该索引函数算法时间复杂度相对较小，所用时间也相对较短，相比之下，时间复杂度基本最优，由此得出此模型的索引时间复杂度符合要求。

6.2.2 对空间复杂度结果的分析

根据基于聚类的空间索引将 DNA 序列数据进行排序，与普遍的排序算法空间复杂度分析做对比，如下表 6 所示：

表 6：几种不同的空间复杂度结果

排序方式	冒泡排序	快速排序	插入排序	堆排序
空间复杂度	$O(1)$	$O(n\log_2 n)$~$O(n)$	$O(n)$	$O(1)$

基于聚类的空间索引的空间复杂度平均情况为:$O(n)$,与数据量 n 呈线性比例关系，该索引的空间复杂度，数据量越大，空间复杂度越大，所占用的内存越多。

7 索引性能的评价

7.1 计算出的相关指标

1. 通过 MATLAB 编程，对于 4 个评判因素进行综合考虑，得到 3 个评估因素的结果如下表 8 所示：

表 8:4 个评估因素结果

因素	索引查询速度	索引内存使用	k 值范围	建立索引时间
指标值	0.0002s	1.306G	1~100	14.1191s
评价结果	95	91.5	100	93

7.2 结果分析

由上表 7、表 8 可知，8G 内存下，所能支持的 k 值范围评价结果为 100，满足题目中所给的最大 K 值，已达到最优；索引的查询速度及内存使用，分数都在 90 分以上且时间为 0.0002s，内存仅为 1.840G。由于本文采用的是先进行排序后索引，所以建立索引的时间较长，其评价分数为 94.92。但整体的评价结果为 93.3961，评价为优。由此可以得到，本文所建立的索引方法很好。

8.问题的拓展研究

8.1 模型的准备

8.1.1 算法思想

该算法的运算过程采用了 Induced-Sorting 与递归的排序思想。算法定义了一种新的后缀类型——*LMS* 类型后缀，并证明了通过排序好的 *LMS* 类型后缀可以推导出所有 *SA* 数组中后缀的正确位置。*SA*−*IS* 算法分为三部分：第一部分，问题域缩减，选择抽样字符串 *LMS* 类型后缀；第二部分，利用 Induced-Sorting 的排序思想递归排序抽样字符串；第三部分，利用排序好的抽样字符串再次利用 Induced-Sorting 过程排序剩余部分 *SA* 元素。

8.1.2 定义名词及名词解释:

(1)字符集∑：具有全序关系的集合，在∑中的任意元素 a 和 b 之间都可以进行大小比较，且一定满足 $a<b$ 或者 $a>b$。

(2)字符串 *S*:由 n 个字符按一定次序排列组成的数组，n 表示 *S* 的长度，且任意 *S* 都以$结尾，定义$是字符集中比任意字符都小的元素，可将 *S* 表示为 $S=[s_1,s_2,s_3,...,s_{n-1},\$]$，*S* 中的第 i 个字符可以表示为 $S[i](0\le i\le n)$。

(3)子串 $S[i\cdots j]$：表示字符串 *S* 中从第 i 个字符到第 j 个字符顺次组成的字符串。

(4)后缀 $suffix(S,i)$：由字符串 *S* 的第 i 个字符开始到末尾$字符串结束的子串，表示为 $suffix(S,i)=S[i\cdots n]$。

(5)后缀数组 $SA(A)$:表示字符串 *S* 的后缀数组，是一个由 *S* 的所有后缀按照字典序排列组成的数组。

(6) *LMS* 前缀与 *LMS* 子串定义：

LMS 子串表示：（1）只有含有字符串 *S* 的哨兵元素；（2）字符串 *S* 的子串 $S[i\cdots j]$ 的首尾字符 $SA[i]$ 和 $SA[j]$ 都是 *LMS* 类型的字符，且他们中间没有 *LMS* 字符。

LMS 字符：*SA* 数组中为 *LMS* 类型的字符。它对应于某个 *LMS* 前缀或者后缀的首字符。

LMS **前缀：** 表示：（1）只含有一个单独的 LMS-character；（2）是 $suffix(i)$ 的前缀子串 $S[i\cdots k]$，其中 $S[k]$ 是其后的第一个 LMS-character。

8.2 模型的建立

对数据的处理与问题一相同，首先对矩阵进行降维处理，成为一维矩阵，再对矩阵进行 K-mean 聚类，将所有相同值的矩阵元素聚为一类，消除重复元素排序带来的时间、空间占用；由聚类类别构成完全不同的元素值的新矩阵，再利用 $SA-IS$ 算法排序。

$SA-IS$ 排序算法：

1. 缩减问题域，选择抽样字符串。

对 *SA* 所有后缀进行分类，命名规则如下：

$$S-type: if S[i] < S[i+1]$$
$$L-type: if S[i] > S[i+1]$$
$$LMS: if\ \ S[i]\ is\ S-type\ \ and\ \ L-type$$

选择小子集 *LMS* 类型的后缀作为缩减的问题域。

2.递归排序抽样后缀，利用 *LMS* 前缀推导 *LMS* 后缀。

递归排序 *LMS* 前缀过程：

(1)将所有的 $size-one\ LMS$ 前缀按照其首字符放入 *SA* 的对应桶的末尾，同一桶中的 $size-one\ LMS$ 前缀的顺序可以是任意的。

(2)从左往右扫描 *SA* 中的 *LMS* 前缀 $SA[i]$，并命名，直到扫描完毕。

Ⅰ $SA[i]$ 是 $L-type$ 那么按照命名规则为其命名；

Ⅱ 由其推导出的前缀 $per(S,SA[i]-1)$ 是 $L-type$，那么将该前缀插入对应桶的桶首，同时将桶首的位置往后移动一位，并将 $SA[i]$ 的名字赋值给 $per(S,SA[i]-1)$ 的命名结构暂时缓存；

(3)从右往左扫描 *SA* 中的 *LMS* 前缀，并命名，直到扫描完毕。

Ⅰ $SA[j]$ 是 $S-type$ 那么按照命名规则为其命名；

Ⅱ 由其推导出的前缀 $per(S,SA[i]-1)$ 是 $S-type$，那么将其插入对应桶的桶末，同时将桶末的位置往前移动一位，并将 $SA[j]$ 的名字赋值给 $per(S,SA[i]-1)$ 的命名结构暂时缓存；

(4)判断命名好的 *LMS* 前缀的名字是否具有相同的 *LMS* 子串。

Ⅰ 如果有，则将排好序的 *LMS* 前缀的名字按照其在字符串 *S* 中的顺序排序后得到新的字符串 S_1，递归调用 Induced-Sorting 排序 *LMS* 前缀的过程。

Ⅱ 如果没有相同的，则得到本次递归中排好序的 *LMS*，即本层排好序的 *LMS* 后缀。

(5)将本层排好序 *LMS* 后缀按照排好的顺序插入本次递归的 *SA*，再次利用

Induced-Sorting 过程则可得到本次递归的 SA 的完整顺序。

(6)以此类推，最后得到最后一层的 LMS 前缀，即 LMS 后缀的顺序。

3. 利用排列好的 LMS 类型后缀 Induced-Sorting 排序 SA。

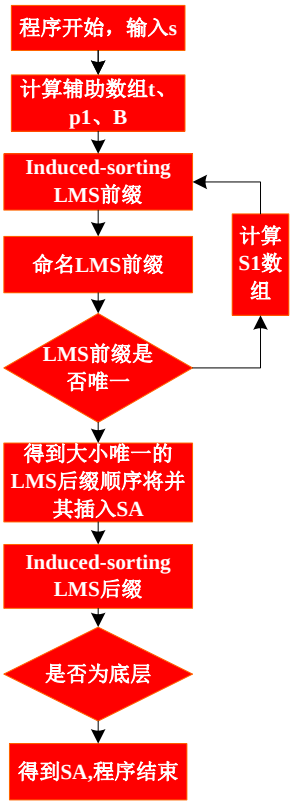
(1)将排序好的 LMS 类型后缀按其顺序逐个放入 SA 中具有相同首字符的桶的尾部。同时桶尾向前移动一位。LMS 类型后缀插入完毕。

(2)推导 $L-type$ 后缀顺序，从左往右扫描 SA，对于任意的 $SA[i]$，如果 $suffix(SA(i)-1)$ 是 $L-type$，则将 $SA[i]-1$ 放在它所属的桶的首部，同时该桶首部位置向后移动一个位置。

(3)推导 $S-type$ 后缀顺序，从右往左扫描 SA，对于任意的 $SA[i]$，如果 $suffix(SA(i)-1)$ 是 $S-type$，则将 $SA[i]-1$ 放在它所属的桶的尾部，同时该桶尾部位置向前移动一个位置。

4. k-mer 序列值矩阵排序完成。

该算法的流程图为：



8.2 模型求解

1.通过 MATLAB 编程，分别求得基于 SA-IS 算法索引模型建立和查询所消耗的时间和所占用的空间如下表 9 所示：

表 9：基于 SA-IS 算法索引模型的相关结果

建立时间	查询速度	运行内存	索引内存	k 值范围

12.5314s	0.00114s	1.519G	0.8320G	1~100

8.3 结果分析

8.3.1 对 SA-IS 的结果分析

1.根据所建立的索引，通过 MATLAB 编程，多次重复建立索引，绘制出其多次重复建立索引的折线图如下图 6 所示：

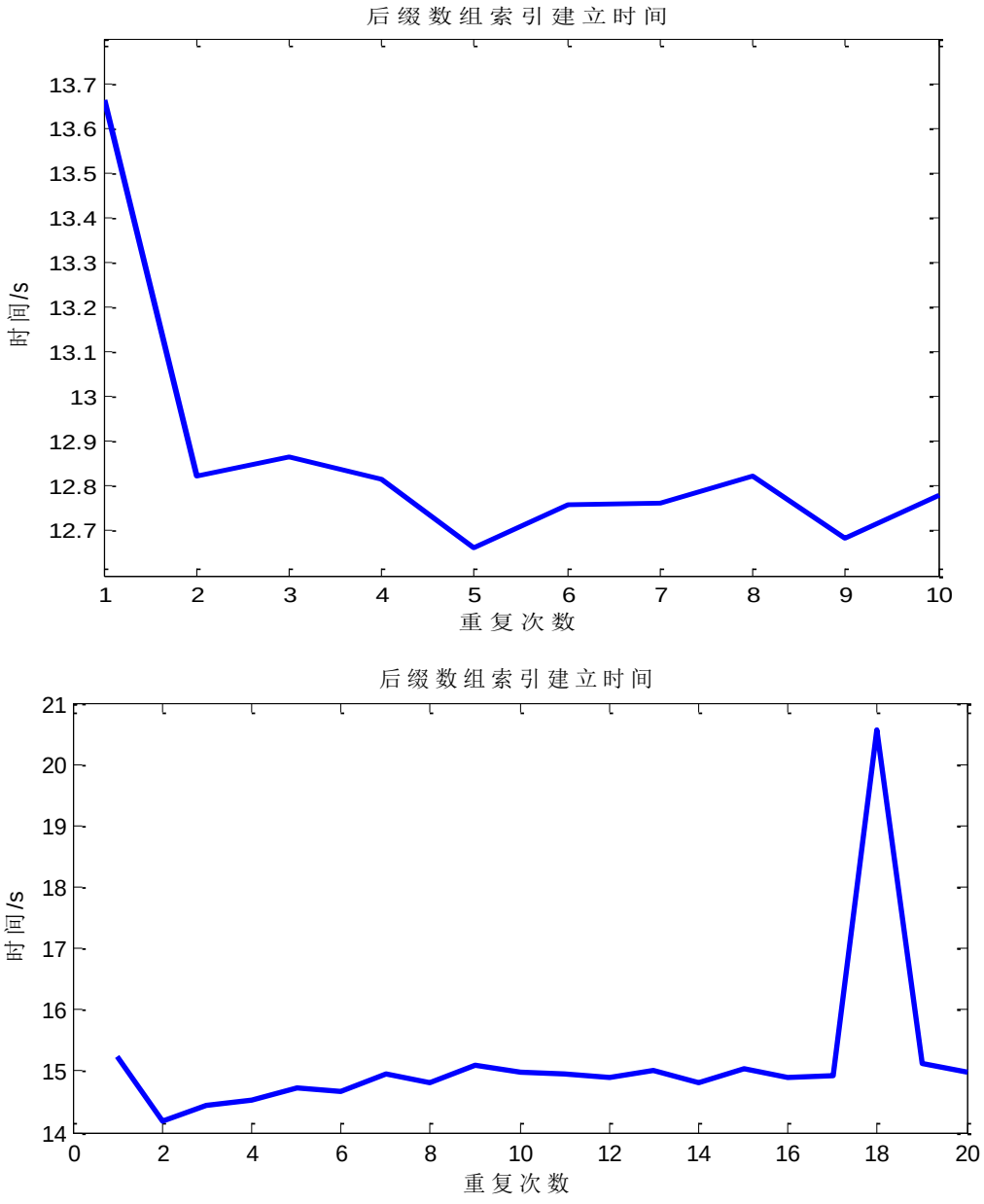


图 6：后缀数组建立所消耗的时间

由图 6（上下两图分别在不同硬件配置的电脑上的测试结果）可知，由于计算机的其他进程占用 CPU，使得每次建立索引的时间略有差别，但是相差不大。求得平均消耗的时间为 12.4s 左右，相对于基数排序的索引略有增长，但是后缀

数组的方法只需要建立一次索引方可查询所有的 K 值，功能更加可观。

2. 根据所建立的索引，通过 MATLAB 编程求得不同数据规模下所占用的时间,并画出建立索引及利用索引查询所占用的时间曲线图，如下图 7 所示：

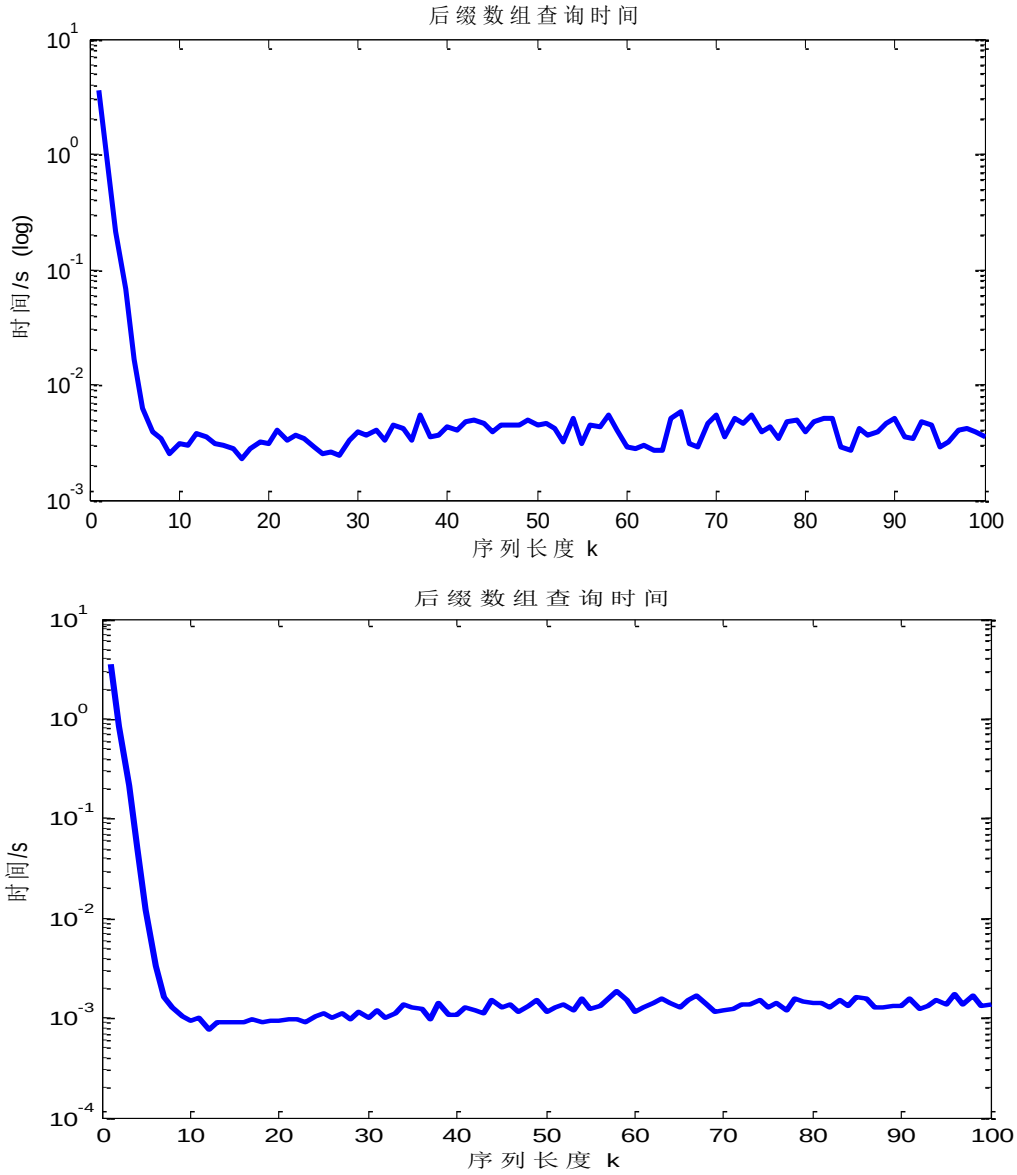


图 7：基于后缀数组查询所消耗的时间

（上下两图分别在不同硬件配置的电脑上的测试结果）

查找时间先随着 k-mer 长度的增加逐渐减少，然后趋于稳定；在 k-mer 长度小于 10 时，查找时间逐渐减少，从最长时间 5s 减少为 0.001s;之后，缓慢的增加到 0.0012s，后保持不变；表明，k-mer 长度为 10 是一个临界的长度，此后，建立索引的时间和查询时间都会逐渐趋于稳定，其中电脑运行其它进程会占用 CPU，导致查询时间会有少许波动，属于正常情况。

8.3.2 与问题一的对比

1.根据两种不同的索引方法，通过 MATLAB 编程，分别绘制出不同 K 值下的建立索引和索引查询所用的时间对比图如下图 8 所示：

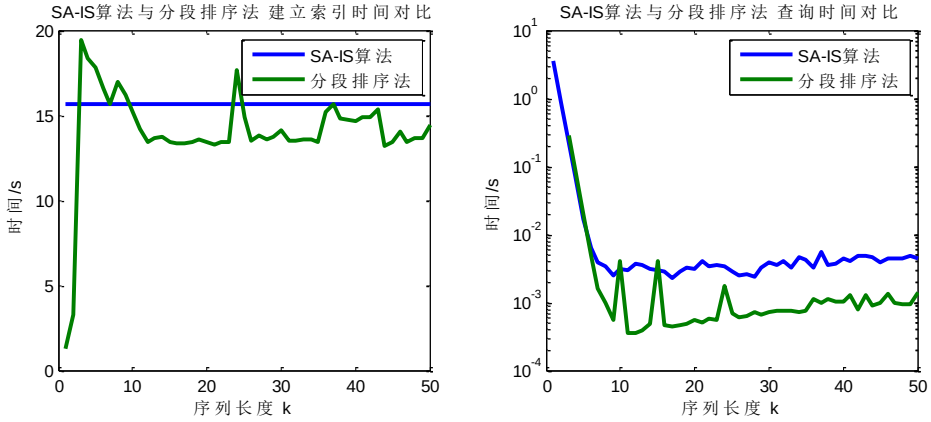


图 8：两种索引建立及查询时间对比图

由图 8 可知，对于建立索引，分段基数排序法每次查询不同长度的 K 值时，都需要重新建立索引，但是 SA-IS 算法只需要建立一次索引即可查询任意长度的 K 值，对于多次不同规模的查询来说，SA-IS 算法更能节约时间，且对于一次查询，SA-IS 算法与分段基数排序法所消耗的时间相差不大。对于索引查询来说，当 K 的范围在 1~10 之间时，两种方法的查询时间基本一样，在 K 大于 10 时，SA-IS 算法略低于分段基数排序法，相差在 0.002s 以内。

2. 根据两种不同的索引方法，通过 MATLAB 编程，分别绘制出不同规模序列下的建立索引和索引查询所用的时间对比图如下图 9 所示：

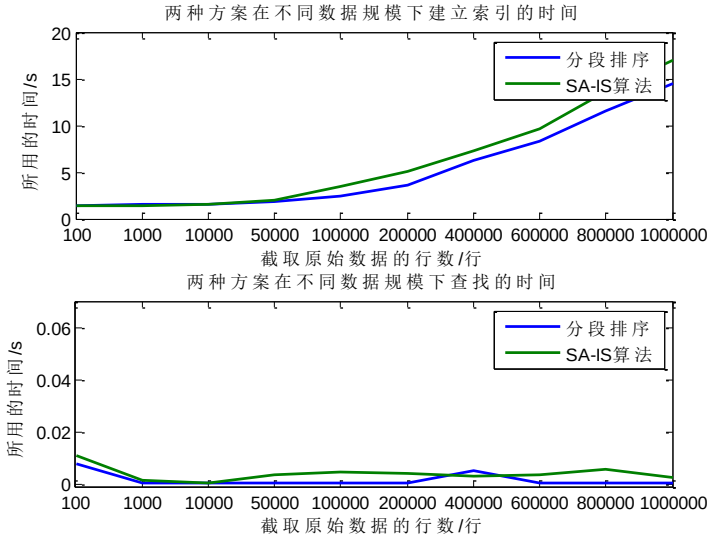


图 9：不同规模序列下的两种方法的对比

由图 9 可知，对于建立索引分析，两种算法在 100000 到 1000000 之间的数据规模逐渐增长，且其增长的函数关系接近线性关系。可以猜想在更大的数据量

下，建立索引时间的变化是类似线性增长的。对索引查询分析，数据规模为 100 时，数据量虽然小，但是消耗的时间较多，是因为通过 MATLAB-C++混合编程时，第一次调用 C++函数会有时间的 延误。且在查找过程中计算机的其他进程占用 CPU，使得查询时间有些许波动，但总体变化不大。

8.4 对 SA-IS 算法的检验

从 DNA 文本中随机选取 1000 个 k-mer 序列，建立索引，在索引中搜索，1000 个 k-mer 序列分布如下：

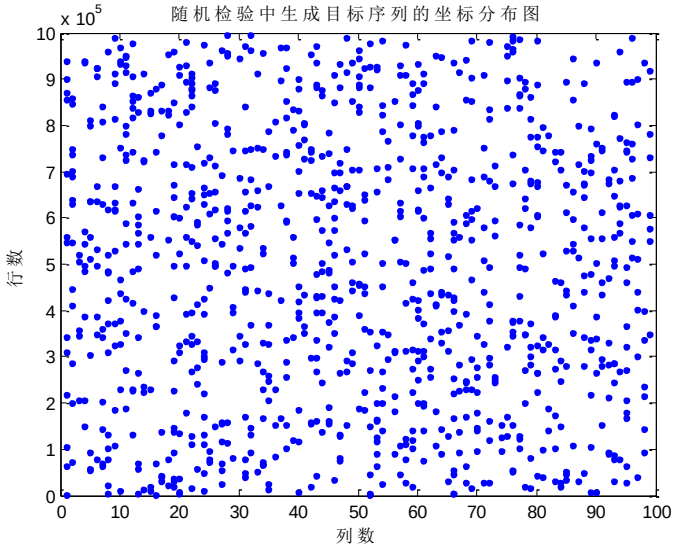


图 10：SA-IS 算法结果检验图

由上图 10 可知，生成的 1000 个 k-mer 序列分布十分均匀，长度从 0-100 均匀分布，选取序列从 1-1000000 中随机选取，选取的所有 k-mer 序列在索引中都能找到，用时 44.359771s，从时间上来看，由于存在一些较短的序列，在查询过程中暂用时间较长，导致整体时间较长，随机生成的 1000 个 k-mer 均能找到表明算法具有很高的准确性。

9.索引算法软件 GUI 构想

基于所建立的模型的基础上，本文已经满足了能够根据任意给定的 K 值所查找的时间及空间的大小，并给出最优索引下的时间与空间。为了能够更方便的求取任意 K 值下的索引，本文采用 MATLAB 的 GUI 界面，初步建立索引。

9.1 软件制作的基本思想

1.产生一个能够简易操作的界面：在此，本文运用了 MATLAB 中的 GUI 界

面，其界面表达更加简便。

2.能够输入 K 值，根据情况，可以输入任意 K 值。

3.能够输出值：当输入 K 值时，可以相应得查询时间。

9.2GUI 制作

9.2.1GUI 基本界面设计

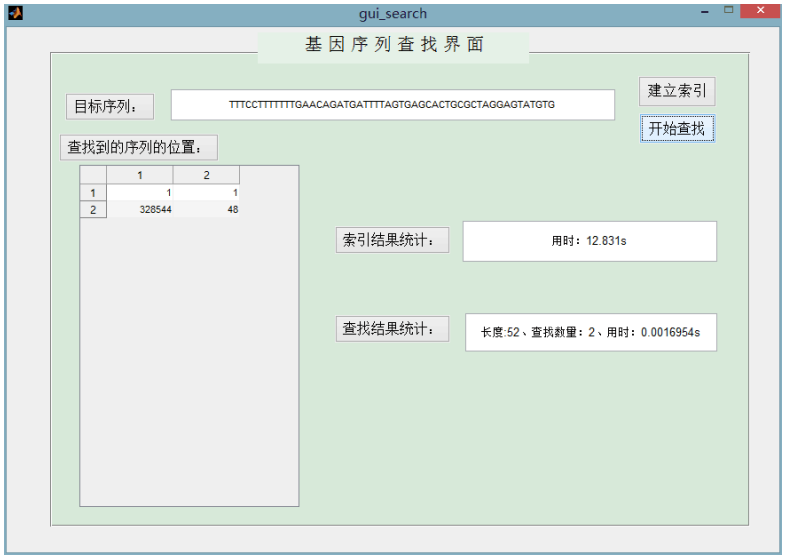
1.输入任意 K 值，基于问题一的模型，将其主函数编译为调用函数，在用户界面中的“建立索引”和“开始查找”按钮上调用函数，求得 GUI 结果界面图如下图所示：

假定 K 值为 8 序列为“TTTCCTTTT”，其输出界面为：

	1	2
1	7086	90
2	14084	69
3	310723	75
4	576259	3
5	795144	8
6	220714	30
7	296156	85
8	317625	85
9	549956	16
10	722239	61
11	56158	21
12	555422	53
13	590682	84
14	611781	8
15	810149	90
16	432388	24
17	856071	81
18	888601	19
19	185801	40

由上图可知，K 值为 10 序列为“TTTCCTTTTT”建立索引需要 12.831s，索引查询需要 0.0018764s。

假定 K 值为 51：“TTTCCTTTTTTTGAACAGATGATTTTAGTGACTGCT”，序列为其输出界面为：



由上图可知，K 值为 52 的序列为：“TTTCCTTTTTTTGAACAGATGATTTTAGTGACTGCT”，建立索引需要 12.8s，索引查询需要 0.0016954s。

10.模型的评价、改进及推广

10.1 模型的评价

10.1.1 模型的优点

优点一：对数据处理前，将其转化为数字矩阵，减少了内存的暂用，并通过基于 k-mean 和二分界值法的基数排序算法建立索引，提高了搜索速度，相比一般的搜索方法有了较大的提高，具有一定的推广性和适用性。

优点二：建立基数排序时，将 k-mer 序列值矩阵进行降维处理，再用二分界值法将序列长度缩短化，对小序列进行基数排序，降低了排序的难度，在时间、空间复杂度上有一定优化。

优点三：利用模糊综合判断对索引进行综合评价，将定性的重要程度转化为定量的指标，清晰直接的呈现了它们之间的关系。

10.1.2 模型的缺点

缺点一：索引的建立过程可能稍微繁琐，占用了一定的内存。

缺点二：计算时间复杂度时，我们计算主要操作的时间复杂度，对其他一些时间复杂度没有计算，可能有一定影响。

缺点三：模型对数据进行处理具有一定的局限性，只含四个碱基，而不存在变异的情形。

10.2 模型的改进

改进一：针对缺点一，将碱基转化为数字后，可以用二进制的形式去表示，这样可以更加减少内存的使用。

改进二：针对缺点二，可将不是四个基本碱基的其他碱基转化为相应的数值，再进行 SA-IS 算法排序，进行相应的搜索，得到所要的结果。

10.3 模型的推广

本文通过将四种碱基数值化，利用基于 k-mean 聚类和二分界值法的基数排序对 k-mer 序列进行排序处理，建立索引方法，极大的提升了查询速度和减小了内存的使用，接着用 SA-IS 算法进行拓展，只用建立索引一次即可查询所有 k-mer 序列，极大地提高了算法的执行效率，然后利用模糊综合判断，将定量的重要程度转换为定性指标，很好的衡量了索性性能。基于 k-mean 聚类和二分界值法的基数排序很好的解决了一类大数据的索引问题，索性性能良好，SA-IS 算法也有很高的效率,有很好的借鉴性。

11.参考文献

[1]Nong, Ge; Zhang, Sen; Chan, Wai Hong (2009). Linear Suffix Array Construction by Almost Pure Induced-Sorting. 2009 Data Compression Conference. p. 193. doi:10.1109/DCC.2009.42. ISBN 978-0-7695-3592-0.

[2] 陈怀谟,王卡佳. 最佳基数排序[J]. 计算机工程与科学. 1992(04)

[3]刘勇.基于 GPU 的内存数据库索引技术研究,计算机应用专业,2013 年 9 月 26 日

[4]王玉梅,模糊综合分析对收益评价商誉改进研究,资产评估专业,2014 年 04 月

[5]潘润秋,刘珺,宋丹妤.基于迷糊综合分析法的农用地分等方法,第 30 卷,第 18 期,2014 年 9 月

12.附录

1 数据导入：

```
function C=getDataAll()
fid = fopen('solexa_100_170_1.fa');
C = textscan(fid, '%s');
fclose(fid);
C=C{1};
C1=C(5:5:end);
fid = fopen('solexa_100_170_2.fa');
C = textscan(fid, '%s');
fclose(fid);
C=C{1};
C2=C(5:5:end);
C=[cell2mat(C1);cell2mat(C2)];
end

clear,clc
tic
lab_data=uint8(getDataAll());
toc
data_uint8=lab_data;
tic
order='ACTG';
for i=1:4
      lab_data((find(data_uint8==order(i))))=i-1;
end
toc
```

2 分段排序程序

```
clear,clc
% time=[];
% for iTime=1:10
    is_C=1;
    [~, b]=memory; b.PhysicalMemory
    %input('    is_C :      ');
    global R_data order

    order='ACTG';
    R_data=1e6;
    % tim=[];
    % for KLen=11
    MM=14;
    tic
    disp('build the index :')
    % tic%'AACTTTTTCGACACCCCTTCTAAATCTCA'

all='TTTCCTTTTTTTGAACAGATGATTTTAGTGAGCACTGCGCTAGGAGTAT
GTGTTGAATCTCCAACCCTCGGAACATAGTCGTTGCTCTTGCGGACTGAGA'
;
    %
GGAATCTGCAACTCGGATGGCCGTCTCAAGTAAGCAACAACATTGTTGCTT
GCGGCTTCCCGGGTGTCGCTAAGCGTTTAAAAAGTTGTTAGGATTAACC

Key='TTTCCTTTTTTTGAACAGATGATTTTAGTGAGCACTGCGCTAGGAGTA
TGTGTTGAATCTCCAACCCTCGGAACATAGTCGTTGCTCTTGCGGACTGAG
A';
%all(1:iTime);
    len_Key=numel(Key);
    fprintf('it''s :::::::::    %d    \n',numel(Key))
    %         disp(KLen)
    if len_Key~=1
    len_lib_data = build_index_fun( min(len_Key,MM) )';
    len_lib_data=len_lib_data(:)';
    if is_C
        [s_len_data s_lab]= sort(len_lib_data);
        s_lab=uint32(s_lab);
        clear len_lib_data ;
```

```
        s_len_data=double(s_len_data);
    end
    else
    end
    %       len_lib_data=double(len_lib_data);
    load data_uint8.mat
    toc
%       time(1,iTime)=toc;
    %% find
%       tim=[];
    % clc
    % for   isC=0:1
    for i=1:1e5
        %       i=i;
    end
%       tic
    %
    %
TTTCCTTTTTTTGAACAGATGATTTTAGTGAGCACTGCGCTAGGAGTATGTG
TTGAATCTCCAACCCTCGGAACATAGTCGTTGCTCTTGCGGACTGAGA
    %
GGAATCTGCAACTCGGATGGCCGTCTCAAGTAAGCAACAACATTGTTGCTT
GCGGCTTCCCGGGTGTCGCTAAGCGTTTAAAAAGTTGTTAGGATTAACC
    %
all='TTTCCTTTTTTTGAACAGATGATTTTAGTGAGCACTGCGCTAGGAGTAT
GTGTTGAATCTCCAACCCTCGGAACATAGTCGTTGCTCTTGCGGACTGAGA'
;
%       Key='CAGATGAATTTA';
    tic

    len_Key=numel(Key);%tim=[tim,toc];tic
    if len_Key~=1
    key=Key(1:min(len_Key,MM)); %tim=[tim,toc];tic
    sys4_key=get_sys4(Key(1:min(len_Key,MM)));% tim=[tim,toc];tic
%       tim=[tim,toc];tic
    if is_C
        f_data=find_C_C(s_len_data,sys4_key,s_lab);
    else
        f_data=find(len_lib_data==uint16(sys4_key));%[row col]
    end
```

```
%         tim=[tim,toc];tic
      [row,col]=solve_row_col(f_data,len_Key,MM,data_uint8,Key);
      else
            [row,col]=find(data_uint8==uint8(Key));
      end
          toc
%               tim=[tim,toc];tic
%         time(2,)=toc;
%         format long;vpa(tim,5)
%         sum(tim)
      result_analyse(row,col,Key,len_Key,data_uint8)
% end
%%
```

3 后缀数组

```
function len_data=SuffixArray()
global R_data
R_data=1e6;
lab_data=load('uint8_lab_data.mat','lab_data');
lab_data=lab_data.lab_data;
len_key=uint32(16);
% type='uint32';
% eval(['lab_data=' type '(lab_data);']);
% eval(['sys_len=' type '(4.^(len_key-1:-1:0));' ]);
% eval(['sys_len_mat=' type '(repmat(sys_len,R_data,1));']);
lab_data=[uint32(lab_data) zeros(R_data,15,'uint32')];
sys_len=4.^(len_key-1:-1:0);
sys_len_mat=repmat(sys_len,R_data,1);
len_data=zeros(R_data,100,'uint32');
i=1;
len_data(:,i)=sum(lab_data(:,i:i+len_key-1).* sys_len_mat    ,2);
% tem=4^(len_key-1);
for i=2:100
      len_data(:,i)=                              (len_data(:,i-1)-lab_data(:,i-1)*4^15)*4
+lab_data(:,i+len_key-1);
end
function [row,col]=solve_row_col(f_data,len_Key,MM,data_uint8,Key)
row=fix(f_data/100)+1;
col=mod(f_data,100);
tem=find(col==0);
```

```
row(tem)=row(tem)-1;
col(tem)=100;
tem=find(col+len_Key-1<=100);
col=col(tem);row=row(tem);
if len_Key>MM
    for i=1:len_Key-MM
        tem_r=[];tem_c=[];
        for j=1:numel(row)
            if
numel(row)>0&&((col(j)+i+MM-1)<=100)&&char(data_uint8(row(j),col(j)+i+MM-1
))==Key(i+MM) %% here should be improved
                tem_r=[tem_r;row(j)];tem_c=[tem_c;col(j)];
            end
        end
        row=tem_r;col=tem_c;
    end
end
function tem=rightLab(s_len_data,sys4_key,nKey)
% sys4_key=get_sys4(Key);
ta=4^(16-nKey);
if fix(double(s_len_data(end))/ta )<sys4_key;
    tem=0;
elseif fix(double(s_len_data(end))/ta )==sys4_key;
    tem=numel(s_len_data);
else
    lab=[1 numel(s_len_data)];
    k=0;temp=0;
    while ~k
        if
fix(double(s_len_data(lab(2)))/ta  )==sys4_key&&fix(double(s_len_data(lab(2)+1))/ta
)>sys4_key
            k=2;tem=lab(k);
        elseif
fix(double(s_len_data(lab(1)))/ta)==sys4_key&&fix(double(s_len_data(lab(1)+1))/ta)
>sys4_key
            k=1;tem=lab(k);
        else

            if
fix(double(s_len_data(fix(mean(lab))))/ta)>=sys4_key&&fix(double(s_len_data(fix(
```

```
mean(lab))+1))/ta)>sys4_key
                    lab(2)=fix(mean(lab));
                else
                    lab(1)=fix(mean(lab));

                end
            end
            if lab(2)-lab(1)<=1&&k==0
                temp=temp+1;
                if temp>=3
                    tem=0;disp('not found')
                    break
                end
            end
        end
end
function tem=leftLab(s_len_data,sys4_key,nKey)
% sys4_key=get_sys4(Key);
ta=4^(16-nKey);
if fix(double(s_len_data(1))/ta )>sys4_key;
        tem=0;
elseif fix(double(s_len_data(1))/ta )==sys4_key;
        tem=1;
else
        lab=[1 numel(s_len_data)];
        k=0;temp=0;
        while ~k
            if
fix(double(s_len_data(lab(1)))/ta )==sys4_key&&fix(double(s_len_data(lab(1)-1))/ta)
<sys4_key
                k=1;tem=lab(k);
            elseif
fix(double(s_len_data(lab(2)))/ta)==sys4_key&&fix(double(s_len_data(lab(2)-1))/ta)
<sys4_key
                k=2;tem=lab(k);
            else

                if
fix(double(s_len_data(fix(mean(lab))))/ta)<=sys4_key&&fix(double(s_len_data(fix(
mean(lab))-1))/ta)<sys4_key
```

```
                lab(1)=fix(mean(lab));
            else
                lab(2)=fix(mean(lab));

            end
        end
        if lab(2)-lab(1)<=1&&k==0
            temp=temp+1;
            if temp>=3
                tem=0;disp('not found')
                break
            end
        end
    end
end
function result_analyse(row,col,Key,len_Key,data_uint8,maxShow)
% fprintf('查找的结果:\n')
% if numel(row)>5||numel(row)==0
    fprintf('查找的结果\n==========%d
==========\n',numel(row)),disp(' ')
    disp([row(1:min(maxShow,numel(row))) col(1:min(maxShow,numel(row)))])
%     disp(' ')
% end
fprintf('目标序列:\n')
disp(Key),disp(' ')
if numel(row)>maxShow
    for i=1:maxShow%numel(row)
        fprintf('row %d col %d :\n',row(i),col(i))
        disp(char(data_uint8(row(i),col(i):col(i)+len_Key-1)))%,disp(' ')
    end
else
    for i=1:numel(row)
        fprintf('row %d col %d :\n',row(i),col(i))
        disp(char(data_uint8(row(i),col(i):col(i)+len_Key-1)))%,disp(' ')
    end
end

clear,clc
is_C=1;
```

```
% [~, b]=memory; b.PhysicalMemory
global R_data order
order='ACTG';
R_data=1e6;
MM=14;
tic
disp('build the Suffix Array :')
len_lib_data=SuffixArray()';%                                    len_lib_data        =
build_index_fun( min(len_Key,MM) )';
len_lib_data=len_lib_data(:);
if is_C
    [len_lib_data s_lab]= sort(len_lib_data);
    s_lab=uint32(s_lab);
%       clear len_lib_data ;
%       s_len_data=double(s_len_data);
end
load data_uint8.mat
toc
%% My Find Way
clear row col
Key='CAGATGATTTTAGTGA';% 由低位到高位
nKey=min(numel(Key),16);%'ACTG'
if nKey<16
    disp('the Key is shorter than 16')
    key=Key;
else
    disp('the Key is longer than 16')
    key=Key(1:16);
end
sys4_key=get_sys4(key);
tic
lab(1)=leftLab(len_lib_data,sys4_key,nKey);
lab(2)=rightLab(len_lib_data,sys4_key,nKey);
disp(lab),fprintf('sum ::   %d     rate::   %f \n',diff(lab)+1,diff(lab)/1e8)
[row,col]=solve_row_col(double(s_lab(lab(1):lab(2))),numel(Key),16,data_uint8,Key
);
toc
result_analyse(row,col,Key,numel(Key),data_uint8,2)%maxShow
numel(Key)
%% the final   test
```