



基于BWT的基因序列比对算法

|  |  |
| --- | --- |
| **专 业** | **数据科学与大数据专业** |
| **姓 名** | **施钰泓** |
| **日 期** | **2022/9/8** |

**摘 要**

此次研究中我们复现了基于BWT的基因比对算法。BWT是一种可以应用于大规模基因序列的存储数据结构，该数据结构利用BWM构造中的特殊性质，建立FM索引，实现了更快的基因比对查找。该查找优于在该算法之前的maq与soap算法。在索引的建立方面，原方法未详细说明，我们设计了一种空间复杂度较低的方法来建立索引结构，使得大规模基因序列的FM索引的建立能够在更小的空间下解决。

关键词：基因序列比对，BWT，FM索引，精确查找

**目 录**

[第1章 研究方向应用简介 - 1 -](#_Toc31756)

[第2章 研究内容介绍 - 2 -](#_Toc16477)

[2.1 构造BWM，获取最后一列数据 - 2 -](#_Toc16401)

[2.2 基于LF-Mapping，构造相关索引信息 - 3 -](#_Toc18093)

[2.3 查找子串在原序列中出现的所有索引位置 - 4 -](#_Toc7834)

[第3章 研究原理、方法介绍 - 5 -](#_Toc7363)

[3.1 获取BWM最后一列数据，建立索引 - 5 -](#_Toc29934)

[3.2 编写EXACTMATCH，查找子串在参考序列中出现的所有索引位置 - 9 -](#_Toc1838)

[第4章 研究过程展示 - 10 -](#_Toc2073)

[4.1 暴力求解 - 10 -](#_Toc1668)

[4.2 桶排序，先分为四类，后在类中进行排序 - 10 -](#_Toc20274)

[4.3 构建四叉树，每次均对当前上一个字符相同的串排序 - 11 -](#_Toc4876)

[第5章 研究结果展示 - 12 -](#_Toc28865)

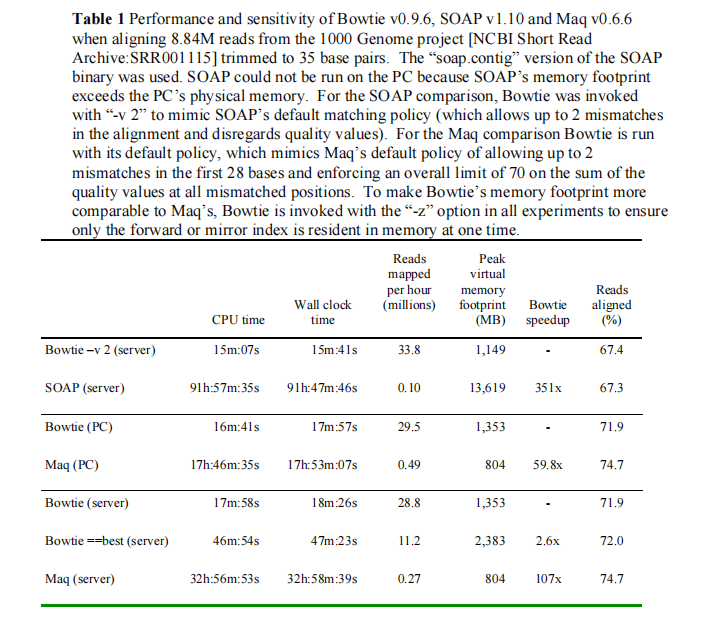
[结 论 - 13 -](#_Toc4871)

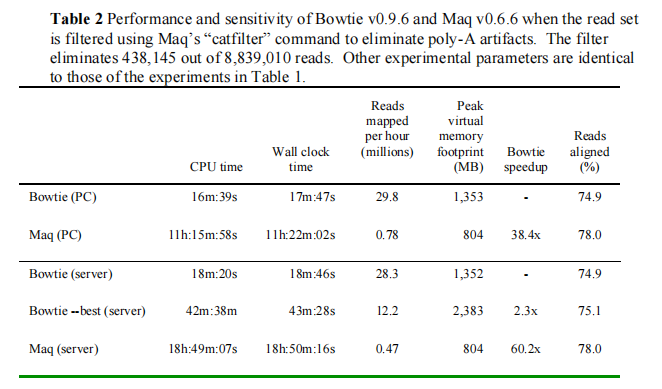
[参考文献 - 14 -](#_Toc5240)

**第1章 研究方向应用简介**

基因序列比对在生物研究方面具有重要的意义。通过基因序列比对，我们可以检测出基因突变位点并研究该突变的影响，可以预测出未来可能会患的疾病，甚至可以探究人类的发展变迁，可以说该方面的研究是解锁人类生命奥秘的重要方式之一,极大地推动了生命科学和生物医学等领域的发展。而基因比对算法发展日新月异，虽然目前基因的匹配算法已经有了很多更加迅速便捷的算法，但是该研究领域的过去的重要算法仍然具有参考与研究价值。因此我们在老师的指导下选择该课题，进行算法的实现。

该课题的目的是为了加快基因比对中的时间与空间成本，使得基因的比对能够更快更方便的进行。在BWT之前，基因测序比对所用的方法大都是基于哈希表的方法。以常见的Maq和SOAP方法为例，使用Maq和SOAP所使用的计算方法将许多短读取与哺乳动物基因组对齐的计算成本非常大。例如，从表1和表2所示的结果推断，Maq将需要超过5个CPU月，SOAP需要超过3个CPU年。尽管通过在多个CPU上并行工作，使用Maq或SOAP实现此目的是可行的，但显然需要新的方法来跟上这些仪器不断增长的吞吐量，同时更经济地使用计算资源。故而本文所述Bowtie方法应运而生。





**第2章 研究内容介绍**

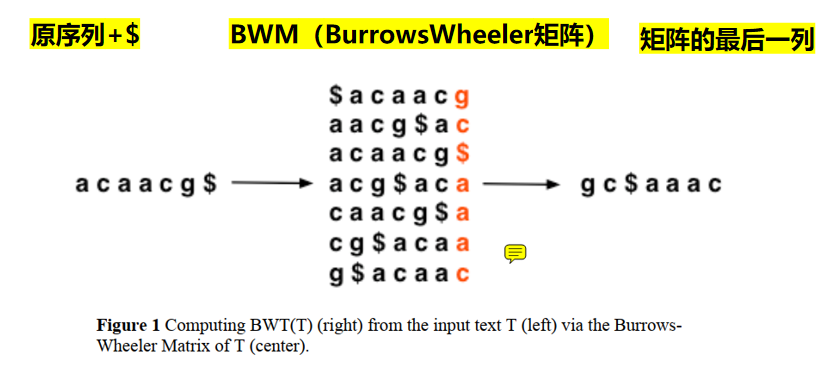
Bowite是一款超快速、内存占用低的短序列比对软件，适用于将短读基因比对至大型参考基因组。采用Burrows-Wheeler Transform算法建立索引的Bowite软件可以在1 CPU时内，将2000万条短读基因比对至人参考基因组，且内存只占有1.3Gb，相较于此前的研究，性能有了极大的提升。

本次研究主要学习Bowite相关的Burrows-Wheeler变换以及核心的基因比对算法，复现代码，并基于4.78 MB的大肠杆菌参考基因序列随机生成短读基因进行运行测试。

Burrows-Wheeler变换（BWT）是其字符的可逆排列。最初是在数据压缩的背景下开发的，基于BWT的索引允许在较小的内存占用中高效地搜索大型文本。它已应用于生物信息学应用，包括寡聚体计数、全基因组比对、平铺微阵列探针设计等等。

核心基因比对算法主要分成以下三步：

## 2.1 构造BWM，获取最后一列数据

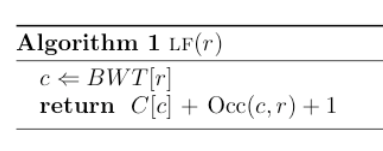


假定用于测试的参考基因序列为acaacg，我们需要利用Burrows-Wheeler Transform构建查询匹配所需要的索引。首先我们在参考序列的末尾插入$符号，作为后续匹配时的终止标识符，然后通过右移循环变换，统共获得7个字符串，对这些字符串进行字典序排序则可获得Burrows-Wheeler矩阵。取出矩阵最后一列数据，后续我们将基于这一列数据，构建匹配时所需的索引。

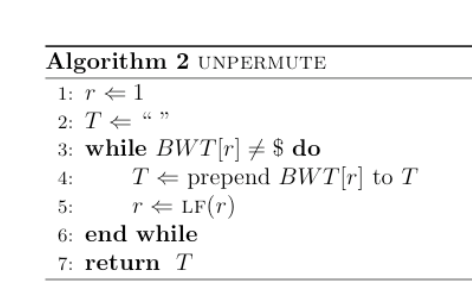
对于BWM而言，有一个重要的性质Last First(LF)Mapping：第一列和最后一列相同字符之间相对位置相同，这个性质是由BWM的构建方法决定的，可以用数学方法加以证明，此处不再赘述。此外还有一个小的性质：最后一列的第一个字符一定是原序列最后一个字符。因为加入的$符号的ASCII码小于A/C/G/T，故BWM第一行一定是“$+参考序列”，不难推知最后一列的第一个字符一定是原序列最后一个字符。整个匹配算法的正确性极大程度上依赖上述两个性质。

当然显而易见，完全建构出BWM的时空复杂度是非常高的。假定参考序列大小为1MB，BWM构建过程中至少需要占用1TB的空间，然而我们知道，基因序列的大小往往会以PB衡量，故而暴力方法在实际应用必不可行。因为其余相关的匹配算法在文献中都有详细介绍，复现难度并不高，我们的工作集中于如何不实际建构出BWM就能获得最后一列的数据。

## 2.2 基于LF-Mapping，构造相关索引信息



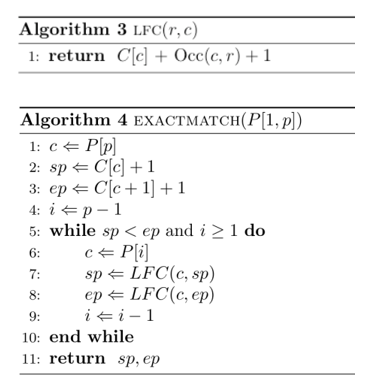
该算法源于参考文献，实际上基于LF-Mapping性质。C[c]是指“所有ASCII码小于c的字符总个数（不含$）”，Occ(c,r)则是指“对于字符c而言，其在相同字符中的相对位置”，因为第一列的数据完全是基于字典序排列的结果，所以C[c]+Occ(c,r)+1就可以求得BWM的最后一列中的字符c在第一列中的相应位置。然后我们可以试图根据最后一列数据来还原参考序列以证明索引建构的正确性。





还原的过程大致如上图所示，已知“最后一列的第一个字符一定是原序列最后一个字符”，故取最后一列的第一个字符，调用LF算法依次求出对应字符在第一列的位置，取出相应行最后一个字符，重复上述步骤直到取出的最后一列的字符为$为止，到此整个原参考序列就已经被还原出来了。

## 2.3 查找子串在原序列中出现的所有索引位置



该算法同样源于参考文献，但是其中Algorithm 4略去了对sp和ep的有效筛选。LFC算法类似于LF算法，其思想核心是一样的。EXACTMATCH实现了对于短读基因在参考序列上的位置的精确查找，其中sp 标定了有效下界，ep标定了有效下界，整个算法利用BWM的性质，利用“两边夹逼”的思想，最终求出短读基因在参考序列上出现的所有位置。整个过程大致如下图所示。



假设随机生成的短读序列为acc，调用EXACTMATCH时，首先计算出初始的sp和ep，sp和ep实际上标定了第一列中字符c出现的范围，然后检查这个范围内的所有字符串的最后一个字符是否为a，如果不是，收缩上下限。接着调用LFC计算出新的sp和ep，重复上述步骤直至整个串匹配结束或者发现sp＞ep即匹配失败。这里其实涉及到一个问题，就是通过“收缩上下限确定有效范围”这一操作的正确性。这其实是由BWM的性质决定的。已知BWM是由参考序列右移循环变化后加以字典序排列得到的，每行最后一个字符实际是原序列中的上一个字符。以上述情况为例，所有首字符为c，尾字符为a的字符串在上一步右移变换前都是以ac为前缀的字符串，按照字典序，他们应该都会集中排列在一起，变换后也不例外，所以这样的操作是合理的。

**第3章 研究原理、方法介绍**

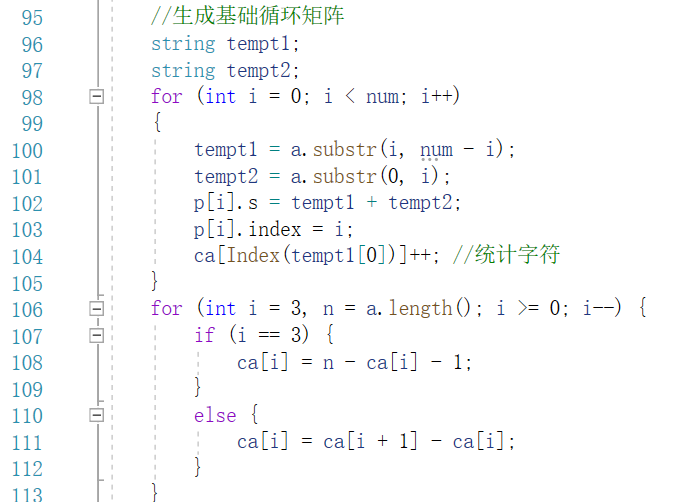
我们小组的研究方案是：首先阅读老师推荐的文献，对BWT相关的算法有相对深入的了解，然后根据论文复现代码，最后再逐步改进。

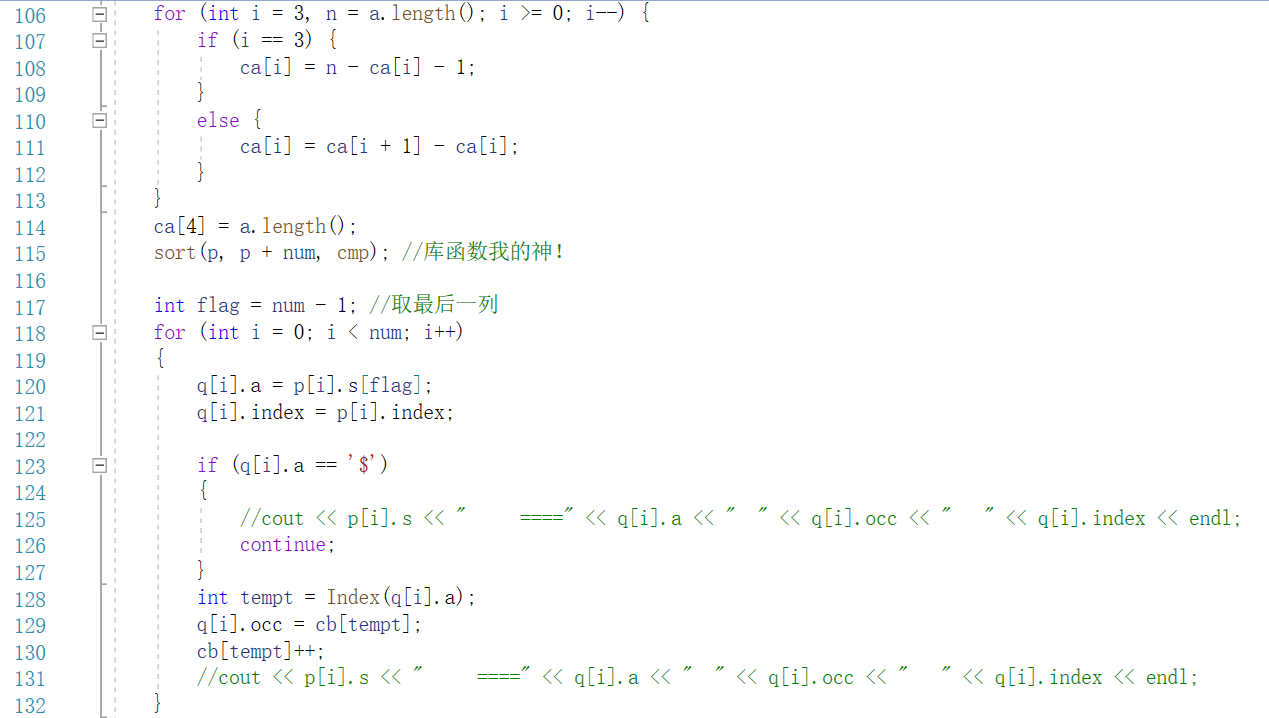
在建立索引的过程中，我们选择最简单的方式，即事先计算出所有的C[c]和Occ(c,r)数据并完全存储下来，并没有采取论文中推荐的更高级的间隔一定距离存储一个字符的C[c]和Occ(c,r)，这样更方便匹配算法的实现，只是需要以增加空间复杂度为代价。

## 3.1 获取BWM最后一列数据，建立索引

3.1.1 暴力求解

首先通过循环罗列出所有右移得到的字符串，调用快排的库函数，排列产生BWM，在提取最后一行数据的过程中，统计C[c]和Occ(c,r)数据，即可建立索引。时间复杂度为。



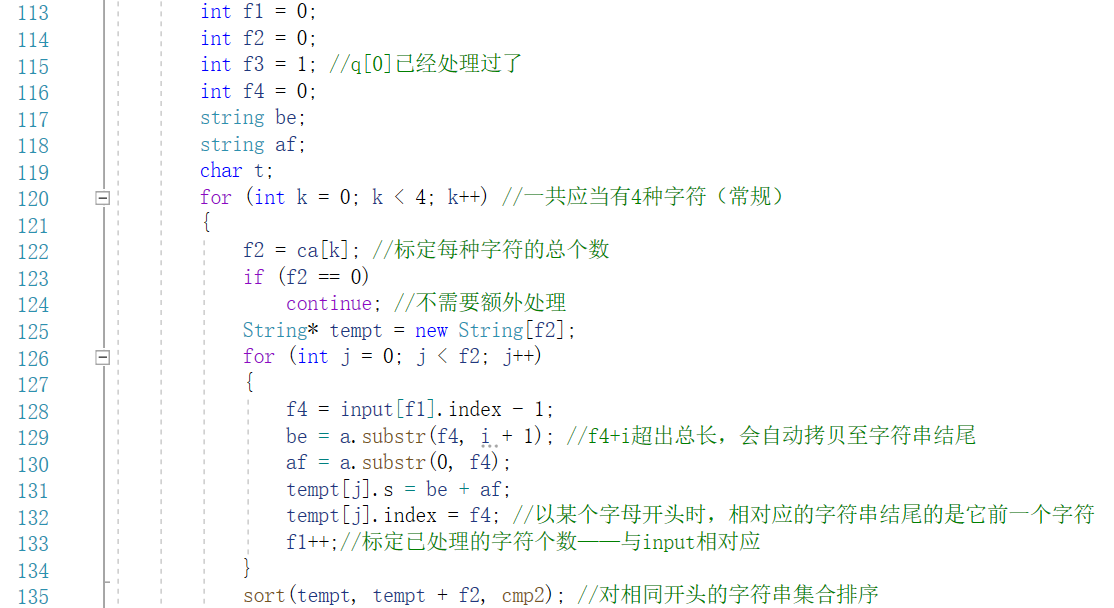


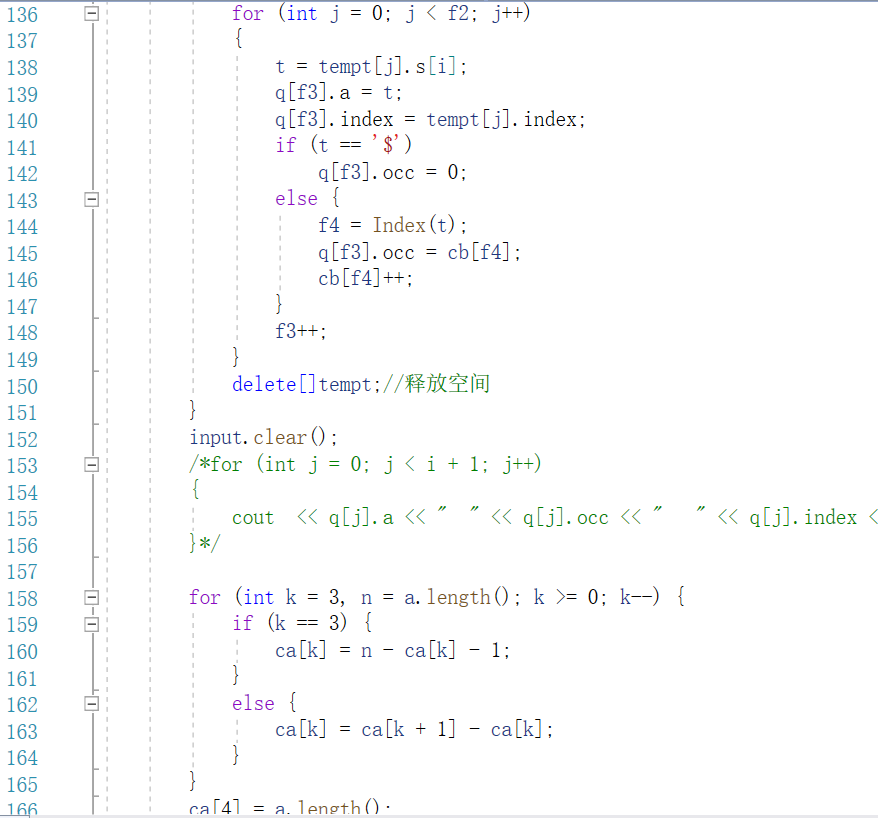
但是实验证明这样的时空复杂度都很糟糕，很难在个人电脑上运行较大规模的数据，需要进一步改进。

3.1.2 桶排序，先分为四类，后在类中进行排序

因为实际的应用场景中，涉及的碱基仅仅有A/C/G/T四种，所以我们可以先扫描一遍读入的参考序列，将衍生的字符串分为以A/C/G/T为首字母的四类，在类内再调用排序的库函数得到正确的顺序，提取最后一列数据同时统计C[c]和Occ(c,r)，这样相较于原来可以将空间消耗降低到原来的1/4左右，但是在实验中，还是不能完整读入给定的大肠杆菌参考序列构建BWM，也需要更进一步的改进。







3.1.3 构建四叉树，每次均对当前上一个字符相同的串排序

对于一个定长的基因序列，每一个位点上的碱基都有四种可能A/C/G/T，如果将所有可能穷举出来，就会产生一个四叉树。当然实际涉及排序的序列串都是由同一个参考序列经过右移循环变换得来的，套入四叉树模型将得到一颗非完全的四叉树（许多枝都是空的），对其进行排序时，许多情况都会被自动剪枝。

所以该方法实际是对上一个方法中的桶排序的更彻底的实现。在排序中扫描的过程类似广度优先搜索，故而引入队列作为容器进行处理。

改进的主要思路为，生成索引只利用原串生成，而不去构建矩阵。因为从原串中我们实际上可以得到矩阵中的所有字符串。首先我们需要考察轮换字符串与原串的关系，我们需要对轮换串编号。我们这样定义，第*i*个轮换串，其首字符为原串中第*i*个字符。那么第*i*个轮换串的第*j*个字符为原串中的第个字符。

其次我们重新设计我们的排序方法。对于基因序列来说，基因序列的碱基只有4种，因此使用桶排序是非常适合的。在使用桶排序的过程中会破坏原本的索引结构，因此我们需要在一个新的列表进行桶排序。这个列表称为排序列表。桶排序原本是对字符进行排序，如果对字符串进行排序则需要进行多次桶排序。对前缀相同的字符串进行多次的桶排序扩大前缀减少前缀相同的字符串数量，直到每个桶中只剩少于等于一个字符串或者遍历完整个字符串为止。这里我们通过队列实现上述过程，通过记录桶中的数量，若里面的字符串仍然需要排序，就按照字典序压入队列通过压入空元素来标记处理层数。这里我们会发现，桶排序的过程就是一个对字符串进行四叉树遍历的过程。得出时间复杂度为.

通过利用原串计算生成的循环串与桶排序，我们便基本实现了在不构建矩阵，仅利用原串生成Last数组的过程。



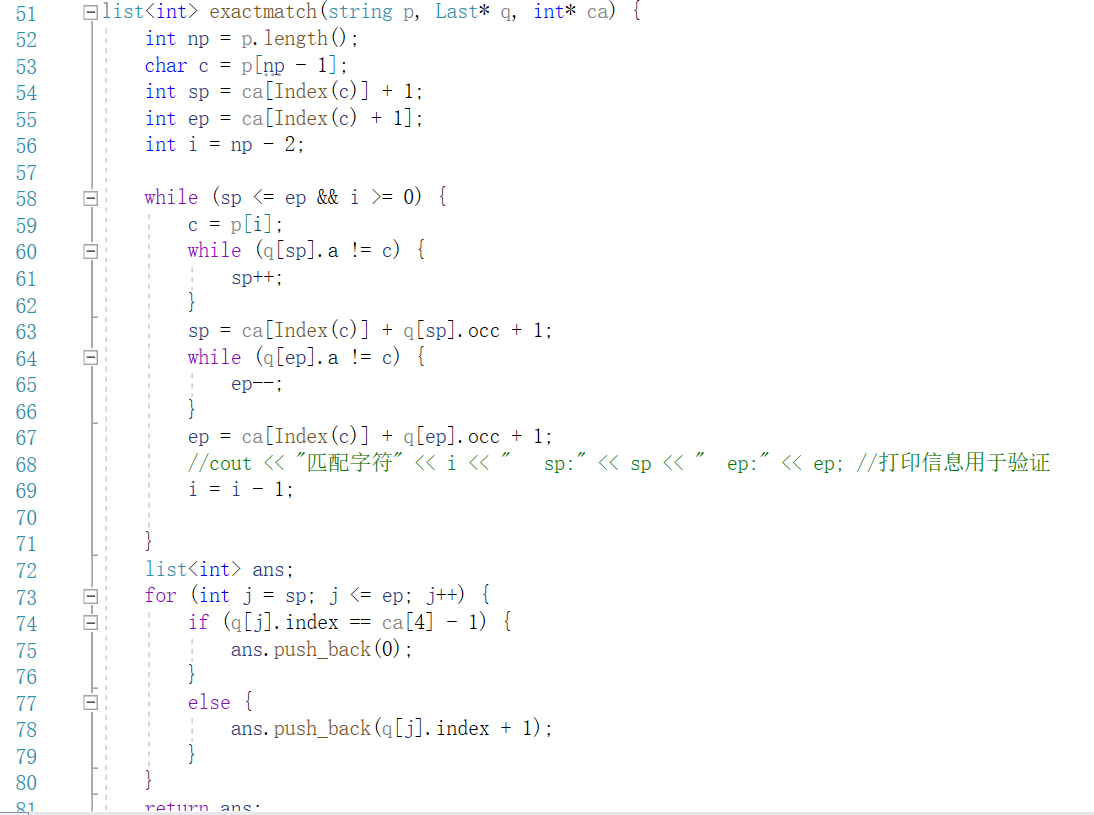




改进后整个程序的占用的运行内存大大减少，因为在这个过程中，并没有实际计算出右移循环产生的字符串，仅仅是利用右移循环的性质不断排序。同时利用队列规避掉递归，提升了程序性能。

## 3.2 编写EXACTMATCH，查找子串在参考序列中出现的所有索引位置

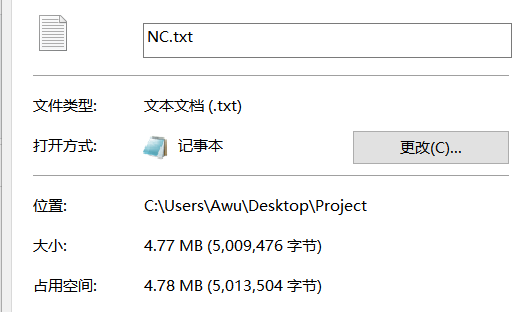
这一部分就是将论文给出的EXACTMATCH伪代码转化成实际可运行的代码，同时在编写过程中，我们发现了论文给出的伪代码省略了对于sp和ep的有效性筛选，这里采用两边夹逼的方法，即得到子串区间上下界后比对上下界的前一个字符。如果与比对串不相同，区间的上界下移，下界上移，直至上下界的Last数组字符与比对串相同为止。之后通过Last数组计算下一次比对First数组的区间，这样我们就得到了比对串的后缀增加一个字符之后在原串中的位置，同时得到了First数组的下一个区间。



**第4章 研究过程展示**

因为在代码编写过程中，我们发现在“获取BWM最后一列数据，建立索引”这一部分，如若真的求解出所有右移循环产生的而衍生串，程序运行内存将大的可怕。所以我们进行了两版优化，基本解决上述问题。

用于测试的参考序列是约5MB的大肠杆菌参考基因序列



## 4.1 暴力求解

暴力求解中需要求出整个BWM，但是5MB的参考序列对应的BWM至少需要占用25TB的内存空间，并不能在实验电脑完整跑出。



经过多次测试，该方法在实验电脑上最多可以处理100kB左右的参考序列。

## 4.2 桶排序，先分为四类，后在类中进行排序

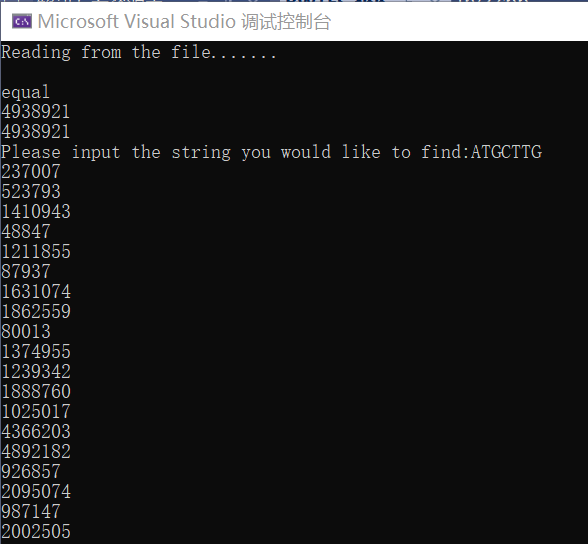
该方法虽然已经做出了优化，理论上占用的最大运行内存减少到上一个方法的1/4，但依然不能对完整的参考序列进行处理，经过多次测试，该方法在实验电脑上最多可以处理400kB左右的参考序列，且运行时长让人难以接受。



## 4.3 构建四叉树，每次均对当前上一个字符相同的串排序

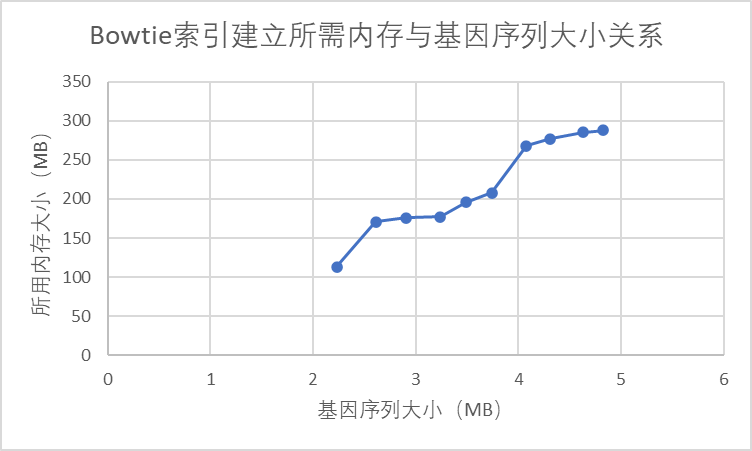
该方法在实际运行中并没有求解出右移循环产生的串，所以占用的空间非常小。大约在45秒时已读入完整的参考序列并生成对应索引，输入带查询串后，几乎在一秒内就能给出查询结果。





**第5章 研究结果展示**

经过测试，运行所用的内存与基因序列的大小呈现如下图所示，经分析还可以在内存回收方面考虑减小内存。在未改进前，程序只能除理100KB左右大小的基因序列，改进后可以完全处理5MB的数据，且运行时间缩短效果明显，由此可见程序的性能得到了极大的提升。



**结 论**

在高通量基因测序中，基因比对的效率至关重要，Bowtie算法通过建立索引的方式，将比对的时间复杂度变为，使得比对所需的时间大大降低。在算法改进过程中，通过充分的利用已有的信息，建立新信息与已有信息的关系，又大大降低了运行时所需的空间。可见建立一个好的索引可以帮助我们实现更快的查找。而充分利用信息与信息之间的关系则可以帮助我们节省空间。

**参考文献**

1. Langmead B T. Highly scalable short read alignment with the Burrows-Wheeler transform and cloud computing[M]. University of Maryland, College Park, 2009.
2. <https://www.jianshu.com/p/49ccbaf7773b>
3. <https://blog.csdn.net/stormlovetao/article/details/7048481>
4. <https://zhuanlan.zhihu.com/p/88263062?utm_id=0>