基于散列表的k-mer序列快速查找算法

蔡琪　朱鹤

摘　　要

生物序列的k-mer频次统计是生物信息处理中一个非常基础且重要的问题.本文将针对特定长度的基因序列分析k-mer的频率分布,尝试使用不同的散列函数建立合适的散列表作为数据索引,分析算法的时间和空间复杂度,并根据计算机处理数据的特点进行算法优化.

关键词：散列函数　k-mer　复杂度

1 问题的重述

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} 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} . 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.

1.1 问题分析

对于问题一,$10^6 \times 100$的DNA序列,当k较小时,对应的k-mer重复率会很高,为了快速检索任意一个k-mer所在的DNA序列编号和相应序列中出现的位置,需要去除序列中重复冗余的信息,建立合适的数据索引.建立索引的方法大致有两种：基于hash的和基于BWT. 题目中要求每次建立的索引只需支持一个k值,且DNA序列的总数不是太大,我们采用对序列建立散列表作为数据索引的方法.通过建立散列表可以实现O(1)的查找速度,并且可以很容易的利用多线程编程加快建立索引的速度.我们将根据分析的序列信息采用合适的参数建立散列表.

对于问题二,散列表在没有冲突的情况下可以实现O（1）的查找速度,为了提高检索速度.我们采用以空间换取时间的策略,在内存限制范围内,设置比较大的散列表,从而减少冲突的概率.冲突解决策略的选择也会影响查询的速度,好的冲突解决策略可以减少因冲突产生的探测次数,提高检索速度.原序列的保存采用字符型变量,为了节省空间,我们采用二进制表示四种字符,可以将数据量压缩到25%.

对于问题三,分析建立索引的时间、空间复杂度.对于不同长度的k-mer,我们分析平均时间复杂度和最坏情况时间复杂度.同时根据序列的概率分布情况可以得到更加准确的时间和空间利用.

于问题四,通过建立散列表查询的时间和空间复杂度都是常数级的,通过分析序列的分布情况可以得到更加准确的时间空间分布.

对于问题五,在允许的8G内存限制下,根据k-mer的长度分析保存地址信息需要的节点数目,进而得出最大允许的k-mer长度.

2 基本假设

1. 假设题目所给数据及建模收集数据可以代表一般情形
2. 假设采用的散列函数对不同序列生成的散列值不同
3. 程序测试都是在win8.1+Intel core i7,2.4GHz+内存8G+ Visual Studio Community 2013平台上完成

3 模型的建立与求解

3.1 问题一:建立数据索引

对基因序列的分析 已有研究发现,DNA序列中的短序列片段（通常长度为6 个碱基以下）不仅具有高度重复性,而且在DNA序列中的分布是具有一定特点的.Trifonov和Sussman提出DNA序列中短序列片段的频率分布呈现为一条具有相对稳定性的曲线,后来Borodovsky和Sprizhitskii 通过实验研究和数据分析验证和支持了这一理论.DNA序列的这一特征是作为生物学数据所具有的特异性表现,从某种程度上代表和反映了生物的本质特征.通过对这些序列分布是否均匀进行分析,可以更好的选择散列表的参数.

KL散度 [1] Kullback-LeiblerDivergence是统计学家Kullback和Leibler于1951年提出的一种相关熵表示方法,能度量两个离散或连续概率分布p和q 之间的差异或接近程度.本文通过将序列的k-mer频数分布映射成相应的概率分布,来对比分析实际DNA序列与等长随机序列之间k-mer 分布的差异. 对于长度为k的k-mer,不同的k-mer种类数为4^k,假设每种k-mer出现的频数为n_i,其中$i = 1, 2, \ldots, 4^k$,则在一条长度为L的DNA序列中,k-mer分布概率表示为$p_i = \frac{n_i}{L-k+1}$,设实际DNA序列的k-mer分布概率表示为p_i,A、G、C、T四种碱基等概率均匀分布的随机序列的k-mer分布概率表示为q_i,则p_i,q_i的KL散度可以由如下公式计算：

$$D_{KL}(p|q) = \sum_{i=1}^{4^k} p_i log(\frac{p_i}{q_i}) \tag{1}$$

由公式(1)可以看出,$D_{KL}(p|q) >= 0$恒成立.当p_i,q_i的概率分布完全相同时,KL散度值为零,表示两个序列完全一样.序列中碱基排列不一致时,KL 散度大于零.试验中假设等长随机序列之间k-mer均匀分布,则:

$$q_i = \frac{1}{4^k} \quad i = 1, 2, \ldots, 4^k \tag{2}$$

郝柏林院士等在 2000 年提出了一种使用2D 分形图像的可视化方法来反映序列的k-mer 的频次分布,并在此基础上设计了基于B 树的快速k-mer 频次统计算法[2].DNA序列中的每一个k-mer都可以映射成坐标平面上的一个点（x,y）,相同的k-mer对应的（x,y）坐标相同,不同的坐标点对应了不同的k-mer,某一个坐标点的值越大,表示该点对应的k-mer在整个DNA序列中出现的频率越高.对于$k < 8$时,我们将ATCG的序列按照下面的对应关系转化为整型数表示.

表 1: ATCG字符与二进制数的对应关系

A	T	C	G
00b	01b	10b	11b

取数字的低位作为x坐标,高位作为y坐标,序列出现的次数决定颜色的深浅,绘制的部分图像如下:

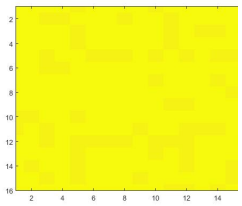

图 1: k=4

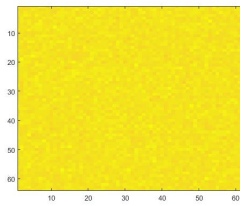

图 2: k=6

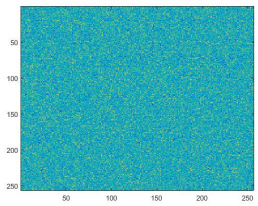

图 3: k=8

对于k>8时,将序列对应的x,y坐标对256取余数后绘制的部分图像如下：

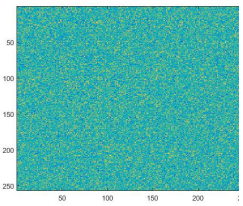

图 4: k=20

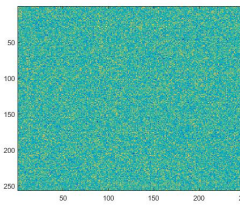

图 5: k=60

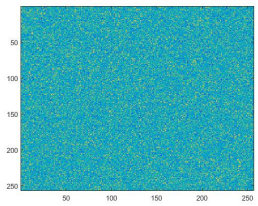

图 6: k=100

根据绘制的图像可以看出来,颜色分布基本均匀,说明不同序列的频数分布大致相同.在下面的分析中我们认为对于任意的k值,不同k-mer的分布为均匀分布.

3.1.1 散列函数的选择

散列函数是一种从任何一种数据中创建小的数字"指纹"的方法.散列函数把消息或数据压缩成摘要，使得数据量变小，将数据的格式固定下来.好的散列函数在输入域中很少出现散列冲突.利用散列值作为数据保存的位置索引，可以使访问速度达到$O(1)$的时间复杂度. 下表[3]是对常用的散列函数的测试,测试数据一是
216553个英文单词,测试数据二是216553个随机GUID，测试数据三是从1到216553的数值.对于每组测试数据记录了平均散列时间和冲突数:

表 2: 散列函数对比

Hash	Lowercase	Random UUID	Numbers
MurmurHash	145 ns 6 collis	259 ns 5 collis	92 ns 0 collis
FNV-1a	152 ns 4 collis	504 ns 4 collis	86 ns 0 collis
FNV-1	184 ns 1 collis	730 ns 5 collis	92 ns 0 collis*
DBJ2a	158 ns 5 collis	443 ns 6 collis	91 ns 0 collis***
DJB2	156 ns 7 collis	437 ns 6 collis	93 ns 0 collis***
SDBM	148 ns 4 collis	484 ns 6 collis	90 ns 0 collis
SuperFastHash	164 ns 85 collis	344 ns 4 collis	118 ns 18742 collis
CRC32	250 ns 2 collis	946 ns 0 collis	130 ns 0 collis
LoseLose	338 ns 215178 collis	- -	- -

散列函数的选择主要考虑计算时间和冲突的数目,综合上面已有的数据,我们采用MurmurHash作为散列函数.MurmurHash 是一种非加密型哈希函数，适用于一般的哈希检索操作.由Austin Appleby在2008年发明[4]，并出现了多个变种，都已经发布到了公有领域.与其它流行的哈希函数相比，对于规律性较强的key，MurmurHash的随机分布特征表现更良好. 对于满足$4^k < H$的k值,利用表2的对应关系把序列对应的二进制数作为散列值,可以节省计算散列值的时间,也完全避免了冲突的产生.如对于序列"ATCG"，其对应的哈希值为"00011011b".

3.1.2 散列表的大小

散列表的大小必须大于序列种类数才可以保存所有的数据.设，H表示散列表的大小，B_k表示长度为k的k-mer的种类数，m表示每条DNA序列的长度，n表示DNA序列的数目。散列表的最小值可以由以下公式确定：

$$H \geq \max_k\{B_k\} = \max_k\{\min\{4^k, (m-k+1)*n\}\} \quad k = 1, 2, \ldots, 100 \tag{3}$$

通过计算可以得到当$k = 14$的时候,K取得最大值,为$87*10^6$,这就要求为了确保可以建立k从1到100的所有索引,必须要求散列表的大小不小于87×10^6.根据对给定的基因序列进行分析,我们得到B_k与k的关系曲线如下：

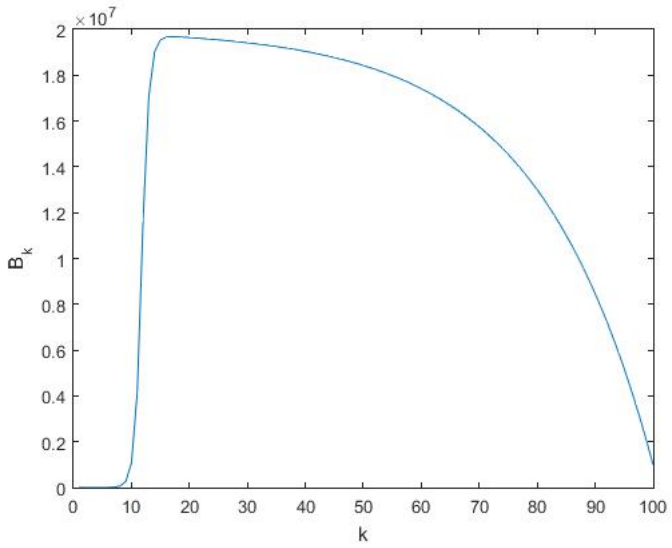

图 7: k=20

根据测算的数据,当k=17是,B_k取得最大值,为19682399. 设散列表中非空位置的数目为H_1,散列表的载荷因子α定义为：$\alpha = \frac{H_1}{H}$. α太大会使冲突的数目增多，降低建立索引和查询的时间，α太小会浪费存储空间，需要根据实际情况调整α的大小.

3.1.3 散列表节点的选择

每个散列表节点要保存对应k-mer序列的标识符和地址,我们采用如下图所示的数据结构

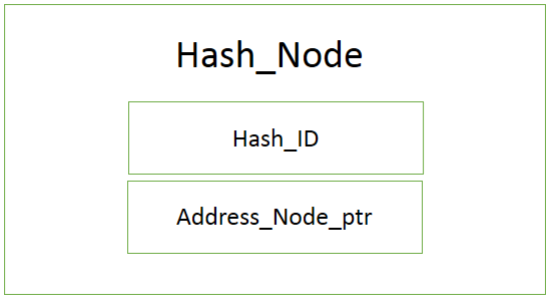

图 8: 散列节点结构

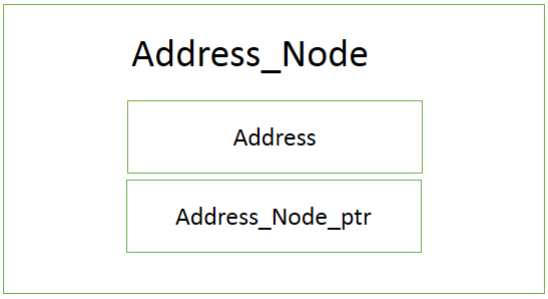

图 9: 地址节点结构

散列节点中的Hash_ID保存序列生成的散列值的高64位作为识别序列的标志, Address_Node_ptr为指向地址节点的指针,地址节点保存该序列的地址的指向下一个同一序列地址节点的指针.添加一个k-mer序列S_K时,若散列表中对应位置的$Hash_ID$为0时,将该序列的$Hash_ID$保存到该节点中,将地址保存到节点的地址空间中；如果散列表对应的位置$Hash_ID$不为0,则分配一个新的Address_Node 保存该序列的信息.最坏情况下,出了第一个插入的序列外其他的序列全部发送冲突。需要申请新的节点数H_N为:

$$H_N = (m - k + 1) \times n - 1 \tag{4}$$

使用X86程序占用的空间大小约为762MB,使用x64程序占用的空间约为1143MB,经测试可以申请足够空间. 散列表的结构如下图所示：

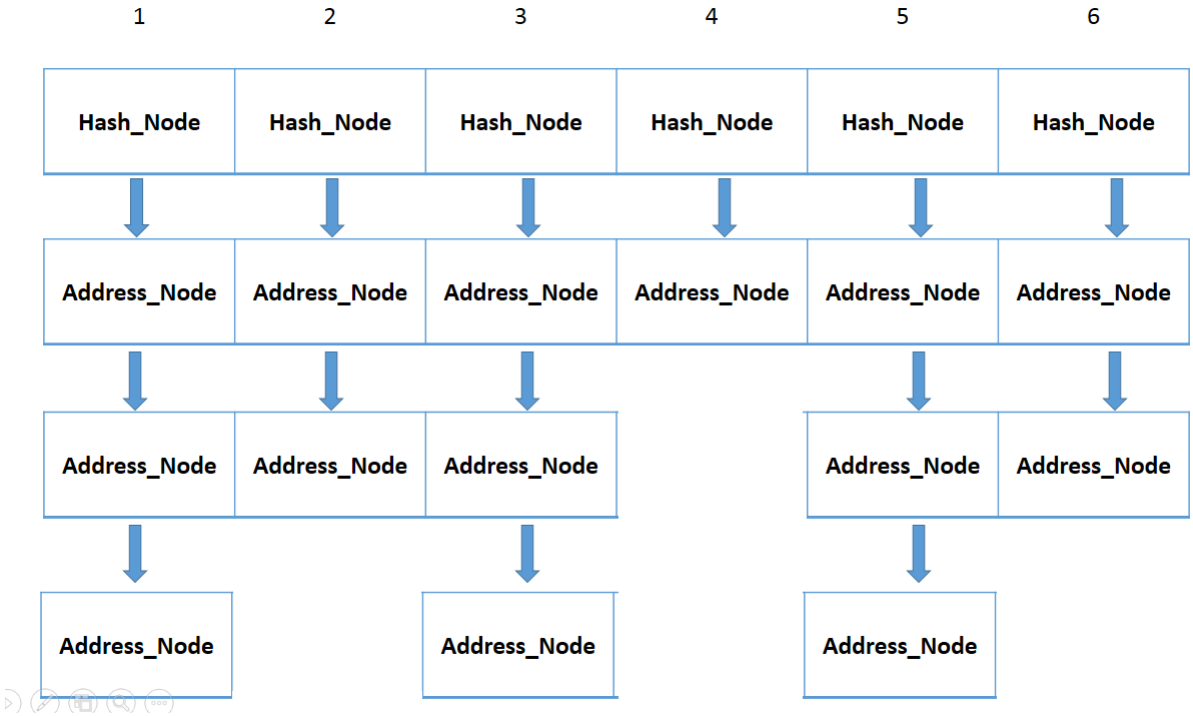

图 10: 散列表结构

3.2 合适的冲突解决方法

解决冲突的方法主要有开放寻址法和链接技术.由于在散列表中需要保存序列的地址,每个节点后面连接了一个链表,故采用开放寻址法解决冲突。开放寻址法主要有：线性探查,二次探查,再散列,双重散列. 线性探查计算下一个节点速度较快,但是可能出现堆积导致探测次数过多.二次探查可能出现再次求出的地址经过几次过程后回到起始地址,从而进入死循环.再散列的方法可以避免线性探查导致的堆积现象,但当散列表中的空位较少时可能很难找到空位. 我们通过散列函数产生的散列值对散列长度取模作为该序列在散列表中的地址.对于不同的k值我们测得在散列表的长度H=49999991大小下冲突的次数,测量数据如下所示：

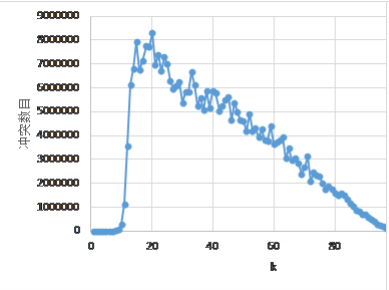

图 11: 线性探查的冲突数与k的关系

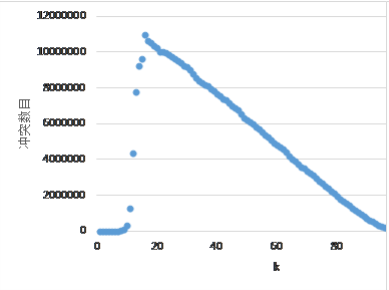

图 12: 一次再散列+ 线性探查冲突数与 k 的关系

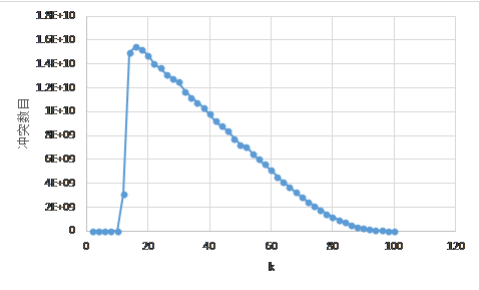

图 13: 二次再散列+ 线性探查冲突数与 k 的关系

3.2.1 充分利用计算机的性能

预先将序列读取到内存中 计算机从内存和磁盘中读取的速度差别在两个数量级左右,为了提高运行时的速度,减少计算机多次从磁盘中读取数据浪费的时间,我们先将所有的基因序列读取到一个大数组中,一共占用10^8B大小的空间,经测试可以保证内存占用不超过限制.

采用多线程的方式 现在的计算机大都支持多线程的方式运行程序,采用多线程可以充分利用CPU资源,提高运行的速度,特别是对于计算量比较大的程序,多线程的优势更加突出.由于开始已经将所有的序列读取到内存中,我们可以简单的把数据平均分成N份,每个线程执行其中的一份,根据实际测试,采用多线程后,对n个每个长度为m的序列求散列值程序的运行时间可以提高接近8 倍.

3.3 建立索引的复杂度

3.3.1 时间复杂度

采用对每个基本操作设定一个时间参数计算时间复杂度。假设散列函数没有发生冲突。设总的时间为T,读取数据的时间为T_1,读取每个字节的平均时间为G_1,则$T_1 = n \times m \times G_1$；计算所有散列值的时间$T_2$,对k长序列计算散列值的时间为$T_{k2}$,$G_2$是与计算散列值相关的时间常数，$T_{k2} = k \times G_2$,根据公式(3)计算出的$B_k$,可得$T_2$=$G_2 \times B_k$,$T_3$表示建立散列表的时间,$G_3$表示插入一条散列值的时间常数,则$T_3 = G_3 \times B_k$.

$$T = T_1 + T_2 + T_3 \tag{5}$$
$$= n \times m \times G_1 + n \times (m - k + 1) \times k \times G_2 + n \times (m - k + 1) \times G_3 \tag{6}$$
$$= O(nm + (nk + n)(m - k + 1)) \tag{7}$$

最坏时间复杂度 最坏情况下，插入的每个序列都发生冲突，插入一条散列值得时间不再是常数，设在散列表中移动一次位置需要G_4的时间，则$T_3 = \frac{B_k \times (B_k - 1)}{2} \times G_4$

$$T = T_1 + T_2 + T_3 \tag{8}$$
$$= n \times m \times G_1 + n \times (m - k + 1) \times k \times G_2 + \frac{n \times (m - k + 1) \times (n \times (m - k + 1) - 1)}{2} \times G_4 \tag{9}$$
$$= O(nm + (nk + n)(m - k + 1) + n^2(m - k + 1)^2) \tag{10}$$

3.3.2 空间复杂度

平均空间复杂度 平均空间复杂度由三部分组成：保存序列的空间,散列表占用空间,新分配的节点占用空间.假设平均情况下没有冲突。设总的空间为S,保存序列占用的空间为S_1,每个字符占用的空间为M_1,则$S_1 = M_1 \times m \times n$,散列表占用空间大小为$S_2$，每个散列表节点的大小为$M_2$,则$S_2 = M_2 \times H$,新分配节点占用的空间为$S_3$,每个新节点占用的空间大小为$M_3$，则$S_3 = 0 \times M_3$.

$$
\begin{aligned}
S &= S_1 + S_2 + S_3 && (11)\\
&= n \times m \times M_1 + H \times M_2 + 0 \times M_3 && (12)\\
&= O(nm + H) && (13)
\end{aligned}
$$

最坏空间复杂度 由于保存序列的空间大小和散列表的大小是固定不变的,最坏的情况是除了第一个序列保存在原散列表下，其余的节点全部保存在新分配的节点中。S_1和S_2不变，$S_3 = (B_k - 1) \times M_3$

$$
\begin{aligned}
S &= S_1 + S_2 + S_3 && (14)\\
&= n \times m \times M_1 + H\ times M_2 + (n \times (m - k + 1) - 1) \times M_3 && (15)\\
&= O(nm - nk + n + H) && (16)
\end{aligned}
$$

3.4 检索的复杂度分析

检索的空间复杂度为$O(1)$.在理想没有冲突的情况下时间复杂度为$O(1)$,在最坏情况下所有的序列都保存在某个哈希节点的下面以链表的形式存储，设检索的时间为Q，遍历一个节点的时间为Q_1，则平均需要遍历一半节点数.

$$
\begin{aligned}
Q &= B_k \times Q_1 && (17)\\
&= n \times (m - k + 1) * Q_1 && (18)\\
&= O(n(m - k + 1)) && (19)
\end{aligned}
$$

3.5 最大支持的k值

内存占用最多的情况为申请新节点最多,当k=1,且DNA 序列全部相同是,占用的节点数目最多,此时新申请的空间大小为762MB(X86程序),1143MB(X64程序).散列表节点申请的空间大小为 H*8B=406MB,保存序列信息的数组占用的空间大小为95MB,一共占用的内存空间大约为1644MB(X64),1263MB(X86),加上运行时分配的其他空间,该算法可以对所有的k=1,2,...,100建立索引.

4 模型的检验

根据上面的分析,我们采用MurmurHash散列函数产生散列值,散列表的大小H=53312517,冲突解决策咯采用线性探测,建立数据索引,对k=1,2,...,100建立数据索引,测得时间和内存占用下表所示：

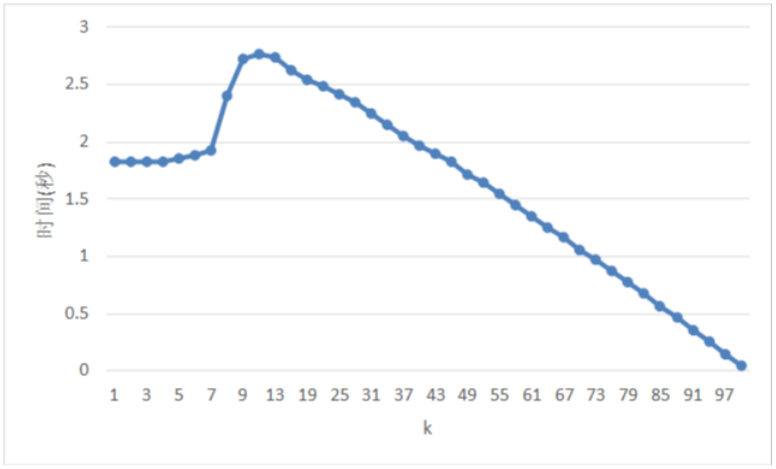

图 14: 建立索引的时间与k的关系曲线

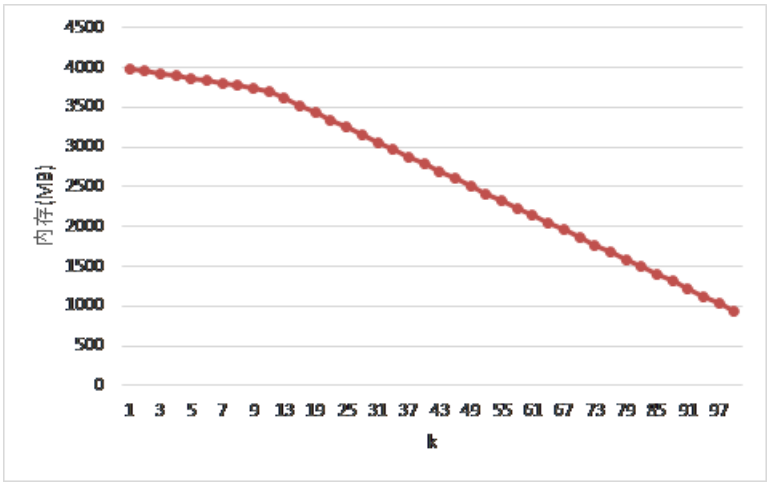

图 15: 内存占用与k的关系曲线

对于任意的k 值从样本数据中随机抽取100 个，平均的查找并输出的时间为1us左右，最长查找及输出时间为2us 左右。

5 模型的优缺点

该模型利用了散列表检索可以在O(1)时间内完成的性质,建表完成后检索速度非常快,并根据序列的信息逐步确定散列函数和散列表的参数并进行测试,减少了建立散列表的时间,利用多线程的方式同步建立数据索引,提高了建表速度. 该模型是针对给定的样本数据进行设计的,可移植性差.在分析时,是按照序列分布完全均匀来做的,尽管样本数据基本符合均匀分布,对其他分布不均匀的数据没有考虑.程序中假设不同序列的Hash_ID不会发生冲突,实际

中可能会出现相同的情况,会导致检索结果出错.程序中对数据文件没有检验,认为所有的数据都是符合要求的.

6 进一步讨论

对于不均匀序列 可以对出现次数较多的序列先求出其在散列表中的位置,预先为其分配一定的空间,利用数组保存地址信息,一方面可以节省程序运行时的时间,另一方面可以减少指针占用的内存空间.

更大数据的检索 本模型中样本数据可以直接在内存中处理,如果数据更大无法一次性在内存中运行,要考虑将计算的结果写入硬盘进行保存,而磁盘的读取性能远不及内存,有很多可以优化的地方.

不同的k值如何一次性的建立索引 本模型建立的索引只能对特定的k进行查询，如果要求一次建立索引可以用于所有k值得查询，可以先对较小的k值建立索引，然后将较长的序列分割为k长的序列进行查询。比如建立k=4的索引，要查找k=8的序列“ATCGGCTT”，可以先查找”ATCG”，根据找到的位置向后移动4个位置，比较是否为”GCTT”,对于最后长度不是4的序列也可以进行比较。另一种方法是分别检索”ATCG”和“GCTT”，然后对查找到的位置排序，如果可以连接在一起说明找到了相关序列。对不满足k长的序列，可以在后面补上所有可能的序列组合进行查询。实际测试中这种方法对于题目给定的要求检索速度不如直接对特定k长建立索引快，故没有采用。但在一次建立多种查询的情境可能更有效。

参考文献

[1] S. Kullback and R. A. Leibler, “On information and sufficiency,” *The annals of mathematical statistics*, pp. 79–86, 1951.

[2] B.-L. Hao, “Fractals from genomes–exact solutions of a biology-inspired problem,” *Physica A: Statistical Mechanics and its Applications*, vol. 282, no. 1, pp. 225–246, 2000.

[3] I. Boyd, “Which hashing algorithm is best for uniqueness and speed?” http://programmers.stackexchange.com/questions/49550/which-hashing-algorithm-is-best-for-uniqueness-and-speed, 2015.

[4] A. Appleby, “Murmurhash on googlepages,” http://murmurhash.googlepages.com/, 2012.