分类 _____ 密级_____

西北师范大学

硕士学位论文

基于后缀数组的字符串模式查找的算法

张利香

导师姓名职称：赵学锋 副教授

专业名称：计算机软件与理论 研究方向：算法设计与分析

论文答辩日期：2010 年 5 月 学位授予日期：2010 年 6 月

答辩委员会主席：

评 阅 人：

二〇一〇年五月

硕士学位论文
M. D. Thesis

基于后缀数组的字符串模式查找的算法

The String Pattern Searching Algorithms Based on Suffix Arrays

张利香

Zhang LiXiang

独创性声明

本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包括其他人已经发表或撰写过的研究成果，也不包含为获得西北师范大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

签名：张利香　　日期：2010.5.29

关于论文使用授权的说明

本人完全了解西北师范大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存论文。

（保密的论文在解密后应遵守此规定）

签名：张利香　导师签名：[illegible]　日期：2010.5.29

摘 要

模式匹配问题是计算机科学的一个基本问题。在早期的模式匹配研究中，多数算法集中于精确模式匹配的研究，如：著名的单模式匹配算法KMP、BM及多模式匹配算法CA、CW、BNDM等。但在实际应用中，近似模式匹配算法在信息过滤及计算生物学等领域显得非常重要。比如人们通常在DNA的片段分析中，只是寻找到与目标模式相似的DNA片段，从而判断它们之间是否有亲缘关系等等。要在海量的信息中快速地找到有意义的模式串，运用一般的方法进行模式查找，所花费的时间让人难以忍受，为了提高模式查找算法的运行效率，这里提出了模式查找的新算法。

本文是针对长篇英文小说中的高频词（模式串）查找问题，应用了近似串匹配的过滤的思想，提出了模式查找的算法Epattern_searcherH和Epattern_searcher。

本论文的主要工作如下：

(1) 提出了模式查找的算法。针对近似模式匹配中过滤思想设计模式查找的算法，首先按照一定的过滤原则（某种条件限定）搜索文本串，过滤掉那些不可能发生匹配的的文本段；然后验证剩下的匹配候选位置处是否真的存在成功匹配。由于过滤去了大部分不可能发生匹配的文本串，加速了匹配的查找过程，从而达到提高模式查找算法运行效率的目的。

(2) 应用后缀数组的数据结构来实现提出的算法。对于字符串处理问题，后缀树是一种很好的数据结构，但后缀树需要占用较大内存空间，而空间问题一直是运用后缀树数据结构来处理超长字符串的一个瓶颈。与后缀树结构相比，后缀数组可节约三分之二以上的空间，因此这里运用后缀数组的数据结构来实现提出的算法。

关键词： 模式查找；串匹配；后缀数组；英文高频词；过滤；编辑距离

Abstract

Pattern-matching problem is a fundamental problem in computer science. In the early years, most of pattern-matching algorithems focus on the exact pattern-matching algorithms study, such as: well-known single pattern matching algorithms KMP BM and multi-pattern matching algorithms CA CW BNDM so. However, in practical applications, it is very important that approximate pattern matching in information filtering and computational biology and other fields. For example, in the analysis of DAN fragments, people usually search similar patterns with the target DNA fragment, in order to determine whether there are genetic relations between them. To search a meaningful pattern string quickly in the vast amounts of information, we must improve the efficiency of pattern matching algorithms. Therefore, it is necessary to improve pattern-matching algorithms furtherly.

Here for the problem of searching the high-frequency words (pattern string) in longer English novel, we applied filtering ideas to present approximate string pattern searching algorithms Epattern_searcherH and Epattern_searcher.

The main work of this paper is as follows:

(1) Propose the algorithms of pattern searching, for the idea of the filter design pattern searching algorithms of approximate pattern matching. First, according to certain filter principles (limited to certain conditions) to search a text string, filter out those unlikely matches the text segment; and then further verify that the positions of the rest of the candidate matches whether it really exists at the success of matching. As filtered out most of unlikely match the text string, the process of matching has greatly accelerated.

(2) To achieve the pattern searching algorithms based on suffix arrays data structure.Suffix tree is a good data structure for the treatment of large-scale string, but the requiems of suffix tree index structure space is too large, thus there is a pattern searching index structure—— suffix arrays. Because the storage space of suffix arrays required is far less than the suffix tree.

Keywords: pattern searching; string matching; suffix arrays; english high-frequency words; filtration; edit distance

目　录

第一章　绪论

1.1　串匹配技术的研究意义

串匹配就是在一个符号序列中查找另一个(或一些)符号序列的搜索问题。也就是说，串匹配主要是用于解决模式搜索问题。模式查找在信息过滤及生物学领域有着广泛的应用。在网络信息的今天，通常，在基于文本信息搜索中，使用较好的搜索工具百度、Google 等可以快速的搜索到与你所查找问题相关的网页内容，所查找的内容按照查找关键词（模式）的相关性大小进行排序，使人们能够迅速地获得自己所需要的信息。比如，对某个问题进行网上调查问卷，获取他们的反馈信息，可以通过多重复近似模式查找的方法迅速得到较准确的答案。随着国际化地交流，英语和计算机是人们生活中使用的两种基本工具，人们越来越重视英语的学习，广泛阅读成为学好语言的基本方法。在长篇英文小说阅读中，为了能够较快地查找到文章中的高颇词，从而较准确的把握文章大意，提出了近似模式查找的算法。与精确模式查找算法相比之下，近似模式匹配具有更广泛的实用价值。

串匹配技术已经广泛地应用于文本编辑处理、文献检索、自然语言识别、入侵检测、病毒检测、信息过滤、计算生物学等领域。近年来，随着串匹配算法的应用领域的扩展，串匹配技术的需求日新月异，经典的串匹配算法已不能满足要求，由于串匹配是这些应用中最耗时的核心问题，好的串匹配算法能显著地提高应用效率，因此研究并设计快速的串匹配算法具有重要的理论价值和实际意义。

1.2　串匹配算法研究现状

1.2.1　串匹配算法的发展简史

串匹配技术的发展与其应用密不可分。自从 1946 年世界上第一台电子计算机问世以来，最初的串匹配技术是应用于文本编辑、全文检索系统、查询系统等许多领域。在二十一世纪，由于网络技术迅猛地发展，串匹配技术已经更广泛应用于网络入侵检测系统[1,2]、内容过滤[3]、生物科学计算[4,5]、新闻主题提取[6]等领域。近年来，在学术

界，对串匹配的研究兴趣与日俱增，特别是在信息检索领域和计算机生物学领域中的发展需求呈不断上升趋势。

在串匹配算法中，依据在文本中搜索模式串的方式不同，字符串匹配算法可归结为三种基本的方法，即：基于前缀的搜索方法、基于后缀的搜索方法和基于子串的搜索方法。由于匹配模式的数量不同，又可分为单模式匹配和多模式匹配两种。下面分别从单模式匹配和多模式匹配两方面介绍模式匹配算法的发展简史。

最早的单模式匹配算法即蛮力算法（Brute-Force 算法），也是最原始的朴素字符串匹配算法，其特点简单、直观。该算法在模式字符串与文本字符串都是随机选择的情况下，工作效率相当的不错，还有此算法并没有其它算法所必须的预处理时间，但是对文本串的扫描常常需要回溯，因而效率较低。在 1970 年，串匹配问题已经可以在$O(m+n)$（m 和 n 分别为模式串和文本的长度）时间内解决，这种串匹配问题是由 S.A.Cook 在理论上进行了证明。D.E.Knuth 和 V.R.Pratt 随后又仿照 Cook 的证明构造了一个算法。与此同时，J.H.Morris 在研究文本处理时也独立地得到与 D.E.Knuth 和 V.R.Pratt 两人在本质上相同的算法。当前最为经典的串匹配算法—Knuth-Morris-Pratt 算法[7]（简称 KMP 算法）是由这三个人殊途同归地用这样两个算法构造出来的。在 1977 年，一个新的串匹配算法 Boyer-Moore 算法[8]（简称 BM 算法）被 R.S.Boyer 和 J.S.Moore 两人共同设计出，成为目前最常用、效率较高的算法之一。随后，又有 Tuned Boyer-Moore 算法[9]、Turbo BM 算法[10]、Boyer-Moore-Horspool 算法[11]、Boyer-Moore-Horspool-Raita 算法[12]以及其它基于 BM 算法的改进算法[13,14,15]等。在 1980 年，从截然不同于 KMP 算法和 BM 算法的途径出发，一种新算法——Karp-Rabin 算法[16]（简称 KR 算法）被 Karp 和 Rabin 合作研究出。在 1990 年，Sunday D.M.提出了 BM 算法的一种简化算法——Quick Search 算法[17]（简称 QS 算法），成为实际应用中的一种高效的算法。

A.V.Aho 和 M.J.Corasick 提出了多模式串匹配算法中最经典的算法 DFSA（Deterministic Finite State Automata）算法[18]。该算法是通过构造有限自动机来实现匹配，其构造有限自动机的过程是将多模式匹配问题转化为单模式匹配问题，因此，完全可以在有限自动机构造完毕之后应用一些现有的快速单模式匹配算法来加快匹配速度。在 1993 年，在结合 DFSA 算法和 BM 算法的基础上，一种新型的 FS 算法被 Jang-Jong Fan 和 Keh-Yih Su 两人设计出[19]，该算法在平均情况下比 DFSA 算法速度更快[19]。另外，由 Sun Wu 和 Udi Manber 两人在基于 BM 算法的框架下派生出了高

效的多模式匹配算法—Wu-Manber 算法[20,21]。Aho 和 Corasick 算法是 KMP 算法基础上扩展的多模式串匹配算法，在实际应用中对该算法进行改进，主要做法是对供给函数模拟的状态转移预先进行计算，获得一个完全的自动机，称为扩展的 Aho—Corasick 自动机，在小字母表和模式串集合规模不大的情况下，这种构造非常有效的。另外，一种简单有效的多模式匹配算法是 SBDM (Set Backward Dawg Matching)和 SBOM (Set Backward Oracle Matching)，它们分别是对单模式算法 BDM 和 BOM 的扩展[22]，是近几年才提出的理论与应用相结合的有效算法。由实验证明，在多模式串匹配算法中，最有效的算法是 Wu-Manber、高级 Aho-Corasick 和 SBOM。随着模式串集合的增大，SBOM 越来越有优势。尤其在查找的模式串较短时，高级 Aho-Corasick 算法则更有竞争力，因为它只对文本扫描一遍。

1.2.2 国内外研究现状

目前国内外研究者提出的串匹配算法有很多种。其算法的思想和使用技巧非常丰富。根据串匹配的不同应用领域，按其功能分为精确串匹配算法和近似串匹配算法两类。

精确字符串匹配算法是在数据序列中找出和一个或一组与特定的模式完全相同的字符串片段的所有出现位置，根据同时要匹配的模式的多少，大致分为单模和多模两种，同时只匹配一个字符串的算法称为单模式算法，同时匹配多个字符串的算法称为多模式算法。

近似字符串匹配可以描述为：给定长度为 n 的目标文本 T、长度为 m 的模式 P，以及允许最大误差 k（k<m）。近似串匹配算法是允许被匹配的字符串和模式之间存在一些误差，也叫可容错模式匹配算法，是字符串匹配问题的重要扩展。同时两个字符串之间的误差可用距离标准来度量，常用的距离标准模型是编辑距离标准模型和汉明距离标准模型。在具体应用中，汉明距离是要求模式的长度和文本串中的待比较的片段的长度相等，而编辑距离则可在模式和文本中待比较的片段长度不等的情况下进行，相比之下编辑距离的运用范围较广些。经典的近似字符串匹配算法包括：动态规划算法、自动机算法、过滤算法以及位并行算法等[23]。

1.3 本文的目标与组织结构

本文主要讨论了一些经典、高效的串匹配算法，针对在英文小说中查找高频词的问题，设计了多重复近似模式查找算法 Epattern_searcherH 与算法 Epattern_searcher，并将这两种算法进行了比较，通过实验结果表明算法 Epattern_searcher 在英文小说中搜索高频词的速度较快。

本文共分六章，各章的主要内容如下：

第一章为绪论，介绍了串匹配算法的研究意义、发展简史以及研究现状，并陈述了本文的目标与组织结构。

第二章为串匹配算法的概述，给出串匹配的定义，主要介绍了一些实用、高效的串匹配算法及近年来的发展状况。根据在文本中搜索方式不同，将字符串匹配算法归结为三种基本的算法进行了介绍，即：基于前缀的搜索算法、基于后缀的搜索算法和基于子串的搜索算法。

第三章为后缀数组，介绍了模式查找算法中通常用到的两种索引结构后缀树和增强后缀数组。针对大规模的字符串搜索问题，后缀树是一种很好的数据结构，但后缀树索引结构的空间需求过大，因此，为了减少索引数据结构的空间占用，本文应用一种面向模式查找的索引结构——后缀数组来实现算法。

第四章为模式查找的算法介绍，针对在英文长篇小说阅读中快速查找高频词的问题，提出了 Epattern_searcherH 算法和 Epattern_searcher 算法；最后分析了提出算法的时间和空间复杂度，相比一般模式查找的方法，其运行速度较快，比较实用。

第五章为实验结果与分析，分别对算法 Epattern_searcher 和算法 Epattern_searcherH 进行了实验测试，且从实验结果表明算法 Epattern_searcher 比 Epattern_searcherH 运行的时间消耗少，即执行速度快；当容错率较高时 $(10\% \le d/p \le 14\%)$，Epattern_searcher 算法中第三种过滤方法 THIRD 运行速度最快，而 Epattern_searcherH 算法不再适用（要求容错率一般在10% 以下）。

第六章为总结与展望，总结了本文的研究内容以及存在的不足，并对以后的工作给予了展望。

第二章　串匹配算法概述

2.1　串匹配的定义

串匹配是模式匹配中最基本的一个问题，也是文本处理领域中一个非常重要的主题。

串匹配的定义：串匹配就是给定一组特定的字符串集合 P，对于任意的一个字符串 T，找出 P 中字符串在 T 中所有出现的位置。这里的 P 为模式的集合，P 中的元素为模式串(或关键词)，T 为文本。字符串中的字符都来自一个有限的符号集合∑，简称字母表(或字符集)[24]。

根据在文本中搜索方式的不同，字符串匹配算法可以归结为三种基本的算法，即：基于前缀的搜索算法、基于后缀的搜索算法和基于子串的搜索算法。基于前缀的算法，如 BF 算法、KMP 算法、Shift-Or 算法[26] 等；基于后缀的算法，如 Boyer-Moore 算法、Horspool 算法[25]、Commentz-Walter 算法[27]、Wu-Manber 算法[28] 等；基于子串的算法，如 BNDM 算法、SBOM 算法[29]等。本章选取其中应用最广泛、效率最高的几个著名算法进行简单介绍。由于大部分多模式串的算法是由单模式的算法推广而得到的，这里对单模式和多模式算法不再作区分。

2.2　基于前缀的搜索算法

基于前缀的算法一般需要从文本中逐个读入字符，每读入一个字符就要更新相应变量，检查是否存在一个可能与模式串的前缀的匹配。它们不具备跳跃能力，所以具有$O(n)$的时间复杂度。BF、KMP 算法是其中的代表。

2.2.1　BF 算法

蛮力算法，简称 BF 算法，它是最早的最简单的单模式匹配算法。算法的思想是把文本串中的每一个字符 $t[i]$ $(i \in [0, n-m])$ 作为一次匹配的开始，并依次比较 $t[i \cdots i+m-1]$ 与模式串 $p[0 \cdots m-1]$ 是否全部相同，如果全部相同，则报告一次匹配，否

则，从文本串中选择下一个字符$t[i+1]$，按照上述方法继续检测，直至 i 把所有可能的取值取尽时结束。

BF 算法的输入是：长度为 n 文本串 T；输出是：匹配信息。

其算法的代码如下：

```
for (int i = 0; i <= n - m; i++)
  {
  for (int j = 0; j < m && p[j] = t[j + i]; j++)
    if (j = m)
    记录匹配信息;    //匹配成功
  }
```

由于 BF 算法具有简单、直观，无需预处理过程的特点，所以不需要额外的存储空间，但对文本串的扫描常常需要回溯，因而效率较低。在匹配过程中，时间复杂度最好为$O(n-m+1)$，最坏为$O(n\cdot m)$。

2.2.2 KMP 算法

Knuth-Morris-Pratt 算法[7]（简称 KMP 算法）是由 D.E.Knuth、J.H.Morris 和 V.R.Pratt 三人设计出的一种高效的单模式匹配算法。KMP 算法采用从前向后的比较方式，在每一次尝试匹配过程中，当发现有不匹配现象时，不需要回朔匹配指针，而是利用在本次尝试过程中已经得到的部分匹配信息进行跳跃，之后继续进行比较。若在某次尝试匹配过程中，当前的匹配窗口为 j，即检测的文本串 T 中$t[j\cdots j+m-1]$是否和长度为 m 的模式串 p 相等；若在该次尝试过程中，$p[i]$和$t[i+j]$发生不匹配，其中$i\in[0,m-1]$，则可知模式串 P 在位置 i 之前的字符与文本中相应的字符匹配，即$p[0\cdots i-1]=j[j\cdots j+i-1]=u$；同时假设$t[i+j]=x$，$p[i]=y$，$x,y\in\Sigma$，且$x\neq y$，此时，KMP 算法依据已匹配的信息 u 进行跳跃。KMP 算法采取的跳跃方法很简单，是在模式串 P 中的前$i-1$ 个字符中找到 u 的最大后缀 v 并且紧接 v 的下一个字符不为$p[i]$，即$z\neq y$，如图 2.2.2 所示。

KMP 算法中跳跃表 kmpNext 的预处理如下：

$$\text{kmpNext[i]}=\begin{cases}-1 & i=0\\ i-\max\{k\mid k\in[2,i-1],\ p[0\cdots k-2]=p[i-k+1\cdots i-1]\} & \\ 0 & \text{其他情况}\end{cases} \qquad (2.2)$$

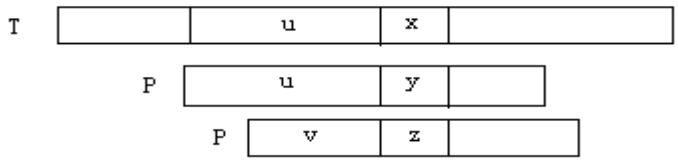


图 2.2.2 kmp 算法的跳跃方法

KMP 算法的主要程序代码如下：

输入：长度为 n 的文本串 T 及其跳转表 kmpNext

输出：匹配信息

```
计算 kmpNext;     //预处理过程
int  i = j = 0;
while ( j < n ) {
Attempts++;       //尝试的次数
      while (i > -1 && p[i]!=t[j])
            i = kmpNext[i];
            i++;   j++;
            if (i >= m) {    //匹配成功信息
            保存匹配信息;
            i = kmpNext[i];
              }
}
```

KMP 算法的空间复杂度为$O(m)$，其预处理阶段的时间复杂度为$O(m)$，匹配阶段的时间复杂度为$O(n+m)$，与字符表$\sum$的大小σ无关。

2.3 基于后缀的搜索算法

基于后缀的搜索算法都是基于滑动窗口。滑动窗口沿着文本 T 滑动，对于任意位置上的窗口，在窗口中从后向前搜索窗口中的文本和模式串 P 的公共后缀。比如：算法 BM、WM，其中 WM 算法是实际应用效果非常好的 BM 的扩展算法。

2.3.1 BM 算法

Boyer-Moore 算法，简称 BM，是由 R.S.Boyer 和 J.S.Moore 在 1977 年设计的一个串匹配算法，成为目前实际应用中最常用且效率较高的单模式匹配算法之一。 不

同与 BF、KMP 算法的是 BM 算法采用从后向前的比较方式，并利用当前尝试中的已匹配信息和匹配失败的字符，查找预处理好的良好后缀跳转表 bmGS（Good-Suffix Shift Table）和不良字符跳转表 bmBC（Bad-Character Shift Table），跳跃式的进行处理。BM 算法的最大可能跳转距离为 m。这里首先分析 BM 算法在匹配过程中如何运用 bmGS 和 bmBC 两个表进行最大可能的跳跃，其次给出其各自定义。

假定在某一次尝试过程中，$t[i+j]=x$，$p[i]=y\ (x,y\in\Sigma,\text{且}x\neq y)$，由于 BM 算法从后向前比较，可知 $p[i+1\cdots m\text{-}1]=t[i+j+1\cdots j+m\text{-}1]=u$，且 $p[i]\neq t[i+j]$。如图 2.3.1 所示。

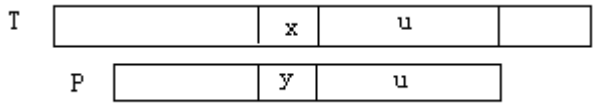


图 2.3.1 在 BM 算法中，假定的初始情况

良好后缀跳转表 bmGS 是在某次尝试过程中，根据发生不匹配位置之前的已匹配信息进行跳跃，它所考察的对象是一个后缀，其后缀可能是一个字符，也可能是多个字符。如在图 2.3.1 中，当 $p[i]\neq t[i+j]$ 时，bmGS 根据已匹配的后缀 u 来进行跳跃。不同与良好后缀跳转表，不良字符跳转表 bmBC 考察的对象是某一字符，即是在某次尝试中发生不匹配的字符。如在图 2.3.1 中，bmBC 是根据文本串 T 与模式串 P 发生不匹配的位置信息，即文本串中的 $t[i+1]$ 来进行跳跃。具体的跳转情况可分为以下几种：

(1) 若在模式 P 中存在含有子串 u 且其前导字符不为 y 的子串 zu，其中 $z\in\sum$ 且 $z\neq y$，则根据 bmGS 进行跳转。如图 2.3.2 所示。

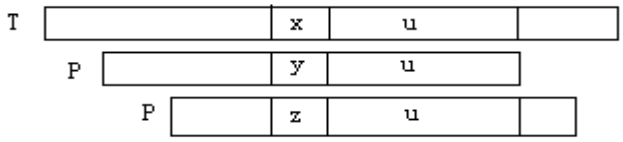


图 2.3.2 bmGS 跳跃：当 P 中存在前导字符不为 y 的子串 zu

(2) 若在模式串 P 中不存在上述那种情况，那么在 P 中找到 u 的最大后缀 v，进行 bmGS 跳转。如图 2.3.3 所示。

(3) bmBC 跳转则根据不相匹配的字符进行跳转。在模式串 P 中找到 $t[i+j]$ 字符（即字符 x）在 P 中的最右出现位置，然后进行 bmBC 跳转。如图 2.3.4 所示。

(4) 若在文本串中 $t[i+j]$ 字符（即字符 x）在 P 中不曾出现，那么 P 在 T 中可能

匹配的位置中一定不包含文本串 T 的第 $i+j$ 个字符，因此将 P 的第一个字符与 T 中的第$i+j+1$个位置对齐，进行下一次匹配。如图 2.3.5 所示。

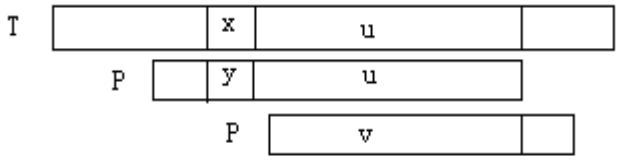


图 2.3.3 bmGS 跳跃：当 P 中只存在 u 的最大后缀 v

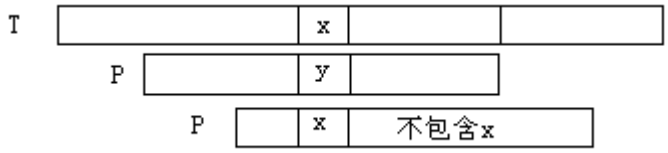


图 2.3.4 bmBC 跳跃：当 P 中找 x 最右出现的位置

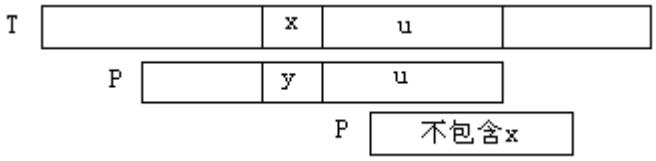


图 2.3.5 bmBC 跳跃：当 P 不包含字符 x

由此，bmGS 和 bmBC 的定义如下：

$\forall i \in [1, m-2], \exists k \in [m-2-i, m-2]$ 使得 k 满足：

$$p[k+i-m+2 \cdots k] = p[i+1 \cdots m-1], p[i] \neq p[k+i-m-1] \tag{2.3}$$

即 P 中存在含有子串 $p[i+1 \cdots m-1]$ 且前导字符不为 $p[i]$ 子串的最右位置 k。

$$bmGS[i] = \begin{cases} \min\{m-k\} & i \in [1, m-2] \text{且} k \text{满足公式2.3} \\ m-1 & i = 0，\ i \in [1, m-2] \text{且} k \text{不满足公式2.3} \\ 1 & i = m-1 \end{cases} \tag{2.4}$$

$$bmBC[c] = \begin{cases} m & c \in \Sigma \text{且} c \notin P \\ \min\{m-i-1\} & P[i] = c (i \in [0, m-2]) \end{cases} \tag{2.5}$$

在实际应用中，BM 算法是最常用最有效的单模式匹配算法之一，其特点是采用从后向前的比较方式，并且使用由良好后缀跳转表 bmGS 和不良字符跳转表 bmBC 计算得到的移动距离中的最大值进行跳跃。BM 算法的空间复杂度为$O(m+\sigma)$，其预处理阶段的时间复杂度为$O(m+\sigma)$，最好情况 BM 算法的时间复杂度达到$O(n/m)$。

2.3.2 Wu-Manber 算法

Wu-Manber 算法克服了 Set-Horspool 的缺点，它用多个字符组成的块 B 来代替单个字符，从而降低了块在模式串中出现的概率，改善了跳跃能力。Wu-Manber 的跳跃距离可以表示为：$s[B]=m-\max\left\{j\mid(j=b-1)\vee\left(B=P_{j-b+1}\cdots P_1\right)\right\}\left(B\in\Sigma^b\right)$。

用字符块代替单个字符的方法，固然改善了跳跃距离，但它需要维护一个$|\Sigma|^b$大小的表，使存储空间急剧增大。WU-Manber 算法通过散列解决了这个问题。由于采用哈希和查表的方法，WU-Manber 计算跳跃距离的代价为$O(1)$，因而是一种效率非常高的多模式串匹配算法。

2.4 基于子串的搜索算法

基于子串的搜索方法出现较晚，在实际应用中，如果模式串 P 足够长，其中一些算法是最有效的。和基于后缀的搜索方法一样，它也使用滑动窗口，并在其中从后向前搜索。不同的是，它也是模式串 P 的一个因子。最早使用这种方法的是 BDM 算法，当 P 足够短时，可以改造成更简单有效的算法 BNDM。对于较长的模式串，BOM 算法是最快的。在 BDM 和 BOM 基础上扩展的多模式匹配的有效算法是 SBOM 算法，它使用了 Factor oracle 结构。

2.4.1 SBDM 算法和 SBOM 算法

SBDM 算法使用后缀自动机在长度为$l\min$的文本窗口中从后向前识别模式串的子串以进行窗口移动。该算法使用后缀自动机建立在所有模式长度为$l\min$的前缀的反转$P_{1\min}^{rv}$上，构造时间复杂度为$O\left(r\times l\min\right)$[BBE87]。搜索在大小为$l\min$的窗口中进行，窗口沿着文本移动。在当前窗口中，从后向前识别文本的最长后缀，即 P 中模式的长度是$l\min$的前缀的子串。搜索过程可能出现下面两种情况：

(1) 无法继续识别子串，即在自动机中无法继续识别文本字符σ。这时，没有模式串的前缀能够完全覆盖窗口内已读入的文本字符。因此，可以安全将窗口移动到字符σ之后。

(2) 到达了窗口的起始位置，并且抵达了自动机的状态 q。这意味着已经识别出

$F(q)$ 中某个字符串的前缀 $L(q)$，因此需要将 $F(q)$ 中的每个模式串与当前文本进行比较验证并报告成功的匹配。然后，将窗口向右移动一个字符，开始新的搜索。

SBDM 的最坏时间复杂度是 $O(n\times|p|)$。然而，当字母表不是太小的时候，该算法的复杂度是亚线性的。在实际应用中，SBDM 的后缀自动机构建代价很大。对于很大的模式串集合，构建时间很难被搜索阶段的时间分摊抵消。而且随着模式串的规模增大，后缀自动机耗用的存储空间也急剧增大。这时 SBOM 算法使用同样的搜索方法并克服了后缀自动机这个瓶颈，它使用了一种更简单的轻量级的数据结构。在所有情形下，SBOM 都比 SBDM 要快。

SBOM 算法[AR99]使用了 Factor Oracle 结构。模式串集合 P 对应的 Factor Oracle 自动机识别的字符串集合，其集合是 P 中的模式的集合的超集。算法的搜索过程与 SBDM 类似，利用 Factor Oracle 自动机，在长度为 lmin 的文本窗口内从后向前识别字符，以进行窗口移动。如果无法继续识别文本字符 σ，那么当前窗口能够安全地移动到 σ 之后；如果完全识别了窗口内的字符，到达了窗口的起始位置，那么需要将 P 的一个子集与文本进行比较验证。

SBOM 算法的最坏时间复杂度是 $O(n\times|p|)$，平均复杂度是亚线性的。Factor Oracle 自动机的构建过程非常快，而且消耗的内存也很少，即使文本规模相对较小，也是很有效的。

2.5 本章小结

本章主要介绍了在实际应用中最广泛且效率最高的几个著名的经典串匹配算法。根据在文本中搜索方式不同，分为三种基本的算法，即：基于前缀的搜索算法、基于后缀的搜索算法和基于子串的搜索算法。为后面的章节奠定了理论基础。

第三章　后缀数组

目前，在模式查找中，后缀树和后缀数组是最常用到的索引数据结构。虽然后缀树在字符串处理中的适用性更强，设计的算法也相对容易理解，但后缀树需要占用较大的内存空间，而空间问题一直是运用后缀树数据结构来处理超长字符串的一个瓶颈。与后缀树数据结构相比，后缀数组数据结构可以节约三分之二以上的空间，但美中不足的是基于后缀数组的各种查找算法要相对复杂一些。

3.1　后缀树

后缀树在与字符串处理相关的各种领域里都有着广泛的应用。它是一种针对大规模字符串处理的有效的数据结构，在1968年，Morrison首次提出了Patricia树的概念，这就是后缀树的原型。在1973年，Weiner第一次明确提出了紧凑型bi-tree[31]。因为对于长度为n的序列，构造后缀trie树需要$O\left(n^2\right)$的时间和空间复杂度，人们对后缀trie树[32]进行了一定的改进，得到了后缀树。

后缀树定义：就是一种压缩的后缀trie树，它将后缀trie树的边进行压缩，使得树中只有一个孩子的节点与其孩子节点合并，并将指向孩子节点的边上的标识连接到上一层的边上，从而使得树中边上的标识是原序列的一个子串。给定长度n的序列S，S的后缀树是一棵有向的树，树中有n个叶子节点。每个内部节点，即非叶子节点（根节点除外）都至少有两个孩子，而且树中的每一条边上标识着S的一个非空子串。任何两条由同一节点发出的边上的标识的第一个字符不允许相同。如果将叶子节点编号，那么由根节点开始到第i个叶子节点的路径上的标识连接起来就是字符串S的第i个后缀。

如图3.2所示的字符串EXNXNXH$的后缀树。其构造过程中，需要在字符串S后加上一个$符号表示串的结束（$为字符串S的结束字符，∑中不包含$），从最长的一个后缀开始插入来构造后缀树，直到插入到只有一个符号$时结束。

人们提出了很多后缀树算法，其时间和空间复杂度是线性的，后缀树的建立时间与被索引的字符串的长度成正比，在后缀树上进行字符串匹配查找时需要的时间与待查找的字符串的长度成正比。Weiner的算法[31]、McCreight的McC算法[33]和UkKonen

的UkK算法[34]是三种经典的后缀树构造算法，它们都在$O(n)$时间内完成长度为n的字符串的后缀树的建立。虽然Weiner的算法是最早提出的线性时间算法，但该算法使用了较多的存储空间；而McC算法使用了后缀链接等技术使算法的时间复杂度减少到线性，与其它两种算法相比McC算法更高效，使用的存储空间更少；对于UkK算法也使用了后缀链接技术，其最大特点在于它是在线算法(On-1ine)，也就是说算法的任何阶段都生成当前子串的后缀树。

后缀树所需的存储空间与索引的序列的长度相关。比如，McC算法要求28倍于序列长度的存储空间[33]，Kurtz提出的算法要求20倍[35]，Manber和Myers提出的算法要求18.8-22.4倍[36]，Karkkainen提出的算法要求15到18倍[37]。虽然，这些算法的性能已经很令人满意，但在信息过滤和计算生物学等领域中，由于一条数据信息的长度可能达到几个G，对于这种超长的数据信息，如果构造后缀树结构，则所需的存储空间是其应用的一个瓶颈。

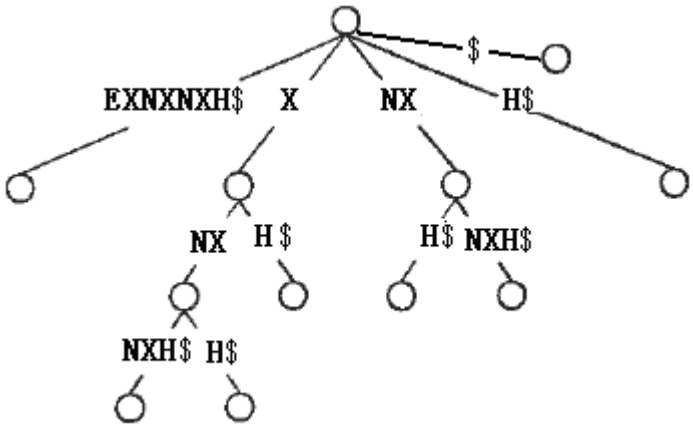


图 3.2 字符串 EXNXNXH$的后缀树

3.2 后缀数组

后缀数组的定义：后缀数组 *suftab* 是一个一维数组，它保存1…*n* 的某个排列 *suftab*[1]，*suftab*[2]，…*suftab*[*n*]，并且保证 *suffix*(*suftab*[*i*]) < *suffix*(*suftab*[*i*+1])，$1 \le i < n$。也就将S的n个后缀从小到大进行排序（按字典序排序）之后把排好序的后缀的开头位置顺次放入 *suftab* 中。

增强后缀数组定义：它是由后缀数组演变而来的，具体是由后缀数组和后缀数组中每两个相邻的后缀的最长公共前缀列表lcptab结合而成的。

后缀数组[30]是由Manber和Myers提出的，它是一种比后缀树更具有空间效率的数据结构。从[38][39]等文献中提出的增强后缀数组是比后缀数组具有更加强大的功能，它能够完成重复模式的查找及最长公共子序列查找，还有字符串匹配等一系列问题。

一个字符串S的后缀数组suftab是一组0到n的整数，其中n是字符串长度，将S$的 $n+1$ 个后缀按照字典序排序（$为字符串S的结束字符，S中不包含$）。 *suftab*[*i*] 表示S中从第 *suftab*[*i*] 位开始的后缀。后缀数组的大小为4n个比特。后缀数组中存放的就是排序好的字符序列S的所有后缀串的起始位置。

给定一个长度为 n 的字符串 S 和 S 的后缀数组 *suftab*，lcptab 是一组 0 到 n 的整数。定义 $lcptab[0]=0$，*lcptab*[*i*] 是 *suftab*[*i*-1] 和 *suftab*[*i*] 的最长共同前缀的长度，其中 i 的范围从 1 到 n。由于 S 的结束字符是并未在序列其它位置出现过的$，$lcptab[n]=0$。一个字符串 S 的增强后缀数组是由 S 的后缀数组和 lcptab 构成的。如图 3.3 所示是S=”HJHHHJHRHR”的增强后缀数组。对于长度为 n 的字符串 S，存储 S 的后缀数组和 lcptab 分别需要 4n 个比特，所以 S 的增强后缀数组需要 8n 个比特的存储空间。建立字符串 S 的后缀数组需要 $O(n\cdot\log n)$ 的时间复杂度，lcptab 可以在后缀数组的建立过程中得到[39]，或者可以在 $O(n\cdot\log n)$ 的时间复杂度内由 S 的后缀数组得到[40]。因此，一个序列的增强后缀数组可以在 $O(n\cdot\log n)$ 时间内建立完成。

i	Suftab	Lcptab	后缀字符串示意
0	2	0	HHHJHRHR$
1	3	2	HHJHRHR$
2	0	1	HJHHHJHRHR$
3	4	3	HJHRHR$
4	6	1	HRHR$
5	8	2	HR$
6	1	0	JHHHJHRHR$
7	5	2	JHRHR$
8	7	0	RHR$
9	9	1	R$
10	10	0	$

图 3.3 字符串 HJHHHJHRHR 的增强后缀数组

3.3 本章小节

本章介绍后缀树与后缀数组的索引数据结构。通常，对于长为n的序列，索引该序列的后缀树所需的空间至少为12.5n比特，而后缀数组所需的空间为4n比特。但对

于超长数据序列处理来说，应用后缀数组索引数据结构将会节约了极大的空间，因此第四章的模式查找算法是采用后缀数组数据结构来实现。

第四章　模式查找的算法

在英文小说中进行高频词的查找，其实质就是在文本中进行多重复近似模式的查找。这里提出多重复近似模式查找算法 Epattern_searcherH 和 Epattern_searcher，它们都是经过两步预处理，第一步是针对英文小说中的标点符号和空格的预处理，以便进行模式查找；第二步是过滤预处理，在算法中运用汉明距离标准或编辑距离标准度量多重复近似模式之间的误差，同时进行过滤预处理来提高模式查找算法的运行效率。

对于一篇英文文章，如果不进行第一步预处理，直接对其进行近似模式查找，对于相似度很高(除相似度为百分之百)的单词，它们的意思相差甚远，对于这样的查找结果则毫无意义。所以，这里的算法都要对英文文本进行第一步预处理，将组成文本的每个单词作为一个单位来处理，且在文本中没有其它标点符号，只有区分每个单位的空格出现。

第一步预处理的操作过程如下：

(1) 把英文文本的每个标点符号用一个空格来代替，至于每段结束时和分段开头位置处多于一个的空格，将删除多余的空格，只保留一个空格，作为区分单词的标识。

(2) 对英文文本的搜索变成对于一个较长字符串的搜索，以遇到的空格为标识把英文的每个单词区分开，这里将每个单词作为一个处理单位。因为每个单词长度不定，同时每个单词之间存在差异，则称这样的单位（单词）为一个 X 因子。这样，原英文文本可看成由 n 个 X 因子组成的字符串，从前向后搜索字符串时，每个 X 因子之间以空格作为区分单词的标志。在英文词库中大概有二十万左右的词汇，也就是说这样的 X 因子大概有二十多万左右，它们都来自于英文词库的数据集σ。

(3) 对于各个 X 因子之间的匹配，是用一般的字符串的精确匹配方法来处理，即相匹配的 X 因子是完全相同的。具体匹配过程描述为：首先判断相互比较的 X 因子的长度是否相等，若长度不等，则这两个 X 因子不同；若长度相等的情况下，再比较每个 X 因子的首字符是否匹配，若不匹配，则这两个 X 因子不同；若两个 X 因子的首字符相同，继续比较 X 因子的第二个字符是否匹配，以此类推，比较完 X 因子后面的其它字符，以此来判断两个待比较的 X 因子是否相同。

这两个算法中都进行了过滤预处理，其过滤的思想是这样描述的。首先，按照一定的过滤原则（某种条件限定）搜索文本串，过滤掉那些不可能发生匹配的的文本段；

然后，验证剩下的匹配候选位置处是否真的存在成功匹配。由于过滤去了大部分不可能发生匹配的文本串，大大的加速了匹配的查找过程，因此提高了模式查找算法运行效率。

这两个算法都是运用后缀数组的索引数据结构来实现的。Epattern_searcherH 算法是将后缀数组与每两个相邻后缀的最长公共前缀 LCP 列表结合起来，处理英文小说中高频词的查找问题。对于字符串 S 的后缀是按照每个 X 因子（单词）字典序来排序形成的子串。一般来说，一长篇英文小说的可长达几个 G，只有采用后缀数组数据结构来实现多重复近似模式的查找，这样才有可能使实现的算法在运行效率和内存占用两方面都达到较理想的效果。

算法 Epattern_searcherH 是应用汉明距离标准来度量多重复近似模式之间的误差，而算法 Epattern_searcher 是应用编辑距离标准来度量多重复近似模式之间的误差。在具体应用中，汉明距离是要求待比较的两个片段的长度相等（两个片段含有相同个数的 X 因子），汉明距离的值等于相同位置处 X 因子不同的个数之和，而编辑距离则可在待比较的两个片段的长度不相等（两个片段所含有 X 因子个数可以不相等）的情况下进行，相比之下编辑距离的运用范围较广些，在实际应用中也相对灵活些。

英文小说中的 X 因子来自英文词库集 σ，$|\sigma|$ 大约为 20 万左右。

X 因子的字符来自于字符集 $\sum$，其集合由以下三种字符组成：

(1) 大小写英文字母，$a\sim z$，$A\sim Z$；共 $26\times 2=52$ 个字符；

(2) 数字字符，$0\sim 9$，共 10 个；

(3) 其他字符，空格；

则集合 $\sum=\{a\ \sim z,0\sim 9,$空格$\}$，即 $|\sum|=63$。

4.1 算法 Epattern_searcherH

4.1.1 问题定义

这里的模式查找算法是基于文本内容的搜索思想。在给定长为 n 个 X 因子的字符串 S($s_{p_1}s_{p_2}\cdots s_{p_m}$)中，查找一种让人感兴趣的长为 p 个单位的近似模式串，其模式片段 p 在字符串 S 的 m 个片段中至少重复出现 r 次($r\le m$)，且任何两个近似模式片段之间的汉明距离最大为 d。通常要求容错率小于 10%($d/p\le 0.1$)。同时，我们称所查找的重复模式为一个(p,d,r)_reduplicate。

4.1.2 相关定义及定理介绍

定义 4.1 共享 k 因子定义：在所查找的重复模式 p 中，任意两个模式 p 之间存在 d 个误差，由于误差 d 的位置将 p 划分为 d+1 个长度不等的小片段，在这些小片段中又存在长均为 k($k \geq 2$)个单位的非重叠公共子串（简称：**共享 k 因子**）。

如图 4.1.1 所示，在长为 n 个 X 因子组成的字符串 S 中，存在长均为 16 的两个近似模式 p 的重复片段 p_1 和p_2，也就是说，近似模式 p 在字符串中重复出现了两次 ($r=2$)，这两个近似模式之间的误差用汉明距离 d 来表示，其值为 2，出现误差的位置将模式串 p 又分成 3 个小片段，在每个小段中又存在长为 4 个单位的 k 因子，在这两个近似模式片段 p_1 和p_2 的相同位置处对应的 k 因子是完全相同，属于精确匹配。

算法 Epattern_searcherH 的过滤的条件用 f 表示，其过滤条件的具体是指所要查找的基本模式 p 中包含的非重叠共享 k 因子的最少个数。

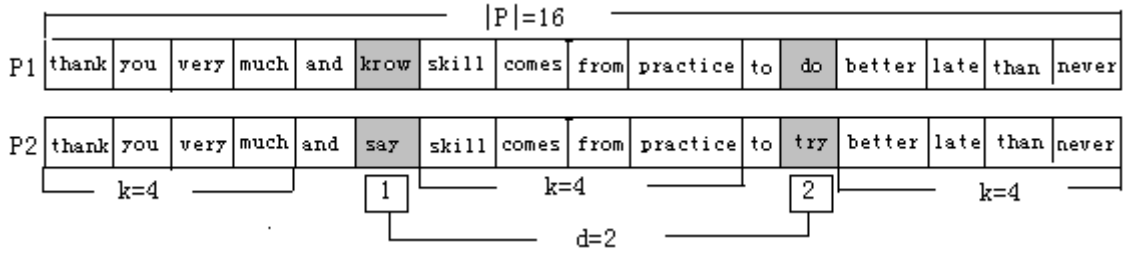


图 4.1.1 重复模式 p1 与 p2，其长度$|p|=16, d=2, r=2, k=4$, 长为 4 个单位 k 因子分别为：thank you very much、skill comes from practice、better late than never.

给定由 n 个 X 因子组成的字符串 S，每个 X 因子来自于英文词库集σ，其 X 因子的字符又属于字符集$\sum$，S 表示为 S[0]S[1]…S[n-1] $(s[i] \in \sigma, 0 \leq i < n)$。S 的长度也可表示为$|S|$。定义$S[i, j]$为 S 从第 i 个 X 因子位置开始到第 j 个 X 因子结束的子串，即表示为$S[i]S[i+1]\cdots S[j]$。如$S=UV$，$U,V \in \sigma$,称 V 为 S 的后缀。

定理 4.1 如果$r \geq 2, d \geq 0$且它们都为整数。$\theta(r,d)$表示相同位置处出现不同 X 因子的个数之和，出现 X 因子不同的这些位置是标注 r 个片段中任意两个片段的汉明距离最大为 d。则有$\theta(r,d)=r \cdot d/2$。

证明：首先，证明$\theta(r,d) \leq r \cdot d/2$，为了证明它，我们先对 d 应用了一个假设。如果$d=0$，$\theta(r,d)=0$。由于寻求$\theta(r,d)$的一个上限，选择 r 个片段中最多的不同 X 因子的点与尽可能多的限制，限制每两个片段的汉明距离最大为 d。为了估计$\theta(r,d)$

实际数量，必须考虑两个情况。第一种情况，没有两个片段之间的汉明距离为精确的d，由归纳假设法得：$\theta(r,d)\le\theta(r,d-1)\le r\cdot(d-1)/2\le r\cdot d/2$成立。第二种情况，至少有两个重复片段间的汉明距离为 d，为了不失一般性，假设 p_1 和 p_2 之间的汉明距离是 d。由 d 引导的 d 的位置 j 处有 $p_1[j]\neq p_2[j]$。让 J 代替这些位置。如果没有其它的点，则证明结束。因为已经证明存在θ(r,d) = d $\le r\cdot d/2$。假设现在开始的都是 $j'\neq J$，因此 $p_1[j']\neq p_2[j']$。按照标注点的定义，存在两个整数$i''>i'$，满足 $p_{i''}[j']\neq p_{i'}[j']$。因此 $i''>2$，也可以假设 $3\le i''\le r$（这个特殊性暗含在$r\ge 3$里），如果$i''\le 2$，取 $i=i''$，则有 $p_{i'}[j']=p_1[j']=p_2[j']$，与 $p_i[j']=p_{i''}[j']$不同。如果$i'>2$，则有 $p_1[j']=p_2[j']$，同样和 $p_{i''}[j']$或者 $p_{i'}[j']$得到的不同。如果取$i\in\{i'',i'\}$，则有 $p_1[j']=p_2[j']$，它们和 $p_{i'}[j']$不同。由于$j\in J$，则有 $p_1[j]\neq p_2[j]$和 $p_i[j]\neq p_1[j]$或 $p_i[j]\neq p_2[j]$。因此$i'''\in\{1,2\}$，p_i 和 $p_{i'''}$不同，至少 d/2 个的位置是 J 同样和 j' 不同。这就暗示着 p_i 和 $p_{i'''}$不可能相同的，比较 $d-d/2=d/2$ 位置上不是 J，则有 $p_i=p_{i'''}$。对于$r-2$个可能的i值，有$r\cdot$d/2 个 j'的可能值。因此$\theta(r,d)\le r\cdot$d/2。

其次，需证明$\theta(r,d)\ge r\cdot$d/2 成立。考虑以下 r 个的片段，所有这些片段的长度都是$r\cdot$d/2。令$1\le i\le r$，i是整数，片段 p_i所有的位置为 A，除了以下的 T：第一个 d/2 个字母在p_1中；接下来 d/2 个字母在 p_2 中，代替 p_1；下一个 d/2 个字母在 p_3 中，代替 p_2；以此类推，知道最后一个 d/2 个字母在 p_r 中。通常，$p_1[0]=T$ 和每一个$1\le i\le r$且 $j>0$，令 $p_i[j]=T$，如果$(i-1)\cdot$d/2 $\le j<i\cdot$d/2；所有其它的位置都用 A 代替。可以验证从 p_1 到任何其它片段之间的汉明距离是 d。由于每一个位置是一个点在这些片段中，即为证明$\theta(r,d)\ge r\cdot$d/2 提供了束缚条件。

因此，$\theta(r,d)=r\cdot$d/2 得证。

定理 4.2 令 $p\ge 0,r\ge 2,d\ge 0$,且它们都为整数。则非重叠共享 k 因子的最少个数为 f，$f=\lfloor p/k\rfloor-r\cdot$d/2。

证明：因为在属于一个(p,d,r)_reduplicate的模式片段 p 中，可能有$\lfloor$p/k$\rfloor$个非重叠的共享 k 因子，且有任何标注点θ(r,d)可以与之匹配或等同(重合)，然后证明出 $f=\lfloor p/k\rfloor-\theta$(r,d)。

定义 4.2 *Tfactor*(k,a)**双因子**，是由两个长均为 k 个单位的共享 k 因子组成，a 表示两个 k 的因子之间的间隔长度。例如在字符串 S 的第 i 个 X 因子的位置开始有一双因子 *Tfactor* 出现，可用形式 $s[i,i+k-1]s[i+k+a,i+2\cdot(k+a)-1]$ 来表示。在双因子 *Tfactor*(k,a)中，其$a\in[(Kfnum-2)\cdot k,p-2k]$。

重复模式 p 是由多个共享 k 因子组成的字符串片段，也是由多个双因子 $Tfactor(k,a)$ 形成的序列 kseq。对于双因子 $Tfactor(k,a)$ 序列的形成过程，其 a 的取值是由最大值 $p-2k$ 开始的，在此过程中 a 是逐渐变小，每次形成过程 a 的值减少 k，a 的最小值为 $(f-2)\cdot k$ 。如图 4.1.2 实例所示。

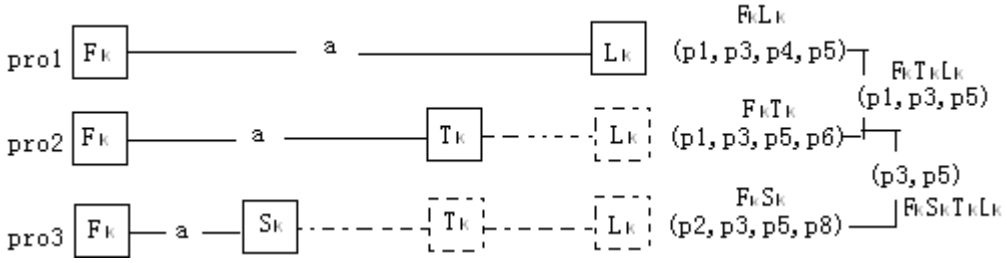


图 4.1.2 字符串 S 分成的 8 个长为 p 片段 $(m=8), r=2, f\geq 4$，
基本模式 p 由 4 个 k 因子组成,即 FkSkTkLk

在图 4.2.2 的实例中，查找到的重复模式 p 的过程如下列步骤所述：

第一步是在给定字符串 S 中，一共可分为 8 个长均为 p 的片段（模式），要求所查找的模式 p 至少在此字符串中重复出现两次 $(r=2)$。双因子 $Tfactor(k,a)$ 的第一个 k 因子 F_k 是固定在模式的首位置处不变，首先寻找间隔 a 取最大值时的另一个 k 因子 L_k，形成第一个双因子 F_kL_k。双因子 F_kL_k 在模式（长为 p 个 X 因子） $p_1\ p_3\ p_4\ p_5$ 中出现。

第二步是 a 的取值开始减小，第一个 k 因子 F_k 不变，找到另一个因子 T_k，则第二个双因子 F_kT_k 在片段 $p_1\ p_3\ p_5\ p_6$ 中出现。总结第一步与第二步的结果，则有三个 k 因子 $F_kT_kL_k$ 形成的串，在片段 $p_1\ p_3\ p_5$ 中出现。

第三步是继续缩小 a 的取值，第一个 k 因子 F_k 仍然固定不变，找到另一个 k 因子 S_k，第三个双因子 F_kS_k 在片段 $p_2\ p_3\ p_5\ p_8$ 中出现。实例中的过滤条件 f $(f=4)$，也就是说，要求所查找到的模式 p 至少含 4 个 k 因子。如果模式 p 中所包含的 k 因子的数量少于 4 时，则此片段直接被过滤掉，不再考虑。

总结前面第一、二、三步操作所述，所给的字符串 S 被分成的 8 个长为 p 的片段 $(m=8)$，要求每两个片段（模式）之间的汉明距离不超过 d，且片段 p 至少重复两次 $(r=2)$，满足过滤条件 $f\geq 4$。查找到的重复模式 p 至少由 4 个 k 因子组成，即 $F_kS_kT_kL_k$，且出现在字符串 S 的两个片段 p_3 和 p_5 中，也就是说，查找到的满足条件的模式串 $F_kS_kT_kL_k$（基本模式）是由四个 k 因子所组成的片段。

算法中的四个参数 p、d、r、k，它们都是由用户根据实际情况给定。在具体应用

中，算法的运行效率受到给定参数值的影响。若 k 取值太大，则会出现 $f=0$，这样使过滤条件失去意义；若 $k=1$ 或 $k=2$ 时，在应用中没有实际意义；因此，k 的取值根据客户自己的需要必须选择适当大小的值。这样才能使模式查找的算法在速度和准确率两方面都能得到较好的效果。

对于双因子 $Tfactor(k,a)$，当 k 与 a 的值固定时，先应用后缀数组来解决双因子 $Tfactor(k,a)$ 发生的一系列问题。再依后缀数组的特性，将从序列 S 的第 i 个 X 因子位置开始的后缀用 $S[i,n\text{-}1]$ 来表示。后缀数组 A 则表示将字符串的后缀按字典序排序后所得的数组，即 $A\in\{0,1,\cdots n\text{-}1\}$。则有 $S[A(0)\cdots]\le S[A(1)\cdots]\le\cdots\le S[A(n\text{-}1)\cdots]$ 成立(按字典序排序后)，Lcp 是用以表示按字典序形成后缀数组中的每两个相连续的后缀之间的最长公共前缀（共同祖先）。

算法 Epattern_searcherH 是一种重复模式查找的过滤算法，即为(p,d,r)_reduplicate 的查找算法。(p,d,r)_reduplicate 的模式是由双因子 $Tfactor(k,a)$ 的形成的序列 kseq 构成的，即由 a 的值从最大到最小变化的过程中构成双因子序列，也为若干个 k 因子形成的序列。序列 kseq（重复模式）形成过程中，其每个双因子依据位置的前后进行了编号，其编号用 No 来表示。如 $k=2$ 时，在序列 kseq 中，AA 的编号为 No0，AB 的编号为 No1，如此类推得出其它 k 因子的编号。(No1,No2) 表示双因子中两个 k 因子发生位置的编号，其中 No1 表示每发生双因子中的第一个因子，而 No2 表示每发生双因子中的第二个因子。存储对应双因子发生的位置列表用 position(S)来表示。在查找的过程中，当 k 和 a 取定值时，对应双因子的位置与当前字符串后缀之间的关系有：假设在字符串 S 的后缀对应着字典序后缀数组 $A[i]$ 位置处，发生双因子的第二个因子 No2，则可推出第一个双因子 No1 的位置 posNo1 为 $A[i]-k-a$。

i	lcp	A	suffix	posNo1	No
0	0	2	love love me me love me	∅	0
1	1	6	love me	4	1
2	2	0	love me love love me me love me	∅	1
3	2	3	love me me love me	0	1
4	0	7	me	4	2
5	1	1	me love love me me love me	∅	3
6	2	5	me love me	2	3
7	1	4	me me love me	1	4

图 4.1.3 当 $k=2$ 和 $a=1$ 时，后缀数组 A 和对应的 No、posNo1、lcp 数组的完成情况

在英文文本 love me love love me me love me 中，假定 k=2 和 a=1 时，在双因子的

序列形成过程中，各个数组之间的关系如图 4.1.3。出现双因子的编号、发生位置及对应双因子如图 4.1.4。

双因子位置编号 No 与每相邻后缀的最长公共前缀 LCP 的函数关系如公式 4.1：

$$No[i]=\begin{cases}0 & i=0,\\ No[i-1]+1 & lcp[i]<k,\\ No[i-1] & lcp[i]\geq k.\end{cases} \tag{4.1}$$

position(s)	(No1,No2)	Tfactor
0,3	(1,1)	love me love love
1	(3,4)	me love me me

图 4.1.4 当 $k=2$ 和 $a=1$ 时，双因子序列以 k 因子 Love me 开始，观察到(2,1)-双因发生位置

4.1.3 算法的实现

算法 Epattern_searcherH 的实现步骤描述如下：

step1 输入n个单词的英文文本S，且进行第一次预处理。

step2 给定算法中参数p、r、d、k的值，且计算出过滤条件 f 的值，其表示所查找的模式中包含的最少k因子的个数，$f=\lfloor p/k\rfloor-r\cdot d/2$。

Step3 第二次预处理过滤的开始。双因子 $Tfactor(k,a)$ 的序列kseq形成。查找过程开始时，当第一个k因子固定在片段的开始位置处不变，且当双因子的间隔长度a取最大值 $p-2k$ 时，查找另一个k因子，保存当前查找到的另一个k因子的位置。在查找另一个k因子的过程中，每次间隔a的取值都递减k，直到a的取值达到最小值时，即 $a=(f-2)\cdot k$，查找另一个k因子的过程结束。在此过程中，另一个k因子的起始位置等于此位置处对应后缀数组的值。

Step4 计算双因子 $Tfactor(k,a)$ 的序列kseq(模式片段p)所包含的k因子的个数为B，然后与 f 进行比较。如果B小于 f，则此片段p被过滤掉，继续在后面剩余文本中进行step3的操作；如果B大于等于 f，保存片段p在序列S中的位置，并对这样的片段p开始计数。

Step5 继续搜索文本串S，重复step3和step4的操作，直至搜索完整个序列S，计算出没有被过滤掉的片段p的个数C，与r(模式重复的限定次数)进行比较。若C小于r，则没有找到满足条件的重复模式p；若C大于等于r，则找到了重复的近似模式串p。

Step6 输出重复的近似模式出现的所有位置，算法结束。

4.1.4 Epattern_searcherH 算法的复杂度分析

在模式查找过程中，算法 Epattern_searcherH 是将长为 n 个单位(X 因子)的字符串 S 划分成 m 个长均为 p 的片段，所以总共输入的大小$|S|=n\cdot|X|=p\times m\cdot|X|$。

(1) 内存分析，在算法的所有步骤过程所用到最多四个数组（后缀数组 A、LCP、No、posNo1），它们所占内存空间为 16n 个比特。在平均情况下，双因子$Tfactor(k,a)$中开始的 k 因子所占内存空间是$O(\frac{n\cdot|X|}{\sigma^k})$，双因子之间的间隔 a 可能取值在区间$[0,p-2k]$上，所以整个双因子序列总占内存空间为$O(\frac{n\cdot|X|}{\sigma^k}\times P)$。

因此，算法的空间复杂度为$O(16n+\frac{p\cdot|X|}{\sigma^k}\cdot n)=O((16+\frac{p\cdot|X|}{\sigma^k})\cdot n)$。

(2) 时间复杂度，第一步预处理所用时间$O(n)$。第二步过滤预处理，在最坏情况下，算法的时间复杂度为$O(p^2\cdot n\cdot|X|\cdot Z^{f-1})$，其中$Z=p\cdot\min(\sigma^k,p)$，因为$p\ll\sigma^k$，所以时间复杂度可表示为$O(|X|\cdot p^{2f}\cdot n)$。

因此，算法的总的时间复杂度为：$O((|X|\cdot p^{2f}+1)\cdot n)$。

4.2 算法 Epattern_searcher

这里将算法 Epattern_searcherH 中重复模式之间的误差的衡量标准由汉明距离换为编辑距离标准，提出了新的算法 Epattern_searcher。

4.2.1 问题定义

给定由 n 个 X 因子（单词）组成的字符串 S，（p,r,d 都取整数，且满足$p>0$, $0\le d<p, 2\le r\le m$）中，将以 S 中每连续的 p 个 X 因子为标准划分片段，共分为 m 个片段（这些片段 p 都含 p 个 X 因子，可以表示为：$s_1,s_2,\cdots,s_m$）简称片段 p。其中有 r($r\le m$) 个片段 p 出现在字符串 S 中的一系列区间$[p\text{-}d,p+d]$上，其 r 个片段中任意两个片段 p 是非重叠的且它们之间的编辑距离最大为 d，则称这样的多重复模式为一个(p,d,r)_repeat 。

长为|S|的字符串 S 是由 n 个 X 因子组成的，可以表示为：$S[0]S[1]\cdots S[n\text{-}1]$，每个 X 因子的字符来自于字符集$\Sigma$，每个 X 因子是来自于英文单词库集$\sigma$。其中S[i] ($0\le i<n$)表示字符串中的第 i 个 X 因子。$S[i,j]$定义为 S 从第 i 个 X 因子的位置开始

到第 j 个 X 因子结束的子串，即 $S[i]S[i+1]\cdots S[j]$。如果 $S=UV$,称 V 为 S 的后缀。令 $v=s[i,j]$ 和 $v'=s[i',j']$ (i 对应 i'，j 对应 j')，如果区间[i, j]和$[i',j']$的交集是非空的，那么片段 v 与 v' 重叠。

4.2.2 编辑距离

编辑距离是 A．Levenshtein 提出的距离函数，以替换、插入和删除这三种基本操作来确定两个字符串的相似程度[41]。编辑距离可以给出任意长度的字符串之间的距离，但组成这两个字符串的字符必须来自同一个字符集合。

在编辑距离模型中，一个差异等价于一个编辑操作：插入一个字符、删除一个字符，或者替换一个字符。设 x，y 是两个字符串，它们之间的编辑距离 $ed(x,y)$ 就是将 x 转变为 y 所需的最少编辑操作次数，例如：$ed(annual, annealing)=4$。如果模式串的长度是 m，出现错误的个数为 k，那么 k 必须满足 $0<k<m$，比值 $\alpha=k/m$ 称为错误水平，通常在很多应用中都要求 $\alpha<1/2$。下面介绍一种用来计算编辑距离的最古老也是最灵活的动态规划算法。

基于动态规划的近似串匹配算法的首要问题是计算编辑距离 $ed(x,y)$。动态规划算法计算编辑距离时要使用一个矩阵 $M_{0\cdots|x|,0\cdots|y|}$，这里 $M_{i,j}$ 的含义是：将 $x_{1\cdots i}$ 变成 $y_{1\cdots j}$ 所需要的最少编辑次数，亦即 $M_{i,j}=ed(x_{1\cdots i},y_{1\cdots j})$。

$M_{0,0}$ 是两个空串的编辑距离，值为 0；对于长度分别为 i 和 j 的两个串 x，y 来说，假定所有长度小于 i 的 x 子串和所有长度小于 j 的 y 的子串的编辑距离已经计算出来，则可以利用它们推导出 $ed(x_{1\cdots i},y_{1\cdots j})$。

考虑两个串的最末字符 x_i 和 y_j：如果 x_i 和 y_j 相等，那么只要把 $x_{1\cdots i-1}$ 转换成 $y_{1\cdots j-1}$ 就可将 x 转换成 y，此时，$M_{i,j}=M_{i-1,j-1}$；如果 x_i 和 y_j 不等，则有且只有三种办法将 x 通过三种编辑操作转换成 y：

用 y_j 替换 x_i，然后将 $x_{1\cdots i-1}$ 转换成 $y_{1\cdots j-1}$，此时 $M_{i,j}=M_{i-1,j-1}+1$

从 x 中删除 x_i，然后将 $x_{1\cdots i-1}$ 转换成 $y_{1\cdots j}$，此时 $M_{i,j}=M_{i-1,j}+1$

从 y 中删除 y_j，然后将 $x_{1\cdots i}$ 转换成 $y_{1\cdots j-1}$，此时 $M_{i,j}=M_{i,j-1}+1$

这里要注意的是，在一个串中插入一个字符的效果等价于另一个串中删除相应的字符。综合上述几种情况，便得到公式 4.2.2。

不论最坏情况还是平均情况下，动态规划这种算法的时间复杂度都为 $O(|x|\cdot|y|)$。一个更易于编程的计算矩阵 M 的公式如下：

$$M_{i,0} \leftarrow i \quad M_{0,j} \leftarrow j$$

$$M_{i,j} \leftarrow \begin{cases} M_{i-1,j-1} & \text{如果} x_i = y_j \\ 1+\min\left(M_{i-1,j-1}, M_{i-1,j}, M_{i,j-1}\right) & \text{否则} \end{cases} \qquad (4.2.2)$$

因为矩阵 M 中相邻元素的值最多相差 1，所以当$\delta\left(x_i, y_j\right)=0$时，$M_{i-1,j-1}$不可能大于$M_{i-1,j}+1$或$M_{i,j-1}+1$。

计算出矩阵 M 后，便可以回溯得到最优路径，也就是从$M_{0,0}$到$M_{|x|,|y|}$最小代价的矩阵元素序列。路径可能存在多条，每条路径都对应一个比对，它是 x 中的每个字符和 y 中的每个字符之间的一幅映射图，表示 x 怎样经过匹配、替换、删除等操作而转变为 y。

如图 4.2.1 给出了用动态规划算法计算$ed\left(annual, annealing\right)$的示意图。

图 4.2.1 为动态规划算法实例：计算“annual”和“annealing”的编辑距离。图中用圆圈标明的矩阵元素给出了唯一的最优路径，右边给出了字符串的比对，其中虚线表示替换操作。

		a	n	n	e	a	l	i	n	g
	⓪	1	2	3	4	5	6	7	8	9
a	1	⓪	1	1	1	0	1	1	1	1
n	2	1	⓪	1	2	1	1	2	1	2
n	3	2	1	⓪	1	2	2	2	2	2
u	4	3	2	1	①	2	3	3	3	3
a	5	4	3	2	2	①	2	3	4	4
l	6	5	4	3	3	2	①	②	③	④

图 4.2.1 动态规划算法计算编辑距离矩阵矩阵示例

定义 4.3 编辑距离：两个字符串 A 和 B 之间的编辑距离的标准定义为：将字符串 A 通过对 X 因子的插入操作、删除操作和替换操作的三种基本操作转换为字符串 B 的最少操作 X 因子的次数。

A = surgery
⇒ survery ——— 将g替换为v
⇒ survey ——— 删除r
⇒ surveys ——— 插入s
= B

图 4.2.2 字符串 A=‘surgery’与 B=‘surveys’之间的编辑距离

例如图 4.2.2 所示，将单词‘surgery’转化为‘surveys’需要进行三次操作。需要注意的是，将字符串 A 转化为 B 可以有很多种操作序列(从而其操作次数也不同)，但 A 和 B 之间的编辑距离定义为所有操作序列所对应的操作次数的最小值。因此，求两个字符串间的编辑距离实际上是一个最优化问题的解。

定义 4.4 编辑距离的扩展定义：这里的编辑距离是指两个由 n 个 X 因子组成的字符串序列，它们之间通过对 X 因子的三种基本操作：插入操作、删除操作、替换操作，将一个字符串序列转换为另一个字符串序列的最少操作 X 因子的次数。

计算两个字符串 s1+ch1 与 s2+ch2 的编辑距离有下面性质：

性质 4.1 $d(s1+ch1,s2+ch2)=\min(d(s1,s2)+ch1==ch2\ ?\ 0:1,$

$$d(s1+ch1,s2)+1,d(s1,s2+ch2)+1)\ ;$$

由于我们定义的三个操作来作为编辑距离的一种衡量方法。于是对 ch1 与 ch2 可能的操作有：

第一种操作是把 ch1 变成 ch2 ；

第二种操作是 s1+ch1 后删除 ch1 ， $d=\left(1+d\left(s1,s2+ch2\right)\right)$；

第三种操作是 s1+ch1 后插入 ch2 ，则 $d=\left(1+d\left(s1+ch1,s2\right)\right)$。

对于第二种和第三种的操作可以等价于：

s2+ch2 后添加 ch1，则有 $d=\left(1+d\left(s1,s2+ch2\right)\right)$ 成立；

s2+ch2 后删除 ch2，则有 $d=\left(1+d\left(s1+ch1,s2\right)\right)$ 成立；

因此，可以得到计算编辑距离的性质 4.1。

4.2.3 算法相关的基本概念、定义、定理

q-gram 的定义：q 个连续字符组成的字符串。

QUASAR[42] (Q - gram Alignment based on Suffix Array)是一种基于后缀数组的数据库搜索算法。QUASAR 算法基于这样的观察：如果两个序列之间存在一个低于某特定值的编辑距离 e (Edit Distance) ，那么，它们就共享一定数量的 q-gram。这里，q-gram 指长度为 q 个单位的匹配信息串。假定重复模式长度为 p，各个片段 p 之间至少共享 t 个这样的 q- gram ，则 $t=|p|-q+1-(qe)$（文献[42]中已证明）。

通常用户对输入的英文文本进行第一次预处理（符号处理）后，然后给定算法中一些参数 p、r、d、q 和 g 的值，计算出算法的过滤条件的限定值，才能在序列中进行多重复模式的查找，即 (p,d,r)_repeat 的查找。由于重复片段很难找到，所以要对序

列 S 进行过滤预处理，将不发生(p,d,r)_ repeat 的部分片段的位置过滤掉，不再考虑，但这些过滤条件很难被快速的验证。过滤的必要条件是以发生一个(p,d,r)_repeat 中的两个 q–gram 的属性为基础的。这里所用到的部分技术曾经被文献[43]中的作者在 1985 年使用过。所以，本算法是按照一些更接近的定义和文献[44]等所给出的属性和技术来进行过滤的预处理。

这里定义的 q–gram 是指长为 q 个 X 因子的片段信息。对于片段 v = s[i, j] 的长度可以用 $|v|$ 来表示。

定义 4.5 给定的一个序列 S，一个 q-hit h 定义为一对 (i, j)，其表示在序列 S 的第 i 个 X 因子的位置和第 j 个 X 因子的位置处有相同的 q-gram 发生，也就是说 $s[i,i+q\text{-}1]=v=s[j,j+q\text{-}1]$。也称 v 是 h 所指的 q-grams。对于任何一对 $h=(i,j)$，称 i(resp.j) 是 h 的第一次投影(对应的 j 是 h 的第二次投影)。

定义 4.6 给出比对 q-hit $h=(i,j)$，h 的对角线 $diag(h)=\{h'=(i',j')\mid j'\text{-}i'=j\text{-}i\}$，$h'=(i',j')$ 位置上的所有可能比对具有相同的值 $j\text{-}i$。简称为 $j\text{-}i$ 对角线。如果对角线 $h=(i,j)$ 和 $h'=(i',j')$，满足 $(j\text{-}i)\text{-}(j'\text{-}i')=1$ 或−1，则这两条对角线是连续的。

考虑到两个近似重复片段 $v=s[x,x+p\text{-}1]$ 和 $v'=s[x',x'+p\text{-}1]$，如果它们之间的编辑距离是 0，则有 $p\text{-}q+1$ 对 q-hits 即 $(x,x'),...,(x+p\text{-}q,\ x'+p\text{-}q)$。总之，序列中的一次编辑操作引起一个 q-hit 最多移动一次对角线，且可能引起最多 q 个 q-hits 的移动。由于重复片段 v 和 v' 两个之间的差异最多需要 d 步编辑操作完成，所以重复片段中必定有 $t=(p\text{-}q+1)\text{-}qd$ 个 q-hits 存在，这也证明了规则 1。

算法 Epattern_searcher 过滤时必须满足下面五个必要条件，也称为规则。其规则 1 曾在文献[43]中定理 5.1 中第一次出现。

规则 1，两个重复片段 v 和 v' 中至少有 $t=(p\text{-}q+1)\text{-}qd$ 个 q-hits ；

在过滤时，任何比对 q-hits $h=(i,j)$ 和 $h'=(i',j')$ 必须满足规则 2、3、4、5：

规则 2，$|diag(h)\text{-}diag(h')|\le d$ ；

规则 3，$i\ne i'$ ；

规则 4，$j\ne j'$ ；

规则 5，有且仅有 $j<j'$ 时，有 $i<i'$。

如图 4.2.3 的实例满足了上面介绍的过滤的 5 个必要条件（规则）。对于一个编辑操作最多移动一个 q-hits 的对角线一个位置。因此 d 步编辑操作最多移动对角线 d 个位置，同时验证了规则 2(文献[44]已经应用过的过滤条件)。规则 3、规则 4 和规则 5

有待进一步被验证，它们在其它算法中还没有提到过，这里是在 Epattern_searcher 算法中首次应用到这三个过滤的必要条件。

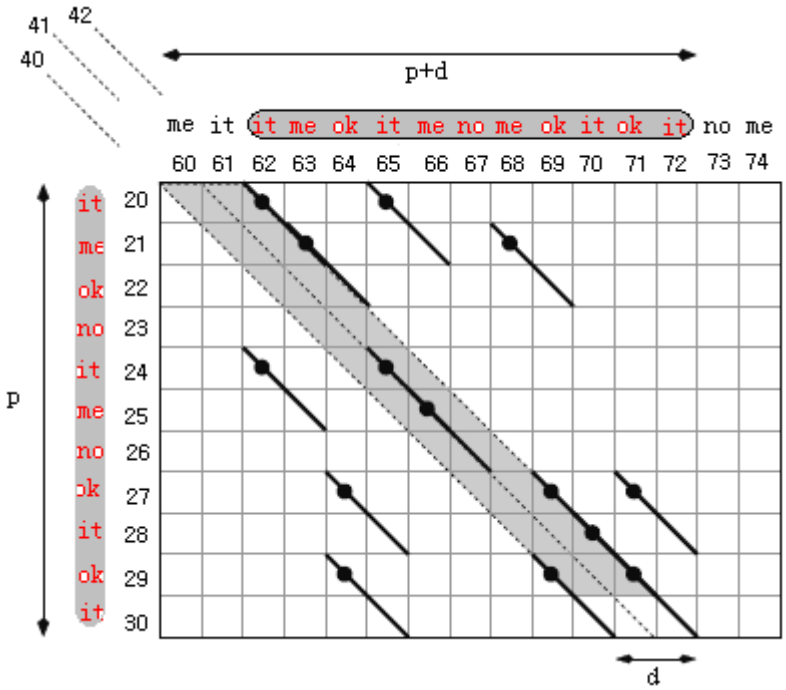


图 4.2.3 p=11,d=2, (p,d,r)-repeat 的实例

定理 4.3 如果 $v=s[x,x+p-1]$ 和 $v'=s[x',x'+p-1]$，它们之间编辑距离最大为 d，那么至少有 $t=(p-q+1)-qd$ 个 q-hits，其 q-hits 可以验证过滤的必要条件 3、4、5 成立。

证明：让 V 和 V' 在字母集 $\sigma\cup\{空格\}$ 上。比如：

(1) 没有 $i\in[0,|V|-1]$ ，使 $V[i]$ 和 $V'[i]$ 等同于''；

(2) $p\le|V|=|V'|\le p+d$；

(3) 这些序列从 $V(resp.\ V')$ 中获得，删去所有的''等同于 $v(resp.\ v')$；

(4) V 和 V' 是一个最优定位的代表，在 v 和 v' 之间符号''代表每个单词间的间隙——空格，其函数值对应编辑距离（0 表示发生一个匹配，1 表示发生任何一个编辑操作）。

现在让 Y 是在字母集 $\{M,D\}$ 中的一个字符串，例如：

(1) $|Y|=|V|=|V'|$；

(2) 对所有的 i 从 1 到 $|Y|$，如果 $V[i]=V'[i]$，则有 $Y[i]=M$，否则 $Y[i]=D$。

引理 4.1 这里有至少 $t=(p-q+1)-qd$ 个不同的位置 i 且 $j\in[0,q-1]$，则有

$Y[i+j]=M$ 成立，即：至少有 $t=(p-q+1)-qd$ 位置 i 处，发生了大小为 q 的匹配。

证明：显然，这里最多有 $|Y|-q-1$ 个大小至少为 q 的匹配在 Y 中发生。进一步说，在 Y 中每个字符 D 表示至多 d 个不匹配发生。如果用 N 来表示在 Y 中发生匹配的数量，其匹配大小至少为 q，则有：$N\geq(|Y|-q+1)-qd\geq(p-q+1)-qd$ 。

由**引理 4.1** 得出 Y 的建立过程。这里至少有 $(p-q+1)-qd$ 个大小至少为 q 的匹配发生。匹配发生对应的每两个 q-qhits，它们对应着两个不同的 q-grams 在 v(在位置 i 和 i')和 v'(在位置 j 和 j')中发生，同时也证明了规则 3 和规则 4。显然，如果 q-hit(i,j)(若(i',j'))先发生，那么 $i>i'$ 和 $j>j'$(相反，则有 $i'>i$ 和 $j'>j$))，同时也证明了规则 5。

在**定理 4.3** 中至少有 $t=(p-q+1)-qd$ 个不同的位置 i, 对于所有的 q-gram, $j\in[0,q-1]$,即至少有 $t=(p-q+1)-qd$ 个位置 i 处产生了大小为 q（q 个 X 因子）的连续匹配。

如图 4.2.3 所示。一个(p,d,2)_repeat 和一个平行四边形的实例图，$p=11,d=2$。有对角线 40、41、42，其中 40 和 42 的距离是 2，而 40 和 41（及 41 和 42）是连续的。假设 $q=2$，一个 q-hit 的比对 q-grams 是由一个长度为 2 个单位大小的较粗斜线加上一个黑色小原点来表示。图中 q-hit（29,69）指向相同 q-gram (ok it)，29 对应 69。q-hit（27,69）也指向相同 q-gram(ok it)，但是不同与第一次的投影。第 20 个至第 30 个 X 因子的片段与第 62 个至第 72 个 X 因子的片段，是属于同一序列 S 中的两个不同的片段，其长为 p 个 X 因子的大小，即可表示为 S[20,30]与 S[62,72]。这两个重复片段之间的编辑距离为 2。通过删除 S[23]（在第 22 与 23 个 X 因子的位置处没有 q-hits 出现）和删除 S[68]（没有 q-hits 在第 67 与 68 的位置出现）两步编辑操作之后，则可从片段 S[20,30]中获得了另一个片段 S[62,72]。由 $t=(p-q+1)-qd$ 得 $t=6$，在斜线 41 和 42 之间有 7 个 q-hits 组成集合 S'是满足五个规则。如果再加上斜线 40 上的一个 q-hit 得到 8 个 q-hits 的新集合 S''，这时规则 1 和 2 仍然满足，但规则 3、规则 4 和规则 5 不再满足。

在图 4.2.3 中，灰色平行四边形是强调平行四边形 Parall(20,11,40,2)。根据**定义 4.6**，成对的 q-hits(29,69)和(29,71)的确属于 Parall(20,11,40,2)。观察到四点：第一，x 和 y 决定平行四边形的左上位置是 $(x,x+y)$。即 x 和 $x+y$ 是包含 q-hits 的平行四边形的起始位置；第二，$v=s[x,x+p-1]$ 的 q-gram $s[i,i+q-1]$ 出现的最大位置 i 取值为 $x+p-q$；第三，平行四边形 Parall(x,p,y,d)中可能有 $(p-q+1)\times(d+1)$ 对 q-grams，这

也决定了平行四边形的大小；第四，比对 $h=(i,j)\in$ Parall(x,p,y,d)，如果有 $v[i,i+q-1]=v[j,j+q-1]$ 成立，则 $h=(i,j)$ 是平行四边形中的比对 q-hit。

给定重复片段 $v=s[x,x+p-1]$ ，平行四边形 Parall(x,p,y,d)是用来验证规则 1 和 2，且 v 与其它 $r-1$ 个 v_k 是否是属于一个 (p,d,2)-repeat 。用下面方法来检查，设 $v=v_k=s[u,w]$ 是一个 (p,d,2)-repeat，比对的位置是在 $d+1$ 条对角线之间。若对角线 y，使 $u\in[x+y,\ x+y+d]$ 和 $w\in[x+y+p-1,x+y+d+p-1]$，那么比对发生匹配的位置是在对角线 $y,y+1,\cdots,y+d$ 之间。这样称平行四边形 Parall(x,p,y,d)是用来检测 v 和 v_k 是否属于一个 (p,d,2)_repeat 。

算法 Epattern_searcher 是必须满足规则 3、规则 4 和规则 5 的要求，从而产生后面实验结果。

第一种过滤方法 FIRST，首先，要求平行四边形内部的 q-hits 的集合须满足规则 3。此规则 3 保证集合中的两个不同的 q-hits 不能共享第一次投影。当且仅当在平行四边形中至少 t 个 q-hits 存在，且任何两个 q-hits 的投影与第一次的投影不同，称平行四边形是 FIRST 。如图 4.2.3 中的集合中包含的 7 个 q-hits，它们的 q-hits 比对满足规则 3，则对应平行四边是 FIRST 。

第二种过滤方法 SECOND，要求平行四边形内部的 q-hits 的集合须满足规则 4，也易得到验证。如果 $v=s[x,x+p-1]$，存在至少 r 个非重叠的平行四边形 FIRST，即：$\text{Parall}(\text{x,p,y}_\text{i},\text{d})\left(y_i\in\{y_1,\cdots,y_r\}\right)$，称这样的平行四边形是 SECOND，同时保存区间 $[x,x+p-1]$ 的一些位置。

第三种过滤方法是 THIRD，再进一步的要求平行四边形中的 q-hits 的集合必须满足规则 5，存在至少 r 个非重叠的平行四边形 SECOND，则平行四边形 SECOND 变成平行四边形 THIRD。由于集合中 q-hits 对都满足规则 5，那么它们也一定满足规则 3 和规则 4。如图 4.2.3 中的平行四边形的集合中包含 7 个 q-hits，它们满足规则 5，也满足规则 3 和 4。这里强调的平行四边形是 THIRD，能在本算法当中有效地验证的最后一个规则 5。若对任何片段 $v=s[x,x+p-1]$ ，存在至少 r 个非重叠的平行四边形 THIRD 即：$\text{Parall}(\text{x,p,y}_\text{i},\text{d})\left(y_i\in\{y_1,\cdots,y_r\}\right)$，则保存区间 $[x,x+p-1]$ 的位置。即为算法 Epattern_searcher 所查找的重复近似模式所在区间的位置。

对于任何片段 $v=s[x,x+p-1]$ ，检查其是否属于一个 (p,d,2)_repeat 。假设存在片段 $v_k\ (k=1,2,\cdots,r-1)$，v 和 v_k 之间的编辑距离不超过 d。在目前最好的算法中，v 和 v_k 的计算花费时间为 $O(d\cdot p)$。计算 v 和 v_k 中的 q-hits 中的个数，验证它们中至少有 t

个 q-hits，由于 v 和 v_k 中的 q-hits 满足规则 1。在理论上，最多有 (p-q+1)×(p-q+d+1) 个 q-hits。然而，任何比对 q-hits 必须满足规则 2，那么仅限在 $d+1$ 条连续的对角线之间来计算 q-hits 的个数（如图 4.2.3 中的对角线 40 41 42），它们可能包含 q-hits 最多为 $(d+1)\times(p-q+1)$ 个，使用对角线对 q-hits 进行排序。下面引入平行四边形的概念（文献[44]已经应用过）。

定义 4. 7 平行四边形定义：给定长为 p 个单位的重复模式片段 $v=s[x,x+p-1]$，有 $d+1$ 条连续的对角线 $[x,x+d](d\le p)$，这里所有比对(i,j)的集合所对应的各个平行四边形可表示为：

$$Parall(x,p,y,d)=\{(i,j)|i\in[x,x+p-q],j-i\in[x,x+d]\}$$

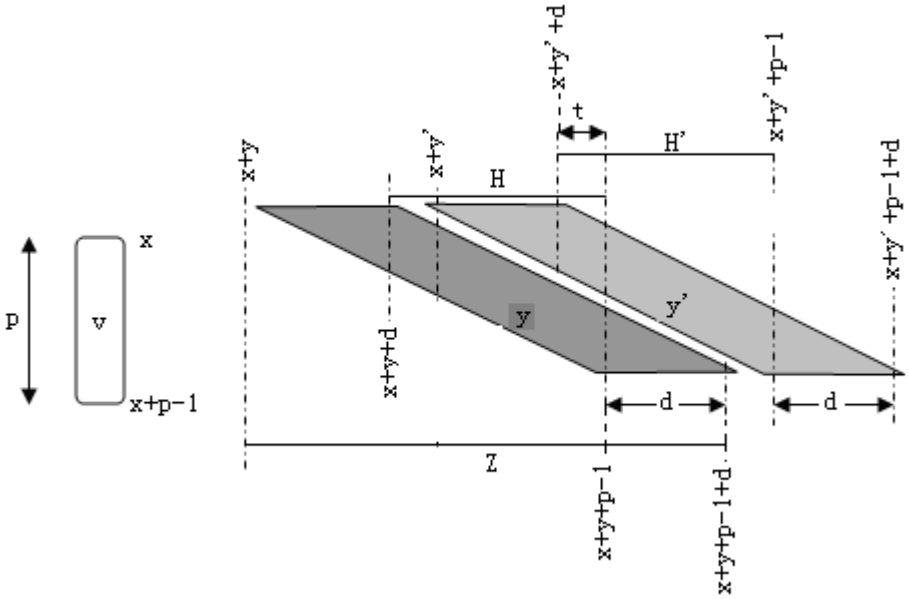


图 4.2.4 检测(p,d,r)_repeat 和两个重叠的平行四边形实例

如图 4.2.4 所示实例，图中深灰色的平行四边形是检测片段 v 和 v_k 之间的 q-hits，其中 $v=s[x,x+p-1]$，$v_k=S[i,j]\left(i\in[x+y,x+y+d],j\in[x+y+p-1,x+y+d+p-1]\right)$。$p+d$ 个 X 因子的片段 Z，$Z=S[x+y,x+y+d+p-1]$，其包含了片段 v_k，而 v_k 又包含了片段 H（H 为 $p-d$ 个 X 因子），$H=S[x+y+d,x+y+p-1]$。对比浅灰色的平行四边形，片段 Z' 为 $p+d$ 个 X 因子大小，$Z'=S[y'+x,y'+d+x+p-1]$，其中包含了片段 v_k'，v_k' 又包含了片段 H'（H' 为 $L-d$ 个 X 因子），$H'=S[y'+d+x,y'+x+p-1]$。在图中 v_k 和 v_k' 都没显示，因为它们中包含的 X 因子的个数是变化的。则 v_k 和 v_k' 它们一定重叠，因为它们都包含了 $t=S[x+y+d,x+y+p-1]$ 片段，即 t 为 H 和 H' 的重叠部分。

一个长为 $p-d$（$p-d$ 个 X 因子的大小）的片段 $H=s[x+y+d,x+y+p-1]$，它包含在片段 v_k 中，同时也包含在片段 Z（$p+d$ 个 X 因子组成）$Z=S[x+y,x+y+d+p-1]$ 中。H 与 Z 表示两个黑色区域的平行四边形。因为 $d<p$ 且 $x+y+d\le x+y+p-1$，所以对于任何一个 (p,d,2)_repeat，$\{v,v_k'\}$ 由同一个平行四边形（v,v_k' 存在部分重叠）来检测，如图 4.2.4 所示。

如果平行四边形 Parall(x,p,y,d)可检测出 $v=s[x,x+p-1]$ 和 $v_k=s[u,w]$ 属于一个 (p,d,2)_repeat，平行四边形 Parall(x,p,y',d) 可检测出 $v=s[x,x+p-1]$ 和 $v_k'=s[u',w']$ 属于一个 (p,d,2)_repeat；如果这两个平行四边形是重叠的，那么片段 $v_k=s[u,w]$ 和 $v_k'=s[u',w']$ 也必会重叠。因而，如果没有重叠的两个片段（片段长均为 p 个单位大小），则也没有重叠的扩大平行四边形。因此，要查找的重复片段是非重叠的，也只有应用非重叠的扩大的平行四边形去检测。

如图 4.2.5 所示，由 $d+1$ 条对角线平行四边形扩大到了 $d+g$ 条对角线的扩大平行四边形的实例图。

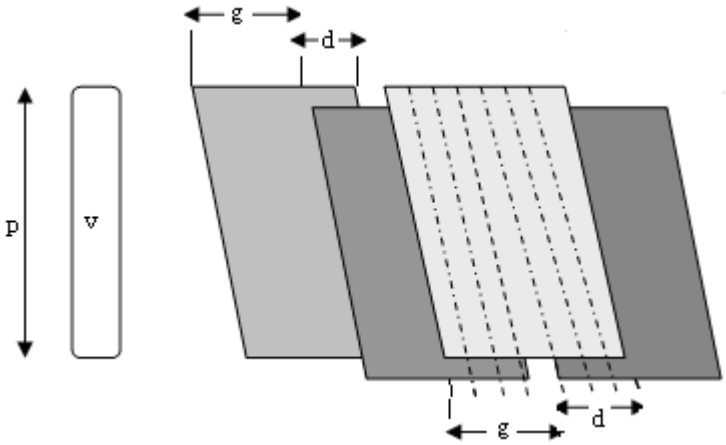


图 4.2.5 扩大的平形四边形的实例图

在图 4.2.5 中，有四个 $d+g$ 条对角线的扩大平行四边形，对于非负整数 k，它们开始于对角线 kg。例如，$d=3$，$g=4$。注意到两个连续的扩大的平行四边形，它们共享 d 条对角线，且任何一个 $d+1$ 条对角线的平行四边形被包含在一个 $d+g$ 条对角线的扩大平行四边形中。

4.2.4 算法 Epattern_searcher 的综述

首先，将输入的英文文本进行第一次预处理，把英文文本变成由 n 个 X 因子组成的字符串。然后再输入算法中各个参数 p、q、r、d 和 g 值，继续对算法进行过滤的预处理。

在过滤开始，首先，计算出算法中过滤第一个必要条件 t，其表示两个近似重复片段所共享的最少 q-gram 的个数，即 $t=(p-q+1)-qd$ 。

建立字符串 S 中出现的任何可能的 q-gram 的列表。总计出现 σ^q 次 q-gram，它的列表大小为 $n-q+1$ 。q-gram 是储存在 $n-q+1$ 大小的数组中，且指定 σ^q 个整型指向每个 q-gram。

沿着字符串S滑动的窗口（窗口的大小是p个X因子大小），出现在窗口中的片段 $v=s[i,i+p-1]$ ，仅考虑与这个滑动窗口相关的q-hits。对于每个位置i，必须考虑所有可能的平行四边形，比如： $Parall(i,p,y,d)\left(y\in[-i,n-i-d+1]\right)$ 。

为了快速的验证平行四边是FIRST或SECOND或THIRD，对每个平行四边形必须结合一个q-hit计数器。首先，计数器初始化为滑动窗口的零位置。在 $[0,p-q]$ 区间上对所有发生的q-grams初始化为零。然后，检查是否每个平行四边形中创建了至少一个q-hit。如果在这种情况下，对应的平行四边形的计数器每发生一次q-gram其值递增一。一旦滑动窗口从位置 $i-1$ 滑向i时，发生在位置 $i-1$ 处的q-hits所涉及的q-gram不再考虑，而那些发生在位置 $i+p-q$ 处所涉及到的新的q-gram必须被考虑。依据平行四边形定义，当 $j\in[y,y+d]$，在比对 $(i-1,j)$ 和 $(i+p-q,j)$ 处，观察所得到的平行四边形 Parall(i-1,p,y,d) 、 Parall(i,p,y,d) 是不同的。因而，为获得平行四边形 Parall(i,p,y,d) 中q-hits的数量，仅需要减去平行四边形 Parall(i-1,p,y,d) 中q-hits的数量。当S的滑动窗口从位置 i-1 滑向i时，仅考虑发生在区间 $S[i-1,i+q-2]$ 上 (搜索的剩余区间)和区间(正在插入X因子的区间) $S[i+p-q,i+p-1]$ 的那些q-grams。任何发生在j位置插入的q-gram，有一个q-hit $(i+p-q,j)$ ，且对这个q-hit所属的所有平行四边形增加它的计数器计数。相反，任何发生j位置处的插入的q-gram，有一个q-hit $(i-1,j)$ 不再涉及到当前滑动窗口的片段，所以对这个q-hit所属于的所有平行四边形递减它的计数器计数。

观察得到的每个q-hit属于 $d+1$ 个连续的平行四边形，例如：q-hit (i,j) 属于平行四边形 $Parall(i,p,(j-i)-k,d)(k\in[0,d])$ 。因此，对每个q-hit，应该更新 $d+1$ 个计数器。为了减少更新计数器次数，采用了参考文献[44]中SWIFT 和文献[42]中QUASAR都已

采用过的一种策略：扩大的平行四边形策略，其是指由$d+1$ 条对角线扩大到$d+g$ 条对角线($g \geq 1$)的平行四边形（SWIFT中实际应用了$d+g+1$对角线）。目的是为了避免同一个扩大的平行四边形中两个非重叠的平行四边形也可能重复发生，必须指定扩大的平行四边形的宽度不得超过p，因此有$d+g<p$成立。

扩大的平行四边形方法：是指平行四边形$Parall(i,p,k,d)\left(k \in [y,y+g-1]\right)$被加入到扩大的平行四边形$Parall(i,p,y,d+g-1)$之中。在实际应用中，所有扩大平行四边形$Parall(i,p,k,d)(k \in [y,y+g-1])$ 需设置唯一的计数器。因此，为了在整个输入序列中查找到重复片段，不再考虑平行四边形$Parall(i,p,y,d)$ $(y \in [-i,\ n\ -\ i-\ d+1])$，足以检查$Parall\left(i,p,y,d+g-1\right)\left(y=k'g,k' \in \left[-\lfloor i/g \rfloor,\left\lceil \left(n-i-(d+1)\right)/g \right\rceil\right]\right)$平行四边形。由于每个平行四边形$Parall(i,p,y,d)$ 被包含在这些扩大的平行四边形之中（如图4.2.5所示）。这种更改的原因是，存在一个q-hit仅被$\left\lceil \left(d+g\right)/g \right\rceil$个平行四边形所共享。对每个q-hit的更新由$d+1$次减少到$\left\lceil \left(d+g\right)/g \right\rceil$次。因为$d+g$条对角线可以形成一个扩大的平行四边形，这个扩大的平行四边形作为判定FIRST或SECOND或THIRD平行四边形的概率就增加了。就过滤条件而言，用d+g 条对角线形成的扩大的平行四边形来代替$d+1$条对角线形成的平行四边形，这种扩大的平行四边形使过滤保持无损且过滤速度更快。

在算法中，若发生在扩大的平形四边形中的q-hits的第一次投影值达到t时，那么对应的扩大平行四边形是FIRST。具体来说，由于发生q-hits的列表是有序的，这里仅更新那些在当前列表中还没有更新的那些q-hits的值。从而计算出q-hits的第一次投影的个数。当q-hits的计数器的第一次投影值达到t时，则直接验证对应的平行四边形是FIRST。

给定一个滑动窗口$v=S[i,i+p-1]$，有且仅有r个FIRST平行四边形被查明时，则称搜索到的平行四边形是SECOND。如果在SECOND平行四边形中至少有r个平行四边形SECOND被发现，则称搜索到了THIRD平行四边形，这个滑动窗口v对应的所有位置$[i,i+p-1]$被保存。

在SECOND平行四边形如何变为一个THIRD平行四边形过程中，假设v和v' 的q-hits集合满足规则5。为了验证规则5成立，必须查明在v和v' 中是否至少有t个q-grams以相同的顺序发生。为了达到这个目的，考虑了在v和v' 中发生的q-grams的最长公共有序子序列的长度。给出第三种过滤方法，它是用来检测一个平行四边形是否为THIRD，也需要知道这个平行四边形内所有q-hits的集合，那么就定义了如下q-hits链问题。

定义4.8平行四边形q-hits链问题：给定重复片段$v=s[x,x+p-1]$和平行四边形$Parall(x,p,y,d)$，求在平行四边形$Parall(x,p,y,d)$内的q-hits集合S'，使其内的q-hits必须满足规则5。

为了检测算法中的SECOND平行四边形是否为THIRD，采用了平行四边形q-hits链式问题来解决所含至少t个q-hits的集合S'。从平行四边形内的简单动态规划方法（平行四边形动态规划策略PDP）到稀疏的动态规划方法（从平均期望很小的几个q-hits实现这种策略）。在实现过程中，仅考虑出现在扩大的平行四边形内部的相关的q-hits的发生。为了提供这些发生q-hits列表的集合，在当前列表中执行了$p-q+1$次二分法搜索，称这个策略为PHS（Paralla Hunt Szymanski）。

这里设计的最优策略是应用滑动窗口的简单递增信息（$v=s[i,i+p-1]$是测试的滑动窗口），目的是尽可能避免下一个滑动窗口的如下的测试。考虑一个滑动窗口$v=s[x-1,x+p-2]$的平行四边形Ｐ，此平行四边形是用于解决q-hits链问题的方案，结果产生了一系列t个q-hits集合。从窗口v滑到$v'=s[x,x+p-1]$，同时也滑动了平行四边形P，解决平行四边形q-hits链问题平行四边形将产生t-1或t或t+1个q-hits链。因此，仅平行四边形P（在v的位置x-1处平行四边形的值等于t-1或t 或 $t+1$）有机会变成一个THIRD平行四边形（在v'的位置x处）。提出的平行四边形q-hits链问题仅针对其值是t-1或t或$t+1$的平行四边形的解决方法。若按PHS上面提出的策略去做，称为PQCP策略，由于这里使用了解决q-hits链问题的方法，从而产生此算法的一些实验结果。

为了检查非重叠的扩大平行四边形的重复，给出了一系列THIRD平行四边形，寻找非重叠的最大基数的平行四边形子集。对有序的平行四边形（按对角线开始位置递增排序）采用贪婪策略容易做到。$Parall(x,p,y,d+g-1)$ 和 $Parall(x,p,y',d+g-1)$两个连续的平行四边形是按照这种贪婪策略，每个平行四边形是由$d+g$条对角线形成的，可成为下一个连续的平行四边形是$Parall(x,p,y'',d+g-1)$。如果最初的两个平行四边形重叠$(y'-y<p-(d+g-1))$，那么则要继续移动下一个相连续的平行四边形parall$(x,p,y'',d+g-1)$，与前面第一个平行四边形$Parall(x,p,y,d+g-1)$重新做是否重叠判断；如果最初相连续的两个平行四边形没有重叠，则保留第一个平行四边形，且在紧接着两个相连续的平行四边形中，即$Parall(x,p,y',d+g-1)$和$Parall(x,p,y'',d+g-1)$中重新做是否重叠判断；以此类推，在剩余的序列中，都是针对两个连续的平行四边形重复上面这一过程的操作。此过程需线性运行时间。

Epattern_searcher算法中检查FIRST平行四边形的过滤方法被命名为FIRST，检查SECOND平行四边形的过滤方法称为SECOND，检查THIRD平行四边形的过滤方法被命名为THIRD。

算法Epattern_searcher采用第三种过滤方法THIRD的实现步骤描述如下：

step1 输入n个单词的英文文本S，且进行第一次预处理。

step2 给定算法中参数p、r、d、q、g的值，计算出第一个过滤的必要条件t的值，其表示重复模式所共享的q-gram的最少个数，$t=(p-q+1)-q\cdot d$。

step3 第二次预处理，即也过滤的开始。每个q-gram的起始位置等于当前位置处后缀数组的值。将窗口初始为0位置处，初始化在区间$[0,p-q]$上的发生每个q-gram的指针，且将所有扩大的平行四边形的计数器的指针初始化为0。

step4 在每个滑动窗口$[i,i+p-1]\left(i\in[0,n-1]\right)$中，更新属于每个扩大平行四边形的q-hit$(i+p-q,j)\left(j\in S[i+p-q,i+p-1]\right)$指针，当q-hit指针计数达到t时，则此时所得的平行四边形为FIRST，且对这样的FIRST开始计数。若滑动窗口中的扩大平行四边形中的q-hit指针计数未达到t，则过滤掉窗口对应的片段；若后面出现的FIRST平行四边形与第一个FIRST重叠，过滤掉后面的FIRST对应的片段，继续移动窗口，查找到与第一个FIRST平行四边形非重叠的FIRST时，才能使FIRST的计数递增一。

step5 继续移动滑动窗口，当非重叠的FIRST平行四边形的计数达到r时，则此时得到了SECOND平行四边形，且对所得到的这样的SECOND平行四边形开始计数。

step6 如果所得非重叠的SECOND平行四边形的个数大于等于r时，则所得的平行四边形是THIRD，且对这样的THIRD平行四边形开始计数。

step7 继续移动滑动窗口，搜索完整个序列，重复step4至step6的操作，当所查到的非重叠的THIRD平行四边形的计数大于等于r时，则保存此时的区间$S[i,i+p-1]$。

step8 输出近似重复模式出现的区间位置，算法结束。

4.2.5 Epattern_searcher 算法复杂度的分析

在算法中输入序列的长度$|S|=n\cdot|X|$，其表示序列S 由n个X因子（单词）组成。在分析中，为了表示方便，应用了参数h和l，其中h是表示q-hits的数目，l表示计算的平行四边形q-hits链问题的数目。

(1) 内存占用分析：主要数据结构q-gram的索引构造，在其构造过程中，需要一

个$n-q+1$大小的数组存储q-gram和$|\sigma|^q$个整型指针，所以占用空间是$O\left(n+|\sigma|^q\right)$。其它的数据结构为：共有$\lfloor n/g \rfloor$个指针指向所有当前的平行四边形，以及$\lfloor n/g \rfloor$个对角线的移动位置。

因此，算法空间复杂度为：

$$O\left(n+|\sigma|^q+\lfloor n/g \rfloor+\lfloor n/g \rfloor\right)$$

特别当$q \le log_{\sigma} n$时，即算法的空间复杂度为$O(n)$。

(2) 时间复杂度分析：有h个q-hits，在算法中每个q-hit的指针共需更新$\lceil (d+g)/g \rceil$次。应用PQCP策略（或PHS策略），每个平行四边形q-hits链问题在最坏的情况下，所需要的时间复杂度为：

$$O\left(p \cdot log\, n + p \cdot (g+d) \cdot log\, p\right)$$

在平均情况下所需要的时间复杂度为：

$$O\left(p \cdot log \frac{n}{|\sigma|^q} + p \cdot log\, p\right)$$

这里的平均情况是指σ中每个单词在文本中均匀出现，即在文本中的n个位置处每一位置出现的概率是$1/|\sigma|$。

这里估计一下h和l的取值。最坏情况下$h=n^2$，在平均情况下$h=n^2 \cdot |\sigma|^{-q}$。关于$l$的值，在滑动窗口的每个位置处最多有$\lfloor n/g \rfloor$个SECOND平行四边形，因此较坏的情况下，则$l=n \cdot \lfloor n/g \rfloor$。假定由序列比对得到的q-gram是均匀分布的，我们所期望的平行四边形是FIRST(SECOND)的概率是：

$$\sum_{i=t}^{p-q} C_y^i \cdot \left(|\sigma|^{-q}\right)^i \cdot \left(1-|\sigma|^{-q}\right)^{y-i}$$

其中$y=(p-q+1)\cdot(d+g)$表示扩大的平行四边形的大小，且$t=(p-q+1)-q \cdot d$。应用Stirling的公式：

$$i! = \sqrt{2\pi \cdot i} \cdot \left(\frac{i}{e}\right)^i$$

及其二项分布近似计算公式，当$n \cdot p$很小时，有：

$$c_n^i p^i (1-p)^i \approx \frac{(n \cdot p)^i}{i!} \cdot e^{-n \cdot p}$$

可得出：

$$\sum_{i=t}^{p-q} C_y^i \cdot \left(|\sigma|^{-q}\right)^i \cdot \left(1-|\sigma|^{-q}\right)^{y-i}$$

$$\approx \sum_{i=t}^{p-q} \frac{\left(y\cdot|\sigma|^{-q}\right)^{i}}{i!}\cdot e^{-y\cdot|\sigma|^{-q}}$$

$$\approx \sum_{i=t}^{p-q} \frac{\left(y\cdot|\sigma|^{-q}\right)^{i}}{\sqrt{2\pi\cdot i}\cdot\left(\frac{i}{e}\right)^{i}}\cdot e^{-y\cdot|\sigma|^{-q}}$$

$$\approx \sum_{i=t}^{p-q}\left(\frac{y\cdot e}{i\cdot|\sigma|^{q}}\right)^{i} = O\left(\frac{y\cdot e}{t\cdot|\sigma|^{q}}\right)^{t}$$

所以，期望的FIRST(SECOND)平行四边形的个数有：即 $l=\frac{n^2}{g}\cdot\left(\frac{y\cdot e}{t\cdot|\sigma|^{q}}\right)^{t}$

算法的最坏时间复杂度为：

$$O\left(\frac{d+g}{g}\cdot n^2+\frac{n^2}{g}\cdot p\cdot\left(\log n+(g+d)\cdot\log p\right)\right)$$

$$=O\left(\frac{n^2}{g}\cdot p\cdot\left(\log n+(g+d)\cdot p\right)\right)$$

同时也注意到当t足够大时，l 趋近于零时，算法的平均时间复杂度下降，其值为：$O\left(\frac{d+g}{g}\cdot n^2\cdot|\sigma|^{-q}\right)$。

因为 $g\cdot|\sigma|^{q} >> d+g$，所以 n^2 的系数 $(d+g)\Big/\left(g\cdot|\sigma|^{q}\right)$ 的值接近于零，所以当 n 趋向无穷大时，其时间复杂度的值也变化不大。

因此，算法的平均时间复杂度为：$O\left(\frac{d+g}{g\cdot|\sigma|^{q}}\cdot n^2\right)$。

4.3 本章小结

本章提出了模式查找的算法 Epattern_searcherH 和 Epattern_searcher，它们都是属于近似模式查找的过滤算法，将它们应用于长篇英文小说的英文高频词查找中。与模式查找的一般算法相比，速度较快。其算法 Epattern_searcher 比 Epattern_searcherH 运行速度快。在算法 Epattern_searcher 中，当误差率较小的情况下 $(d/p\le 0.1)$，它的第二种过滤方法 SECOND 速度较快；而在误差率比较大的情况下 $(d/p\ge 0.14)$，它的第三种过滤方法 THIRD 速度较快且比较实用，而 Epattern_searcherH 算法不再适用。

第五章　实验结果与分析

第四章已介绍了用于英文小说高频词查找的算法 Epattern_searcherH 和算法 Epattern_searcher，且对它们做了实验测试。它们都是基于过滤的思想的情况下的模式查找算法，且运用加强后缀数组索引数据结构来实现的，用于解决相同的问题——长篇英文小说中高频词的查找。因此，这两个算法之间是具有可比性。我们是在同一环境下实现算法，随机选取英文文本对它们进行了测试，通过多组实验表明，算法 Epattern_searcher 比 Epattern_searcherH 的执行时间快，具有较好的效果。

5.1　实验环境说明

为了对算法进行测试，根据实验室的条件和数据大小，我们选用了下列的平台作为测试环境：

(1) CPU：Intel Pentium4　4×2.0　GHz；

(2) 内存：512MB；

(3) 硬盘：80GB；

(4) 网卡：100MB/S；

(5) 操作系统：Microsoft Windows XP；

(6) 开发工具 Visual C++。

在上述平台上，我们实现了基于后缀数组构建的模式查找算法，且根据近似模式之间的距离标准不同进行相应的查找处理。然后，对相应算法的性能进行了测试和比较。

5.2　实验结果与分析

5.2.1　Epattern_searcherH 算法的实验测试

由于算法结果是受参数 p、r、d 和 k 取值的影响，一般是根据用户的需要，取合适的参数值。这里对算法 Epattern-searcherH 分别进行如下实验测试。

实验一：输入随机英文小说大小为 50000 byte，应用的参数为 $r=3$，$d=p/10$，

$k=5$，给出 p 的一些变化值，查找 $(p,d=p/10,3)\_reduplicate$。随着模式长度 p 的变化时间和误判率变化情况如图 5.2.1 所示。

实验二：输入随机英文小说大小为 10000 byte，应用的参数为 $p=100,k=5,d=3$，给出 r 变化值是从 2 到 10，查找 (100,d,3)_reduplicate。时间和误判率随参数 r 的变化情况，如图 5.2.2 所示。

实验三：输入随机英文小说大小为 50000 byte，应用的参数为 $r=3$， $p=100$ 给出 d 变化值是从 0 到 14，查找 (100,d,3)_reduplicate。随着参数 d 的变化，时间和误判率变化情况如图 5.2.3 所示。

实验四：输入随机英文小说大小为 50000 byte，应用的参数为 $r=3,p=100,d=10$，给出 k 变化值是 4、5、6，查找 (100,d,3)_reduplicate。时间、误判率和内存占用随参数 k 取值的相应变化情况，如图 5.2.4 所示。

	k=4	k=5	k=6
时间（秒）	25655	32	6
误判率（%）	0.12	0.15	0.20
内存（kb）	11524	11744	11160

图 5.2.4 取不同 k 值时，对应的速度和误判率变化情况

实验一结果：

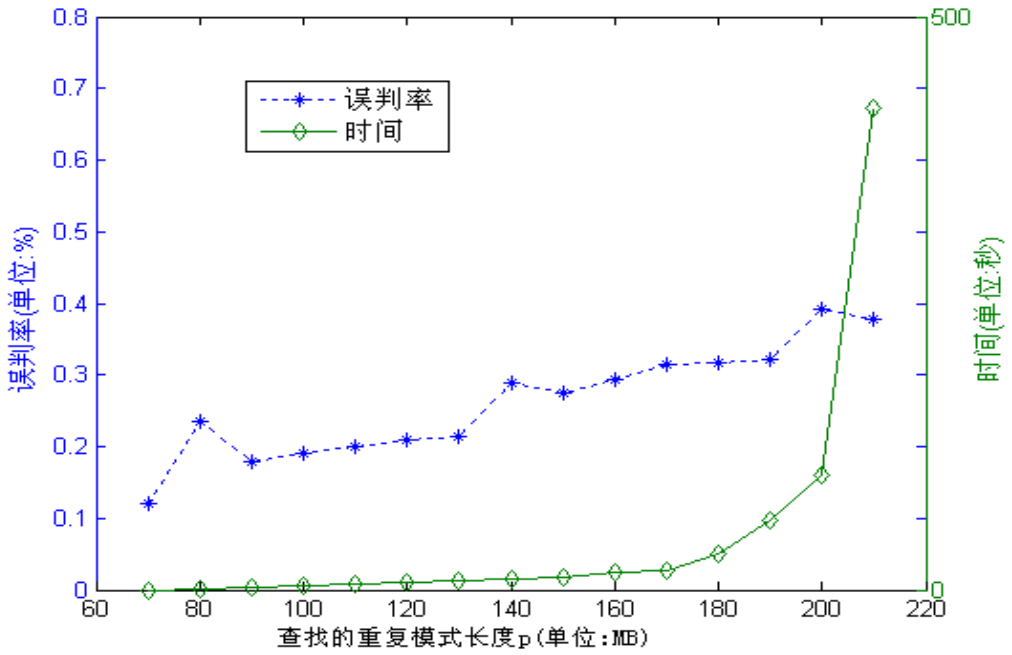
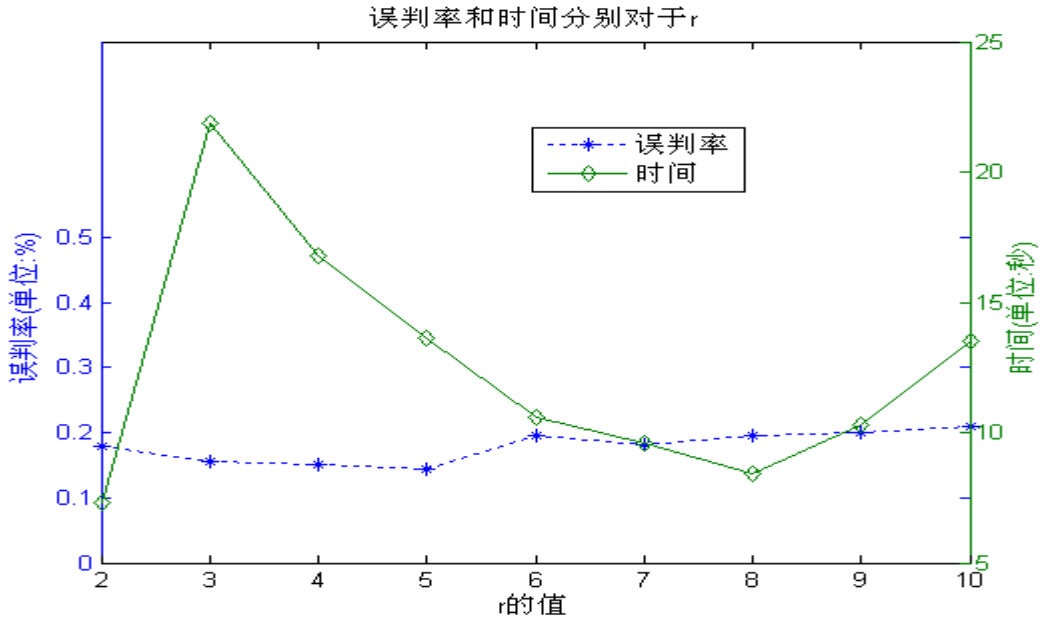


图 5.2.1 时间与误判率随参数 P 的变化情况

实验二结果：



5.2.2 时间与误判率随参数 r 的变化情况

实验三结果：

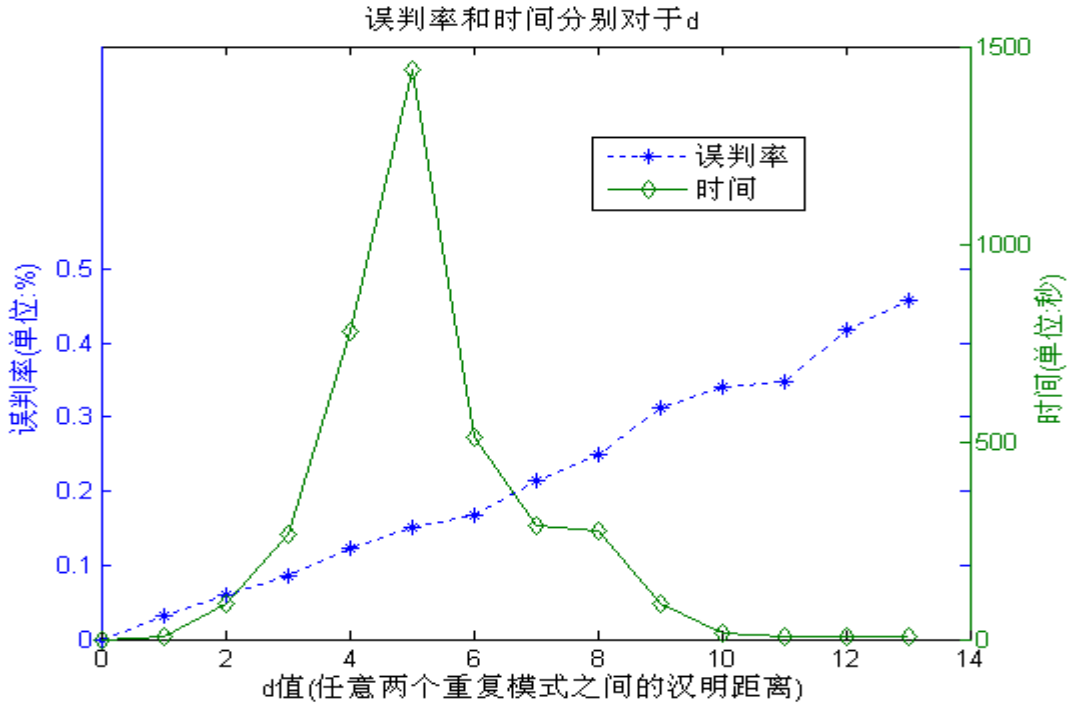


图 5.2.3 时间与误判率随参数 d 的变化

5.2.2 Epattern_searcher 算法的实验测试

实验一：随机输入英文大小为50Mb，取相应的参数 $p=260, d=13, g=16$，$r=280$ 随q-gram的长度q（q个X因子的长度）的变化，对于FIRST、SECOND、THIRD三种过滤方法所进行测试，其时间消耗和可选择性的变化如图5.2.5所示。

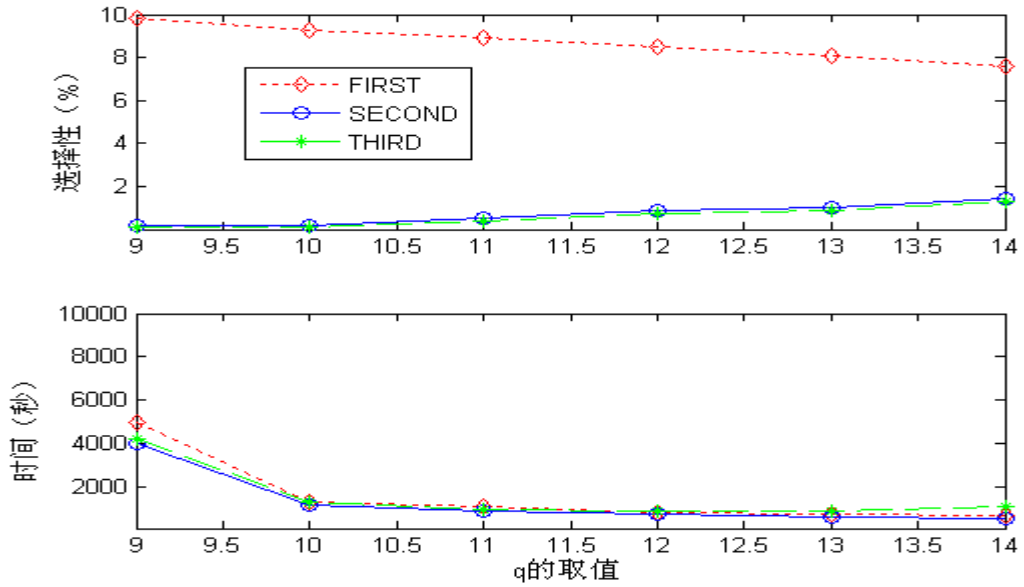


图 5.2.5 q-gram 的长度为 q(q 个 X 因子的长度)变化时，时间和选择性变化情况

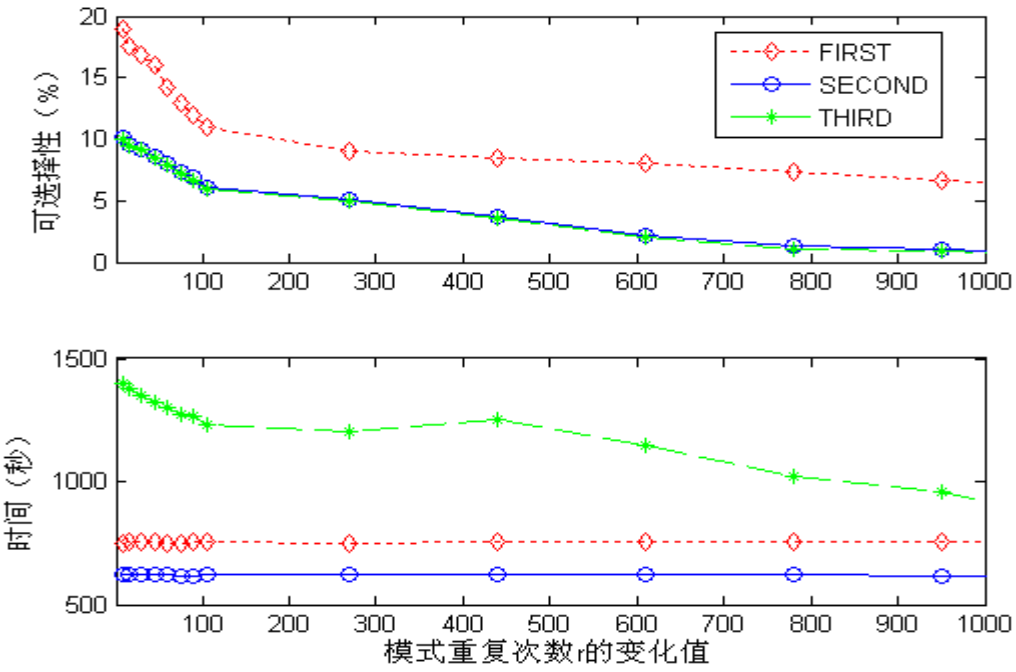


图 5.2.6 模式重复次数 r 变化时，时间和选择性的变化情况

实验二：随机输入英文字符串大小为50Mb，取相应的参数 $p=260,d=13,g=16$，$,q=14$ 随重复模式的次数r的变化，对于FIRST、SECOND、THIRD三种过滤方法所进行实验，其时间消耗和可选择性的变化情况如图5.2.6所示。

实验三： 随机输入英文小说大小为50Mb，取相应的参数 $p=260,d=13,g=16$，$q=14$ 随着编辑距离d的变化，对于FIRST、SECOND、THIRD三种过滤方法所进行测试，其时间消耗和可选择性的变化情况如图5.2.7所示。

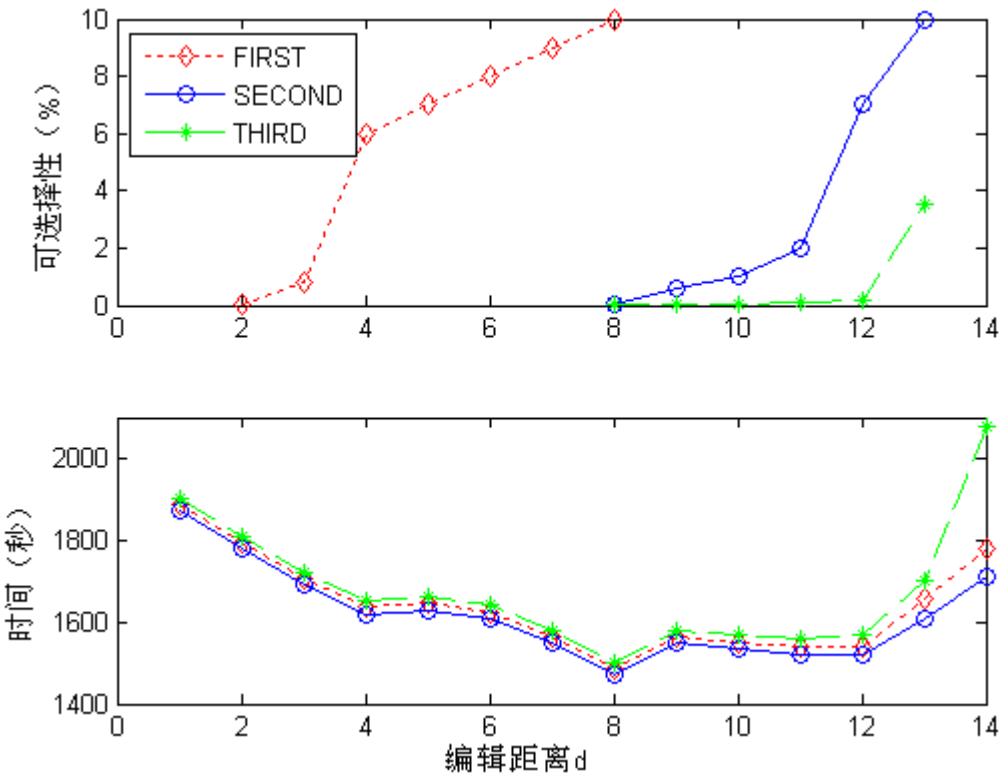


图 5.2.7 编辑距离 d 变化时，时间和可选性的变化情况

5.2.3 两种算法的实验结果比较

实验一：取参数 $p=100,k=6,d=7,r=3,m=8$，随输入数据长度的变化，其算法 Epattern-searcherH 的执行时间和内存占用情况如图 5.2.8 所示。

实验二：输入随机英文文本大小为 50000B，算法 Epattern-searcher 对参数 d、q 取不同的值如下面表格里的数据所示，对三种过滤方法 FIRST、SECOND、THIRD 的可选性的时间消耗情况做了比较。且对三种过滤算法两两进行比较。其测试结果如

图 5.2.9 所示。

实验三：算法 Epattern-searcherH（取参数 $p=100,k=6,d=8,r=23$ ）与算法 Epattern-searcher（取参数 $p=100,d=14,q=6,$，采用它的第三种过滤方法 THIRD），随输入文本 T 的长度变化时，两种算法运行时间变化情况如图 5.2.10 所示。

结果对比实验一：

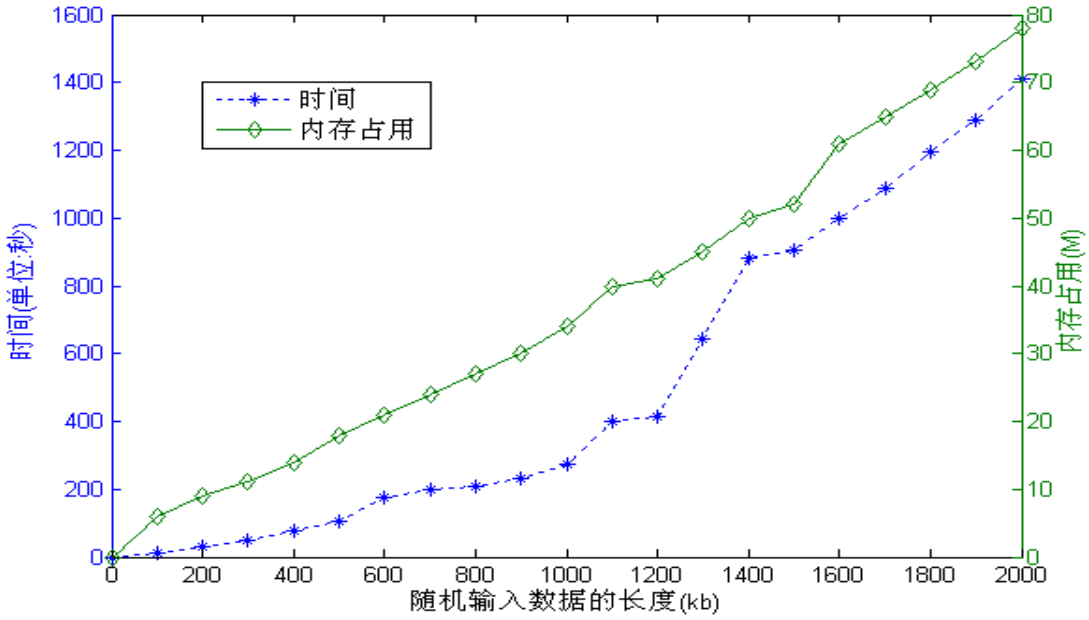


图 5.2.8 算法 Epattern_searcherH 的时间和空间占用情况

结果对比实验二：

d	q	t	可选性			运行时间			FIRST-SECOND		THIRD-SECOND		THIRD-FIRST	
			FIRST	SECOND	THIRD	FIRST	SECOND	THIRD	S-U	S-I	S-D	S-I	S-D	S-I
	8	13	13,84	5,10	1,36	123	116	120	1,06	2,71	1,13	3,76	1,07	10,21
	7	24	12,85	1,01	0,09	377	370	375	1,01	6,71	1,03	41,09	1,03	281,13
10	6	35	14,28	0,89	0,08	1286	1292	1241	0,99	16,01	1,04	81,23	1,05	1304,20
	5	46	21.91	0,65	0,02	5080	5138	5069	0,98	33,95	1,03	235,91	1,02	3011,80
	4	57	57,90	1,02	0,01	13441	13362	13280	1,01	56,77	1,02	391,15	1,01	8011,11
	7	10	50,91	24,51	13,52	405	382	401	1,06	2,07	1,22	18,02	1,16	380,01
	6	23	28,15	3,99	0,13	1274	1262	1255	1,01	7,04	1,04	30,42	1,05	214,01
12	5	38	34,82	2,30	0,04	4972	4952	4756	1,02	16,21	1,03	53,41	1,03	899,35
	4	49	85,12	3,53	0,03	13612	13610	13560	1,03	24,12	1,01	162,12	0,99	3916,11
	6	11	99,61	9,08	85,71	1609	1405	1305	1,15	1,02	1,54	1,12	1,34	1,17
14	5	26	75,21	12,51	0,35	5017	4794	4669	1,05	6,03	1,03	35,71	1,01	162,61
	4	41	99,21	25,13	0,08	14290	13949	13292	1,02	3,97	1,09	289,21	0,98	1192,34

图 5.2.9 输入文本大小为 50000B, (p,d,r)_repeat 是针对 d 和 q 取不同值时，三种过滤方法的时间和可选性变化情况, 用 S-U、S-D 和 S-I 分别表示可选性的增加、减少和改进

结果对比实验三：

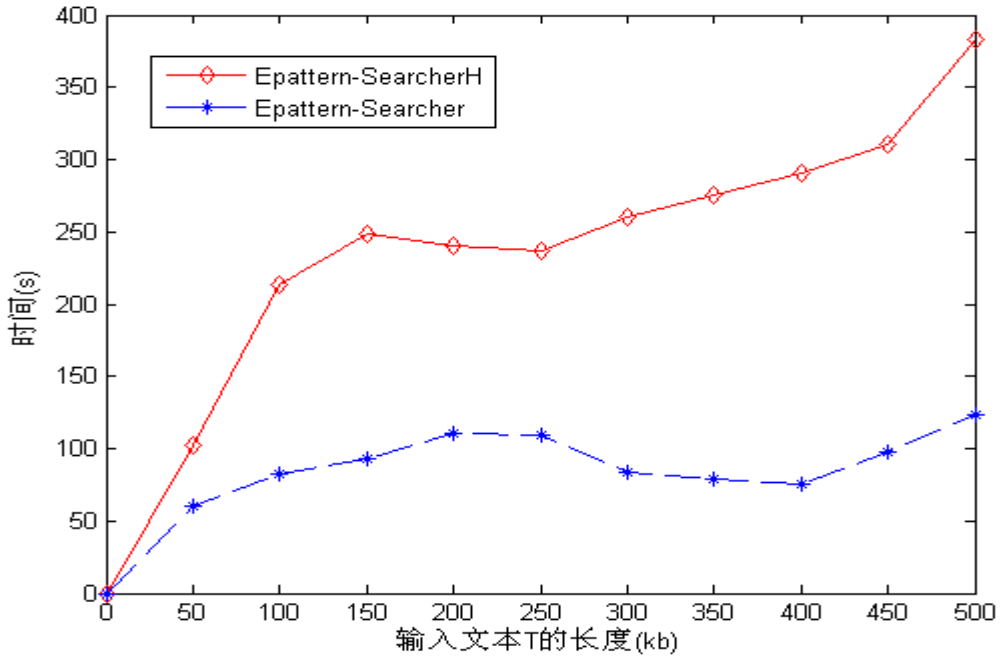


图 5.2.10 随输入文本 T 变化时，两种算法的运行时间变化情况

5.3 本章小结

算法 Epattern-searcherH 和 Epattern-searcher 都是基于过滤的思想，针对随机英文文本进行了测试。从测试结果的对比得知，Epattern-searcher 比算法 Epattern-searcherH 的运行消耗的时间少，从而验证 Epattern-searcher 算法较优。

第六章　总结与展望

6.1　总结

模式查找在文本编辑处理、文献检索、自然语言识别、入侵检测、病毒检测、信息过滤、计算生物学等领域中有着广泛地应用价值。我们是将重复模式查找算法应用在长篇英文小说阅读中，为能够快速地查找到文章中出现的高频词汇，以便较准确地把握全文大意，从而为英文语言学习者的阅读提供了必要的帮助。由于模式匹配是这些应用中最耗时的核心问题，要在海量数据中进行模式查找，按一般的查找方法，所花费的计算时间代价比较昂贵，而好的串匹配算法能显著地提高应用效率，因此，这里提出了模式查找的算法 Epattern_searcherH 和 Epattern_searcher，其运行效率相比一般的模式查找算法有所提高。

本文所做工作如下：

(1) 长篇英文小说可看成一个超长的字符串序列，要对它进行模式查找，采用一般的模式匹配算法，所需的时间和空间上消耗，使人难以忍受。这里应用增强后缀数组的数据结构来处理超大规模的字符串，使算法的运行效率和空间占用两方面都能得到良好的效果。

(2) 在模式查找算法中进行了过滤预处理，加快了算法运行效率。

6.2　不足与进一步的工作

本文是对大规模的字符串中进行重复模式查找方面做了初步的研究，但有许多问题需待进一步解决。未来的研究工作主要包括如下几个方面。

(1) 将匹配算法中的编辑距离变为旋转距离或者其他距离的应用是一个有待研究的问题。

(2) 算法的复杂度分析方面是有待研究的方向。

(3) 在字符串匹配算法设计中，重复模式查找的索引结构后缀数组有待进一步改进，寻找更好的改进策略以提高算法的匹配速度。

本文的研究内容的看法可能存在不足，恳请各位专家、老师、读者提出批评意见。

参考文献

[1] C. Jason Coit, Stuart Staniford, Joseph McAlerney. Towards Faster String Matching for Intrusion Detection or Exceeding the Speed of Snort. http://cnscenter.future.co.kr/resource/security/ids/speed_of_snort_03_16_2001.pdf.

[2] Mike Fisk, George Varghese. Applying Fast String Matching to Detection. http://public.lanl.gov/mfisk/papers/setmatch-raid.pdf.

[3] 夏亮,郑万波,王智.包过滤系统中关键字过滤的实现及其性能分析[J].吉林大学学报(信息科学版),2003,21(2):167-171.

[4] Lok-Lam Cheng,David W.Cheung,Siu-Ming Yiu.Approximate String Matching in DNA Sequences.http://www.cs.hku.hk/～dcheung/publication/dasfaa2003_2.pdf.

[5] 北京大学生物信息中心. http://www.cbi.pku.edu.cn/wnet/software2.html.

[6] YIN Zhong-hang, WANG Yong-cheng, CAI Wei, HAN Ke-song. Extracting Subject from Internet News by String Match. Journal of Software, 2002, 13(2): 159-167.

[7] D.E. Knuth, J.H. Morris, V.R. Pratt. Fast pattern matching in strings. SIAM Journal on Computing, 1977, 6(1): 323-350.

[8] R.S. Boyer, J.S. Moore. A fast string searching algorithm. Communications of the ACM, 1977, 20(10): 762-772.

[9] A. Hume, D.M. Sunday. Fast string searching. Software-Practice & Experience, 1991, 21(11): 1221-1248.

[10] Crochemore M., Czumaj A., Gasieniec L., Jarominek S., Lecroq T., P landowski W., Rytter W. Speeding Up Two String Matching Algorithms. Algorithmica, 1994, Vol 12, No.4-5: 247-267.

[11] R.N. Horspool. Practical fast searching in strings. Software Practice & Experience, 10(6): 501-506,1980.

[12] T.Raita. Tunning the Boyer-Moore-Horspool String Searching Algorithm.Software-Practice and Experience, 1992 2, 22(10): 879-884.

[13] Z.Galil. On improving the worst case running time of the Boyer-Moore string matching algorithm. Communication of the ACM, 1979, 22(9): 505-508.

[14] W.Rytter. A correct preprocessing algorithm for Boyer-Moore string-searching. SIAM J. Comput, 1980, 9(3): 509-512.

[15] R. Schaback. On the expected sublinearity of the Boyer-Moore algorithm. SIAM J. Comput, 1988, 17(4): 648-658.

[16] Karp R.M., Rabin M.O. Efficient randomized pattern-matching algorithms. IBM J. RES. DEVELOP, 1987, 31(2): 249-260.

[17] D.M. Sunday. A very fast s rithm. Communications of the ACM, 1990, 33(8): 132-142.

[18] A.V.Aho, M.J.Corasick. Efficient String Matching: An Aid to Bibliographic Search. Communication of the ACM, 1975, 18(6): 333-340.

[19] Jang-Jong Fan, Keh-Yih Su. An Efficient Algorithm for matching Multiple Patterns. IEEE Transactions on Knowledge and Data Engineering, 1993, 5(2): 339-351.

[20] Sun Wu, Udi Manber. A Fast Algorithm for Multi-pattern Searching[D]. http://webglimpse.net/pubs/TR94-17.pdf.

[21] Sun Wu, Udi Manber. Agrep-A Fast Approximate Pattern-matching Tool[M]. Usenix Winter Technical Conference, 1992: 153–162.

[22] Gonzalo Navarro,Mathieu Raffinot.Flexible Pattern Matching in Strings.Publishing House of Electronics Industry,2007:42－68.

[23] 范立新.用位并行法进行过滤的中文近似串匹配算法[D].浙江大学计算机科学与技术学院.2006.

[24] Gonzalo Navarro Mathieu Raffinot.Flexible Pattern Matching in Strings.北京电子工业出版社.2007.

[25] R.N. Horspool. Practical fast searching in strings. Software - Practice & Experience, 1980, 10(6): 501-506.

[26] Baeza-Yates, R.A.and Gonnet, G.H. A new approach to text searching. Communications of the ACM, 1992, 35(10): 74–82.

[27] Commentz-Walter. A string matching algorithm fast on the average. Proc. 6th International Colloquium on Auto mata, Languages, and Programming, 1979, 11-132.

[28] S. Wu, U. Manber. A fast algorithm for multi-pattern searching.Report TR-94-17,

Department of Computer Science,University of Arizona,Tucson,AZ,1994.

[29] C. Allauzen and M. Raffinot. Factor oracle of a set of words. Technical report 99-11,Institut Gaspard-Monge, Universite de Marne-la-Vallee,1999.

[30] Abouelhoda M.I.，Kurtz S.，Ohlebusch E.The enhanced suffix array and its applications to genome analysis[c]，Proc．Workshop on Algorithms in Bioinformatics,Lecture Notes in Computer Science，Springer-Verlag ,Berlin,2002,449-463.

[31] Weiner P. Linear Pattern Matching Algorithm[C],Proc.14th IEEE Symposium on Switching and Automata Theory,1973,1-11.

[32] Grossi R，Italiano G F.Suffix Trees and their Applications in String Algorithms[C]，Proc．Ist South American Workshop on String Processing(WSP 1993)，1993，57-76.

[33] McCreight E M．A Space-economical Suffix Tree Construction Algorithm[J]，J.ACM，1976，23：262-272.

[34] Ukkonen E．On-line Construction of Suffix Trees[J],Algorithmica,1995,14:249--260.

[35] Kurtz S．Reducing the Space Requirement of Suffix Trees[J]，Software Practice and Experience，1999，29：1149-1171.

[36] Manber U．Myers E W．Suffix Arrays:A New Method for Online String Searches [J],SIAM Journal on Computing,1993,22(5):935-948.

[37] Karkkainen J．Suffix cactus：a cross between suffix tree and suffix array [C]，Proc．of the Annual Symposium on Combinatorial Pattern Matching (CPM'95)，1995,191-204.

[38] Abouelhoda M.I.，Kurtz S.，Ohlebusch E．Replacing Suffix Trees with Enhanced Suffix Arrays [J],Journal of Discrete Algorithms,2004,2:53-86.

[39] Kasai T.，Lee G.，Arimura H.，Arikawa S.，Park K．Linear-time longest-common-prefix computation in suffix arrays and its applications[C]，Proc．Annual Symposium on Combinatorial Pattern Matching，Lecture Notes in Computer Science，Springer-Verlag，Berlin,2001，2089:181-192.

[40] Knight J.,Gusfield D.,Stoye J．The strmat software-package [EB/OL].
http: //www.CS.ucdavis/～gusfield/strmat.tar.Gz,1999.

[41] I.Levenshteim Binary codes capable of correcting deletions，insertions and reversals. DokladyAkademii Nank SSSRl63(4)p845—848,1965,also Soviet Physics DoklvAyl0(8)p 707-710,Feb 1966． Discovered the basic DM for edit distance.

[42] Burkhardt S, Crauser A, Ferragina P, Lenhof HP, Vingron M: q-Gram Based Database Searching Using a Suffix Array (QUASAR) [C],In RECOMB,Lyon,April 1999, 77-83.

[43] Ukkonen E: Approximate String-Matching with q–Grams and Maximal Matches. Theoretical Computer Science 1992, 92:191–211.

[44] Rasmussen K, Stoye J, Myers E: Efficient q-gram Filters for finding all epsilon-matches over a given length. Journal of Computational Biology 2006,13(2):296–308.

致谢

转眼间，研究生生涯即将结束。在这论文完成之际，我的心情万分激动。从论文选题，资料收集到论文撰写编排整个过程中，我得到了许多热情的帮助。

感谢我的导师赵学锋副教授，在本论文选题及研究过程中得到赵老师的悉心指导。

感谢计算机系张贵仓老师、冯百明老师、王彩芬老师、张明新老师、杨勇老师、马满福老师、蒋芸老师、王治和老师、党小超老师、任小康老师给予的指导、关心和帮助。在三年的研究生学习生涯中，各位老师给了我无价的知识和真诚的关爱，让我见识了众多研究领域的精华，在此表示衷心的感谢。

感谢在算法设计与分析专业研究生阶段一起渡过美好时光的师弟师妹们：杨海斌、王秀花、高红玉等的热心帮助，他们给了我很多重要的建议。

感谢同学陈文华、史变霞、郝瑞芝、李华以及其他同学帮助和鼓励，与她们在生活中友好相处，使我受益匪浅，在此致以诚挚的谢意。

最后，向我的父母、家人和所有朋友致谢，感谢他们对我理解、支持和帮助。

攻读硕士期间发表论文

[1] 张利香，赵学锋，王宇.电路布线问题的一种快速算法.光盘技术,2008.9(23): 31-32.

[2] 张利香，赵学锋.电路布线问题求解的改进算法.电脑知识与技术,2008.12(8):2205-2206.