块字符SSE指令在基因单模式串匹配里的应用

王力

2014年12月

中图分类号：TQ028.1

UDC分类号：540

块字符SSE指令在基因单模式串匹配里的应用

作 者 姓 名 王力

学 院 名 称 计算机学院

指 导 教 师 戴林教授

答辩委员会主席 \*\*教授

申 请 学 位 工学硕士

学 科 专 业 计算机科学与技术

学位授予单位 北京理工大学

论文答辩日期 2015年1月

**Synthesis and Application on textile of the Shape Memory Polyurethane**

Candidate Name： Wang Li

School or Department: \*\*\*\*

Faculty Mentor: Prof. \*\*

Chair, Thesis Committee： Prof. \*\*

Degree Applied: \*\*\*\*

Major： \*\*\*\*

Degree by: Beijing Institute of Technology

The Date of Defence： \*，\*\*\*\*

块字符SSE指令在基因单模式串匹配里的应用 北京理工大学

**研究成果声明**

本人郑重声明：所提交的学位论文是我本人在指导教师的指导下进行的研究工作获得的研究成果。尽我所知，文中除特别标注和致谢的地方外，学位论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得北京理工大学或其它教育机构的学位或证书所使用过的材料。与我一同工作的合作者对此研究工作所做的任何贡献均已在学位论文中作了明确的说明并表示了谢意。

特此申明。

签 名： 日期：

**关于学位论文使用权的说明**

本人完全了解北京理工大学有关保管、使用学位论文的规定，其中包括：①学校有权保管、并向有关部门送交学位论文的原件与复印件；②学校可以采用影印、缩印或其它复制手段复制并保存学位论文；③学校可允许学位论文被查阅或借阅；④学校可以学术交流为目的,复制赠送和交换学位论文；⑤学校可以公布学位论文的全部或部分内容（保密学位论文在解密后遵守此规定）。

签 名： 日期：

导师签名： 日期：

**摘要**

本文……。（摘要是一篇具有独立性和完整性的短文，应概括而扼要地反映出本论文的主要内容。包括研究目的、研究方法、研究结果和结论等，特别要突出研究结果和结论。中文摘要力求语言精炼准确，硕士学位论文摘要建议500~800字，博士学位论文建议1000~1200字。摘要中不可出现参考文献、图、表、化学结构式、非公知公用的符号和术语。英文摘要与中文摘要的内容应一致。）

**关键词：**形状记忆；聚氨酯；织物；合成；应用 **（**一般选3～8个单词或专业术语，且中英文关键词必须对应。）

**Abstract**

In order to exploit …….

**Key Words**: shape memory properties; polyurethane;textile;synthesis;application

目录

[第1章 绪论 1](#_Toc407094261)

[1.1 课题研究背景及意义 1](#_Toc407094262)

[1.1.1 课题的研究背景 1](#_Toc407094263)

[1.1.2 课题的研究意义 1](#_Toc407094264)

[1.2 针对生物学的字符串匹配算法的国内外研究现状及发展趋势 1](#_Toc407094265)

[1.2.1 相关研究工作 1](#_Toc407094266)

[1.2.2 存在的问题 2](#_Toc407094267)

[1.2.3 发展趋势 3](#_Toc407094268)

[1.3 本文的目标与组织结构 3](#_Toc407094269)

[第2章 精确字符串匹配算法综述 4](#_Toc407094270)

[2.1 基于前缀搜索的算法 5](#_Toc407094271)

[2.1.1 Knuth-Morris-Pratt算法 5](#_Toc407094272)

[2.1.2 Shift-And/Shift-Or算法 7](#_Toc407094273)

[2.2 基于后缀搜索的算法 8](#_Toc407094274)

[2.2.1 BM算法 8](#_Toc407094275)

[2.2.2 QS算法 11](#_Toc407094276)

[2.3 基于子串搜索的算法 11](#_Toc407094277)

[2.3.1 BOM算法 11](#_Toc407094278)

[2.3.2 SBDM算法和SBOM算法 12](#_Toc407094279)

[第3章 生物信息学基本概念 14](#_Toc407094280)

[3.1 核酸 14](#_Toc407094281)

[3.2 蛋白质 14](#_Toc407094282)

[3.3 高通量测序结果文件 15](#_Toc407094283)

[3.3.1 FASTA格式 15](#_Toc407094284)

[3.3.2 FASTQ格式 16](#_Toc407094285)

[3.3.3 数据样例 16](#_Toc407094286)

[第4章 SIMD指令集在串匹配算法中的应用 17](#_Toc407094287)

[4.1 Intel SSE指令介绍 17](#_Toc407094288)

[4.2 已有的应用 18](#_Toc407094289)

[4.2.1 位并行 19](#_Toc407094290)

[4.2.2 块字符 19](#_Toc407094291)

[4.2.3 Four-Russians 20](#_Toc407094292)

[第5章 改进算法IEPSM详述 21](#_Toc407094293)

[5.1 综述 21](#_Toc407094294)

[5.1.1 wscmp(a,b)字长比较指令 21](#_Toc407094295)

[5.1.2 wsmatch(a,b)字长匹配指令 21](#_Toc407094296)

[5.1.3 wsblend(a,b)字长翻转指令 22](#_Toc407094297)

[5.1.4 wscrc(a)字长CRC码计算指令 22](#_Toc407094298)

[5.2 模式串串长m大于等于16 22](#_Toc407094299)

[5.3 模式串串长m大于等于4且小于16 24](#_Toc407094300)

[5.4 模式串串长m等于3 24](#_Toc407094301)

[第6章 实验结果与分析 25](#_Toc407094302)

[6.1 测试算法选取 25](#_Toc407094303)

[6.2 测试集选取 25](#_Toc407094304)

[6.3 参数选取实验 25](#_Toc407094305)

[6.4 算法对比实验 27](#_Toc407094306)

[结论 30](#_Toc407094307)

[参考文献 31](#_Toc407094308)

[攻读学位期间发表论文与研究成果清单 33](#_Toc407094309)

[致谢 34](#_Toc407094310)

**第1章 绪论**

**1.1 课题研究背景及意义**

**1.1.1 课题的研究背景**

随着人类基因组计划的顺利实施和信息技术的迅速发展，GeneBank、EMBL和DDBJ国际三大核酸序列数据库数据量呈指数增长。生物学家、数学家和计算机科学家都面临着一个相同且严峻的问题，如何利用、表达这些数据进而分析与解释基因序列间的潜在关系，从中发掘出对人类有利的信息。为了迎接这一挑战，一门涉及生物、数学、物理、化学、计算机科学等诸多科学的新型学科应运而生，并且日益成为二十一世纪自然科学的核心领域之一。

数据库是信息处理的基础，而生物信息数据库则是一切生物信息学工作的出发点。目前已建立的数据库涵盖了生命科学各个领域，包括核酸序列数据库、蛋白质序列数据库、蛋白质片段数据库、蛋白质三维结构数据库、文献数据库等数据库。当今国际上三大主要的核苷酸、蛋白质数据库分别是：美国国家生物技术信息中（NCBI）的GenBank数据库（http://www.ncbi.nlm.nih.gov），欧洲生物信息学研究所（EBI）的核酸序列数据库EMBL（http://www.ebi.ac.uk/embl）和日本信息生物学中心（CIB）的DNA数据库DDBJ（<http://www.ddbj.nig.ac.uk/embl>）。据统计，生物学数据库现已达500个以上，仅仅与分子生物学相关的数据源就多达337个。每个数据源包含有大量的数据，如登记在GenBank数据库中的DNA序列总量已经达到32549400条，超过37893844733亿碱基对。

因此由于新一代的高通量测序技术产生测试数据的速度越来越快，产生了海量的生物数据信息，生物基因信息研究对字符串匹配算法的性能提出了更高的要求。

**1.1.2 课题的研究意义**

生物信息学的主要研究对象是DNA序列和蛋白质序列，字符串匹配计算又是该领域的基础。所以找出一个适用于基因序列且更有效率的算法显得十分必要。

**1.2 针对生物学的字符串匹配算法的国内外研究现状及发展趋势**

**1.2.1 相关研究工作**

[串匹配与序列查找并行算法研究序列部分]

自20世纪70年代以来，许多学者一直关注串匹配问题的研究，并不断取得一些新的理论或者实验性的成果，其中特别著名的精确串匹配算法主要有KMP算法、BM算法和KR随机算法等，后来大部分改进算法都是基于这些经典算法的。

【单模式字符串匹配算法效率的研究-国内外现状】

* KMP的改进算法

文献【2】将KMP算法的计算过程实现并行化，加快算法匹配流程、简化算法匹配步骤、提高了算法效率。

文献【3】针对文本串和模式串可能存在相同的重复字符的情况对KMP算法进行了改进，当发生不匹配时，不是将模式串的指针向左滑动，而是根据当前匹配字符的特征值计算窗口的滑动距离，将指针指向模式串首字符后开始匹配，提高了算法的性能。

文献【4】对模式串进行预处理，提出了一个计算距离的函数，每次发生不匹配后，使用函数对模式串两端求取移动距离，选择其中的较大值，继续下一轮匹配。

* BM的改进算法

文献【5】当文本串的字符是由字母表随机生成时，直接使用BMHS算法对匹配窗口进行移动，有时候效率反而会更低。因此，当使用BMHS算法得到移动距离后，将此数值和BMH算法的移动距离进行比较后再确定窗口最终的移动距离。这样可以综合两种算法的优势，提高匹配效率。

文献【7]不再使用坏字符或者好后缀规则，而是将这两个规则和它们的相邻关系判断匹配窗口的移动距离，即从后向前查找最大后缀的出现，并依据该后缀的偏移量来向后移动。

文献【8]通过在模式串中找到以不重复的字符结尾的最长前缀子串。使用此子串长度作为跳跃标准代替BM算法的好后缀和坏字符规则进行跳跃。当且仅当此子串成功匹配后才对模式串的其他字符继续比较；如果在此子串匹配过程中发现不匹配的字符，则将匹配窗口移动至少此子串长度的距离。即当不匹配发生在此子串的匹配过程中时，跳跃距离等于子串长度；当不匹配发生在子串之后的字符，则移动距离大于子串长度。将具体的移动距离通过函数表示，简化匹配的过程。

文献【9]对BM的改进类似于Sunday算法，它们之间的区别在于当Sunday算法发现不匹配时，匹配窗口的移动距离决定于位于窗口内文本子串尾字符的下一个字符。若此字符不属于模式串，则跳过此字符，将窗口移动至此字符的下一个字符的位置。因为当窗口内发生匹配失败时，此字符都必将参与下一次的匹配过程。而改进算法的第一个规则是当发生不匹配时，如果不匹配的字符不属于模式串，则跳过此字符，将窗口移动至此字符的下一个字符的位置；第二个规则是当发生不匹配时，且模式串成功匹配的子串内存在重复的字符，那么将模式串右移直到重复字符位置重合。

文献【10]将匹配窗口内的匹配顺序由从前向后修改为从两端到中间。具体的匹配规则如下：从模式串尾字符开始匹配，不成功则将匹配窗口右移，通过跳转函数计算移动距离；匹配成功则对首字符进行匹配，若首字符匹配不成功，则查找文本串中对应模式串尾字符的下一字符，若此字符不属于模式串，则模式串右移m+l；若首字符匹配成功，则继续匹配尾字符的前一字符按交替顺序向中间靠拢，直到全部字符匹配完成。

文献【12】KMP模式匹配算法在最差情况下的执行时间是线性的，并且和字母表大小以及模式串长度无关，BM算法具有接近最优的平均效率和最好效率，并且实际运行速度也非常快；设计一个包含了KMP和BM主体思想(受到Sunday算法的启发)，结合两者优点的混合算法。

Sunday的改进算法

文献【1 3】在每次匹配进行前，首先查看窗口内文本子串的尾字符是否属于模式串，如果不属于，则直接将匹配窗口右移m+l位，减少不必要的匹配过程，极大提升算法效率，即先判断再进行匹配。当文本串属于大字符集，而模式串属于小字符集时，文本串中不属于模式串的概率较高。此时改进算法可以减少许多算法的无效匹配步骤，提升算法效率。因为在实际使用中文本串的无效字符出现概率比率非常高，所以在实践中有很好的意义。

文献[14]首先试图在模式串P中找到一个最长的真首子串substring(substring<P)，把它作为新的模式串和正文T匹配时，当发生不匹配时，都能移动尽可能远的距离；当匹配时，然后逐个地匹配endstring(endstring=T-substring)；当endstring匹配失败时，再移动substring到endstring刚刚匹配失败的位置，继续匹配；当endstring也匹配成功时，则整个匹配过程完成，匹配成功。

文献[16】基于CPU进行一次字节长度的字符比较和进行一次机器字长长度的整数比较所花费的时间完全相同的事实，以及QS算法对当前尝试中比较顺序和匹配失败位置不关心的特点，将字符比较映射到整数域进行；由于比较次数被成倍减少，算法的平均复杂度被降低，效率相应得到提高。

文献【18]因为Sunday算法的无论是从前往后还是从后往前开始比较，若字符在模式串的出现是等概率的，那么就都会发生无效匹配的情况，占用了大量的时间。而改进算法将匹配窗口内的匹配顺序由从前向后修改为从两端到中间，按交替顺序向中间靠拢，直到全部字符匹配完成；同时记录下模式串字符出现的次数，在具体的匹配时从出现概率高的字符开始匹配，直到所有字符匹配完成。这样可以减少无效的移动和匹配次数。

文献【20]使用两个字符作为判断模式串是否匹配的跳转标准。具体匹配时这两个字符选取的是匹配窗口内模式串尾字符对应文本串子串的字符和此字符的下一个字符；当发生不匹配时，匹配窗口移动的距离是这两个字符中移动位数较大的那个。

文献【21]当发现不匹配时，匹配窗口的移动距离决定于位于窗口内文本子串尾字符的下一个字符。若此字符属于模式串，则将模式串中最右端的该字符与此字符对齐；若此字符不属于模式串，则跳过此字符，继续判断下个字符是否是否属于模式串，直到发现属于模式串的字符，才移动匹配窗口。

文献[23】通过将实际的字母表映射为较小体积的新字母表加速近似字符串匹配的效率；将这中方法应用到使用Four-Russian算法的近似BM算法修改版中；实验表明，缩小字母表加快了算法的运行速度，尤其是处理后不匹配的问题，当k比较小时，新算法比原算法在处理英文数据的速度要快。

文献[24】描述了一种提高Boyer．Moore．Horspool算法效率的方法，不再通过逐字符的方式比较模式串和文本串，而是通过比较机器字长的方式。能够明显的提高匹配效率，尤其是对长模式串效果更显著。

但专门针对生物序列的串匹配算法并不多，大部分算法还是通用的算法，最新的算法在向并行串匹配研究方向发展。

【基于连续r位匹配—研究动态】 。

如文献【33]提出了一种基于分组的串匹配算法。文中证明了最优分组定理，并提出了基于最短路径和动态规划的两种最优分组策略。分组算法适合模式串长度变化幅度非常大的情形，是解决大规模串匹配的一种有效方法。

文献【4】提出了一种是家复杂度最优的精确串匹配算法。该算法将文本分成个长度为2m-的相互重叠的窗口。在每个窗口内，综合使用后缀自动机DAWG和AC自动机进行扫描，保证了Ο(n)的最坏时间复杂度和的平均时间复杂度。

文献【5】使用Band-Row的方法来压缩SNORT中使用的AC算法，速度比预案算法提高了17%。这种表示法能够获取的状态结点转换速度，只需要两次额外的边界检查。但是当一行中的非空元素个数大于3时，空间压缩的效果不明显。此外，两次边界检查，对算法性能影响较大，使得算法速度的提高有限。

文献【6】使用位图压缩（Bitmap Compression）和路径压缩（Path Compression）两种方法来节省存储空间，以利于在硬件上实现。该文的实验结果表明，用该方法在ASIC等器件实现AC算法，速度有较大幅度的提高，但该方法并不适合于软件实现。

从硬件并行和体系结构方面来设计串匹配算法的专用芯片也是近年来的一个研究热点。文献【7】提到使用TCAM来进行串匹配算法的硬件并行化，获得GB级的匹配速度。文献【8】从体系结构方面对串匹配算法进行了研究，它将AC自动机进行分割，分配到并行的匹配引擎进行扫描，然后将匹配结果进行集成。这些方法都是利用硬件的高速和并行特性，以获取GB级的匹配速度。但是，专用硬件的存储空间有限，功能单一，软件上有效的并行算法很难应用到硬件上。

串匹配算法的另一发展方向是利用GPU的并行能力来处理。【基于GPU的串匹配算法】因为随着计算技术和集成电路技术的发展，图形硬件的更新速度也越来越快。GPU自1999年由nVidia公司提出以来，其发展速度是CPU的三倍多。多流水线结构、向量处理特性以及32位IEEE标准浮点精度的实现使得它对于计算密集型的科学应用有非常大的吸引力，越来越成为通用计算的一个有效的并行平台【l’2]。近年来很多的科学计算都已经被移植到了GPU这个平台上，诸如矩阵乘、3-sAT等经典的数值计算等。【基于GPU并行串匹配算法的研究】如CMATH算法是在GPU上使用CUDA的高效执行并行化的精确字符串匹配。

**1.2.2 存在的问题**

大部分算法还是沿用常用的字符串处理思路，逐个读入字符，未考虑到DNA字符集大小问题和DNA字符出现特征。因为基因数据的字典大小仅为4，所以如果按QS之类的算法每次只读下一个字符，就不能实现很好的跳转。另外尽管有些算法已经开始考虑利用CPU和GPU的硬件指令去加速算法，但这些改进很多只是将原来的经典算法换成硬件并行版本，并且有些改进算法也忽视了编程实现上的一点，大部分使用跳转表，通过读表得到不同值再去跳转，实验表明，常值跳转在程序实现上能达到极好的性能，以一个较小的常值跳转在多数情况下会比较大的变值跳转快。比如作者最早开始的算法改进是生成一个跳转数组，该数组能根据读入的串选择跳转距离，在多数情况下跳转距离都能达到模式串串长，极少几种情况会很短。但后来经实验发现，这种跳转距离的优化还没有一个安全的常值距离跳转来得快。

**1.2.3 发展趋势**

以后的串匹配算法会进行块字符组合，使用SIMD指令实现多字符一次性处理，并根据程序实际去设计算法。

## 1.3 本文的目标与组织结构

本文第二章根据分类描述了现有的一些单模式精确串匹配算法。在第三章和第四章分别介绍了核酸、蛋白质和SSE指令的相关知识。第五章详述新算法iepsm。第六章是新算法与其他算法的比较实验并给出了结论。

作者的主要工作是设计出一款针对生物基因序列特性的单模式串匹配算法，使其的运行效率最好。

**第2章 精确字符串匹配算法综述**

一个字符串是一个定义在有限字符表Σ上的字符序列。例如，GTACTGGTAATG是字符表Σ={A,C,G,T}上的一个字符序列。单模式精确串匹配问题即在长串T中检索出给定字符串p的所有出现位置。其中，T称为文本，p称为模式串，T和p都定义在同一个字母表Σ上。给定字符串x，y和z，称x是xy的一个前缀，x是yx的一个后缀，x是xyz的一个因子。

根据在文本中搜索模式串方式的不同，字符串匹配算法可以归结为三种基本的方法。

第一种方法就是从文本中逐个读入字符，每读入一个字符就更新相应变量，检查是否存在一个可能的匹配。KMP算法就属于这种方法，另外一种是更快的Shift-Or算法，对其进行扩展后可以支持对更复杂模式串的搜索。

第二种方法基于滑动窗口。滑动窗口沿着文本T滑动，对应任意位置上的窗口，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。Boyer-Moore算法就使用了这种方法。但在一般情况下，Boyer-Moore算法比它的一个简化版本Horspool要慢。实际上，Boyer-Moore系列算法中除了Horspool，都要比使用其他方法的一些算法慢。

第三种方法出现较晚，实际应用中如果模式串p足够长，其中一些算法是最有效率的。和第二种方法一样，第三种方法也使用了滑动窗口，并在其中从后向前搜索。不同的是，它搜索的是窗口中文本的最长后缀，并且这个最长后缀同时也是模式串p的一个因子。最早使用这种方法的算法是BDM算法，当p足够短时，可以将其改造成更简单、更有效的算法BNDM。对于较长的模式串，一个称为BOM的新算法是最快的。

在所有的单模式串匹配算法中，都有一个搜索窗口，其长度与模式串长度相等。搜索窗口沿着文本从左向右滑动，搜索模式串的过程在窗口内进行。

因此根据在文本中搜索方式的不同，我们可以将单模式串匹配算法归结为三种基本的搜索方法，并简要介绍对应的算法。

前缀搜索在搜索窗口内从前向后（沿着文本的正向）这个读入文本字符，搜索窗口中文本和模式串的最长公共前缀。

后缀搜索在搜索窗口从后向前（沿着文本的反向）逐个读入文本字符，搜索窗口中文本和模式串的最长公共后缀。使用这种搜索方法的算法可以跳过一些文本字符，从而具有亚线性的平均时间复杂度。

子串搜索在搜索窗口内从后向前逐个读入文本字符，搜索满足如下条件的最长字符串u：u既是窗口中文本的后缀，也是模式串的子串。与后缀搜索方法一样，使用这种搜索方法的算法也具有亚线性的平均时间复杂度，甚至能达到最优的时间复杂度。

## 2.1 基于前缀搜索的算法

即在搜索窗口内从前向后(沿着文本的方向)逐个读入文本字符，搜索窗口中文本和模式串的最长公共前缀。应用此种搜索方法能跳过部分字符，从而获取亚线性时间复杂度。

### 2.1.1 Knuth-Morris-Pratt算法

Knuth-Morris-Pratt算法【章20】（简称KMP算法）是由D.E.Knuth、J.H.Morris和V.R.Pratt共同设计出的极为高效的精确单模式串匹配算法，其本质是前缀匹配算法，对比前缀蛮力匹配算法，区别在于它会动态调整每次模式串的移动距离，而不仅仅是加一，从而加快匹配过程。下图通过一个直观的例子展示前缀蛮力匹配算法和KMP算法的区别，前文提过，这二者唯一的不同在于模式串移动距离。



图1.1 KMP算法模式串移动示意图

上图中，前缀蛮力匹配算法发现匹配不上，就向右移动距离1，而KMP算法根据已经比较过的前缀信息，了解到应该移动距离为2，换句话说针对母串的下一个匹配字符，KMP算法了解它下回应该匹配模式串的哪个位置，比如上图中，针对母串的第i+1个字符，KMP算法了解它应该匹配模式串的第k+1个字符。为什么会是这样，这是因为母串的子串T[i-k, i]=aba，而模式串的子串P[0,k]=aba，这二者正好相等。所以模式串应该移动到这个位置，从而让母串的第i+1个字符和模式串的第k+1个字符继续比较。

另外注意上图中，模式串位置j已经匹配上母串的位置i，也就是T[i-k, i] = P[j-k, j] = aba。而根据前文的T[i-k, i] = P[0, k] = aba，从而得出P[0, k] = P[j-k, j] = aba。通过观察发现，就是在模式的子串[0, j]中寻找一个最长前缀[0,k]，从而使得[j-k, j] = [0,k]。

于是可以定义一个jump数组，jump[j] = k，表示满足P[0, k] == P[j-k, j] 的最大k值，或者表述为：如果模式串j+1匹配不上母串的i+1,那跳转到模式串k+1继续比较。有了这个jump数组，易知KMP算法如下。

表2.1 KMP算法

|  |  |
| --- | --- |
| **KMP算法** | |
| 01  02  03  04  05  06  07  08 | J ← 0  For i ← 1 to n do  While (j > 0) and (P[j+1]!=T[i])  Do j ← jump[j]  If P[j+1] = T[i]  J ← j+1  If j == m  Report pattern occur, i-m |

KMP算法中jump数组的构建可以通过归纳法来解决，首先确定jump[1] = 0，假设jump[j]=k，也就是P[0, k] == P[j-k, k]，如果P[j+1] == P[k+1]，那么得出[0,k+1] = P[j-k, j+1]，从而更加定义得出jump[j+1] = k+1；

如果P[j+1] != P[k+1]，那就接着比较P[j+1] ?= P[k1+1]，其中（jump[k] = k1），根据（jump[k] = k1）的定义，P[0,k1] == P[k-k1, k]，根据（jump[j] = k）的定义，P[0, k] == P[j-k, k]，根据这两个等式，推出P[0, k1] == P[j-k1, j]，如果此时P[j+1] == P[k1+1]，则得出：jump[j+1] = K1 +1 = jump[k] +1。

如果P[j+1] != P[K1+1]，继续递归比较P[j+1]和P[jump[jump[k]]+1]至P[1]；

如果依次比较都不相等，那么jump[j+1] = 0。写成伪代码如下，可以看出其实就是模式串自我匹配的过程。

表2.1 Jump数组

|  |  |
| --- | --- |
| **Jump数组** | |
| 01  02  03  04  05  06  07 | Jump[1] ← 0; j ← 0;  For i ← 2 to m do  While (j > 0) and (P[j+1]!=T[i])  Do j ← jump[j]  If P[j+1] = T[i]  J ← j+1  Jump[i] ← j |

考虑模式串匹配不上母串的最坏情况，前缀蛮力匹配算法的时间复杂度最差是O(n×m)，最好是O(n),其中n为母串的长度，m为模式串的长度。KMP算法最差的时间复杂度是O(n)；最好的时间复杂度是O(n/m)。

### 2.1.2 Shift-And/Shift-Or算法

穷举算法的效率比较低，几乎文本串的每个字符在匹配过程中都进行了m(模式串长度)次比较，

二十世纪90年代初，在Baez-aYates的博士论文11”[1]中最早出现了采用位并行方法(Bit一Parallelism)进行字符串匹配的思想，而后出现了经典的shift-or算法，以及又在此基础上进行了改进和提高的shift-and算法[2]，shift-and算法又称为BAP(Bit一ParallelAutomaton)算法。

位并行算法用来模拟经典算法，是一种加速算法实现的手段。在搜索中，通过并行模拟，可以加快经典算法的运行速度。位并行算法非常适合模式串比较短的情况。

位并行利用了计算机机器字位运算的内在并行性，可以把多个值装入同一个长度为ω的机器字内，然后只需一次运算就能更新所有值。利用位并行，一个算法所执行的运算次数最多能减少到原来的1/ω，这里ω是机器字的位数。

鉴于64位电脑已经成为主流，而每位可以存储一个字符比较的结果，因此可以在一次匹配时同时对64个字符进行比较。最终可以将穷举算法提高64倍的匹配速度。

Shift-And算法思想：设模式字符串为P，文本为text。核心是维护一个记录了最长前缀串的字符串集合D。D中的元素满足既是移动窗口内文本串子串的后缀，又是模式串前缀的条件。窗口每向后移动一次，集合D就会被位并行机制更新一次。

Shift-And算法思想：设模式字符串为P，文本为text。它主要通过维护一个字符串集合D（D中记录了P中所有与当前已读text的某个后缀相匹配的前缀），集合D中的每个字符串既是模式串p的前缀，同时也是已读入文本的后缀，每当从text中读入一个新的字符，算法立即利用位并行机制来更新集合D。[3]

我们可以这么具体理解。

* 设P长度为m，则集合D可表示为D = dm…d1 而用D[j]代表dj；
* D[j]=1，当且仅当p1…pj 是 t1…ti 的某个后缀；
* 当D[m]=1时，就认为P已经于text匹配；
* 当读入下一个字符 ti+1, 需要计算新的集合 D′；

当且仅当D[j]=1并且 ti+1等于pj+1时D'[j+1]=1。这是因为D[j]=1时有 p1…pj是 t1…ti 的一个后缀，而当ti+1 等于 pj+1可推出p1…pj +1是 t1…ti+1 的一个后缀。这个集合可通过位运算来更新。

Shift-and算法首先建立一个数组B, 数组长度为字符集长度（例如A-Z的话数组B的长度为26），如果P的第j位等于c(P[j]=c)，则将B[c]中第j位置为1，否则为0。

首先D=0m，对于每个新读入的文本字符ti+1，可以用如下公式对D进行更新：

公式：D ((D << 1) | 0m-11) & B[ti+1]

直观上来讲，左移位操作<<将D’的第i+1位的值置为D的第i位。因为空字符串也是文本的后缀，所以D<<1需要在最低位与0m-11进行位或操作。因为要找到那些满足ti+1=pm+1的位置，所以需要再讲上面的结果与B[ti+1]进行位与操作。

当模式串的长度不超过几个机器字长时，公式(1)中的操作能够在常数时间内完成，这是shift-and算法的时间复杂度为O(n)。

## 2.2 基于后缀搜索的算法

基于后缀搜索的算法的难道在于不漏扫的情况下，寻找合适的窗口移动方法。

### 2.2.1 BM算法

Boyer-Moore为了做到这点定义了两个规则：坏字符规则和好后缀规则。

* 坏字符规则

一是如果坏字符没有出现在模式字符中，则直接将模式串移动到坏字符的下一个字符。如图所示，坏字符c，没有出现模式串P中，直接将P移动c的下一个位置。



图1.1 KMP算法模式串移动示意图

二是如果坏字符出现在模式串中，则将模式串最靠近好后缀的坏字符与母串的坏字符对齐。如下图所示。

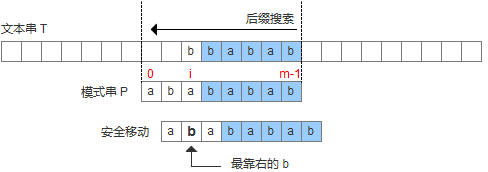


图1.1 KMP算法模式串移动示意图

* 好字符规则

一是模式串中有子串匹配上好后缀，此时移动模式串，让该子串和好后缀对齐即可，如果超过一个子串匹配上好后缀，则选择最靠靠近好后缀的子串对齐。

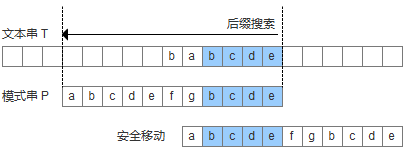


图1.1 KMP算法模式串移动示意图

二是模式串中没有子串匹配上后后缀，此时需要寻找模式串的一个最长前缀，并让该前缀等于好后缀的后缀，寻找到该前缀后，让该前缀和好后缀对齐即可。

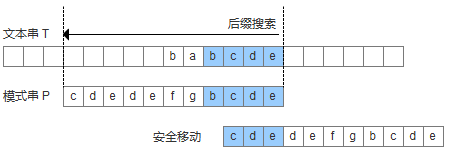


图1.1 KMP算法模式串移动示意图

三是模式串中没有子串匹配上后后缀，并且在模式串中找不到最长前缀，让该前缀等于好后缀的后缀。此时，直接移动模式到好后缀的下一个字符。

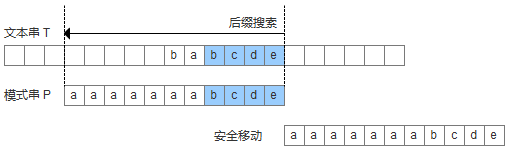


图1.1 KMP算法模式串移动示意图

BM算法的算法描述可以查询相应文档，这里不再给出。

BM算法是实际应用中最有效、最常用的单模式精确串匹配算法之一，它采用从后往前的扫描顺序，并且使用好字符表和坏字符表来获去最大移动距离进行跳跃。BM算法的空间复杂度为，其预处理阶段的时间复杂度为，最好情形下BM算法的时间复杂度可达

### 2.2.2 QS算法

Sunday算法是Daniel M.Sunday在1990年提出的有一个BM的简化算法，它将Horspool算法进一步优化，因此又称为BMHS算法。

Sunday算法与BM算法的核心相同，但是匹配效率更高。它的主要思想是：当发现不匹配时，匹配窗口的移动距离决定于位于窗口内文本子串尾字符的下一个字符。若此字符属于模式串，则将模式串中最右端的该字符与此字符对齐；若此字符不属于模式串，则跳过此字符，将窗口移动至此字符的下一个字符的位置。因为当窗口内发生匹配失败时，此字符都必将参与下一次的匹配过程。这个算法的关键点：

* 为pattern串建立hash表，以提高查找字符的速度。
* 对齐跳转，快速的后移比较，使比较次数减少。

Sunday算法在最坏情况下的计算复杂度为Ο(m\*n)，但由于该算法只根据窗口内文本的最后一个字符进行跳转，所以对于字符表大的字符串会有很好的效果。

## 2.3 基于子串搜索的算法

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。

【高性能精确单模式串匹配-增加BNDM】

【快速精确字符串匹配-子串搜素】

### 2.3.1 BOM算法

BOM 算法使用确定性的有向无环自动机来识别子串，它在窗口文本的所有后缀中寻找能够和模式串的子串匹配的最长符号串，而不是按照固定顺序逆向匹配模式串。如果发现模式串与当前窗口文本相等，则报告成功匹配；否则根据最长匹配的情况移动窗口，在下一个可能匹配的位置进行匹配。

为了实现Factor Oracle结构，最简单的方法是使用一个（m+1）\*A的表，其中A是模式串的字母表大小。这种表示的优点是能够在的时间进行状态间的跳转，从而加快搜索过程。然后对于很长的模式串，则需要考虑采用的空间来实现。

BOM算法具有最坏时间复杂度，平均时间复杂度。

### 2.3.2 SBDM算法和SBOM算法

SBDM 算法使用后缀自动机在长度为 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 min 的前缀的反转串，上，构造时间复杂度为Ο(r× min)。搜索在大小为 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lmin的前缀的子串。搜索过程可能出现下面两种情况。

* 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。
* 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Ο(n×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长。这是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算法使用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Ο(n×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。

**第3章 生物信息学基本概念**

生物信息学的主要任务是分析、处理和研究DNA序列（蛋白质序列）数据中所包含的各种生物学信息。下面简要介绍这两种序列和高通量测序结果文件格式。

## 3.1 核酸

核酸是由许多核苷酸聚合成的生物大分子化合物，为生命的最基本物质之一。核酸广泛存在于所有动植物细胞、微生物体内，生物体内的核酸常与蛋白质结合形成核蛋白。不同的核酸，其化学组成、核苷酸排列顺序等不同。根据化学组成不同，核酸可分为核糖核酸（简称RNA）和脱氧核糖核酸（简称DNA）。DNA是储存、复制和传递遗传信息的主要物质基础。RNA在蛋白质合成过程中起着重要作用——其中转运核糖核酸，简称tRNA，起着携带和转移活化氨基酸的作用；信使核糖核酸，简称mRNA，是合成蛋白质的模板；核糖体的核糖核酸，简称rRNA，是细胞合成蛋白质的主要场所。

核酸是一种一维高分子链，链中包含4种单体，每个单体叫做核苷酸。核酸中携带着遗传信息，遗传信息主要表现在核苷酸的排列次序上。根据核苷酸类型的不同，核酸分为脱氧核糖核酸（DNA）和核糖核酸（RNA）。核苷酸由磷酸、脱氧核糖或核糖和碱基组成。构成核苷酸的碱基分为嘌呤和嘧啶两大类。前者主要指腺嘌呤（adenine，A）和鸟嘌呤（guanine，G），DNA和RNA中均含有这两种碱基。后者主要指胞嘧啶（cytosine，C）、胸腺嘧啶（thymine，T）和尿嘧啶（uracil，U），胞嘧啶存在于DNA和RNA中，胸腺嘧啶只存在于DNA中，尿嘧啶则只存在于RNA中。其中，DNA是存储、复制和传递遗传信息的主要物质基础，RNA在蛋白质合成过程中起着重要作用。

## 3.2 蛋白质

蛋白质是构成生物体的直接元素，使用不同的蛋白质构造出了不同的生物体，是生命活动的主要承担者。没有蛋白质就没有生命。氨基酸是蛋白质的基本组成单位。它是与生命及与各种形式的生命活动紧密联系在一起的物质。机体中的每一个细胞和所有重要组成部分都有蛋白质参与。蛋白质占人体重量的16%~20%，即一个60kg重的成年人其体内约有蛋白质9.6~12kg。人体内蛋白质的种类很多，性质、功能各异。这些蛋白质都是由二十种氨基酸通过肽键连接而成，并在体内不断进行代谢与更新。这二十种氨基酸是蛋白质的基本单位，赋予蛋白质特定的分子结构形态。表2.1列出了这二十氨基酸的名称、字母表示和符号。

表2.1 蛋白质的氨基酸名称及字母符号

|  |  |  |
| --- | --- | --- |
| **名称** | **三字符号** | **单字符号** |
| 丙氨酸 | Ala | A |
| 精氨酸 | Arg | R |
| 天冬氨酸 | Asp | D |
| 半胱氨酸 | Cys | C |
| 谷氨酰胺 | Gln | Q |
| 谷氨酸 | Glu | E |
| 组氨酸 | His | H |
| 异亮氨酸 | Ile | I |
| 甘氨酸 | Gly | G |
| 天冬酰胺 | Asn | N |
| 亮氨酸 | Leu | L |
| 赖氨酸 | Lys | K |
| 甲硫氨酸 | Met | M |
| 苯丙氨酸 | Phe | F |
| 脯氨酸 | Pro | P |
| 丝氨酸 | Ser | S |
| 苏氨酸 | Thr | T |
| 色氨酸 | Trp | W |
| 酪氨酸 | Tyr | Y |
| 缬氨酸 | Val | V |

## 3.3 高通量测序结果文件

### 3.3.1 FASTA格式

[百度百科]

在生物信息学中，FASTA格式（又称为Pearson格式），是一种基于文本用于表示核苷酸序列或氨基酸序列的格式。在这种格式中碱基对或氨基酸用单个字母来编码，且允许在序列前添加序列名及注释。

序列文件的第一行是由大于号">"或分号";"打头的任意文字说明（习惯常用">"作为起始），用于序列标记。从第二行开始为序列本身，只允许使用既定的核苷酸或氨基酸编码符号。通常核苷酸符号大小写均可，而氨基酸常用大写字母。使用时应注意有些程序对大小写有明确要求。文件每行的字母一般不应超过80个字符。

对于核酸序列，除了为大家所熟知的A、C、G、T、U外，R代表G或A（嘌呤）；Y代表T或C（嘧啶）；K代表G或T（带酮基）；M代表A或C（带氨基）；S代表G 或C（强）；W代表A或T（弱）；B代表G、T或C；D代表G、A或T；H代表A、C或T；V代表G、C或A；N代表A、G、C、T中任意一种[1]。

该种格式数据库被用来存放原始数据，以及一系列附加的注释。不同的检索工具和程序利用了这些信息中的不同部分。纵观各种格式，我们可以发现其中应用了一些共同的规则，以使得多种情况下在不同格式之间生成和交换数据成为可能。最便于人阅读的格式对计算机程序来讲很可能并非是最有效率的（例如GenBank flatfile，这是一种人可以阅读的ASN.1版本）。这些记录还有二进代码版，更加紧凑，计算机处理也更快。但不幸的是，由于历史的原因，对一种固定格式的频繁使用使得引入另一种格式极为困难，尽管新的格式可能更加富含信息，更加准确，易于复制和计算，易于抽取信息，易于使用。GBFF的简单性，使我们都可以获得易用的工具，这也是EMBL和GBFF极大通用性的重要原因。

### 3.3.2 FASTQ格式

FASTQ格式是序列格式中常见的一种，它是存储了生物序列（通常是核酸序列）以及相应的质量评价的文本格式。

它们都是以ASCII编码的。现在几乎是高通量测序的标准格式。NCBI Short Read Archive也是这格式，多了一些描述性词汇而已。

FASTQ格式的序列一般都包含有四行，第一行由'@'开始，后面跟着序列的描述信息，这点跟FASTA格式是一样的。第二行是序列。第三行由'+'开始，后面也可以跟着序列的描述信息。第四行是第二行序列的质量评价（quality values，注：应该是测序的质量评价），字符数跟第二行的序列是相等的。

### 3.3.3 数据样例

一个FASTQ数据的典型格式如下。

@SEQ\_ID

GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT

+

!''\*((((\*\*\*+))%%%++)(%%%%).1\*\*\*-+\*''))\*\*55CCF>>>>>>CCCCCCC65

其中第一行由@开头表示数据的描述信息，第二行是数据序列本身，第三行为+号，后面也可以跟随描述信息，第四行为数据质量描述。

而一个典型的FASTA数据格式如下。

>SEQ\_ID

GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT

很明显，描述信息是以>开头。没有数据质量信息。

**第4章 SIMD指令集在串匹配算法中的应用**

## 4.1 Intel SSE指令集介绍

单指令流多数据流（英语：Single Instruction Multiple Data，缩写：SIMD）是一种采用一个控制器来控制多个处理器，同时对一组数据（又称“数据矢量”）中的每一个分别执行相同的操作从而实现空间上的并行性的技术。

在微处理器中，单指令流多数据流技术则是一个控制器控制多个平行的处理微元，例如Intel的MMX或SSE，以及AMD的3D Now!指令集。

图形处理器（GPU）拥有强大的并发处理能力和可编程流水线，面对单指令流多数据流时，运算能力远超传统CPU。OpenCL和CUDA分别是目前最广泛使用的开源和专利通用图形处理器（GPGPU）运算语言。

本文将应用Intel SSE指令集。

SSE(Streaming SIMD Extensions)是英特尔在AMD的3D Now!发布一年之后，在其计算机芯片Pentium III中引入的指令集，是继MMX的扩充指令集。SSE 指令集提供了 70 条新指令。AMD后来在Athlon XP中加入了对这个新指令集的支持。

该指令集主要是为提高流媒体处理性能而推出的, 包括SSE、SSE2、SSE3、SSE4，目前已更新到SSE4.2指令集，这些指令集并不能完全说是一个独立的新技术，可以说是处理器众多新技术之一【基于SSE4指令集的H.264】，它将多批次的指令组编辑成为一条单一的指令，从而达到提升数据处理的能力。集成。SSE系列指令集定义了8个全新的128位寄存器，支持在128位数据的并行操作。从理论上来说，SSE指令的处理速度可以达到普通指令的4倍。如今，SSE指令集在多媒体处理领域已经得到广泛应用，如图像处理、视频处理、音频处理等。在信息加密和科学计算方面，SSE指令集也在发挥着作用。鉴于SSE指令集在数据并行处理上的高效性能，我们将SSE指令应用到串匹配领域，利用SSE指令集提供的大位宽寄存器对串匹配算法进行优化。

该指令集的扩展版本简介如下。

SSE2是Intel在Pentium 4处理器的最初版本中引入的，但是AMD后来在Opteron 和Athlon 64处理器中也加入了SSE2的支持。SSE2指令集添加了对64位双精度浮点数的支持，以及对整型数据的支持，也就是说这个指令集中所有的MMX指令都是多余的了，除了主要负责64位双精度浮点数及整型运算符和对Cache控制延迟降低两部分之外，更重要的是避免了占用浮点数寄存器。这个指令集还增加了对CPU缓存的控制指令。AMD对它的扩展增加了8个XMM寄存器，但是需要切换到64位模式（x86-64/AMD64）才可以使用这些寄存器。Intel后来在其Intel 64架构中也增加了对x86-64的支持。

2004年，以Prescott为核心的英特尔奔腾4处理器加入了SSE3指令集。该指令集是Intel引入的第三代SIMD指令集，AMD在Athlon 64的第五个版本，Venice核心中也加入了SSE3的支持。新指令集仅13条，包含寄存器的局部位之间的运算，例如高位和低位之间的加减运算，浮点数到整数的转换。主要是对水平式暂存器整数的运算符，可对多笔数值同时进行加法或减法运算，令处理器能大量执行DSP及3D性质的运算。此外，SSE3更针对多线程应用进行最佳化，使处理器原有的Hyper-Threading功能获得更佳的发挥。

作为SSE3指令集的额外扩充版本，2006年SSSE3出现在人们已经相对比较熟悉的酷睿位架构处理器中，新增有32条指令，进一步增强CPU在多媒体、图形图像和Internet等方面的处理能力。而英特尔方面本来是计划将该32调指令收录到后来的SSE4指令集中，但考虑到当时硬件升级速度的大幅提升，最终决定提早加入至酷睿微架构产品中。

2007年11月，随着penryn(酷睿双核)处理器到来的SSE4，被视为继2001年以来最重要的多媒体指令集架构的改进，除扩展Intel 64指令集架构外，还加入有关图形、视频编码及处理、三维成像及游戏应用等指令，指令涉及音频、图像和数据压缩算法等多方面性能大幅度提升。SSE4分为4.1版本及4.2版本，4.1版本共新增47条指令，主要针对向量绘图运算、3D游戏加速、视频编码加速及协同处理加速动作。2008年1月45 nm的Ndahem处理器追加了SSE4.2版本，新增7条指令，两个版本合共54条指令。如图3.1所示，为SSE指令集的发展简史【基于SSE4指令集的H.264】。

Intel指出，加入的SSE4指令集让45nm Penryn处理器增加了2个不同的32Bit向量整数乘法运算单元，并加入8位无符号最大值及最小值运算，以及16Bit及32Bit有符号运算。在面对支持SSE4指令集的软件时，可以有效地改善编译器效率及提高向量化整数及单精度代码的运算能力。同时，SSE4改良插入、提取、寻找、离散、跨步负载及存储等动作，令向量运算进一步专门化。

SSE4还计入了六条浮点运算指令，支持单精度、双精度浮点运算及浮点产生操作，可立即转换其路径模式，大大减少延迟，这些支持在3D游戏及对浮点运算能力非常敏感的领域起积极的效果。

SSE4指令集还加入了传流式负载指令，能够提升帧缓冲区的读取数据频宽，理论上可获取完整的快取缓存行，及每次读取64Bit而非8Bit，并可以将其保存至临时缓冲区内，让支持SSE4指令集的读取频宽效能提升最高至8倍。

SSE4指令集进一步强化编码效果，例如可同时处理8个4-byte宽度的SAD（Sums of Absolute Differences）运算，常用于新一代高清影像编码如VC.1及H.264等规格中，令视频编码速度进一步提升。AMD也开发了属于自己的SSE4多媒体指令集，并内置在Phenom与Opteron等K10架构处理器中，不过无法与Intel的SSE4系列指令集兼容。

SSE5是AMD为了打破Intel垄断在处理器指令集的独霸地位所提出的，SSE5初期规划将加入超过100条新指令，其中最引人注目的就是三运算对象指令（3-Operand Instructions）及熔合乘法累积（Fused Multiply Accumulate）。其中，三运算对象指令让处理器可将一个数学或逻辑库，应用到运算对象或输入数据。借由增加运算对象的数量，一个 x86 指令能处理二至三笔数据， SSE5 允许将多个简单指令汇整成一个指令，达到更有效率的指令处理模式。提升为三运算指令的运算能力，是少数 RISC 架构的水平。熔合乘法累积让允许创建新的指令，有效率地运行各种复杂的运算。熔合乘法累积可结合乘法与加法运算，通过单一指令运行多笔重复计算。通过简化代码，让系统能迅速运行绘图着色、快速相片着色、音场音效，以及复杂矢量演算等性能密集的应用作业。目前AMD已放弃下一代Bulldozer核心内置SSE5指令集，改内置Intel授权SSE4系列指令集。

AVX(Advanced Vector Extensions) 是Intel与AMD在X86指令集的SSE延伸架构，如IA16至IA32般的把暂存器XMM 128bit提升至YMM 256bit，以增加一倍的运算效率，它借鉴了一些AMD SSE5的设计思路，进行扩展和加强，形成一套新一代的完整SIMD指令集规范。此架构支持了三运算指令，减少在编码上需要先复制才能运算的动作。在微码部分使用了LES LDS这两个不常用的指令作为延伸指令Prefix。业界普遍认为支持AVX指令集是Sandy Bridge最重要的进步。

FMA是Intel的AVX指令集的一个分支，因此FMA指令使用的是与AVX指令相同的编码设计规则。如名称上熔合乘法累积（Fused Multiply Accumulate）的意思一样，它允许建立新的指令并有效率地执行各种复杂运算。熔合乘法累积可结合乘法与加法运算，通过单一指令执行多次重复运算，从而简化程序，从而使系统能快速执行绘图、渲染、相片着色、立体音效及其他复杂向量运算等计算量大的工作。

FMA提高了浮点运算能力。Haswell架构中拥有2个新的FMA单元，每个FMA单元支持8个单精度或4个双精度浮点数。

## 4.2 已有的应用

近二十年间，探索使用计算机字并行ram模型来加速串匹配算法的尝试有很多。在这个模型中，计算机操作长度为ω的字，使得能一次性读入块字符。这意味着许多算法中的一些对字的操作步骤能在一个CPU单位时间内完成。

许多使用字并行ram模型的算法都基于以下两种技术，位并行技术和块字符技术。

### 4.2.1 位并行

位并行技术利用了计算机机器字位运算的内在并行性，即可以把多个值装入同一个机器字内，然后只需一次运算就能更新所有值。利用位并行，一个算法所执行的运算次数最多可能减少到原来的1/ω，这里ω是机器字的位数。当前的系统结构中，ω为32或64，因此实际应用中的加速效果是相当显著的。位运算非常适合基于不确定自动机算法的实现。Shift-Or（SO）算法就是第一个使用位并行的算法，该算法高效地模拟了KMP自动机的非确定性实现并且算法时间复杂度在。并且在小字母表的短串模式匹配中，SO算法仍然是一个相当快速的实用算法。不久后快速BDM改进算法（BNDM）产生，该算法使用位并行去实现非确定性前缀自动机【31】。其中一些BNDM算法的变体也是目前最快的串匹配算法。

然而，位并行技术最大的限制在于它只能处理长度小于计算机位长的串，当串长m小于该长度时，位并行算法非常高效。一旦超过该长度，对超过该长度的比较就只能使用BF算法。虽然有一些改进技巧能让位并行在长串模式匹配时仍能保持优势，但实际效果没有强太多，因为内在的位长限制是不可改变。

### 4.2.2 块字符

在块字符技术中，多个字符被打包成一个较大的字，使得这些字符能同时被计算而不用逐个去比较。在这种情形下，若模式串的字符是由字符表大小为σ组成，那么能组成一个单字的字符个数为。在这里，我们定义块字符因子α=。

第一个使用块字符的方法是由Fredriksson提出【23】。他提出了一个方案能给许多串匹配算法加速。他的方法。。。而第一个使用块字符获得很好实验结果的算法是由Ben-kiki等人于2011年提出的ssecp算法【4】。该算法只基于两条特殊的SSE指令，pcmpestrm和pcmpestri指令，但算法时间复杂度只有。该作者通过实验在论文中得出结论在短串模式匹配中ssecp算法是最快的。

### 4.2.3 Four-Russians

所谓Four-Russians技术，是指将某个问题划分成若干小问题，如果某个小问题在求解大问题的过程中反复出现，则可以将这个小问题事先计算出来，则求解大问题过程中如果碰到这个小问题只需要查表或者通过一些简单计算就可以得到这个小问题的结构，从而加速整个问题的求解。

当模式串的长度增长时，ssecp算法的优势逐渐减弱。这时，Kulekci提出了流式SIMD扩展过滤算法ssef，该算法利用Four-Russians的思想使用一种过滤方法去检查块字符而不是逐个去比较。尽管该算法在理论上存在最差的时间复杂度，但在实际应用中，它是长串模式匹配中最快的。

**第5章 改进算法IEPSM详述**

## 5.1 综述

尽管可供使用的SSE指令有很多，但大多数指令在实际执行需要的CPU时间周期数仍然较大。因此在本文中，我们将挑选四条执行周期数少的指令来实现我们的精确块字符匹配算法（iepsm）。

但在设计算法之前，我通过编程实验发现计算机处理串匹配问题有如下特性，如果不将这些特性运用进去，只根据理论预测的算法可能无法达到预期的效果。

这些特性如下：

* 查询表的大小（即数组定义）最好设置在2048左右。因为大部分串匹配算法都引入了查询表，而从计算机体系结构来看，过大的数组会导致cache不命中，从而带来的硬件中断，降低效率。Iepsm将大小设置为2048，这样既能保证查询表足够大又能降低cache不命中率。
* 模式串的指针跳转长度定义成常量。经实验，指针的常量跳转速度比读变量跳转有倍数的性能提升。

由于基因串的特征和字符表大小相同的随机串表现类似，且由于基因模式串的长度一般较长，所以我们重点考虑长串匹配的情况。考虑到大部分SSE指令的输入参数为128位的数据，所以iepsm算法根据串长的不同以16个字符为界分为两种不同的优化方案，并在长串匹配中将上述的两点特性运用进去。

首先，我们需要通过组合SSE指令来为我们的算法设计合适的伪指令。

### 5.1.1 wscmp(a,b)字长比较指令

### 5.1.2 wsmatch(a,b)字长匹配指令

### 5.1.3 wsblend(a,b)字长翻转指令

### 5.1.4 wscrc(a)字长CRC码计算指令

Wscrc指令通过传入一个8bytes长度的字符串来生成一个32bit的CRC值。该指令通过该SSE指令模拟\_\_mm\_crc32\_u64(α)，由于该指令只有一个时间周期，因此具有很快的速度和很好的鲁棒性。

但因为该指令生成的是64bit的数字，而我们的数组大小只有2048，所以还需将结果与上掩码0x11111111111（2047）。

## 5.2 模式串串长m大于等于16

表2.1 IEPSM-算法

|  |  |
| --- | --- |
| **IEPSM算法** | |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14 | Prepare(P,m,T,n)  For i ← 0 to m-α do  h = wscrc(P+i)  shift[h].pos = i  shift[h].fingerprint = P+i  Search(P,m,T,n)  For i ←0 to n-α do  h = wscrc(T+i)  f = T+i  p = shift[h]  while(p!=null)  if f == p.fingerprint  memcmp(P,T)  p = p.next |

算法分为两个阶段。

首先我们定义了查询表，它记录采样串在模式串的出现位置，其中采样串的长度是个试验参数B，取值在4bytes和8bytes之间。程序里由一指针数组Node\* shift[2048]来实现。其中结点Node，数据结构如下。

表2.1 Node数据结构

|  |  |
| --- | --- |
| **Node数据结构** | |
| 01  02  03  04  05  06 | typedef struct node  {  struct node \*next;  int pos;  unsigned long long val;  }NODE; |

它存储某一采样串p1在模式串P中的位置，其中p1通过计算hash值h去定位到它的结点shift[h]。

那么在预处理阶段，对于模式串P，从位置i起取8bytes强转成64位ulong类型做为该子串的指纹值，然后将该子串与上掩码并调用wscrc(a)计算得到串hash值h，那么对于采样串p1，算得hash值h后，将p1在p中出现的位置i存储在shift[h]中，对于hash冲突造成的重值，存储在shift[h]对应的下一链表结点中。

在搜索阶段，首先指针指向匹配串T的m-8位置，并取8bytes子串，算得hash值h，若shift[h]有值i，说明该子串可能是串P在i位置的一部分，然后判断该子串是否和shift[h]的指纹值相等，若相等，再根据i值将串P的i位置和该子串对齐，并调用memcmp函数进行比较，若shift[h]有多个链表结点，则需根据i值进行多次对齐和比较。完成后指针再继续前进m-B的长度，并重复刚才的搜索步骤。

在这里，我们采用了m-B的常值跳转，而根据鸽笼原理，m-B是iepsm不漏扫的最大距离。所以容易看出，如果采样串长度B较小，即能获取较大的跳跃距离m-B，但与此同时数组结点对应的链表会很长，这样每次比较都会去遍历这个链表。另外B越小，采样串的匹配概率越大，导致mencmp全匹配函数的调用次数加大，抵消改进；而根据经验，shift数组越大，程序会因CPU缓存原因查询越慢，所以合适的大小是2048。但对于基因串来说，4的8次方为65536，会造成较大的hash冲突，同样降低性能。

因此我们可以通过实验去找到合适的采样串长度，使得它既能获取较大的跳跃距离，同时又不造成很高的hash冲突。

另外解释下指纹值的作用。不论B选多大，我们实际上都是读取8bytes的采样串p0，但通过掩码来获取变长的采样串p1， 所以我们通过在调用memcmp前比较p0对应的64bit数值与对应模式串位置的64bit数值，这样就能大大减少全比较函数调用次数。具体实现是在Node定义中加一字段存储原采样串的64bit数值，同时在预处理阶段根据模式串P将其初始化。

## 5.3 模式串串长m大于等于4且小于16

## 5.4 模式串串长m等于3

**第6章 实验结果与分析**

本实验的测试机为CPU型号为Intel(R) Xeon E3-1230 V2，工作频率为3.30GHz，4G内存，操作系统为Linux Mint 13。实验中所有的测试算法均由C语言实现，并由GCC编译。

## 6.1 测试算法选取

我们选取了最新的基于前缀搜索的算法ufndmq、基于后缀搜索的算法hashQ、基于子串搜索的算法fsbndmq、针对生物学的串匹配算法tvsbs和使用SSE指令处理块字符的ssecp这五种算法和我们的新算法进行比较。

## 6.2 测试集选取

我们选取了三种类型生物的基因序列和氨基酸序列做为测试集。测试集详细如下。

表2.1 测试数据集选择

|  |  |  |
| --- | --- | --- |
| **种类** | **基因序列** | **蛋白质序列** |
| 大肠杆菌 | ftp://ftp.ncbi.nlm.nih.gov/genomes/Bacteria/Escherichia\_coli\_K\_12\_substr\_\_MG1655\_uid57779/NC\_000913.fna | ftp://ftp.ncbi.nlm.nih.gov/genomes/Bacteria/Escherichia\_coli\_K\_12\_substr\_\_MG1655\_uid57779/NC\_000913.faa |
| 水稻 | ftp://ftp.ensemblgenomes.org/pub/plants/release-24/fasta/oryza\_sativa/dna/Oryza\_sativa.IRGSP-1.0.24.dna.genome.fa.gz | ftp://ftp.ensemblgenomes.org/pub/plants/release-24/fasta/oryza\_sativa/pep/Oryza\_sativa.IRGSP-1.0.24.pep.all.fa.gz |
| 人类 | ftp://ftp.ensembl.org/pub/current\_fasta/homo\_sapiens/dna/Homo\_sapiens.GRCh38.dna.primary\_assembly.fa.gz | ftp://ftp.ensembl.org/pub/current\_fasta/homo\_sapiens/pep/Homo\_sapiens.GRCh38.pep.all.fa.gz |

## 6.3 参数选取实验

在该实验中，我们通过选取五种不同的参数来获取不同种类序列数据下的最优参数。下表中最优数据用斜体加粗标出。

大肠杆菌基因数据

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 | 44 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| iepsm4 | ***3.27*** | 3.08 | 2.09 | 2.21 | 2.12 | 2.02 | 1.31 | 1.81 | 1.36 | 1.76 | 1.31 | 1.22 | 1.46 | 1.22 | 1.7 |
| iepsm5 | 3.84 | 2.77 | 2.2 | 1.37 | 1.74 | 1.96 | 1.71 | 0.97 | 1.37 | 1.05 | 1.25 | 1.13 | 1.11 | 0.71 | 0.71 |
| iepsm6 | 3.54 | ***2.31*** | ***2.09*** | ***1.14*** | ***1.18*** | ***1.03*** | ***0.83*** | 1.13 | 1.31 | 0.5 | 0.65 | 0.34 | 0.6 | 1.05 | 0.3 |
| iepsm7 | 3.95 | 2.42 | 2.31 | 1.27 | 1.84 | 1.61 | 1.57 | 1.34 | 1.28 | 1.25 | 1.21 | 0.9 | 0.84 | 0.62 | 0.51 |
| iepsm8 | 4.85 | 3.6 | 2.46 | 1.44 | 1.36 | 1.51 | 1.51 | ***0.69*** | ***0.57*** | ***0.52*** | ***0.81*** | ***0.79*** | ***0.17*** | ***0.4*** | ***0.43*** |

水稻基因数据

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 | 44 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| iepsm4 | ***269.64*** | 192.34 | 154.25 | 134.13 | 130.48 | 116.81 | 108.32 | 103.69 | 100.46 | 86.73 | 74.67 | 72.00 | 71.69 | 68.10 | 60.38 |
| iepsm5 | 271.99 | 190.55 | 145.98 | 120.71 | 108.45 | 96.35 | 88.26 | 79.59 | 77.07 | 60.81 | 53.69 | 48.03 | 48.64 | 46.16 | 37.02 |
| iepsm6 | 296.24 | ***188.66*** | ***137.71*** | ***113.11*** | ***94.04*** | ***81.05*** | 73.69 | 64.99 | 64.88 | 49.67 | 42.68 | 35.22 | 27.21 | 23.70 | 19.37 |
| iepsm7 | 358.05 | 207.44 | 144.98 | 118.34 | 95.81 | 82.95 | 74.96 | 64.81 | 60.37 | 47.32 | 40.90 | 33.56 | 23.38 | 17.28 | 13.45 |
| iepsm8 | 440.13 | 226.16 | 153.86 | 118.23 | 96.98 | 84.19 | ***72.78*** | ***64.14*** | ***59.39*** | ***45.95*** | ***40.89*** | ***30.60*** | ***20.59*** | ***15.19*** | ***10.92*** |

人类基因数据

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 | 44 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| iepsm4 | ***1833.65*** | 1416.69 | 1265.22 | 1089.91 | 954.36 | 937.78 | 918.14 | 848.27 | 728.48 | 719.65 | 591.28 | 621.64 | 497.39 | 536.42 | 406.07 |
| iepsm5 | 1976.54 | 1350.44 | 1045.03 | 865.67 | 763.62 | 685.66 | 663.47 | 612.89 | 567.02 | 480.07 | 386.43 | 374.30 | 346.34 | 364.98 | 273.49 |
| iepsm6 | 2134.28 | ***1345.82*** | ***992.15*** | ***807.76*** | ***683.10*** | ***582.12*** | ***517.52*** | 500.12 | 459.53 | 386.18 | 322.16 | 282.56 | 213.18 | 191.98 | 176.22 |
| iepsm7 | 2516.66 | 1455.80 | 1025.39 | 817.82 | 697.70 | 595.98 | 544.09 | 485.27 | 449.21 | 360.28 | 308.06 | 252.78 | 193.30 | 144.38 | 122.68 |
| iepsm8 | 3077.91 | 1549.02 | 1078.94 | 824.01 | 699.93 | 594.25 | 532.21 | ***466.46*** | ***445.42*** | ***351.45*** | ***303.27*** | ***236.61*** | ***151.72*** | ***116.49*** | ***92.90*** |

大肠杆菌蛋白质数据

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 | 44 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| iepsm4 | ***1.58*** | ***1.2*** | ***0.79*** | ***0.63*** | ***0.36*** | ***0.46*** | 0.49 | ***0.33*** | ***0.22*** | 0.44 | 0.66 | 0.69 | 0.99 | 0.87 | 1.11 |
| iepsm5 | 1.61 | 1.54 | 1.08 | 0.98 | 0.86 | 1.02 | 1.19 | 0.44 | 0.78 | 0.52 | 0.96 | 0.84 | 0.82 | 0.54 | 0.48 |
| iepsm6 | 1.72 | 1.68 | 0.85 | 1.03 | 0.89 | 1.16 | ***0.31*** | 0.67 | 0.84 | ***0.35*** | ***0.42*** | ***0.16*** | 0.48 | 0.93 | 0.18 |
| iepsm7 | 1.89 | 1.66 | 1.01 | 0.85 | 0.79 | 0.65 | 1.23 | 0.39 | 0.36 | 0.43 | 0.46 | 0.73 | 0.31 | 0.34 | 0.39 |
| iepsm8 | 1.97 | 1.66 | 1.05 | 0.94 | 0.42 | 0.46 | 0.87 | 1.08 | 0.93 | 0.9 | 0.98 | 0.38 | ***0.28*** | ***0.19*** | ***0.15*** |

水稻蛋白质数据

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 | 44 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| iepsm4 | ***9.7*** | ***7.25*** | ***5.56*** | ***4.92*** | ***4.3*** | ***3.91*** | ***3.61*** | 3.57 | 2.59 | 3.12 | 2.31 | 1.63 | 1.58 | 1.22 | 1.09 |
| iepsm5 | 11.55 | 7.56 | 6.2 | 5.25 | 4.68 | 4.31 | 4.36 | 3.02 | 3.01 | 2.4 | 2.55 | 1.9 | 1.23 | 0.83 | 0.76 |
| iepsm6 | 11.89 | 8.57 | 6.38 | 5.55 | 4.89 | 4.63 | 3.88 | 3.42 | 3.13 | ***2.27*** | ***2.01*** | ***1.39*** | 1.36 | 1.34 | 1.21 |
| iepsm7 | 14.13 | 8.31 | 6.48 | 5.45 | 4.31 | 4.43 | 4.58 | ***2.74*** | ***2.8*** | 2.31 | 2.05 | 1.73 | 0.74 | 0.85 | 0.64 |
| iepsm8 | 17.74 | 9.31 | 6.58 | 6.08 | 4.42 | 4.04 | 4.22 | 3.67 | 3.34 | 2.84 | 2.39 | 1.72 | ***0.69*** | ***0.69*** | ***0.19*** |

人类蛋白质数据

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| m | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 | 44 | 64 | 128 | 256 | 512 | 1024 | 2048 |
| iepsm4 | ***21.97*** | ***15.36*** | ***11.56*** | ***9.23*** | ***9.90*** | ***6.99*** | ***6.34*** | ***5.80*** | ***5.29*** | ***4.51*** | 4.44 | ***2.83*** | 2.22 | 1.39 | 1.36 |
| iepsm5 | 26.37 | 17.30 | 13.17 | 10.78 | 10.73 | 7.65 | 6.84 | 6.10 | 5.48 | 4.75 | 4.47 | 3.08 | 2.30 | 1.60 | 1.35 |
| iepsm6 | 30.75 | 18.80 | 13.85 | 11.14 | 10.88 | 7.95 | 7.12 | 6.50 | 5.57 | 4.58 | ***4.13*** | 3.06 | 2.40 | 1.49 | 1.38 |
| iepsm7 | 36.49 | 20.66 | 14.94 | 11.65 | 11.34 | 8.27 | 7.27 | 6.52 | 5.59 | 4.93 | 4.40 | 3.23 | 2.36 | 1.52 | 1.44 |
| iepsm8 | 44.74 | 22.52 | 15.73 | 11.99 | 11.50 | 8.25 | 7.33 | 6.49 | 5.49 | 4.68 | 4.21 | 2.87 | ***2.11*** | ***1.35*** | ***1.24*** |

通过表？对大部分数据而言，可以得知。

表2.1 基因数据参数选择

|  |  |
| --- | --- |
| **模式串串长m** | **基因序列** |
| 12≤m<16 | 4 |
| 16≤m<40 | 6 |
| 40≤m | 8 |

表2.1 蛋白质数据参数选择

|  |  |
| --- | --- |
| **模式串串长m** | **蛋白质序列** |
| 12≤m<256 | 4 |
| 256≤m | 8 |

## 6.4 算法对比实验

在该实验中，iepsm选取了6.3实验中得出的最优参数。

结论

本文针对生物信息学中的单模式串匹配问题，使用Intel SSE指令提出了一个优化算法。由于基因串检索一般较长，所以本文重点考虑了模式串串长大于等于16的情形，另外对小串也给出了优化算法，只是没有过多考虑生物信息特点。

对于长串情形，该算法针对生物信息的特点，采用优化的块字符大小来生成hash表，。除此之外，新算法采用一个64bit的大整数来表示字符串的指纹，该指纹通过8个字符生成。算法在进行逐字节的比较前通过该指纹过滤掉很多逐字匹配的调用。

采用……。（结论作为学位论文正文的最后部分单独排写，但不加章号。结论是对整个论文主要结果的总结。在结论中应明确指出本研究的创新点，对其应用前景和社会、经济价值等加以预测和评价，并指出今后进一步在本研究方向进行研究工作的展望与设想。结论部分的撰写应简明扼要，突出创新性。）

**参考文献**

[1]姜敏,彭少贤,郦华兴.形状记忆聚合物研究现状与发展[J].现代塑料加工应用,2005,17(2): 53-56.

[2]Akahashi T, Hayahi N, Hayashi S．Structure and Properties of Shape-Memory Polyurethane Block Copolymers[J].Appl PolymSci,1996,60：1061-1069.

[3]董永春，滑钧凯．纺织品整理剂的性能与应用[M].北京:中国纺织出版社，1999,34-35.

[4]曲飞. 纺织品阻燃剂的现状及发展趋势.全国纺织材料最新研究进展研讨会论文集[C].北京：北京理工大学出版社,2007,145-148.

[5]Mao Xia. Analysis of Affective Characteristics and Evaluation of Harmonious Feeling of Image Based on 1/f Fluctuation Theory. International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems (IEA/AIE ) [C]. Australia Springer Publishing House, 2002: 17-19.

[6]张和生. 地质力学系统理论[D]. 太原: 太原理工大学, 1998.

[7]姜锡洲. 一种温热外敷药制备方案[P] : 中国, 881056078, 1983-08-12.

[8]GB/T 16159-1996, 汉语拼音正词法基本规则[S]. 北京: 中国标准出版社, 1996.

[9]毛峡. 情感工学破解‘舒服’之迷[N] . 光明日报, 2000-4-17(B1).

[10]冯西桥. 核反应堆压力容器的LBB分析[R]. 北京: 清华大学核能技术设计研究院, 1997.

[11]王明亮. 中国学术期刊标准化数据库系统工程的[EB/OL]. (1998-08-16)[1998-10-04] http://www.cajcd.cn/pub/wml.txt/980810-2.html.

**附录**

附录相关内容…

**攻读学位期间发表论文与研究成果清单**

[1] 高凌.交联型与线形水性聚氨酯的形状记忆性能比较[J].化工进展,2006,25(1)：532－535.（核心期刊）

**致谢**

不知不觉，来到北京已经快三年了，而在北京理工大学的两年半硕士研究生生活也接近尾声，回首往昔，深深地觉得自己对这个城市、这个学校有着割舍不了的情感，离别的思绪开始蔓延。值此毕业论文完成之际，向所有帮助过我、关心过我的老师及同学表示诚挚的感谢和衷心的祝愿。

首先要感谢我的导师戴林副教授，他那严谨的治学态度，渊博的学识，宽广的胸襟，敏捷的思维，忘我的工作精神，以及平易近人的为人，为我树立了人生的榜样。平易近人的戴老师亦是我的良师益友，他严格把关、循循善诱，从本论文的选题，系统的设计一直到论文撰写都倾注了大量的心血，使得本论文得以顺利完成，他在软件开发，数据库设计领域的许多看法也让我受益匪浅。

感谢朱玉文、刘万春、黄河燕、鉴萍、李侃等老师，以及刘峡壁、马波、刘庆辉等老师，为我们传授知识；感谢王景如、宋健龙等同学，让我在实验室的生活中很舒适。

特别要感谢王景如同学，为我的论文和毕业设计提供了很多有价值的参考信息。

最后要感谢我的父母，他们25年来对我的关怀和照顾无微不至，养育之恩，无以为报。同时也是他们对我的鼓励和支持才使得我能走到今天，让我有勇气不断进取，不断攀登。

最后衷心所有关心、支持和帮助过我的人们！。