南京理工大学

硕士学位论文

# 基于层次分类的网络内容监管系统中串匹配算法的设计与实现

姓名：章张

申请学位级别：硕士

专业：计算机软件与理论

指导教师：黄河燕;王树梅

20040601

# 摘　要

互联网上的反动、暴力、淫秽等非法信息使得网络内容监管日益重要。本文在对网络内容监管方面的研究现状进行分析后，采用一种层次分类的方法，即在信息识别过程中，先将网络信息进行关键字过滤，然后再将含指定关键字的信息内容传送给语义分析模块进行处理，从而减轻语义分析的压力，提高系统效率，并且适合于非法信息多变的特点。

关键字过滤通过串匹配算法实现，并且已成为系统性能的瓶颈。为此，本文分析了现有串匹配算法（包括单模式串匹配算法和多模式串匹配算法）的特点，设计并实现了一种适合于中文大字符集和网络内容监管的串匹配算法：基于 QS 算法的单模式串匹配算法（IQS 算法）和基于 Wu-Manber 算法的多模式串匹配算法（IWM 算法）。IQS 算法主要用于工作人员的日后查阅，而 IWM 算法主要用于网络信息过滤。并且在中、英文两种环境下以及模式串的不同长度和个数情况下对 IQS 算法和 IWM 算法进行了实验比较分析，结果表明在运行时间与尝试次数方面整体优于其它算法，获得了较好的效果。

IQS 算法和 IWM 算法已集成实现在网络内容监管系统中，用真实数据对该系统进行了测试，实验结果表明该算法表现出了较快的运行速度，系统的各项性能指标均已达到预定目标。

**关键字**：网络内容监管，串匹配，单模式，多模式

# Abstract

Network Content Monitor has become more and more important just because there is illeagl information about subversion, violence, obscenity etc. found on the network. In this paper, we analyse the current development of Network Content Monitor and introduce a method of hiberarchy classification for it. This method first filters the network information by keywords filter, and then with the help of semantic analysis analyses those information, which contain the keyword(s) given by the user. It can ease the pressure of semantic analysis, improve the efficiency of the system and adapt well to the caprice of illeagl information.

Keywords filter is realized by string matching algorithm and is the performance bottleneck of Network Content Monitor indeed. Therefore, this paper analyses the existing algorithms of string matching including single-pattern string matching and multi-pattern string matching. Then we design and implement two algorithms suitable to Chinese character set and Network Content Monitor: one is IQS, a single-pattern string matching algorithm based on QS algorithm, and the other is IWM, a multi-pattern string matching algorithm based on Wu-Manber algorithm. The aims of the two algorithms mentioned above are to search a keyword among files saved in local hard disks and to filter the network information, respectively. Meanwhile, we make experiments on IQS algorithm and IWM algorithm under the circumstance of Chinese and English, and different lengths and numbers of patterns. The experiments' results indicate that the two algorithms have a better performance on speed and attempt.

Finally, we integrate IQS algorithm and IWM algorithm into the system of Network Content Monitor, and make experiments on this system. The results show that the high speed of string matching algorithms is achieved, and the performance targets of this system are also obtained.

**Key Words:** Network Content Monitor, string matching, single-pattern, multi-pattern

# 声　明

本学位论文是我在导师的指导下取得的研究成果，尽我所知，在本学位论文中，除了加以标注和致谢的部分外，不包含其他人已经发表或公布过的研究成果，也不包含我为获得任何教育机构的学位或学历而使用过的材料。与我一同工作的同事对本学位论文做出的贡献均已在论文中作了明确的说明。

研究生签名：童张　　2004年6月8日

# 学位论文使用授权声明

南京理工大学有权保存本学位论文的电子和纸质文档，可以借阅或上网公布本学位论文的全部或部分内容，可以向有关部门或机构送交并授权其保存、借阅或上网公布本学位论文的全部或部分内容。对于保密论文，按保密的有关规定和程序处理。

研究生签名：童张　　2004年6月8日

# 1 绪论

## 1.1 研究背景及意义

网络内容监管是随着计算机的广泛使用和网络化的普及而产生的。在享受着计算机和网络所带来的便利的同时，人们除了会遭受到病毒、恶意代码和恶意网络链接、垃圾邮件等攻击外，最令人担忧的问题还有互联网上的反动、淫秽、赌博、毒品、暴力等不健康的信息，以及恐怖、欺诈、盗窃机密信息、非法攻击国家政府网站、有碍国家和平与发展等非法行为，这些不仅会干扰人们正常的生活、学习和工作，危害青少年的健康成长，而且影响国家经济的正常发展，甚至造成不可挽回的损失。

由于世界各国意识形态和社会制度的不同，作为全球最大电子媒体的互联网络在各国所受到的待遇也不尽相同，但各国政府在对网络上散布危害国家安全、反动、淫秽、色情、暴力等非法信息的处理态度上都十分重视，纷纷采取了相应的措施。

我国公安部 1997 年 12 月 30 日发布的《计算机信息网络国际联网安全保护管理办法》和国务院 2000 年 9 月 25 日发布的《互联网信息服务管理办法》中规定："任何单位和个人不得利用国际联网危害国家安全、泄露国家秘密，不得侵犯国家的、社会的、集体的利益和公民的合法权益，不得从事违法犯罪活动"，"任何单位和个人不得利用国际联网制作、复制、查阅和传播含有下列内容的信息：危害国家安全，泄露国家秘密，颠覆国家政权，破坏国家统一的；散布淫秽、色情、赌博、暴力、凶杀、恐怖或者教唆犯罪的"等等[1]。2001 年 7 月 11 日中共中央总书记江泽民在中南海怀仁堂主持内容为运用法律手段保障和促进信息网络健康发展的法制讲座，并指出对网络中的有害信息和不良信息必须进一步研究并采取切实有效的措施，在加强网络安全保障体系建设的同时，更要注意充分运用法律手段，搞好对信息网络的管理工作，以推动信息网络的快速健康发展。

1998 年 10 月，美国众议院通过了《儿童在线保护法案》（Child Online Protection Act），以限制未成年人访问国际互联网上的成人内容，保护未成年人的健康成长[2]。有人称其为是保护未成年人不受到网络色情影响的《通信规范法》（Communications Decency Act）之子，因为该法案完全是因为《通信规范法》被最高法院裁定为违宪而制定的。"9·11"事件发生后，美国政府和企业非常担心国际恐怖分子采取网络袭击的办法攻击和破坏美国庞大的信息基础设施，随后美国政府立即采取行动，成立"国土安全办公室"，将打击网络恐怖作为其主要职责之一。同时，针对网络中错误、虚假、色情、暴力等信息，美国政府将有关

网络信息方面的处理技术应用于美国公共图书馆中[3]。

1999 年，欧盟 15 国通讯部长在布鲁塞尔会议上决定提供一笔资金，在今后的四年内实施一项清除互联网上的淫秽和非法内容的行动计划。德国政府还宣布实施《多媒体法》，该法对经营网络信息提出了明确的责任界限，还规定了社会对多媒体信息的许可限度，以免未成年人被信息误导和毒害。英国政府制定了与网络内容监管相关的 RIP 法案[4]（Regulation of Investigatory Powers Bill），并且由政府提议，在民间发起成立了一个因特网监护基金会，专门负责因特网的监督管理事务。荷兰司法当局于 1998 年出台的法律要求所有的电信公司必须安装“网络警察”监控设备，以便对实施犯罪的互联网用户进行有效的追踪。

由此可见，网络内容监管有着十分重要的意义，它不仅关系到个人的正常生活和青少年的健康成长，更影响着国家经济的正常发展。为了更好的监管互联网上的信息，除了要制定相关的法律和法规进行约束外，还需要完善先进的技术手段进行识别、分析和处理。由于互联网信息量的庞大（根据中国互联网络信息中心（CNNIC）《2003 年中国互联网络信息资源数量调查报告》[5]显示，2002 年全国域名数 940329 个，网页总数 157091220 个，网页总字节数 2877754095KB，而截止到 2003 年 12 月 31 日，全国域名数增至 1187380 个，网页总数增至 311864590 个，网页总字节数增至 6059431526KB）、信息种类和格式的复杂（包括静态网页、动态网页、txt 文件、doc 文件、pdf 文件等，以及图像文件、音频文件和视频文件等），因而，非法信息识别和处理的准确度和速度直接影响到网络内容监管的效果。如何设计一个系统以实现对网络内容的监管，即快速准确的鉴别出网络中的信息是否合法，具有时代的迫切性和重要的意义。

## 1.2 国内外现状分析

### 1.2.1 网络内容监管

在面向互联网的内容监督和管理技术中，国内外都对敏感信息识别进行了较为广泛和深入的研究。国家级的研究有美国联邦调查局（Federal Bureau of Investigation）的特征鉴别工具 Carnivore[6]、美国国家安全局（National Security Agency）的全球监测系统 ECHELON[7]、法国的与 ECHELON 类似的 SGDN[8]以及其它国家的相关研究[8]。美国在 2001-2003 年度计划开发一个反恐的自动监测、分类和识别原型系统，为恐怖预警分析提供有效资源。应用的技术主要有信息流挖掘技术以及某些自然语言理解技术。这些研究的主要分析对象为危害国家和民族的、反动的、邪教的等有害动态信息流。

另外，一些软件企业也先后开发出了关于相应识别过滤敏感信息的产品，代表产品有“网络巡警（Cyber Patrol）”、“网络保姆”、“网络爸爸”等，它们的分

析目的主要集中在各种商业秘密、有害于青少年健康成长的色情、淫秽、暴力、毒品、赌博等内容，用户主要是中小企业和家庭用户，主要采用关键字技术，或黑名单技术，即将事先已经发现有问题的网站列入被禁止访问的名单里，从而控制访问。对于信息流的获取一般采用被动方式，即对流过服务器的信息流进行识别和过滤，不让非法信息进入安全域内。在国内这方面的研究也在积极进行[9,10]，如哈尔滨中唯信息安全技术有限公司推出的“互联网有害信息过滤报警管理系统”等，但大多采取了基于关键字或简单语义的识别技术，或是采用被动获取信息的方法。

由于基于关键字或简单语义倾向性识别的信息处理技术简单、快速，因而在企业、家庭和个人中有着广泛的应用，成为互联网监控的基本手段，但该手段也有着很大的局限性，由于它为了最大限度的实现过滤的实时性，只采用相对简单的分析技术，很容易导致误报率较高，识别效果也比较差。因此，当采用主动获取信息并对其中的敏感信息进行自动识别、侦测时，上述手段已经不能满足需求，必须采用高效的识别手段。鉴于国内的网络状况和中文信息处理技术的发展，研究适合国内环境、准确性高的敏感信息侦测和识别技术具有很大的迫切性和重要的意义。下一节我们主要介绍在网路内容监管中常用到的技术手段之一，串匹配技术。

### 1.2.2 串匹配

串匹配（String Matching，Pattern Matching），是指在一个符号序列中查找出一个或多个特定符号序列的过程，是计算机科学领域中最基本的问题之一。串匹配问题可以描述为：已知需要匹配的子串（通常称之为模式串，Pattern）和需要检索的字符串（通常称之为文本，Text），查找在文本中所有出现的模式串，并报告其出现的位置。

依据在匹配过程中所要匹配的模式串个数，串匹配算法可分为单模式串匹配算法和多模式串匹配算法。依据其功能来分，串匹配算法可分为三类：精确串匹配算法(Exact String Matching)、近似串匹配[11]（Approximate String Matching）算法和正则表达式匹配（Expression Matching，Predicate Matching）算法。精确串匹配算法是指在数据序列中查找出一个或多个特定的模式串完全相等的子串的出现位置，主要应用在文本检索和网络安全的入侵检测系统领域中；近似匹配算法是指按照算法定义的相似程度，在数据序列中查找所有和一个或多个特定的模式串的相似程度在一定范围内的所有子串的出现位置，主要应用在计算生物学和信号处理等领域；正则表达式匹配算法是指根据正则表达式的描述，在数据序列中查找能够被正则表达式所接收的所有子串的出现位置，主要用来处理简单的序列匹配无法描述的问题。另外，还可将某些串匹配问题用硬件实现[12,13]，这就是

所谓的硬件匹配（Hardware Matching）。

串匹配技术的发展是与其应用密切相关的。在计算机发展的早期阶段，串匹配常常用于文本编辑、全文检索系统、查询系统等。随着网络技术的发展、数据量的激增以及生物信息学的发展，串匹配已经广泛应用于网络入侵检测系统[14,15]（Network Intrusion Detection System）、内容过滤[16]、生物科学计算[17,18]、新闻主题提取[19]等。

- 入侵检测系统（Intrusion Detection System，简称 IDS）：入侵检测系统是网络安全问题的一个主要研究方向。入侵检测技术的目的是建立一种计算机系统的安全保护机制，防止在未授权的情况下操作计算机系统，访问系统内的数据和资源。入侵检测系统针对各种入侵行为的特点建立了一组入侵行为的特征模式集，同时入侵检测系统针对各种行为加入相应采取的步骤，构成系统的安全规则。所以入侵检测系统的主要工作就是在监听到的数据中匹配系统的安全规则，根据发现的入侵情况采取相应的步骤。精确模式匹配技术在此领域得到了广泛的应用，并且精确模式匹配算法已成为影响入侵检测系统性能的瓶颈[14]。
- 内容过滤：串匹配算法广泛应用于内容过滤系统。将串匹配算法应用在对于关键字进行过滤的包过滤系统中是行之有效的。同时，在实现了对普通文本信息进行按照关键字过滤这一特殊功能的基础上，还可以通过对数据包的重组，实现对病毒等其它信息类型的过滤，进而实现病毒防火墙等功能更加完备的过滤系统[16]。
- 生物科学计算：近似模式匹配技术对于生物信息学（Bioinformatics）中海量的 DNA 序列比较与分析有着重要的作用[17]。生物信息学是上世纪 80 年代末随着人类基因组计划（Human Genome Project，简称 HGP）的启动而兴起的一门新的交叉学科，涉及生物学、计算机科学以及应用数学等学科。它通过对生物学实验数据的获取、加工、存储、检索与分析，进而揭示数据所蕴含的生物学意义。HGP 正式启动于 1990 年，其目的在于确定约 30 亿个碱基对的人类基因组完整核苷酸顺序，找出人类全部约 10 万个基因在染色体上的位置以及包括基因在内的各种 DNA 片段的功能，即“读懂”人类基因，破译人类全部遗传信息。2003 年 4 月 14 日，人类基因组计划的所有目标全部实现。面对如此海量的生物信息数据，如何对其进行处理，是生物信息学的中心任务。在生物信息学领域中，由于遗传过程中发生的突变或进化改变等微小因素，使得两个描述同一物质的正确序列数据有可能含有微小的差异。所以传统的精确模式匹配技术不能满足生物计算的需求。为适应生物计算容许一定错误的匹配，串匹配技术中的“近似匹配”获得了空前的发展和广泛的应用。

目前，用于生物信息学中最常用的工具 BLAST 和 FastA 都基于串匹配理论[18]。

- 新闻主题提取：新闻主题提取的传统提取方法主要是依据词典加匹配的模式，由于无法同步于网上新闻中新词汇涌现的速度而且词典的内容也无法完全涵盖网上新闻的范围，因此这种方法不适用于网上新闻的主题提取。串匹配技术应用于新闻主题提取方面，解决了以上问题，以此提出并实现了一种不用词典即可提取新闻主题的新方法，即利用串匹配技术在标题和正文间寻找重复的字串。结果表明该方法能够准确有效地提取出绝大部分网上新闻的主题，满足新闻自动处理的需要[19]。

由于网络内容监管的特定需求，本文主要介绍精确串匹配算法。依据在匹配过程中所要匹配的模式串数量，依次阐述单模式串匹配算法和多模式串匹配算法的发展状况。

最早的单模式串匹配算法是蛮力算法（Brute-Force 算法），该算法的特点是简单、直观，但是对文本串的扫描常常需要回溯，因而效率较低。1970 年，S.A.Cook 在理论上证明了串匹配问题可以在 O(m+n)（m 和 n 分别为模式串和文本的长度）时间内解决。随后 D.E.Knuth 和 V.R.Pratt 仿照 Cook 的证明构造了一个算法。与此同时，J.H.Morris 在研究正文处理时也独立地得到与前述两人本质上相同的算法。这样两个算法殊途同归地构造出当前计算机课本中最为经典的串匹配算法，称为 Knuth-Morris-Pratt 算法[20]（简称 KMP 算法）。1977 年，R.S.Boyer 和 J.S.Moore 两人设计了一个新的串匹配算法 Boyer-Moore 算法[21]（简称 BM 算法），该算法是目前最常用、效率较高的算法之一。在 BM 算法的基础上，又派生出了许多算法，例如：Tuned Boyer-Moore 算法[22]、Turbo BM 算法[23]、Boyer-Moore-Horspool 算法[24]、Boyer-Moore-Horspool-Raita 算法[25]以及其它基于 BM 算法的改进算法[26-28]等。1980 年，Karp 和 Rabin 合作从截然不同于 KMP 算法和 BM 算法的途径研究出一种新算法，称为 Karp-Rabin 算法[29]（简称 KR 算法）。1990 年，Sunday D.M.提出了 Quick Search 算法[30]（简称 QS 算法），该算法实质上是 BM 算法的简化，在实际应用中是一种高效的算法。

多模式串匹配算法中最经典的算法是由 A.V.Aho 和 M.J.Corasick 提出的 DFSA（Deterministic Finite State Automata）算法[31]。该算法通过构造有限自动机来实现匹配，而构造有限自动机的过程其实就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。由此，1993 年 Jang-Jong Fan 和 Keh-Yih Su 二人在 DFSA 算法的基础上并结合 BM 算法，设计出了一种新型的 FS 算法[32]，该算法在平均情况下比 DFSA 算法速度更快[32]。另外，Sun Wu 和 Udi Manber 在基于 BM 算法的框架下派生出了多模式匹配算法，称为 Wu-Manber 算法[33,34]。

## 1.3 研究内容及目标

基于以上的分析，针对网络内容监管现有的识别手段大多采取了基于关键字或简单语义的识别技术，或是采用被动获取信息的方法，由于这些技术方法简单、快速，并可以实现信息过滤的实时性，因而成为网络内容监管的基本手段，但是这些方法都不能从根本上很好的解决网络内容监管的问题。

因此，本文主要研究当采用主动方式获取信息后对其进行分析、处理和识别的工作。考虑到语言形式的多样性，单纯采用对关键字过滤的方式进行信息处理，并不能达到十分理想的效果。另外，由于网络信息流量巨大（根据中国互联网络信息中心（CNNIC）2004 年 1 月的《中国互联网络发展状况统计报告》显示[35]，截止到 2003 年 12 月 31 日，我国国际出口带宽的总容量为 27216M，与半年前相比增加了 8617M，增长率为 46.3%，和去年同期相比增加 190.1%。），仅仅采用语义分析的方法会影响系统的效率，无法实现网络内容监管的实时性。因此，在如此海量的网络信息的条件下，如何提高网络内容监管系统的效率和准确度已迫在眉睫。鉴于国内的网络状况和中文信息处理技术的发展，本文研究一种适合国内环境、信息主动获取、运行速度快、准确性高的网络内容监管系统，并在此基础上，着重分析设计适合于中文环境和网络内容监管的串匹配算法，以此来减轻语义分析的压力，提高系统的效率。

## 1.4 论文的安排

本文的内容总体安排如下：

第一章主要阐述研究网络内容监管的背景和意义，分析网络内容监管的发展现状，并介绍在网络内容监管中常用的串匹配技术的发展情况，最后介绍本文的研究内容及其目标。

第二章主要介绍与网络内容监管相关的串匹配技术，包括单模式串匹配算法和多模式串匹配算法。

第三章主要介绍网络内容监管系统的设计，即基于层次分类的网络内容监管系统，并阐述了串匹配技术在该系统中的作用和地位，由此引出下一章的内容。

第四章设计实现了适合于网络内容监管工作的串匹配算法，即单模式串匹配算法 IQS 和多模式串匹配算法 IWM，并分别对这两个算法进行了实验比较分析。

第五章主要介绍网络内容监管系统的集成实现，包括 IQS 算法和 IWM 算法在系统中的集成，同时介绍了系统的主要功能，最后用真实数据对系统进行实验测试，并给出测试结果。

第六章对本文进行总结，并介绍了以后的研究方向。

# 2 串匹配算法

## 2.1 概述

依据在匹配过程中所要匹配的模式串数量，本章介绍现有主要的几个单模式串匹配算法和多模式串匹配算法。当然，还有其它的算法，例如：单模式串匹配算法中的 Boyer-Moore-Horspool 算法[24]、基于数值的 ShiftOr 算法[36]、Simon 算法[37]，Apostolico-Crochemore 算法[38]、Colussi 算法[39]等，以及多模式串匹配算法中的 Commentz-Walter 算法[40]等。

## 2.2 单模式串匹配

单模式串匹配问题可以描述为：已知 PAT = pat[0...m-1]表示要匹配的模式串，长度为 m，TXT = txt[0...n-1]表示正文文本，长度为 n。模式串和文本串都由字符表 Σ 组成，字符表 Σ 大小为 σ。串匹配的任务就是要在给定的 TXT 中发现所有的 PAT，并报告所有的出现位置。将匹配过程中 PAT 与 TXT 中长度为 m 的子串的一次比较称为一次尝试（Attempt）。在本小节内，我们约定：模式串 PAT ＝zxyzyzyz，文本串 TXT＝zxyaxzxyzyzyzayayxyzayxz，以此作为实验用例。

### 2.2.1 BF 算法

蛮力算法（Brute-Force，简称 BF）是最早最简单的单模式串匹配算法，其匹配思想是把文本串中的每一个字符 txt[i]（i∈[0，n-m]）作为一次匹配的开始，并依次比较 txt[i...i+m-1]与模式串 pat[0...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 txt[i+1]，依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 算法的程序代码如下。

**算法 2.2.1** BF 算法

输入：文本串 TXT 及其长度 n

输出：匹配信息

方法：

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

}

BF 算法的特点是简单、直观，无需预处理过程，因而不需要额外的存储空间，但是对文本串的扫描常常需要回溯，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*m)。

### 2.2.2 KMP 算法

Knuth-Morris-Pratt 算法[20]（简称 KMP 算法）是由 D.E.Knuth、J.H.Morris 和 V.R.Pratt 共同设计出的一种高效的单模式串匹配算法。

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。

假定在某次尝试匹配过程中，当前的匹配窗口为 j，即考察文本串 TXT 中 txt[j...j+m-1]是否和长度为 m 的模式串 PAT 相等。假设在该次尝试过程中，pat[i]和 txt[i+j]发生不匹配，其中 i∈[0,m-1]。那么我们可知模式串 PAT 在位置 i 之前的字符与文本串相应的字符匹配，即 pat[0...i-1]＝txt[j...j+i-1]＝$u$，并且我们假设 txt[i+j]＝x，pat[i]=y，x,y∈Σ，且 x≠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，就是在模式串 PAT 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 pat[i]，即 z≠y，如图 2.2.2.1 所示。

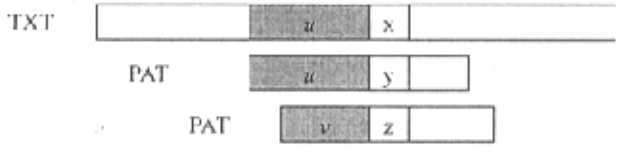



图 2.2.2.1 KMP 算法的跳跃方法

KMP 算法中跳跃表 kmpNext 的预处理如下：

$$kmpNext[i]=\begin{cases}-1 & i=0\\ i-\max\{k \mid k\in[2,i-1], pat[0...k-2]=pat[i-k+1...i-1]\} & \\ 0 & \text{其它情况}\end{cases} \quad (2.2.2)$$

KMP 算法的主要程序代码如下。

**算法 2.2.2** KMP 算法

输入：文本串 TXT 及其长度 n，跳转表 kmpNext

输出：匹配信息

方法：

```
计算 kmpNext;      //预处理
int i=j=0;
while(j<n) {
    Attempts++;     //尝试次数
    while(i>-1 && pat[i]!=txt[j])
        i=kmpNext[i];
    i++;    j++;
    if(i>=m) {   //匹配成功
        记录匹配信息;
        i=kmpNext[i];
    }
}
```

下面，通过一个实例分析 KMP 算法的匹配过程。已知模式串 PAT＝zxyzyzyz 和文本串 TXT＝zxyaxzxyzyzyzayayxyzayxz，其预处理结果如图 2.2.2.2 所示，匹配过程如图 2.2.2.3 所示，阴影部分表示已经比较过的字符。

| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| pat[i] | z | x | y | z | y | z | y | z | |
| kmpNext[i] | −1 | 0 | 0 | −1 | 1 | −1 | 1 | −1 | 1 |

图 2.2.2.2 KMP 算法的预处理结果

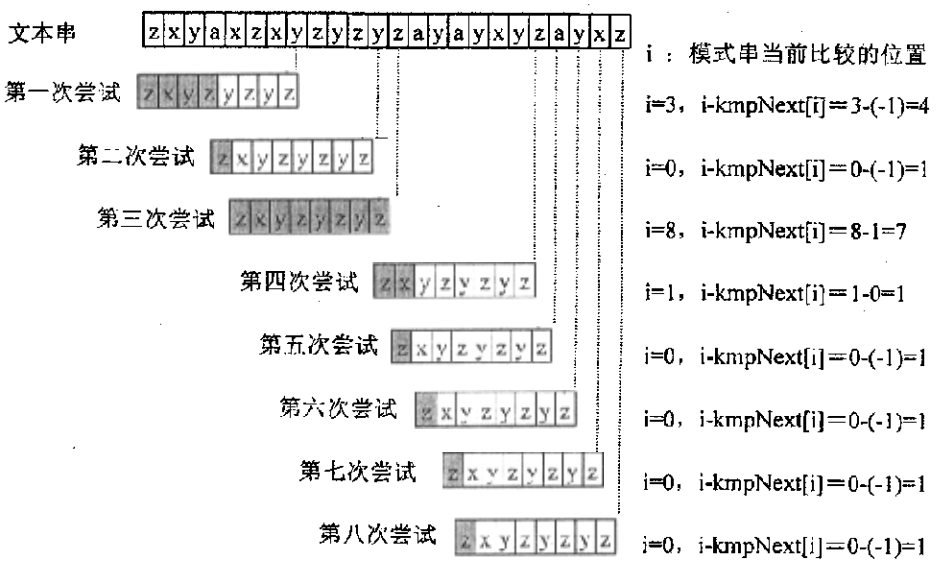



图 2.2.2.3 KMP 算法匹配过程实例

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 Σ 的大小 σ 无关。

### 2.2.3 BM 算法

Boyer-Moore 算法[21]（简称 BM 算法）是由 R.S.Boyer 和 J.S.Moore 两人在 1977 年设计实现的一个串匹配算法，是目前实际应用中最为常用、效率较高的单模式串匹配算法之一。

不同与 BF 算法、KMP 算法等，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（Good-Suffix Shift Table，简称 bmGS）和不良字符跳转表（Bad-Character Shift Table，简称 bmBC），跳跃式的进行处理。BM 算法的最大可能跳转距离为 m。

现在我们具体分析 BM 算法在匹配过程中如何运用 bmGS 和 bmBC 这两个表进行最大可能的跳跃，之后给出其各自定义。假定在某一次尝试过程中，txt[i+j]=x，pat[i]=y（x,y∈Σ，且 x≠y），由于 BM 算法是从后向前比较，因此可知 pat[i+1…m-1]=txt[i+j+1…j+m-1]＝*u*，并且 pat[i]≠txt[i+j]。如图 2.2.3.1 所示。

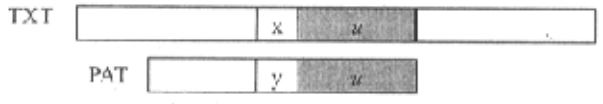



图 2.2.3.1 BM 算法假定的初始情况

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，即可能是一个字符，也可能是多个字符。例如，在图 2.2.3.1 中，当 pat[i]≠txt[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，确切的说是在某次尝试中发生不匹配的字符。例如，在图 2.2.3.1 中，bmBC 是根据文本串 TXT 与模式串 PAT 发生不匹配的位置信息，即文本串中的 txt[i+j]来进行跳跃。具体的跳转情况可分为以下几种：

1) 假如在模式串 PAT 中存在含有子串 *u* 并且其前导字符不为 y 的子串 *zu*，其中 z∈Σ 且 z≠y，则根据 bmGS 进行跳转。如图 2.2.3.2 所示。

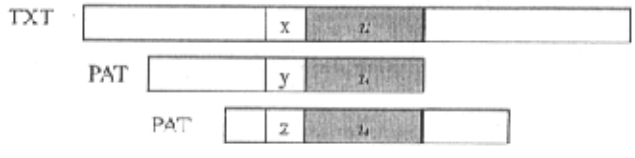



图 2.2.3.2 bmGS 跳跃：当 PAT 中存在前导字符不为 y 的子串 *zu*

2) 假如在模式串 PAT 中不存在上述那种情况，那么在 PAT 中找到 $u$ 的最大后缀 $v$，进行 bmGS 跳转。如图 2.2.3.3 所示。

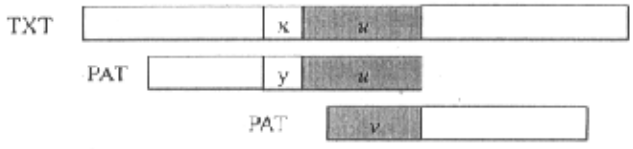



图 2.2.3.3 bmGS 跳跃：当 PAT 中只存在 $u$ 的最大后缀 $v$

3) bmBC跳转则根据不相匹配的字符进行跳转。在模式串 PAT 中找到txt[i+j]字符（即字符 x）在 PAT 中的最右出现位置，然后进行 bmBC 跳转。如图 2.2.3.4 所示。

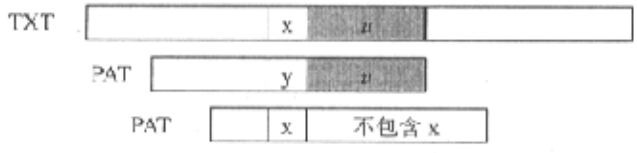



图 2.2.3.4 bmBC 跳跃：当 PAT 中找到 x 最右出现位置

4) 假如文本串中 txt[i+j]字符（即字符 x）在 PAT 中不曾出现，那么 PAT 在 TXT 中可能匹配的位置中一定不包含文本串 TXT 的第 i+j 个字符，因此将 PAT 的第一个字符与 TXT 中的第 i+j+1 个位置对齐，进行下一次匹配。如图 2.2.3.5 所示。

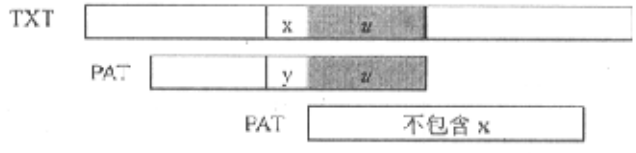



图 2.2.3.5 bmBC 跳跃：当 PAT 不包含字符 x

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 pat[k+i-m+2...k]= pat[i+1...m-1]，且 pat[i]≠pat[k+i-m-1]，即 PAT 中存在含有子串 pat[i+1...m-1]且前导字符不为 pat[i]的子串的最右位置 k。　　(2.2.3.1)

$$bmGS[i]=\begin{cases}\min\{m-k\} & k\text{满足式}(2.2.3.1), i\in[1,m-2]\\ m-1 & i\in[1,m-2], \text{且}k\text{不满足式}(2.2.3.1), \text{或}i=0\\ 1 & i=m-1\end{cases} \quad (2.2.3.2)$$

$$bmBC[c]=\begin{cases}m & c\in\sum \text{且} c\notin PAT\\ \min\{m-i-1\} & pat[i]=c(i\in[0,m-2])\end{cases} \quad (2.2.3.3)$$

BM 算法的主要程序代码如下。

**算法 2.2.3** BM 算法

输入：文本串 TXT 及其长度 n，良好后缀跳转表 bmGS，不良字符跳转表 bmBC

输出：匹配信息

方法：

```
计算 bmGS 和 bmBC;        //预处理
int j＝0;
while (j<=n-m) {
    Attempts++;      //计算尝试次数
    for(int i = m-1; i>=0 && pat[i]==txt[i+j]; i--);
    if (i = =－1) {          //匹配成功
        j+=bmGS[0];
        记录匹配信息;
    }
    else
        j+= max(bmGS[i],bmBC[y[i+j]]-m+1+i);
}
```

下面，以模式串 PAT＝zxyzyzyz，文本串 TXT＝zxyaxzxyzyzyzayayxyzayxz 为例，具体分析 BM 算法的匹配过程。

首先计算出 bmGS 表和 bmBC 表。如下所示：

bmBC[z, x, y, t(t∈Σ 且 t∉ PAT )]＝{2, 6, 1, 8}

bmGS[0...7]={7, 7, 7, 2, 7, 4, 7, 1}

匹配过程如图 2.2.3.6 所示，阴影部分表示已比较过的字符。

BM 算法是实际应用中最常用、最有效的单模式串匹配算法之一，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+σ)，其预处理阶段的时间复杂度为 O(m+σ)，最好情况下 BM 算法的时间复杂度可以达到 O(n/m)。

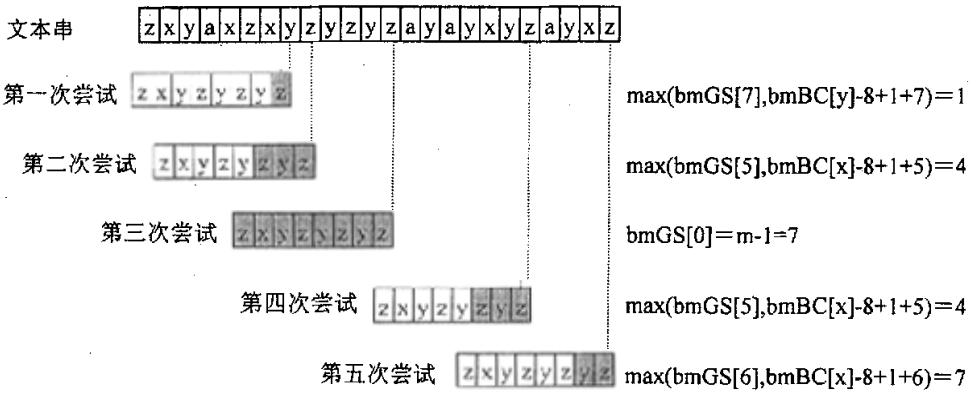



图 2.2.3.6 BM 算法匹配过程实例

## 2.2.4 KR 算法

Karp-Rabin 算法[29]（简称 KR 算法）是由 Karp 和 Rabin 在 1980 年合作从截然不同于 KMP 算法和 BM 算法的途径研究出的一种新算法。

KR 算法的基本思想是：对文本串 TXT 中任意 m 个字符即 txt[j...j+m-1]，以及模式串 PAT 的 m 个字符，分别使用哈希函数并比较其计算结果，如果两值相等，则说明文本串在位置 j 的 m 个字符有可能与模式串匹配，否则，选取文本串下一个位置继续进行比较，直到 j>n－m 结束。

因此，在 KR 算法中，建立一个良好的哈希函数至关重要，除了要具有计算速度快的特点外，还必须对不同的字符串有较好的唯一性，并且通过文本串任意 m 个字符（txt[j...j+m-1]）的哈希值和字符 txt[j+m]，能方便快捷的计算出 txt[j+1...j+m]的哈希值。假设 txt[j...j+m-1]的哈希值为 hash(txt[j...j+m-1])，那么 hash(txt[j+1...j+m])＝rehash(txt[j],txt[j+m], hash(txt[j...j+m-1]))。

在预处理阶段，KR 算法的主要工作是分别计算 hash(PAT)和当 j＝0 时文本串 TXT 中第一次待比较的 m 个字符的哈希值，即 hash(txt[0...m-1])，之后进入匹配阶段。当进行下一次尝试时，计算 hash(txt[j+1...j+m])，即 rehash(txt[j],txt[j+m], hash(txt[j...j+m-1]))。由此，KR 算法的主要程序代码如下。

**算法 2.2.4** KR 算法

输入：文本串 TXT 及其长度 n

输出：匹配信息

方法：

// REHASH：计算下一位置的哈希值

```
#define REHASH(a, b, hash_txt)  ( ( (hash_txt)－(a)*d) << 1 ) + (b) )
//预处理，hash_pat 为模式串 PAT 的哈希值，hash_txt 为当前 TXT 中待匹配的 m 个字符的哈希值
for (d = i = 1; i < m; ++i)
    d = (d<<1);
for (hash_txt = hash_pat = i = 0; i < m; ++i) {
    hash_pat = ((hash_pat <<1) + pat[i]);
    hash_txt = ((hash_txt <<1) + txt[i]);
}
/* 匹配过程*/
j = 0;
while (j <= n-m) {
    if (hash_pat == hash_txt)       //可能存在匹配
        for( int k=0; k<m && pat[k]==txt[j+k]; k++);
        if(k==m)     //匹配成功
            记录匹配信息;
    }
    hash_txt = REHASH(txt[j], txt[j + m], hash_txt);
    j++;
}
```

我们还是以模式串 PAT 和文本串 TXT 为例，具体分析 KR 算法的匹配过程。

首先，可以计算得到模式串 PAT 的哈希值，即 hash(PAT)＝hash(zxyzyzyz)＝30940。之后对文本串 TXT＝zxyaxzxyzyzyzayayxyzayxz 进行匹配，KR 算法的匹配过程如下所示：

第一次尝试：j=0，hash(txt[0…7])=30529≠hash(PAT)；

第二次尝试：j=1，hash(txt[1…8])=29948≠hash(PAT)；

第三次尝试：j=2，hash(txt[2…9])=29297≠hash(PAT)；

第四次尝试：j=3，hash(txt[3…10])=27740≠hash(PAT)；

第五次尝试：j=4，hash(txt[4...11])=30769≠hash(PAT)；

第六次尝试：j=5，hash(txt[5…12])=30940=hash(PAT)。因此将模式串 pat[0…m-1]与 txt[j…j+m-1]分别依次进行比较，结果匹配成功；

……

第十七次尝试：j=16，hash(txt[16…23])=30614≠hash(PAT)。

KR 算法最大的特点是利用哈希函数来决定可能存在匹配的位置，尝试次数为 n－m＋1。KR 算法的空间复杂度为常数，其预处理阶段的时间复杂度为 O(m)，

最好情况下 KR 算法的时间复杂度可以达到 O(n+m)。

### 2.2.5 QS 算法

Quick Search 算法[30]（简称 QS 算法）是由 D.M. Sunday 在 1990 年提出的，在实际应用中 QS 算法是一种高效的算法。

QS 算法实质上是 BM 算法的简化，只使用了不良字符跳转表。在对文本串 txt[j...j+m-1]的一次尝试结束后，不论匹配成功与否，跳转的距离至少为 1。因此，字符 txt[j+m]必然在下一次的尝试之中。所以，可以由字符 txt[j+m]查阅不良字符跳转表 qsBC 来决定跳转距离。假设当前匹配窗口为 j，qsBC 表依据当前匹配窗口的下一个字符即 qsBC[j+m]进行最大限度的跳跃，其中 x=txt[j+m]，则跳跃方式分为两种：

1) 假如模式串 PAT 中不含有字符 txt[j+m]（即字符 x），那么跳跃距离为 m+1。如图 2.2.5.1 所示。

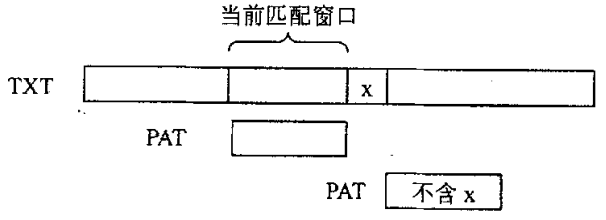



图 2.2.5.1 qsBC 跳跃：当 PAT 不含字符 x 时

2) 假如模式串 PAT 中含有字符 txt[j+m]（即字符 x），那么在模式串 PAT 中找出字符 x 的最右出现位置 i，则跳跃距离为 m－i。如图 2.2.5.2 所示。

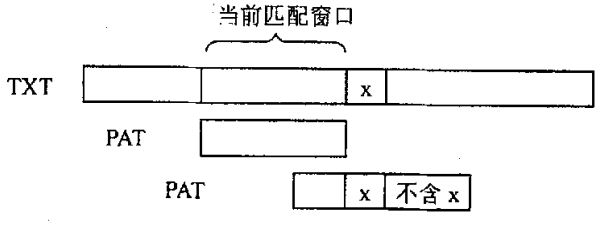



图 2.2.5.2 qsBC 跳跃：当 PAT 含有字符 x 时

QS 算法的不良字符跳转表 qsBC 定义如下：

$$qsBC[c]=\begin{cases} m+1 & c\in\sum \text{且} c\notin PAT \\ \min\{m-i \mid pat[i]=c, i\in[0,m-1]\} \end{cases} \tag{2.2.5}$$

QS 算法的主要程序代码如下。

**算法 2.2.5** QS 算法

输入：文本串 TXT 及其长度 n，不良字符跳转表 qsBC

输出：匹配信息

方法：

```
计算 qsBC;      //预处理
int j=0;
while (j<=n-m) {
    Attempts++;      //计算尝试次数
    for(int i = m-1; i>=0 && pat[i]==txt[i+j]; i--);
    if (i==-1) {      //匹配成功
        记录匹配信息;
    }
    j+= qsBC[y[j+m]];
}
```

下面，还是通过模式串 PAT 和文本串 TXT 为例具体分析 QS 算法的匹配过程。已知模式串 PAT＝zxyzyzyz，在预处理阶段计算出 qsBC 表。如下所示：

qsBC[z, x, y, t(t∈Σ 且 t∉ PAT )]＝{1, 7, 2, 9}

然后进入匹配阶段，QS 算法的匹配过程如图 2.2.5.3 所示。

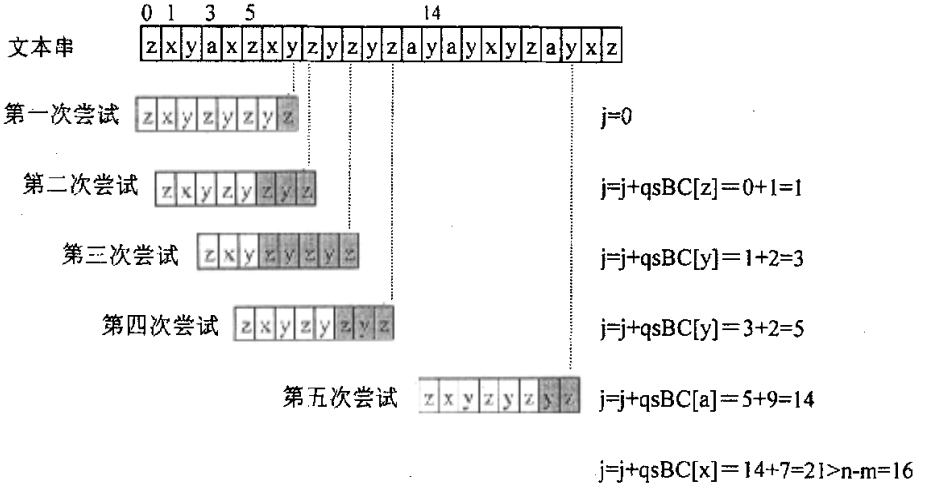



图 2.2.5.3 QS 算法匹配过程实例

QS 算法实质上是 BM 算法的简化，只用了不良字符跳转表，其主要特点是：考察紧邻在当前匹配窗口的下一个字符，利用文本串中未出现在模式串里的字符来加快匹配速度。如果待匹配模式串中未使用的字符大量出现在文本串中，并且

其模式串较短时，QS 算法将会比 BM 算法更快。但对于小字符集语言的字符串匹配时，在非最优情况下，QS 算法的匹配速度慢于 BM 算法。因此在实际应用中，QS 适合于大字符集、模式串较短时的情况。QS 算法的空间复杂度为 $O(\sigma)$，其预处理阶段的时间复杂度为 $O(m+\sigma)$，最好情况下 QS 算法的时间复杂度可以达到 $O(n/(m+1))$。

## 2.3 多模式串匹配

多模式串匹配问题可以描述为：已知 $PAT = \{pat_1, pat_2, \ldots pat_m\}$为一个大小为 m 的模式串集合，minlen 为最短模式串的长度，TXT = txt[0...n-1]表示正文文本，长度为 n，其中每个模式串和文本串都由字符表 Σ 中的字符组成，字符表 Σ 的大小为 σ。多模式串匹配的任务是在文本串 TXT 中发现所有包含于 PAT 中的模式串，并报告出现位置。在本小节内，我们约定：PAT＝{be, eat, beat, bye}，TXT ＝“upbeat”，以此作为实验用例。

### 2.3.1 DFSA 算法

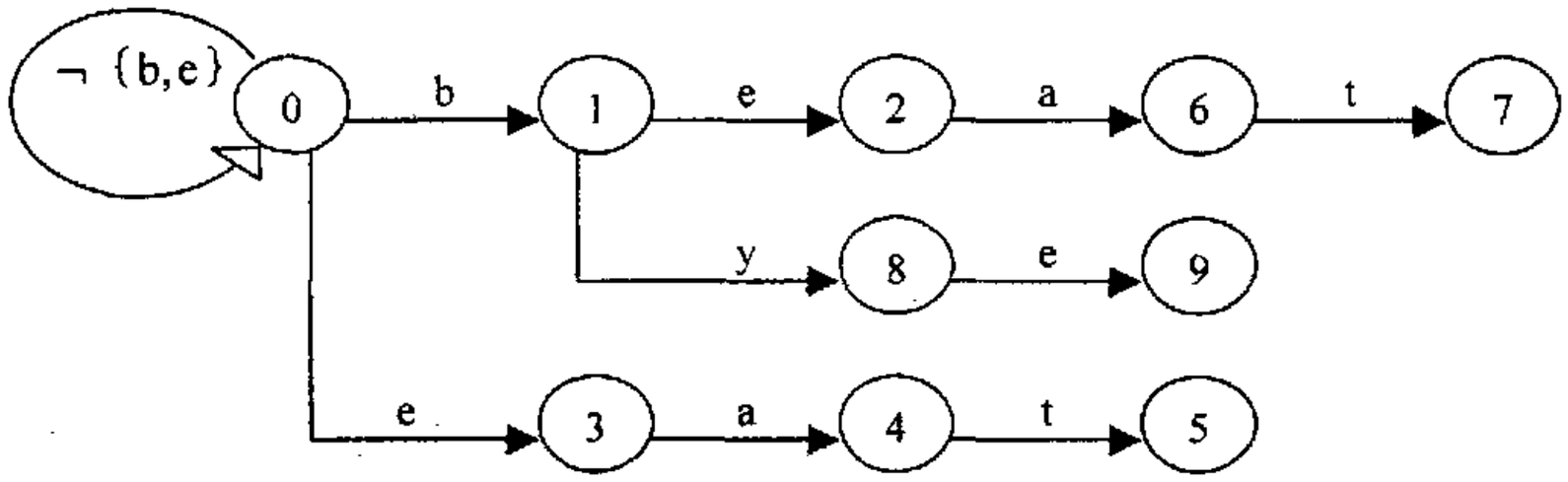
多模式串匹配算法中最经典的算法是由 A.V.Aho 和 M.J.Corasick 提出的 DFSA（Deterministic Finite State Automata）算法[31]，该算法通过构造有限自动机来实现匹配。

DFSA 算法的基本思想是：首先，在预处理阶段通过模式串集合构造有限自动机，而构造有限自动机的过程其实就是将多模式串匹配问题转化为单模式串匹配问题，然后将文本串作为自动机的输入参数进行匹配，对文本的扫描只需一次。有限自动机由一系列的状态组成，每一个状态可以用某个数字来表示。自动机处理文本的过程，实质上就是顺序读入文本中的每个字符，并依据当前所处的状态进行跳转，假如跳转后的状态是终止状态，则输出匹配结果。DFSA 算法主要由三个函数组成：goto 函数，failure 函数和 output 函数。goto 函数负责进行状态的跳转，假如当前状态表示为 cur_state，当前输入的字符为 input_char，则跳转到 goto(cur_state,input_char)；failure 函数负责当 goto 函数的跳转值等于 FAIL 时的状态转移；output 函数记录某一状态是否有匹配结果出现。

我们以模式串 PAT＝{be, eat, beat, bye}为例，构造出的有限自动机如图 2.3.1.1 所示。

在 DFSA 算法中，设定开始状态为 0，goto 函数的取值由当前状态和输入字符决定，或者为 FAIL。在图 2.3.1.1 的(a)图中，从状态 0 到状态 1 的有向边 b 可以表示为 goto(0,b)＝1，并且，除状态 0 外的其它所有状态，没有画出的其它字符的跳转值都为 FAIL，例如：goto(3,x)＝FAIL（$x \in \Sigma$ 且 $x \neq a$）。对于所有的 $x \in \Sigma$，

状态 0 的跳转值 goto(0,x)≠FAIL。failure 函数用于当 goto 函数为 FAIL 时从一个状态跳转到另一个可能存在匹配的状态。例如：当状态 2 的 goto 函数为 FAIL 时，根据已经匹配的 b，e 两个字符跳转到某个可能存在匹配的状态，因此 failure(2)＝3。output 函数用来表示某一状态是否存在某个或多个模式串的匹配。例如：当匹配过程进行到状态 7 时，由 output 函数可知已经匹配的模式串有 beat 和 eat。




(a) goto 函数

| 状态 i | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| failure[i] | 0 | 3 | 0 | 0 | 0 | 4 | 5 | 0 | 3 |

(b) failure 函数

| 状态 i | output[i] |
|---|---|
| 2 | {be} |
| 5 | {eat} |
| 7 | {beat，eat} |
| 9 | {bye} |

(c) output 函数

图 2.3.1.1 DFSA 算法构造的有限自动机 (模式串 PAT＝{be, eat, beat, bye})

假设当前状态为 cur_state，当前输入的字符为 x，state 为自动机某一状态，则 DFSA 算法的匹配过程分下述两种情况：

1) 假如 goto(cur_state, x)=state, 则跳转到状态 state，并且检查 output 函数在此状态是否为空，如不为空，表明有模式串发生匹配。
2) 假如 goto(cur_state, x)==FAIL, 则表明该状态下输入字符 x 发生失败，查找 failure 函数进行跳转，令 state＝failure(cur_state)，则将 state 作为当前状态，x 作为当前输入字符，继续进行匹配。

以 TXT＝“upbeat”为例，分析 DFSA 算法的匹配过程。如图 2.3.1.2 所示。

| 输入字符 | | u | | p | | b | | e | | a | | t | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 状　态 | 0 | | 0 | | 0 | | 1 | | 2 | | 6 | | 7 |
| 是否匹配 | | | | | | | | | Yes | | | | Yes |

图 2.3.1.2 当 TXT＝“upbeat”时 DFSA 算法的匹配过程

DFSA 算法匹配过程的主要代码如下所示。

**算法 2.3.1** DFSA 算法

输入：文本串 TXT 及其长度 n，跳转函数 goto，失效函数 failure 和输出函数 output

输出：匹配信息

方法：

```
int state = 0;       //起始状态
for(int i=0; i < n; i++) {
    while (goto(state, TXT[i])==FAIL)
        state = failure(state);
    state = goto(state, TXT[i]);
    if(output(state)≠EMPTY)     //发生匹配
        记录匹配信息;
}//end for
```

对于串匹配算法而言，在分析时空间复杂度时，主要考察算法的预处理阶段和匹配阶段。在对较大的文本串进行匹配时，与匹配阶段所需的时间相比，预处理阶段所需的时间相对较小，可以忽略不计。并且，在实际应用中，大多数将预处理结果存储为文件格式，匹配时只需载入即可。所以，我们主要关注匹配阶段的时空间复杂度。

DFSA 算法根据自动机原理构造出 goto 函数、failure 函数和 output 函数，goto 函数的大小是状态总数 Num(state)×256，failure 函数和 output 函数的大小都为状态总数 Num(state)，因此 goto 函数的性能直接影响到算法的速度。DFSA 算法的空间复杂度依赖于模式串长度的的总和，因为它决定了自动机的状态总数，时间复杂度为 O(cn)，其中 c 是常数并且与模式串个数无关[31]。

## 2.3.2 FS 算法

FS 算法[32]是由 Jang-Jong Fan 和 Keh-Yih Su 二人在 DFSA 算法的基础上并

结合 BM 算法设计出的一种新型的算法。FS 算法用类似于 DFSA 算法的方法构造有限自动机，然后采用现有的快速单模式串匹配 BM 算法来加快匹配速度，该算法在平均情况下比 DFSA 算法速度更快[32]。

FS 算法结合 DFSA 算法和 BM 算法的思想，但与 DFSA 算法又有所不同，它没有使用 DFSA 算法的中的失效函数 failure，而是通过两个跳跃函数 Skip1 和 Skip2 来进行文本串扫描位置的跳跃。以下是与 DFSA 算法的不同之处：

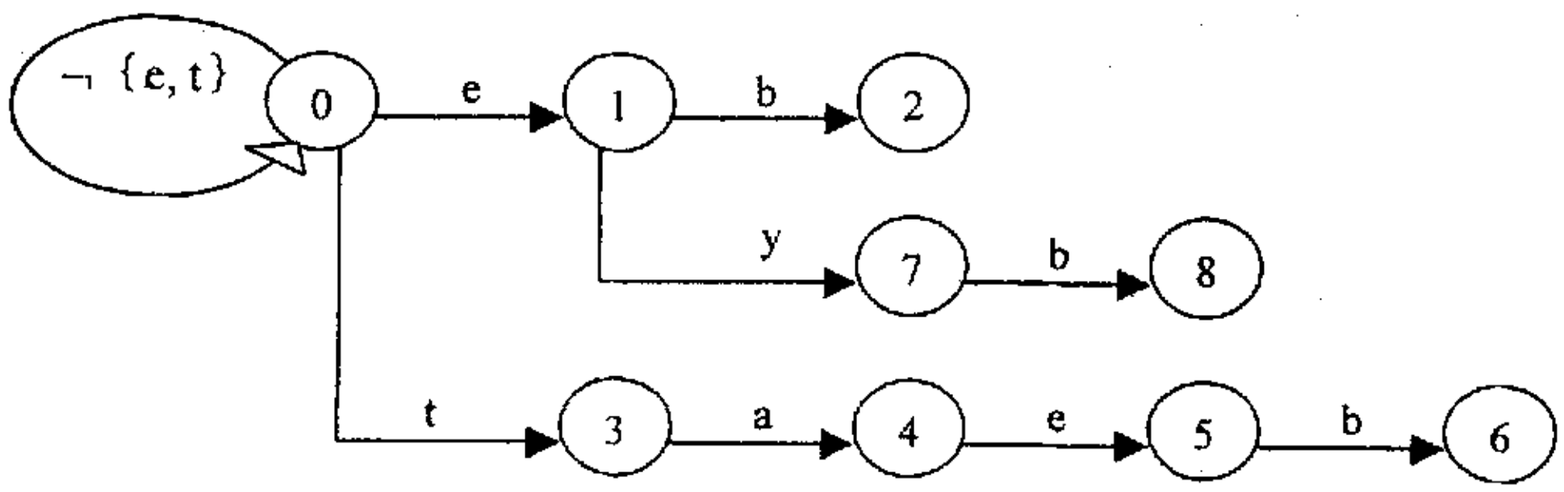
1) 由于 FS 算法在构造好自动机后采用 BM 算法的匹配思想，所以不同于 DFSA 算法的 goto 函数是从模式串的首字符开始，FS 算法的 goto 函数是从模式串的尾字符从后向前构造的。
2) 在开始匹配中，FS 算法从文本串的第 minlen（minlen 为最短模式串的长度）个字符开始匹配，并且采用 BM 算法思想，最大限度的跳过不可能发生匹配的位置，而 DFSA 算法要对文本的每一个字符进行扫描。
3) 在匹配过程中，假如当前状态为开始状态 0 并且通过 goto 函数又跳转到状态 0 时，则访问 Skip1 函数；假如当前状态不为 0 并且通过 goto 函数跳转到状态 0 时，则访问 Skip2 函数。函数 Skip1 和 Skip2 是为了在匹配过程中尽可能的跳过不会发生匹配的字符，加快匹配速度。

FS 算法的 Skip1 函数是指模式串集合中的字符最右出现的位置，Skip2 函数是指根据当前的状态 state 以及已经匹配的字符，向右进行最大可能的跳跃。

FS 算法的匹配思想是在预处理时，首先把模式串集合转换成一个反向的有限自动机，构造出 goto 函数和 output 函数，然后构造 Skip1 和 Skip2 函数。在匹配过程中，由于采用 BM 算法的思想，从输入字符 i=minlen-1 并且状态 state＝0 开始，根据 goto 函数进行状态跳转。如果 goto(state,TXT(i))≠0，当前状态为 state＝goto(state,TXT(i))，并且检测该状态下的 output(state)，判断是否有匹配发生，如果是，则记录匹配信息，然后 i=i－1；如果 goto(state, TXT(i))＝0 并且 state= 0 时，访问 Skip1 函数，文本串跳跃 Skip1(TXT(i))个字符，即 i＝i＋Skip1(TXT(i))，如果 goto(state, TXT(i))＝0 并且 state≠0 时，访问 Skip2 函数，文本串跳跃 Skip2(state, TXT(i))个字符，即 i＝i＋Skip2(state, TXT(i))。

我们还是以模式串集合 PAT＝{be, eat, beat, bye}，文本串 TXT＝“upbeat”为例，说明 FS 算法的匹配过程。图 2.3.2.1 为 FS 算法对模式串 PAT 的预处理结果。FS 算法中 goto 函数和 output 函数与 DFSA 算法的构造方法类似，这里主要以介绍函数 Skip1 和 Skip2。

函数 Skip1 是指模式串集合中的字符最右出现的位置。例如：字符 b 在模式串 be、beat 和 bye 中的最右出现位置分别为 1、3 和 2，函数 Skip1 取其字符出现在模式串中最右位置的最小值，因此，Skip1[b]＝1。凡是不出现在 PAT 中的字符，其 Skip1 值为最短模式串的长度，即 Skip1[x]＝minlen，$x \in \Sigma$ 且 $x \notin$ PAT。




(a) goto 函数

| 字符 char | a | b | e | t | y | 其它 |
|---|---|---|---|---|---|---|
| Skip1[char] | 1 | 1 | 0 | 0 | 1 | 2 |

(b) Skip1 函数

| 状态 i | output[i] |
|---|---|
| 2 | {be} |
| 5 | {eat} |
| 6 | {beat} |
| 8 | {bye} |

(c) output 函数

| 状态 (state) | 字符 (char) | Skip2(state, char) |
|---|---|---|
| 1 | b, y | 0 |
| 1 | Σ－{b, y} | 3 |
| 2 | Σ | 4 |
| 3 | a | 0 |
| 3 | Σ－{a} | 4 |
| 4 | e | 0 |
| 4 | Σ－{e} | 4 |
| 5 | b | 0 |
| 5 | Σ－{b} | 5 |
| 6 | Σ | 6 |
| 7 | b | 0 |
| 7 | Σ－{b} | 4 |

(d) Skip2 函数

图 2.3.2.1 FS 算法的预处理结果（模式串 PAT＝{be, eat, beat, bye}）

函数 Skip2 是指根据当前的状态 state 以及已经匹配的字符，向右进行最大可能的跳跃。例如：在状态 7，由图中可知已经匹配的字符串为 ye，则假如当前输入字符为 b，那么 Skip2 值为 0，继续向左扫描；否则，当前的扫描指针就要向右跳跃，在跳过 ye 两个字符的同时，还要跳过两个字符，因为模式串集合中最短模式串的长度为 minlen＝2，所以安全的跳跃值为 4。

下面以 TXT＝“upbeat”为例说明 FS 算法的匹配过程，如图 2.3.2.2 所示。

FS 算法匹配过程的主要代码如下所示。

**算法 2.3.2** FS 算法

输入：文本串 TXT 及其长度 n，跳转函数 goto，输出函数 output，函数 Skip1

和 Skip2

输出：匹配信息

方法：

```
int i = minlen-1, state = 0;
while (i <= n) {
    Attempts++;      //记录尝试次数
    if (goto(state, TXT[i])==0) {
        if (state==0) {
            i += Skip1(TXT[i]);
        }
        else {
            i += Skip2(state, TXT[i]);
            state = 0;
        }
    }
    else {
        state = goto(state, TXT[i]);
        if (output[state]≠EMPTY)    //匹配成功
            记录匹配信息;
            i--;
        }//end of if (goto(state, TXT[i])==0)
    }//end of while
```

| 文本串(TXT) | 扫描指针 | 当前状态 | goto | i=i+Skip1 | i=i+Skip2 | i=i-1 | output |
|---|---|---|---|---|---|---|---|
| upbeat | 1 | 0 | 0 | 1+2 | | | |
| upbeat | 3 | 0 | 1 | | | 2 | 空 |
| upbeat | 2 | 1 | 2 | | | 1 | be |
| upbeat | 1 | 2 | 0 | | 1+4 | | |
| upbeat | 5 | 0 | 3 | | | 4 | 空 |
| upbeat | 4 | 3 | 4 | | | 3 | 空 |
| upbeat | 3 | 4 | 5 | | | 2 | eat |
| upbeat | 2 | 5 | 6 | | | 1 | beat |
| upbeat | 1 | 6 | 0 | | 1+6 | | |

图 2.3.2.2 FS 算法的匹配过程（文本串 TXT＝upbeat，模式串 PAT＝{be, eat, beat, bye}）

FS 算法结合 DFSA 算法和 BM 算法的思想，其算法性能高于 DFSA 算法，这是因为 FS 算法不必对文本串中的每个字符逐个进行扫描[32]。FS 算法在预处理阶段构造出 goto 函数、output 函数、Skip1 函数和 Skip2 函数，goto 函数和 Skip2 函数的大小都是状态总数 Num(state)×256，output 函数的大小为状态总数 Num(state)，Skip1 函数的大小为字符表 Σ 的大小 σ。FS 算法的时间复杂度在最好情况下为 O(n/minlen)，在最差情况下为 O(n*maxlen)，其中 maxlen 为最长模式串的长度。

### 2.3.3 Wu-Manber 算法

Wu-Manber 算法[33]是由 Sun Wu 和 Udi Manber 在基于 BM 算法的框架下派生出的一种多模式匹配算法，该算法继承 BM 算法的不良字符跳转表的思想，但与 BM 算法不同的是，Wu-Manber 算法依据长度为 B 的字符块进行跳转，B 的实际取值为 2 或 3。

在预处理阶段，Wu-Manber 算法构造三张表：SHIFT、HASH 和 PREFIX。假设当前窗口的最后 B 长度的字符块为 Block[1...B]，其哈希值为 h。SHIFT[h]表示可以安全跳跃的字符数，HASH[h]表示所有模式串最后 B 个字符的哈希值为 h 的模式串链表，前缀表 PREFIX 的大小等于模式串个数，每个模式串的前缀值由其长度为 2 的前缀字符块生成，PREFIX 表的作用在于过滤具有相同 HASH 值而具有不同前缀的模式串，以此尽量减少需要实际进行比较的次数。

Wu-Manber 算法按照以下规则生成进行跳跃表 SHIFT：

1) 如果 Block 不出现在任何关键字中，则 SHIFT[h] = minlen-B+1。
2) 如果 Block 出现在某些关键字中，且在所有模式串中最右的结束位置为 q，则 SHIFT[h] = minlen-q。

按照该规则，当 Block 出现在某些模式串的末尾，即 q = minlen 时，此时 SHIFT[h]被设置为 0，且 HASH[h]非空。事实上只有这些模式串才可能与文本串的当前位置匹配成功，因此 Wu-Manber 算法将 SHIFT 表取值为零作为进一步比较的入口。

Wu-Manber 算法的匹配过程如下：

1) 使用当前窗口的最后 B 个字符计算散列值 h。
2) 如果 SHIFT[h] > 0，跳跃相应距离并转第 1 步继续；否则转 3。
3) 计算当前匹配窗口的前缀的散列值 prefix_hash。
4) 对于 HASH[h]指向的列表中的每个指针，检查是否等于 prefix_hash。如果相等，将其模式串与文本做比较，如果发现匹配则记录匹配信息。
5) 将当前窗口向后跳跃一个字符，转 1 继续进行匹配，直到扫描完全部文本串。

Wu-Manber 算法匹配过程的主要代码如下所示。

**算法 2.3.3** Wu-Manber 算法

输入：文本串 TXT 及其长度 n，SHIFT 表，HASH 表，PREFIX 表

输出：匹配信息

方法：

```
while (textbegin <= textend) {
    Attempts++;      //记录尝试次数
    计算当前长度为 B 的字符块的值 hashblock;
    shift = SHIFT[hashblock];     //查找不良字符块的跳转表
    if(shift == 0) {        //当前字符块出现在某个模式串的末尾
        p = HASH[hash];     //取出可能匹配的模式串
        while(p) {
            结合 PREFIX 表提供的信息，然后判断是否匹配;
            p = p->next;      //继续取出可能存在匹配的模式串
        }
        shift = 1;
    } //end of if
    textbegin = textbegin + shift;      //跳转至下一个可能匹配的位置
} //end of while
```

Wu-Manber 算法由于采用字符块技术，降低了部分匹配的可能性，增加了直接跳跃的机会，而且采用散列技术和前缀表进一步减少了进行实际匹配的次数，使得算法获得了很高的实际运行效率。Sun Wu 的实验表明，其算法的速度高于先前的 egrep、fgrep、GNU-grep 和 gre 算法，并成为 agrep 实用工具的一部分[33]。

# 3 网络内容监管系统概述

## 3.1 项目背景

本课题源于新闻出版总署网络内容监管系统的实际工程项目。新闻出版总署是负责管理国家新闻、报刊、出版、印刷、版权、发行等工作的政府机构。在新闻出版总署众多的职责中，与网络内容监管相关的主要有以下几条[41]：

1) 对新闻出版活动（包括出版物的出版、印刷、复制、发行、进出口贸易等）实施监督管理；查处或组织查处违禁出版物和出版、印刷、复制、发行、进出口单位的违规活动。
2) 审核互联网从事出版信息服务的申请，对互联网出版信息内容实施监督管理。
3) 拟定出版物市场“扫黄打非”的方针、政策和计划并指导实施；查处或组织查处非法出版物和非法出版活动；组织、协调各有关部门和地方的“扫黄打非”工作；组织、协调、指导“扫黄打非”集中行动和大案要案的查处工作。

在我国社会主义市场经济体制改革不断推进和加入世贸组织、对外开放日益扩大的新形势下，新闻出版管理部门面临着由行业管理向社会管理的职能转变，这就要求新闻出版总署必须从思想观念、管理职能、任务重点和工作作风等方面有一个大的转变。在新的形势下，新闻出版总署需要处理的信息量的增加、监管范围的扩大以及面向公众服务需求的增加，使得新闻出版总署的工作日益繁重。

正是为了适应上述形势的发展和净化互联网的环境，国家新闻出版总署提出要建设网络内容监管系统，以此杜绝互联网非法出版的现象，包括黄色信息、反动信息等。国家新闻出版总署预期的项目投资大概是 1.2 亿元人民币。在建设该项目前，出版总署会用 300 万建设一个小型的监管系统，通过这个系统的实际运行情况来论证整个项目的可行性。

基于以上的情况，我们决定开发网络内容监管的原型系统。该系统由中科院华建集团和中国科学院计算机语言信息工程研究中心开发，利用其自身先进的自然语言理解技术、中文信息处理技术以及相应的研究成果，开发具有实用性的商业软件，在准备新闻出版总署网络内容监管的原型系统过程中，除了满足上述几点要求外，还希望在这个系统的基础上做一些包装，以便以后可以推出关于网络内容监管方面的产品。

## 3.2 需求分析

新闻出版总署要求对指定的网站进行监管，每个网站的页面数量假设平均在2000个左右，其具体需求如下：

1) 处理周期要短，要求系统一两天的处理量为100～200指定的网站。
2) 对于非法信息，用户可以自己定义。定义的方式：用户只要输入一定量的训练样本就可以得到非法信息的特征，从而对非法信息进行过滤。
3) 用户可以修改、添加和删除所要监管的对象，并可以通过手动或自动的方式开启或关闭监管操作。
4) 系统提供人机交互的方式。对于一些可疑的信息，可以让工作人员进行进一步的确认。
5) 提供工作人员查询的手段。
6) 工作人员可以通过各种条件查询历史记录。
7) 对于一些确认的可疑信息，计算机主动向工作人员报警，由工作人员做进一步处理。

为适应信息量不断增加的需求，新闻出版总署要求系统能够主动的获取网络中的信息，在保证监管速度的同时，最大限度的提高信息识别准确率。对于计算机无法判别的信息，应及时提示工作人员，并且对于确认后的非法信息应做保存，为日后的行政处罚提供相应的法律证据。同时，保存历次监管的信息，以为工作人员日后查阅和处理。

## 3.3 系统设计目标

当前，大部分中文网络内容监管系统为了最大限度的实现识别过滤的实时性和降低投入，只采用相对简单的识别分析技术，因此，很容易导致较高的误报率，敏感信息的识别率并不是太高，这种类型的系统远远不能满足实际的需要。因此，针对新闻出版总署网络内容监管的需求，该系统应满足以下要求：

1) 不能给网络速度造成影响。由于主动去获取存在于特定区域网络中的内容，监管服务器所在的位置可以能够访问互联网的任何一个地方，只要能够保证足够的接入带宽和访问速度，而不要求放置于骨干网的主要节点。监管服务器的故障也不会给网络造成任何影响。
2) 监管方式的灵活性。这样就完全避免了对相同网站上相同内容的多次重复分析，而且还可以定制一些时间规则，只对某个时间之后发布的内容或上次分析之后的“新”内容进行监管。这种选择性的监管将使得遍历某个区域网络内容的周期大大缩短，总体效率要高出很多。
3) 系统的高效性及信息识别的准确性。采取主动的方式进行监管，不但可

以在获得大块内容甚至整个文档之后在对之进行分析，而且还可以采用更加高级准确的自然语言理解技术分析得到的内容，获得更加良好的效果。

4) 信息识别的可靠性。针对网络信息的特点，我们采用了计算机识别与人工识别相结合的方式，即对含有敏感信息关键字的某一特定信息，当计算机无法识别的时候，可标记为“可疑信息”，并提示用户或工作人员做进一步处理。

5) 要对存在“非法信息”或“可疑信息”的网页进行本地存储，为以后的经济或行政处罚提供有力证据。

6) 安全性高。监管服务器可以位于某一个“安全”的中心，通过一定安全保障途径确保不被攻击。

7) 易于管理。由于监管服务器的分布比较集中，而且每个服务器都可以单独停机而不对网络造成任何影响，因此可管理性高。

8) 能够过滤加密协议。对于通过加密协议如 SSL 发布的网站内容，只要在网页获取的时候采用对应的协议，就能获取到真实内容。然后采用同样的分析手段，就可以对内容进行分析和判断。

9) 智能选择性。可以对某些非法的网站使用事后处理的办法，减少对正常网页的异常阻断。

另外，系统要能够快速、准确的对网络信息进行监管。如果系统不能快速的进行网络内容的监管，或是系统响应时间过长，都会降低系统效率，并且很有可能，一些非法信息已经造成了不可挽回的损失。当系统发现某一非法信息后，要求立刻采取处理措施，把该非法信息所造成的破坏降到最低限度。

## 3.4 总体设计与系统架构

由于现有的网络内容监管系统大部分只采用了相对简单的识别分析技术，来提高系统的运行效率，因此导致信息的识别率不是很高，而且大多都采用被动的方式获取信息，即对流过服务器的信息流进行识别和过滤。为此，针对新闻出版总署的特定需求，本文采用基于层次分类的方法来设计实现网络内容监管系统，采取两层过滤的方法。如图 3.4.1 所示。

第一层采用简单的识别分析技术，如串匹配技术。该层的特点是处理速度较快，但信息的识别准确率较低，因此可以作为第一层处理，为后继的信息识别工作减轻压力，提高系统效率。

第二层采用精确的语义分析技术，分析判断出网络中的非法信息，对于无法识别判断的信息，可标识为可疑信息并提示工作人员以做进一步处理。该层的特点是分析处理速度相对较慢，但是可以获得较高的信息识别准确率。

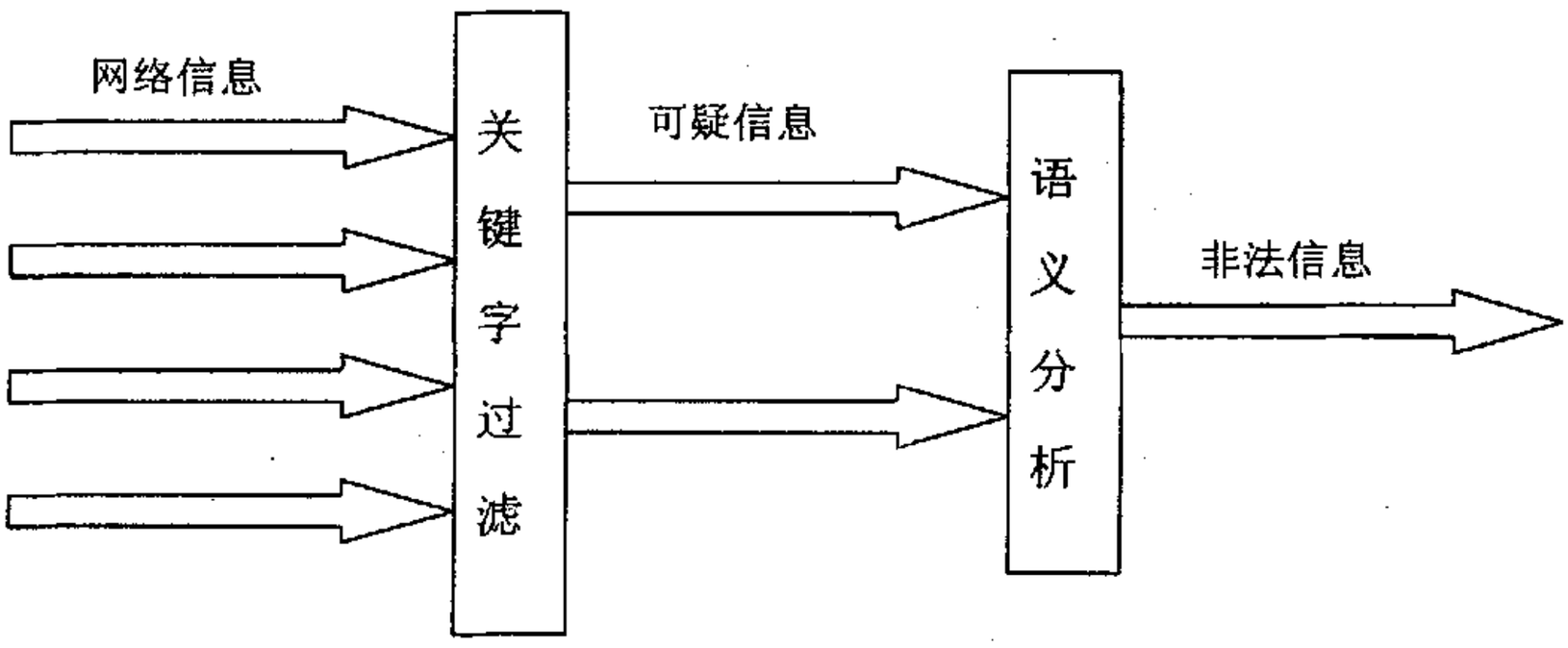



图 3.4.1 网络内容监管系统的信息识别模型

基于层次分类的网络内容监管系统，能够最大限度的提高系统的运行速度，其中的关键字过滤模块实质是一个串匹配过滤模块，将不含指定关键字的信息过滤掉，然后将含有关键字的网络信息提交给语义分析模块处理，当语义分析模块对某一信息无法判断时，可将该信息标识为可疑信息并提示工作人员作进一步处

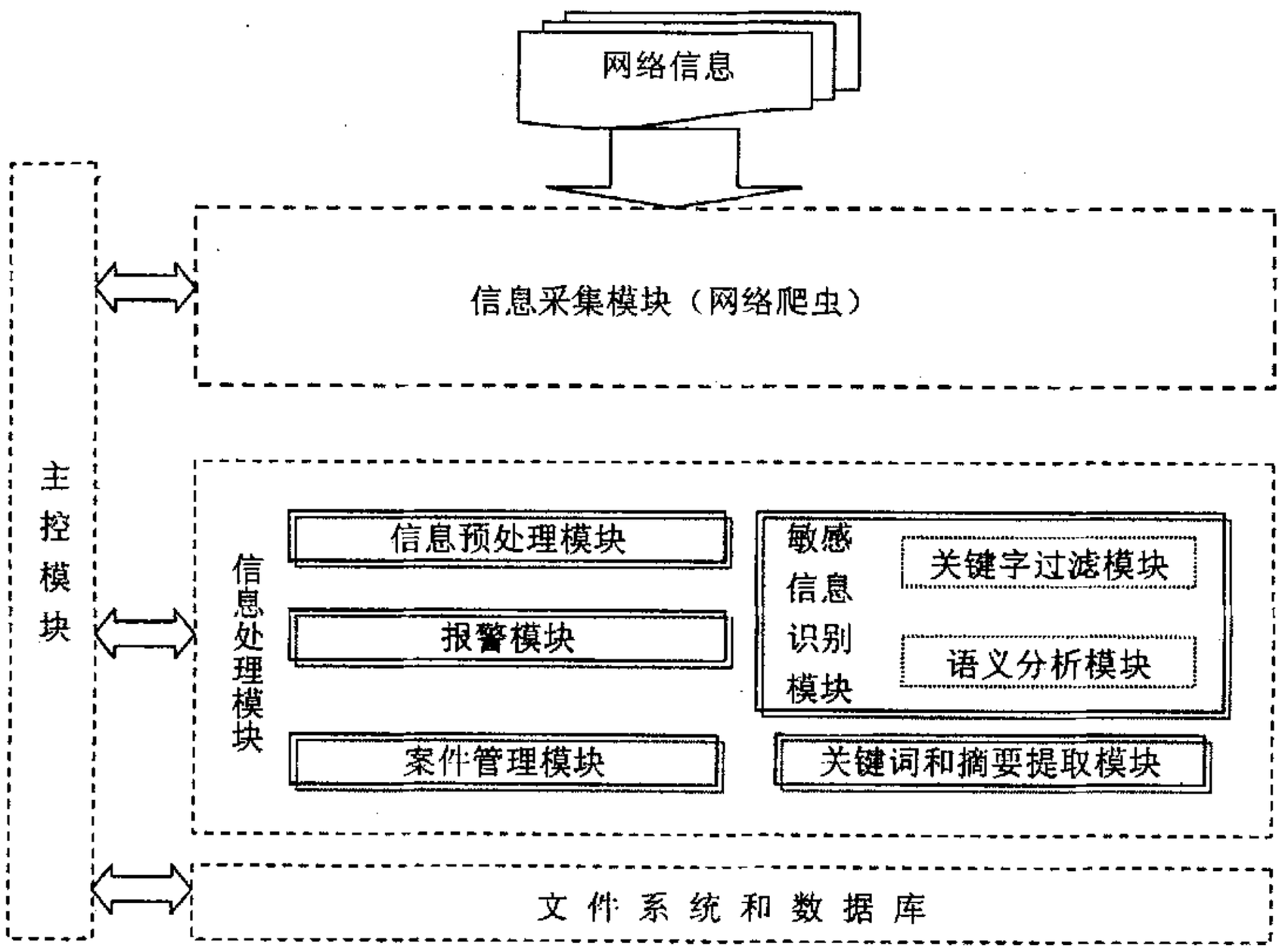



图 3.4.2 网络内容监管系统总体结构

理。这样做可以在提高系统吞吐量的同时，还使得系统具有较高的信息识别率。图 3.4.2 是该系统的总体结构图。

## 3.5 功能模块说明

从图 3.4.2 中我们可以看出，网络内容监管系统主要由四个部分组成：

1)　主控模块

主控模块的主要功能是协调信息采集模块和信息处理模块协同工作，传送两者之间的接口参数，负责对文件系统和数据库的操作，并在终端上显示用户所关心的信息。

2)　信息采集模块（网络爬虫）

针对新闻出版总署的需求，信息的获取要采取主动的方式，因此信息获取模块的主要功能是从指定的网站主动的获取信息。具体来说是由主控模块调用，并传入所要监管网站的相关信息，然后信息采集模块主动获取目标网站的每个页面，并将抓取回来的网页提交给主控模块以做相应处理操作。

除此之外，信息采集模块还要满足以下要求：

- 多线程的爬虫模式，线程个数可以由用户设定。由于过于频繁的访问网站，会给网站造成比较大的压力，所以我们必须限定爬虫的最大线程数量。同时为了保证爬虫的速度，最大限度的利用现有的网络资源，我们又要尽可能多的扩大线程数量，认为爬虫线程在 20～50 之间比较合理。
- 爬虫爬行的深度可以由用户指定，默认的选择是遍历整个网站。
- 用户可以控制爬虫的进度：开启爬虫线程、结束爬虫线程、暂停爬虫线程、恢复爬虫线程。
- 在爬虫爬行过程中很可能出现一些问题，比如超时、IP 地址转移、主机不存在等，此时要求要有相应的处理策略。
- 网页更新判别：如果目标网页在上次搜索后更新过，则要对网页进行进一步的处理，否则不予处理。
- 能够处理多种文件格式：包括 html、shtml、txt、doc、pdf 等文件。
- 是否遵守 robot.txt 文件。
- 对网页中本域名外的连接的处理：不处理还是只处理一层。
- 保证爬虫在同一 IP 地址或者是同一域名中遍历。
- 文件下载的层次：同一网站的所有网页下载于同一目录中还是在网站的根目录下开启不同的文件夹，默认为同一文件夹。

3)　信息处理模块

信息处理模块负责对信息采集模块从网络中获取来的信息进行分析、处理和识别，其中包含以下模块：

- 信息预处理模块：负责网络信息的格式转换；
- 报警模块：当某一信息经敏感信息识别模块识别后被认为是非法信息或是可疑信息时，报警提示工作人员；
- 案件管理模块：对于发现的非法信息，及时备案并协助工作人员组织相关的法律证据等；
- 关键词和摘要提取模块：当用户人员对其历史信息进行复查或是对可疑信息进行分析处理时，为提高用户的工作效率，该模块负责提取网页内容的关键词和摘要；
- 敏感信息识别模块：它是信息处理模块中最为核心的部分。该模块分为关键字过滤模块和语义分析模块。具体来讲，对某一网页信息首先经关键字过滤模块处理，将不含用户指定关键字的网页信息过滤掉，然后把含有用户指定关键字的信息提交给语义分析模块进行处理。如果语义分析模块无法判定是否为非法信息，则可标记为可疑信息并提示工作人员以做人工判断。

4) 文件系统和数据库

文件系统主要存储非法信息和可疑信息的原始文件，为日后的行政处罚做法律证据。数据库主要存放与监管工作相关的信息。

## 3.6 串匹配技术在网络内容监管系统中的应用

从网络内容监管系统总体结构图可知，关键字过滤模块在网络内容监管中有着十分重要的地位，它的性能好坏会直接影响语义分析模块的分析工作。设计一个性能较好的串匹配算法不仅能减轻语义分析模块的压力，加大系统的吞吐量，提高系统的运行效率，在某种程度上更能体现出基于层次分类的网络内容监管系统的优越性，同时也能适应非法信息多变的特点。因此，在下一章里我们主要设计一种适合于网络内容监管的多模式串匹配算法，以此作为关键字过滤模块的核心内容。

此外，在对新闻出版总署网络内容监管的需求进行详细分析之后，发现用户在将非法信息存储在本地后（目的是为了提供法律证据）会关心某个关键字在这些非法信息中出现的情况，或是在某个非法信息中某个关键字出现的情况。为此，我们也需要设计一个单模式串匹配算法。在下一章里，我们将具体阐述上述这两个算法的设计过程。

# 4 串匹配算法的设计与实现

## 4.1 设计考虑

从上一章可以看出，串匹配技术在网络内容监管系统中有着十分重要的作用，它已成为影响网络内容监管系统性能的瓶颈。因此，面对网络中的海量数据，要想提高网络内容监管系统的性能和效率，设计一个快速准确的串匹配算法是最基本、最重要的条件之一。本文在设计面向网络内容监管的串匹配算法时主要遵循以下几条原则：

1) 在保证串匹配算法准确性的同时，最大限度的提高运行效率。这是由网络内容监管自身的特点所决定，主要原因是由于网络中的数据量十分庞大，网络内容监管系统要求能够快速准确的查找出用户所指定的多个关键字，并提供相关的匹配信息。这也是串匹配算法最基本的一条。
2) 算法设计要考虑到中文大字符集这一情况，分析当前的串匹配算法哪个更适合于中文环境或是改进后更能符合网络内容监管这一需求。
3) 在网络内容监管工作中，与网络中的海量数据相比较，非法信息内容相对来说比较小。
4) 设计出的算法要提供良好的接口，以便于系统的集成以及今后的版本升级。在实际的项目中，串匹配算法最终以动态链接库的形式提供给系统调用。

在对网络内容监管的需求进行分析之后，我们提出设计一个单模式串匹配算法和一个多模式串匹配算法的方案。多模式串匹配算法主要针对敏感信息的识别而设计，过滤掉那些不含指定关键字的信息内容，以此为后继的敏感信息识别工作减轻压力，进而提高系统的效率；而单模式串匹配算法主要面向用户人员，为满足日后的查阅工作而设计。以下两节主要阐述这两个算法的设计过程以及实验结果。

## 4.2 单模式串匹配算法 IQS

### 4.2.1 现有算法分析

在众多的单模式串匹配算法中，BM[21]算法和 QS[30]算法被认为具有最高的效率[42]。BM 算法采用从后向前的比较方式，通过良好后缀跳转表（Good-Suffix Shift Table）和不良字符跳转表（Bad-Character Shift Table）进行跳跃式的扫描，最大可能跳转距离为 m，在小字符集环境并且模式串自身有部分匹配的情况下，

BM 算法的工作效率最高。QS 算法只使用了不良字符跳转表（Bad-Character Shift Table），并且考察紧邻在当前匹配窗口的下一个字符，利用文本串中未出现在模式串里的字符来加快匹配速度，最大可能跳转距离为 m+1。因此在实际应用中，QS 算法更适合于待匹配模式串中未使用的字符大量出现在文本串中的情况，在大字符集、模式串较短的环境下工作效率最高。

### 4.2.2 IQS 算法

基于以上对 BM 和 QS 两种算法的分析，结合网络内容监管和中文大字符集的情况，QS 算法更适合于网络内容监管的需求。而且，在大字符集、短模式串的情况下，QS 算法的实际运行效率要优于 BM 算法及其 BM 算法的派生算法[43]。尽管 QS 算法适合于短模式串的情况，但考虑到中文属于大字符集和监管的需求，文本串中出现模式串里的字符的概率要比西文小字符集情况小的多，进而就可以更好发挥出 QS 算法的最大跳转距离为 m+1 的特点，以此缩短文本串的扫描时间，提高系统性能。

#### 4.2.2.1 算法描述

本文提出的单模式串匹配算法基于 QS 算法改进，为方便叙述，记为 IQS（Improved Quick Search）算法。

IQS 算法继承 QS 算法的匹配思想，并在此基础上采用双向扫描的方法，其匹配思想是：设有两个指向文本串的指针 j 和 k，指针 j 从文本串的起始位置 txt[0] 开始扫描，指针 j 自左向右的扫描，其扫描方式与 QS 算法相同；而指针 k 从文本串的第 n-m+1 个字符（即 txt[n-m]）开始扫描，指针 k 自右向左的扫描，其扫描方式与 QS 算法类似，当 j>k 时匹配结束。

接下来，我们具体说明指针 k 的扫描方式。指针 k 与指针 j 的扫描思想类似，都是依据模式串中的字符进行跳跃，最大跳跃距离都可以达到 m+1，不同之处在于：

1) 指针 k 跳跃时依据的是字符在模式串中的最左出现位置，而不是最右出现位置。
2) 某次尝试结束后，指针 k 依据紧邻在当前窗口左边的字符进行跳跃，而不同于指针 j 要考察的紧邻在当前窗口的右边字符。

当对文本串进行匹配时，QS 算法从文本串的起始位置开始扫描。当文本串起始部分的字符大量出现在模式串中时，QS 算法的扫描速度会减慢，而采取双向扫描方法的 IQS 算法，就可以尽量避免上述情况的发生，从而可使得两个扫描指针 j 和 k 尽快出现 j>k 的情况，以此尽早结束扫描。

以下具体阐述 IQS 算法的预处理过程、匹配过程及其时空间复杂度分析。

#### 4.2.2.2 预处理过程

IQS 算法在预处理阶段构造两个表 iqsBc 和 iqsBc1，其中 iqsBc 表与 QS 算法在预处理阶段所构造的表相同，即以模式串中字符的最右出现位置并考察紧邻在当前窗口右边的下一个字符来构造。iqsBc1 表的构造与 iqsBc 表类似，不同之处是以模式串中字符的最左出现位置来构造，并考察紧邻在当前窗口左边的下一个字符。

iqsBc 表的构造与 QS 算法相同，可参照文献[30]或本文 2.2.5 节的内容。下面主要介绍 iqsBc1 表的构造。假设当前扫描窗口为 txt[k...k+m-1]，无论本次尝试匹配成功与否，由于跳跃距离至少为 1，因此考察紧邻在本次窗口的左边的字符进行跳跃，即 iqsBc1 表依据 txt[k-1]进行最大限度的跳跃，其跳跃方式分为两种：

1) 假如模式串 PAT 中含有字符 txt[k-1]（即字符 x），那么找出字符 x 在模式串 PAT 中的最左出现位置，然后将此位置与文本串中 txt[k-1]的位置对齐再进行匹配。跳转如图 4.2.2.2.1 所示。

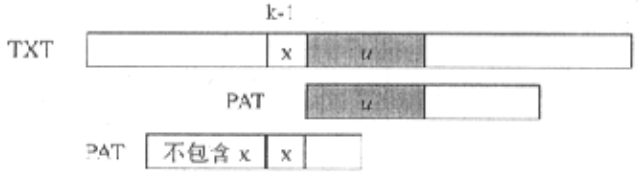



图 4.2.2.2.1 IQS 算法中 iqsBc1 表的跳跃：当 PAT 含有字符 x

2) 假如模式串 PAT 不含有字符 txt[k-1]（即字符 x），则向左跳转的距离为 m+1。如图 4.2.2.2.2 所示。

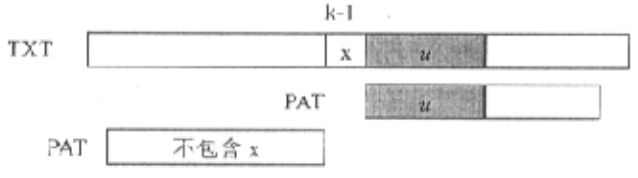



图 4.2.2.2.2 IQS 算法中 iqsBc1 表的跳跃：当 PAT 不含有字符 x

由此 iqsBc1 表的预处理定义如下：

$$iqsBc1[c]=\begin{cases} m+1 & c\in\sum 且 c\notin PAT \\ \min\{i+1 \mid pat[i]=c,\ i\in[0,m-1]\} \end{cases} \quad (4.2.2.2)$$

以下给出 iqsBc 表和 iqsBc1 表的构造过程。

iqsBc 表的构造分为两步：

1) 对于大小为 σ 的字符表 Σ，所有字符的 iqsBc[c]=m+1，其中 c∈Σ。
2) 从左向右依次考察模式串 PAT 中出现的字符，假设某个字符的位置为 i，则 iqsBc[pat[i]]=m–i，其中 i∈[0,m-1]。

iqsBc1 表的构造也分为两步：

1) 对于大小为 σ 的字符表 Σ，所有字符的 iqsBc1[c]=m+1，其中 c∈Σ。
2) 从右向左依次考察模式串 PAT 中出现的字符，假设某个字符的位置为 i，则 iqsBc1[pat[i]]=i+1，其中 i∈[0,m-1]。

#### 4.2.2.3 匹配过程

IQS 算法采用双向扫描的方法，扫描指针分别为 j 和 k，其匹配阶段的主要步骤如下：

1) 指针 j 从文本串的起始位置开始扫描，如若发生匹配，记录相关信息。本次扫描结束后依据当前尝试窗口的下一个字符 txt[j+m]并结合 iqsBc 表进行跳跃，即 j=j+iqsBc[txt[j+m]]。
2) 指针 k 从文本串的第 n-m+1 个字符开始扫描，如若发生匹配，记录相关信息。本次扫描结束后依据紧邻在当前尝试窗口左边的字符 txt[k-1]并结合 iqsBc1 表进行跳跃，即 k=k－iqsBc1[txt[k-1]]。
3) 判断 j 和 k 是否满足关系 j≤k，如果满足转到 1 继续扫描，否则结束。

#### 4.2.2.4 时空间复杂度分析

IQS 算法基于 QS 算法的思想，采用了双向扫描的方法，即两个扫描指针分别从文本串的开始部分和结束部分进行扫描，因此，在预处理阶段的 IQS 算法的时间复杂度为 O(2σ)，空间复杂度为 O(2σ)，在匹配阶段，IQS 算法的时间复杂度与文本串长度 n 和模式串长度 m 有关，在理想情况下每次的跳跃距离为 m+1。IQS 算法的程序代码见附录 A。

### 4.2.3 实验与分析

在评价算法性能时，考虑到算法的实际运行情况比较复杂，并且算法的好坏

只是相对某个环境而言，因此本文所提出的 IQS 算法和下一节将要介绍的 IWM 算法只是针对网络内容监管这一需求，并且适合于中文大字符集和模式串较长且出现部分匹配的概率较小的环境。

#### 4.2.3.1 实验准备

在实际评价时，我们考察 IQS 算法运行在真实语料上的测试数据，因为这些数据将更能说明算法的效率。并且，选择单模式串匹配算法中性能较好的 BM 算法和 QS 算法作为对照。测试语料为 reuters21578[44]，该语料为英文语料，总量为 27,982,337 个字节。对于串匹配算法来说，效率的最好体现是 CPU 运行时间。模式串是从一个单词个数为 25481 的词典中选取的，分别取长度为 2 到 11 的模式串各 100 个进行实验，并保证模式串之间不存在重复现象。然后将 BM、QS 和 IQS 算法分别进行 100 次循环，记录所有模式串在语料中的总出现次数，总消耗的 CPU 时间和总尝试次数。

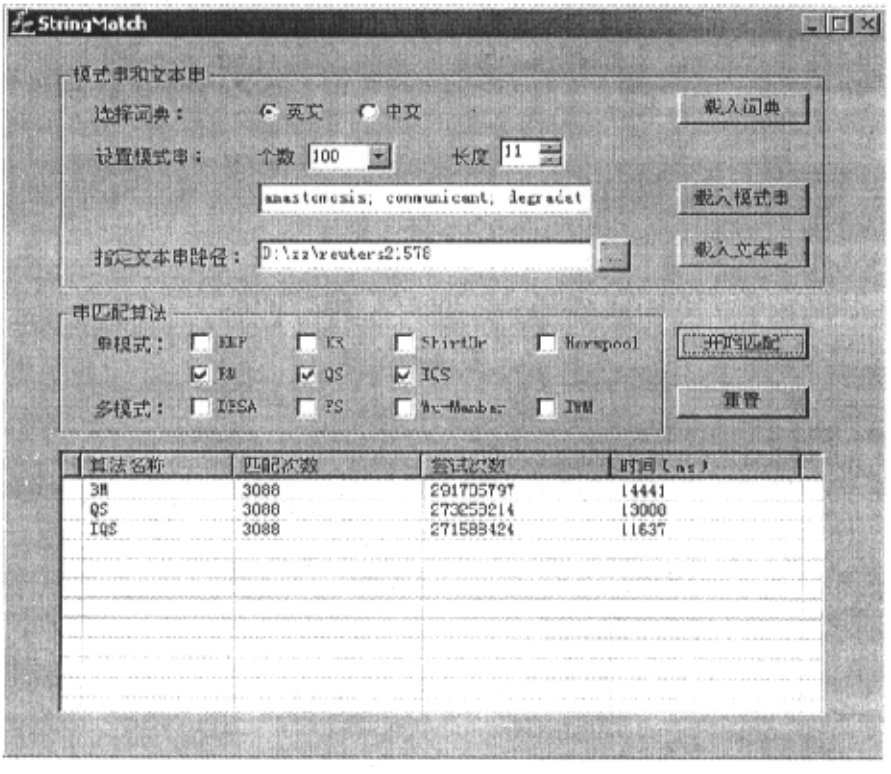



图 4.2.3.1 串匹配算法的实验比较用例

下面，简要介绍本文的实验用例，如图 4.2.3.1 所示。根据测试语料的不同，

可加载中文和英文两种不同的词典；根据设定模式串的个数和长度，模式串从词典中随机选取；测试语料可由用户指定选择。图 4.2.3.1 为 BM、QS 和 IQS 算法在 100 个长度为 11 的模式串的条件下的运行结果。该用例的编译环境为 Microsoft Visual C++ 6.0，使用 C 库函数中的 clock()函数记录各算法所耗的 CPU 时间（不包含 I/O 时间和预处理时间）。

### 4.2.3.2 实验结果

实验环境为 PentiumIII870MHz，256MB 内存，Windows 2000 Advanced Server。图 4.2.3.2 为 BM、QS 和 IQS 算法的实验运行时间，表 4.2.3.2 为相对应的匹配次数。

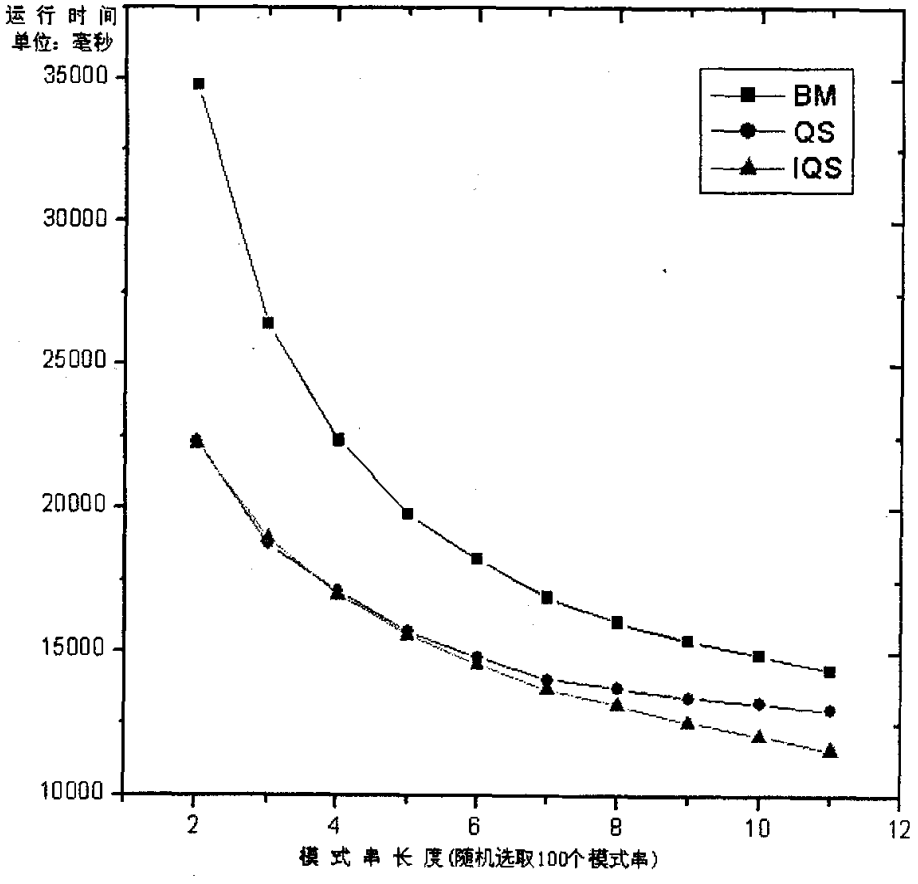



图 4.2.3.2 IQS 算法与 BM、QS 算法的运行时间比较

表 4.2.3.2　IQS 算法与 BM、QS 算法的尝试次数比较

| 模式串长度（100 个） | BM 算法 | QS 算法 | IQS 算法 |
|---|---|---|---|
| 2 | 1.41132415E9 | 9.50516310E8 | 9.50135494E8 |
| 3 | 9.56017122E8 | 7.24431223E8 | 7.23526756E8 |
| 4 | 7.27155868E8 | 5.90124422E8 | 5.89024086E8 |
| 5 | 5.89281628E8 | 4.99431804E8 | 4.98141328E8 |
| 6 | 4.99247145E8 | 4.36561701E8 | 4.34234274E8 |
| 7 | 4.33986577E8 | 3.86286365E8 | 3.84807608E8 |
| 8 | 3.85904509E8 | 3.49917510E8 | 3.48062318E8 |
| 9 | 3.47297917E8 | 3.19125688E8 | 3.16966798E8 |
| 10 | 3.16436276E8 | 2.94217831E8 | 2.92555170E8 |
| 11 | 2.91705797E8 | 2.73258214E8 | 2.71588424E8 |

#### 4.2.3.3 分析比较

首先，就上节的实验结果考察分析算法的运行时间和尝试次数。然后，分析其原因。最后，给出对 IQS 算法的评价。

由图 4.2.3.2 和表 4.2.3.2 可以看出，当模式串长度较短（2～3）时，IQS 算法的优势并不明显，运行时间与 QS 算法大致相等。IQS 算法保持了 QS 算法的特性，和 QS 算法的一样，其运行时间小于 BM 算法。随着模式串长度的增加，IQS 算法渐渐显示出其双向匹配的优势，运行时间比 BM、QS 算法都小。同时，模式串长度的增加使得 BM 算法与 QS 算法、IQS 算法的差距缩小，三个算法的运行时间都在减小。在尝试次数方面，IQS 算法要优于 BM 算法和 QS 算法，其尝试次数不会受模式串长度的影响，在通常情况下，IQS 算法的尝试次数小于 BM 算法和 QS 算法。

由于 IQS 算法继承了 QS 算法的匹配思想，所以当模式串较短时，IQS 算法并不能加快匹配速度，跳跃距离最大为模式串距离加一。当模式串长度变长时，由于 IQS 算法采用双向匹配的方法，最大跳跃距离有了很大提高，能够加快算法的匹配速度，并且也能够避免因文本串的起始部分有部分匹配而导致匹配速度减慢的情况。因此 IQS 算法在模式串较长时的运行时间要小于 BM 算法和 QS 算法。IQS 算法的尝试次数小于 BM 算法和 QS 算法，且不随模式串长度的变化而变化。这主要是由于 IQS 算法采用的双向匹配的方法。当文本串的起始部分有部分匹配的现象发生时，QS 算法的匹配速度将会减慢。IQS 算法采用从文本串的开始部分和结尾部分双向扫描的方法，避免了上述现象的发生：当开始部分有部分匹配而速度减慢时，结尾部分的扫描不受其影响，依旧进行最大限度的跳跃式扫描，所以结尾扫描指针越来越接近开始部分的扫描指针，使得扫描能够尽早结束，因

而尝试次数小于 QS 算法。

从以上的分析可知，IQS 算法总体来说表现出了良好的优势：尝试次数在通常情况下小于 BM 算法和 QS 算法，且不随模式串长度的变化而变化；在模式串较短（2～3）时，IQS 算法的运行时间与 QS 算法大致相等，且小于 BM 算法，在模式串较长时，IQS 算法的运行时间小于 QS 算法和 BM 算法，且随着模式串长度的增加，这种优势将愈加明显（当模式串长度为 11 时，IQS 算法的运行时间约为 QS 算法的 89%，BM 算法的 80%）。

## 4.3 多模式串匹配算法 IWM

### 4.3.1 现有算法分析

在现有的多模式串匹配算法中，最为经典的算法莫过于 DFSA 算法[31]、FS 算法[32]和 Wu-Manber 算法[33]。

DFSA 算法和 FS 算法都是通过构造有限状态自动机来实现匹配的，不同之处在于 FS 算法在基于 DFSA 算法之后结合 BM 算法的思想来加快匹配速度。对于类似于英文这种单字节字符，DFSA 算法和 FS 算法所构造的自动机的 goto 表所占空间为自动机状态总数 Num(state)×256，而对于中文大字符集而言， 其自动机的 goto 表所占空间为自动机状态总数 Num(state)×65536，为单字节情况的 256 倍。随着状态数的增加，会产生存储空间膨胀的问题，导致消耗内存过大，使得算法难以实用，无法符合网络内容监管的需求。

Wu-Manber 算法通常被认为在现有的多模式串匹配算法中具有最高的效率[33]。Wu-Manber 算法建立在 BM 算法基础之上，使用长度为 B 的字符块代替坏字符，进行最大限度的跳跃，并且使用散列技术和前缀表来减少需要进行实际匹配的次数。Wu-Manber 算法的存储空间不会受到小字符集或是大字符集环境的影响，在中文情况下能够表现出较好的性能，符合网络内容监管的需求。

另外，还有一些基于 DFSA 算法和 FS 算法的改进算法[45-49]，但在大字符集环境下多少都存在存储空间膨胀的问题，并且在效率上与 Wu-Manber 算法相比不太理想。还有基于 Wu-Manber 算法的改进算法，如文献[50]通过引入精确的不良字符转移和弱化良好后缀转移来提高算法性能。但在中文大字符集的情况下，并考虑到中文自身的特点和网络内容监管的需求，模式串自身出现部分匹配的概率较小，因而对于中文环境和网络内容监管来说，弱化良好后缀转移对算法的改进并不是太大，而精确的不良字符转移的最大跳跃距离为最短模式串长度 minlen，并为达到实际最大的跳跃距离 minlen+1。

### 4.3.2 IWM 算法

#### 4.3.2.1 算法描述

基于以上的分析，我们知道由于 Wu-Manber 算法采用字符块技术，降低了部分匹配的可能性，增加了直接跳跃的机会，而且使用散列技术和前缀表进一步减少实际匹配的次数，使得算法获得了很高的实际运行效率。在性能方面比其它多模式串匹配算法要相对好的同时，Wu-Manber 算法很好的满足了在中文环境下网络内容监管的需求。但是对 Wu-Manber 算法进行仔细分析之后，仍可发现其尚存在如下不足之处：

1) HASH 表和 SHIFT 表的空间利用率不高。对某一哈希值，当其 SHIFT 表值为零时，对应的 HASH 表值非空（表明该哈希值所对应的字符块出现在某个模式串中）。同时，当 HASH 表中的值为零的时，对应的 SHIFT 表值大于零（表明该哈希值所对应的字符块不曾出现在所有的模式串中）。事实上，HASH 表和 SHIFT 表都可以用来作为继续比较的入口，而在 Wu-Manber 算法中把 SHIFT 表所对应的值为零作为继续比较的入口。
2) 当长度为 B 的字符块不出现在任何模式串中时，跳跃距离 minlen-B+1 是比较保守的，使得算法不够高效，在模式串长度较短时对效率的影响尤其明显。当字符块不出现在任何模式串中且所有模式串的前缀不包含字符块任意长度的后缀时，可以获得更大的跳跃距离 minlen+1。
3) 当 SHIFT 表值为零时，继续深入比较以此判断是否有匹配出现，之后，跳跃距离只能为 1。在长度为 minlen 的字符块中如果有较多的部分匹配现象发生，该策略会在较大程度上影响算法的效率。可以在前 minlen-1 个连续字符中寻找长度为 B 的相同字符块，实际安全的跳跃距离可以大于 1。
4) 考虑到中文大字符集的情况，前缀表 PREFIX 的作用相对减弱。当进入比较阶段时，每次要查询 PREFIX 表以减少匹配次数，实际上在增加访存操作的同时，会降低算法的效率。

本文提出的多模式串匹配算法基于 Wu-Manber 算法改进，为方便叙述，记为 IWM（Improved Wu-Manber）算法。IWM 算法在吸取 Wu-Manber 算法同时，尽量改进了以上那些不足，并进一步增大跳跃距离。为实现最大的跳跃距离，IWM 算法结合了 QS 算法的思想，继承使用 Wu-Manber 算法的字符块技术和散列技术，舍弃了用于减少匹配次数的前缀表 PREFIX，只使用了 SHIFT 表和 HASH 表。其中 HASH 表的构造与 Wu-Manber 算法中的对应表完全相同，不同之处主

要集中在 SHIFT 表，如下所示：

1) Wu-Manber 算法中 B 的取值可以为 2 或 3，IWM 算法考虑到当进行跳跃时可能要考察字符块的后缀，因此 B 的值为 2 是一个较好的选择，因为考察长度 2 的字符块的后缀时只需分析一个字符，使得操作简单易行，而且 B=2 时算法能获得较好的效果[33]。
2) SHIFT 表的跳跃依据当前窗口的最后一个字符和紧邻在该字符的下一个字符，两字符组合成长度为 B（B=2）的字符块以此查表进行跳跃，其最大可能跳跃距离为 minlen+1。

IWM 算法的 SHIFT 表存储了长度为 B（B=2）的字符块的跳跃距离，使用不同于 Wu-Manber 算法的计算方式，通常情况下，跳跃距离大于 Wu-Manber 算法。IWM 算法使用 HASH 表作为进一步比较的入口，且每次的跳跃距离与 HASH 表值是否为零无关。同时，IWM 算法舍弃了 PREFIX 表。

以下是 IWM 算法的预处理过程、匹配过程及其时空间复杂度分析。

#### 4.3.2.2 预处理过程

IWM 算法的预处理过程的主要工作是生成匹配过程中所需要的两个表：SHIFT 表和 HASH 表。

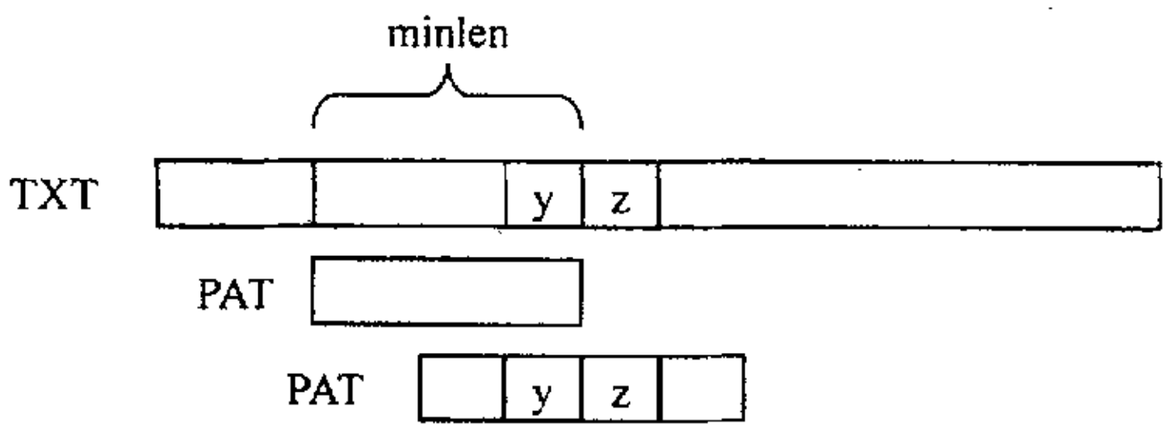
在计算出所有模式串的最短长度 minlen 后，HASH 表针对每一个模式串考察前 minlen 个字符的最后 B（B=2）个字符，计算出所对应的哈希值，并存储为带有模式串编号的链表。HASH 表的构造可以参照 Wu-Manber 算法中的预处理过程。

IWM 算法结合 QS 算法的思想，在查找 SHIFT 表进行最大跳跃时，将当前窗口的最后一个字符和紧邻在当前窗口的下一个字符组合成长度为 B（B=2）的字符块一起进行考虑。设该字符块为 Block[0...B-1]，即 Block[0]和 Block[1]，通过一个哈希函数计算出 Block[0...B-1]的散列值，以此来访问 SHIFT 表值，即该字符块此时可以安全跳过的字符数。设 Block[0...B-1]的散列值为 hash，y=Block[0]，z=Block[1]，其中 y,z∈Σ，则 SHIFT 表的构造如图 4.3.2.2 所示，具体分为以下三种情况：

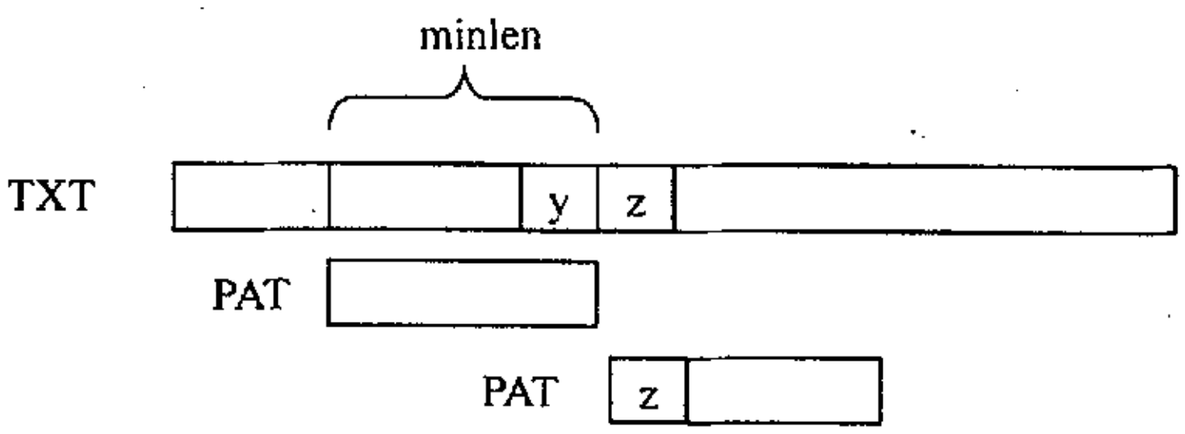
1) 当 Block[0...B-1]出现在某些模式串的前 minlen 个字符时，假设在所有的所有模式串中 Block[0...B-1]的最右出现位置为 i，则安全的跳跃字符数为 minlen-i-1，即 SHIFT[hash]= minlen-i-1。如图 4.3.2.2 中(a)所示。
2) Block[0...B-1]不出现在任何模式串中，但是其后缀即 Block[1]出现在某些模式串的前缀中，则安全的跳跃字符数为 minlen，即 SHIFT[hash]=

minlen。如图 4.3.2.2 中(b)所示。

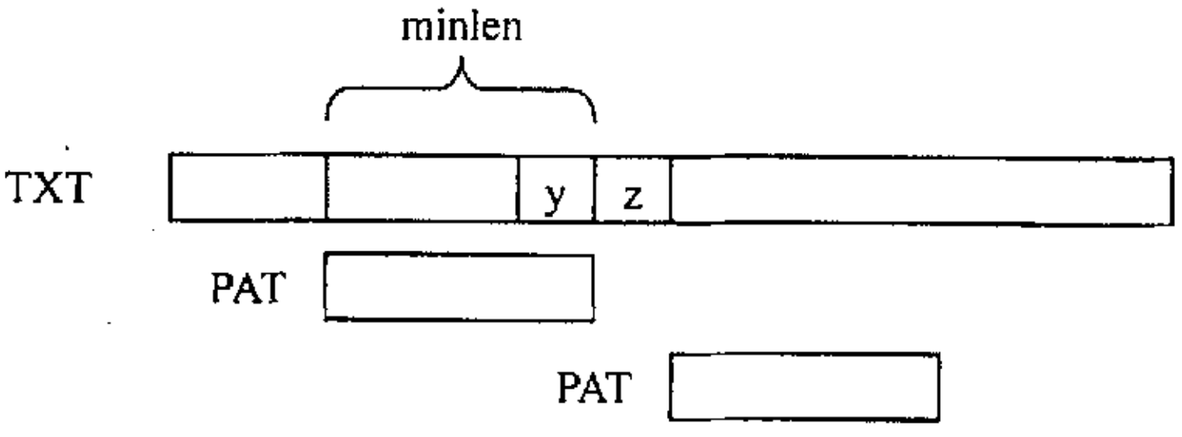
3) 当所有的模式串的前 minlen 个字符中不曾出现 Block[0...B-1]，并且所有模式串的前缀不存在 Block[0...B-1]的后缀时（因为字符块长度 B=2，所以 Block[0...B-1]的后缀长度只能为 1，判断所有模式串的前缀是否存在Block[0...B-1]的后缀只需考察字符Block[B-1]是否与各模式串的首字符相同），则安全的跳跃字符数为 minlen+1，即 SHIFT[hash]= minlen+1。如图 4.3.2.2 中(c)所示。




（a）当模式串含有子串 yz 时




（b）当模式串不含有子串 yz 并且 Block[0...1]的后缀出现在模式串的前缀时




（c）当模式串不含有子串 yz 并且 Block[0...1]的后缀不出现在模式串的前缀时

图 4.3.2.2 IWM 算法的 SHIFT 表跳跃方法

经过预处理，IWM 算法所构造的 SHIFT 表与 Wu-Manber 算法的 SHIFT 表相比具有两个明显的优点：联合考虑 HASH 表，其空间利用率大大提高；更大的安全跳跃距离，最大的跳跃距离为 minlen+1。

#### 4.3.2.3 匹配过程

由于 IWM 算法设定字符块长度 B=2，所以假设当前窗口所要考察的字符块为：字符 Block[0]和 Block[1]。假设紧邻在当前窗口的下一个字符为 char，Block[0]和 Block[1]所组成的字符块的哈希值为 hash，Block[1]和 char 组成的字符块的哈希值为 hash_skip。

IWM 算法使用 HASH 表作为进一步比较的入口。当 hash 所对应的 HASH 值为空时，表明 Block[0]和 Block[1]不曾出现在所有模式串的前 minlen 个字符的最后两位；当 hash 所对应的 HASH 值不为空时，表明 Block[0]和 Block[1]出现在某些模式串的前 minlen 个字符的最后两位，此时的 HASH 表是这些可能会发生匹配的模式串编号的链表，依次取出这些模式串进行比较，如若发生匹配记录相关信息。在判断完 hash 所对应的 HASH 值后，不管 HASH 值是否为空，计算哈希值 hash_skip 所对应的 SHIFT 值以此获得跳跃距离。IWM 算法匹配阶段的主要步骤如下：

1) 使用当前窗口的最后两个字符 Block[0]和 Block[1]计算散列值 hash；
2) 判断 HASH[hash]是否为空，如果是，转 4，否则继续；
3) 依次把 HASH[hash]所指向的每个模式串与文本串比较，如果发现匹配记录相关信息；
4) 使用当前窗口的最后一个字符 Block[1]和紧邻在当前窗口的下一个字符 char 计算散列值 hash_skip，跳跃 SHIFT[hash_skip]个字符。然后判断是否扫描结束，否则转 1 继续扫描。

#### 4.3.2.4 时空间复杂度分析

假设模式串集合的大小为 m，最短模式串长度为 minlen，最长模式串长度为 maxlen，文本串长度为 n，其中每个模式串和文本串都由字符表 Σ 中的字符组成，字符表 Σ 的大小为 σ，跳转表 SHIFT 和哈希表 HASH 的最大长度分别为 MAXSHIFT 和 MAXHASH。由于 IWM 算法继承了 Wu-Manber 算法的散列技术和字符块技术，因此，实际应用中的 MAXSHIFT 和 MAXHASH 值都为 32768。

在预处理阶段，IWM 算法要构造 SHIFT 表和 HASH 表，因此空间复杂度为 O(MAXSHIFT＋MAXHASH)。预处理阶段的时间复杂度较为复杂，主要由 SHIFT 表和 HASH 表的构造决定。HASH 表的构造分为两步，即首先全部置零，然后计算每个模式串前 minlen 个字符里最后 B（B=2）个字符所组成字符块的 HASH 值，所以构造 HASH 表的时间复杂度为 O(MAXHASH＋m)；SHIFT 表的构造分为三步，即首先全部置为 minlen+1，然后计算每个模式串前 minlen 个字符中紧

邻在一起的 B（B=2）个字符所组成的字符块的跳跃值，最后计算由每个模式串的首字符和字符表 Σ 中的任意字符所构成的字符块的跳跃值，所以构造 SHIFT 表的时间复杂度为 O(MAXSHIFT＋m×minlen＋m×σ)。

在匹配阶段，IWM 算法的时间复杂度与文本串长度 n、每个模式串在 HASH 表的分布情况以及模式串自身有关。在最优情况下，IWM 算法的每次跳跃距离为 minlen+1，时间复杂度为 O(n/(minlen+1))；在最坏情况下，IWM 算法的时间复杂度为 O(n×maxlen)。IWM 算法的程序代码见附录 B。

### 4.3.3 实验与分析

#### 4.3.3.1 实验准备

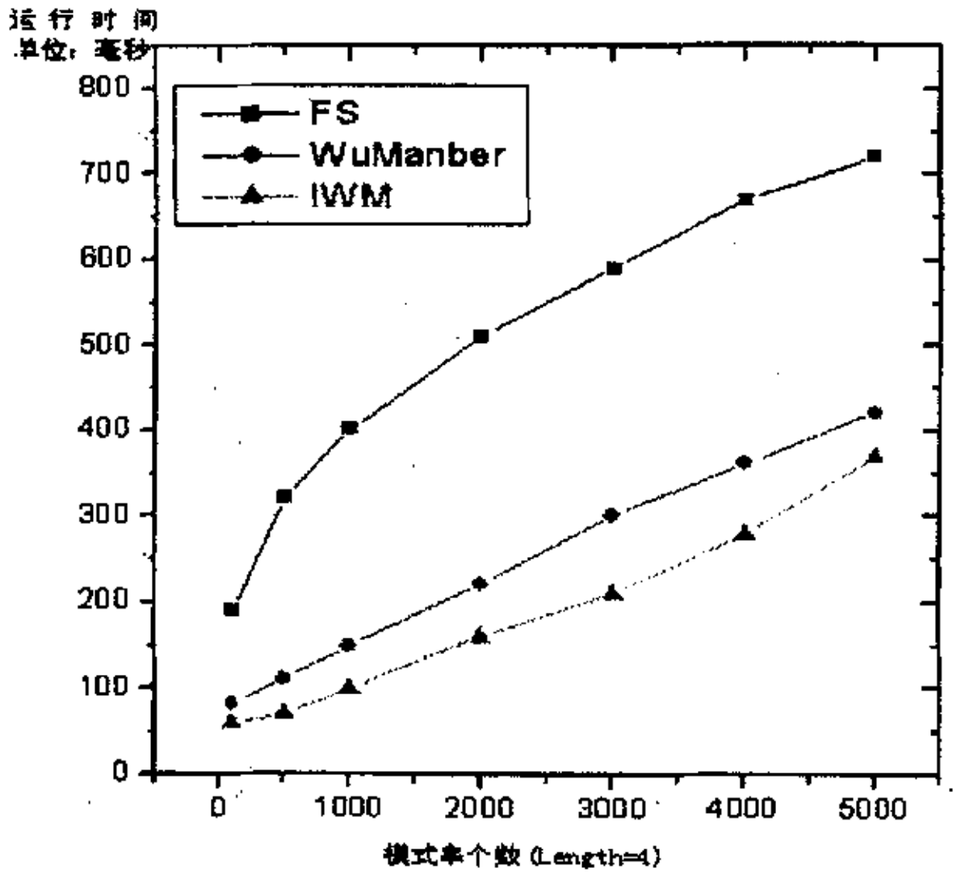
为更好的评价 IWM 算法，我们分别在一个中文语料和一个英文语料上对其进行测试，其中中文测试语料为外国文学小说（HTML 格式），总量为 10,486,265 字节，英文测试语料为 reuters21578[44]，总量为 27,982,337 字节。

本节依然使用 4.2.3 一节中所使用的实验比较用例，并且参照多模式串匹配算法中性能较好的 FS 算法和 Wu-Manber 算法对 IWM 算法进行对比分析。在对 IWM 算法进行实验分析时，我们进行两种方式的实验：在模式串长度相等时，考察模式串数量的不同对算法性能的影响；在模式串数量相同时，考察模式串长度的不同对算法性能的影响。

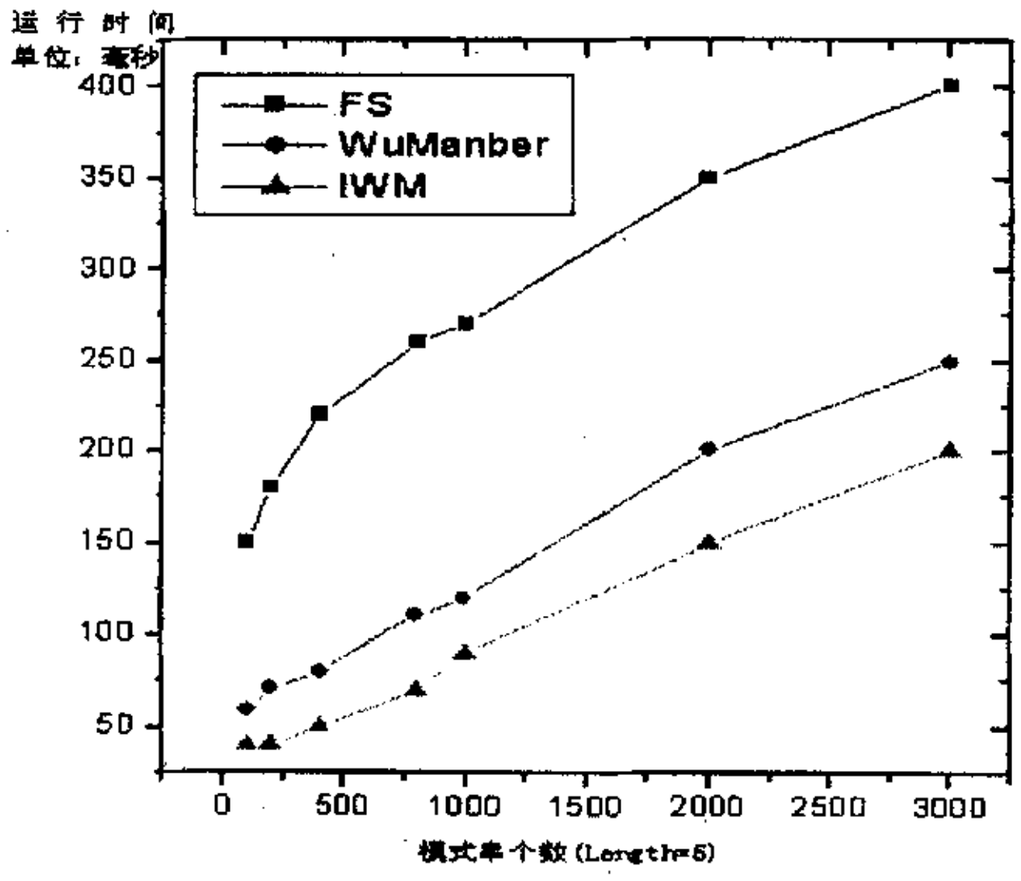
在中文语料上，我们进行两组测试：第一组分别考察在双字、三字、四字的情况下，模式串数量对算法的影响；第二组当模式串数量设定为 1000 时，考察模式串为单字、双字、三字、四字及五字以上时对算法的影响。模式串是从拼音加加（版本 2.204）词库里的 41372 个词语里随机选取的，由于该词典只有词语没有单字，上述第二组实验中所用到的单字是把词典的词语拆分得到的（由于是随机选取，所以某个长度的词语的随机可选取的最大个数会有所不同）。在英文语料上，对长度为 2 到 10 的单词进行测试，该环境下的测试分为两组：第一组分别考察模式串长度为 2 到 10 的情况下，模式串数量对算法的影响；第二组当模式串数量设定为 500 时，考察模式串长度对算法的影响。模式串依然是从一个单词个数为 25481 的英文词典中随机选取。在中英文语料中，测试实例保证所选取的模式串不会重复。测试记录所有模式串在语料中的总出现次数，消耗的 CPU 时间和总尝试次数。

#### 4.3.3.2 实验结果

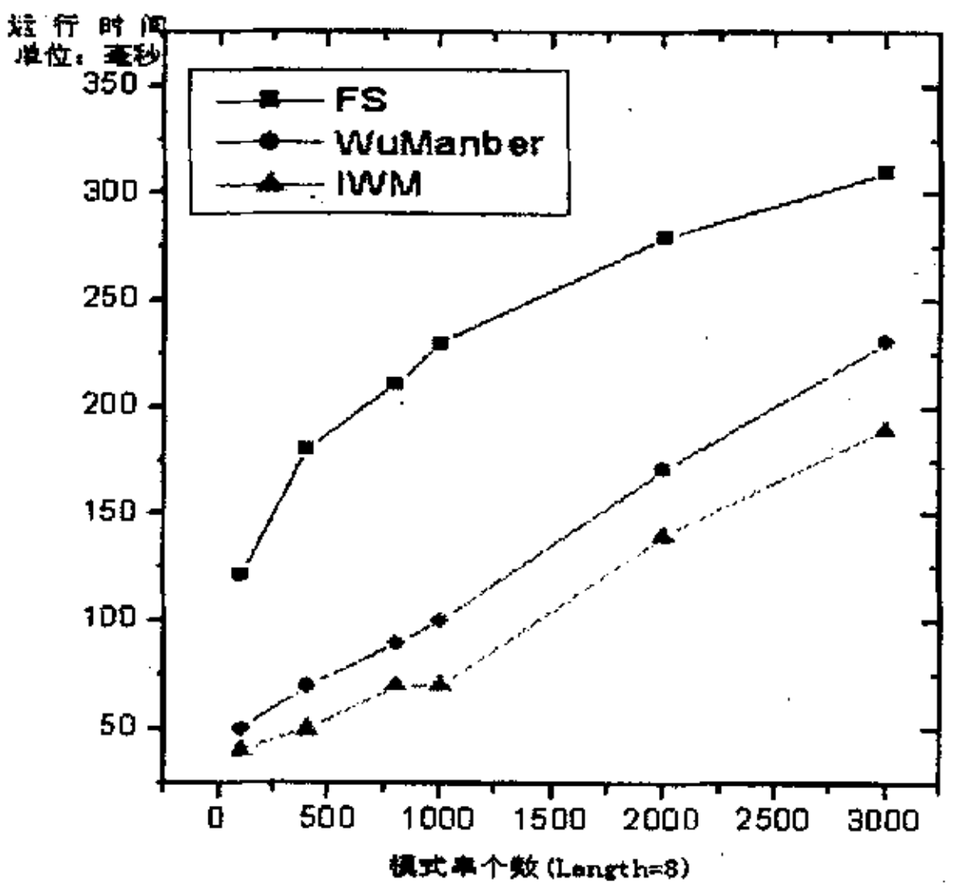
实验环境为 PentiumIII870MHz，256MB 内存，Windows 2000 Advanced Server。以下为实验结果。



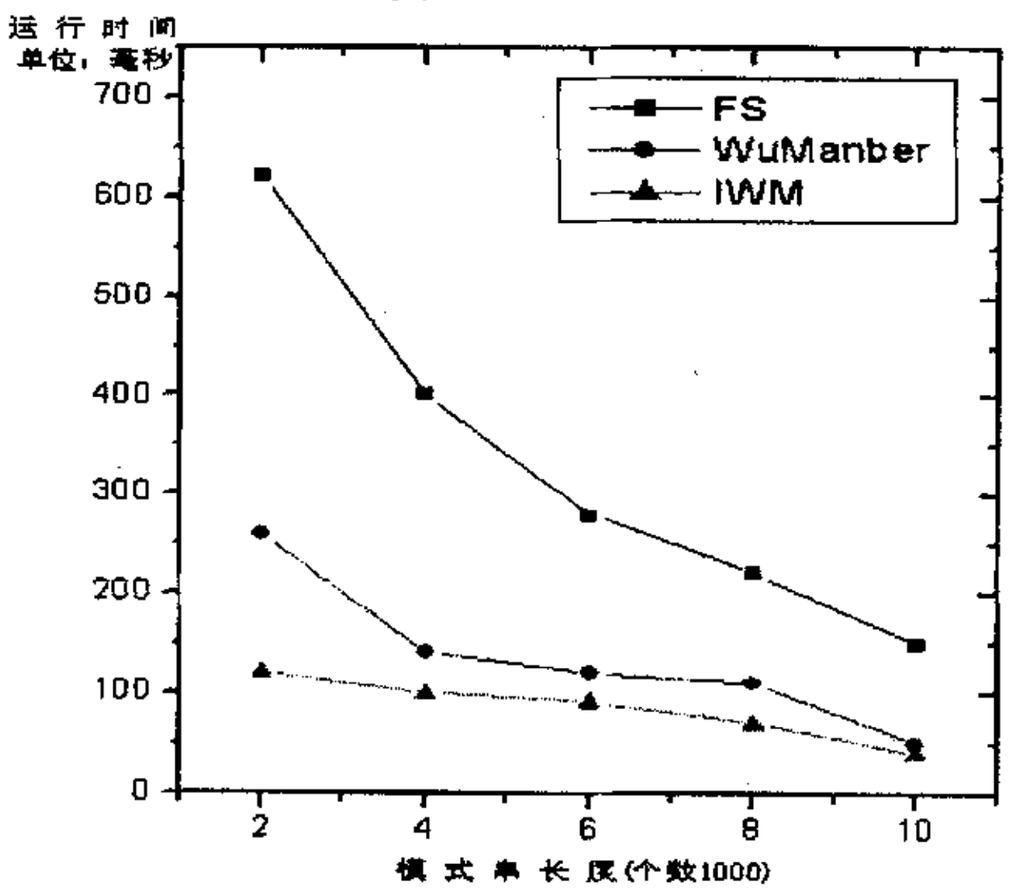

(a) 模式串长度为 4




(b) 模式串长度为 6




(c) 模式串长度为 8

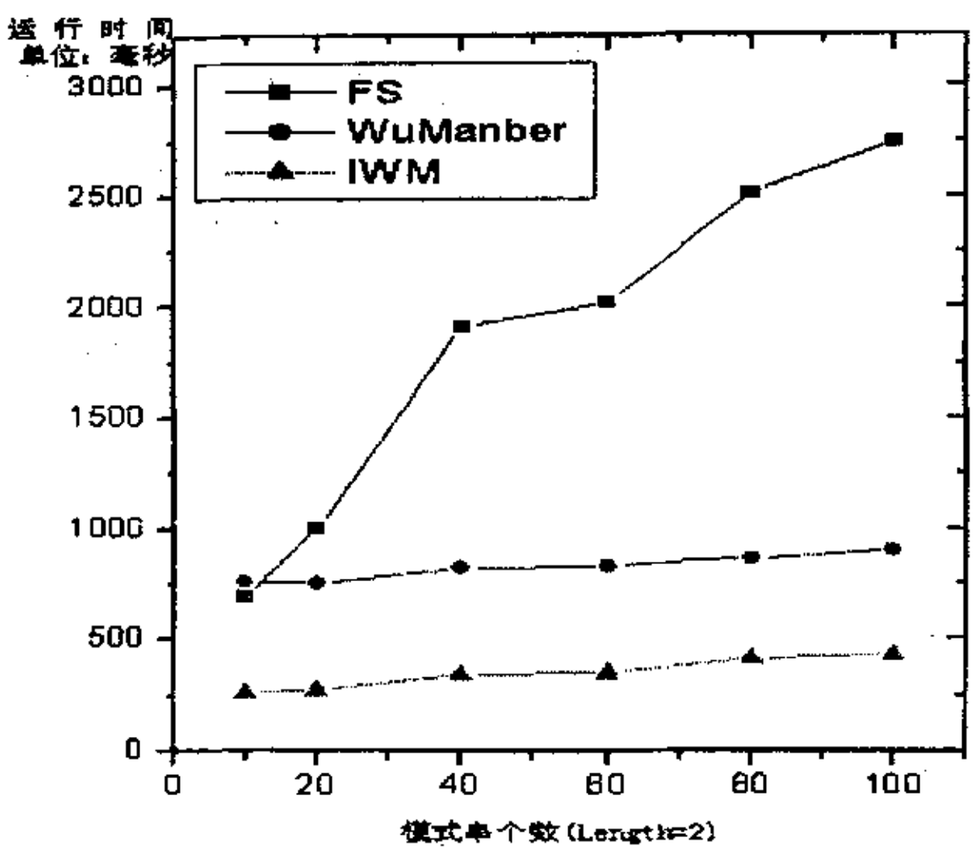



(d) 模式串个数为 1000

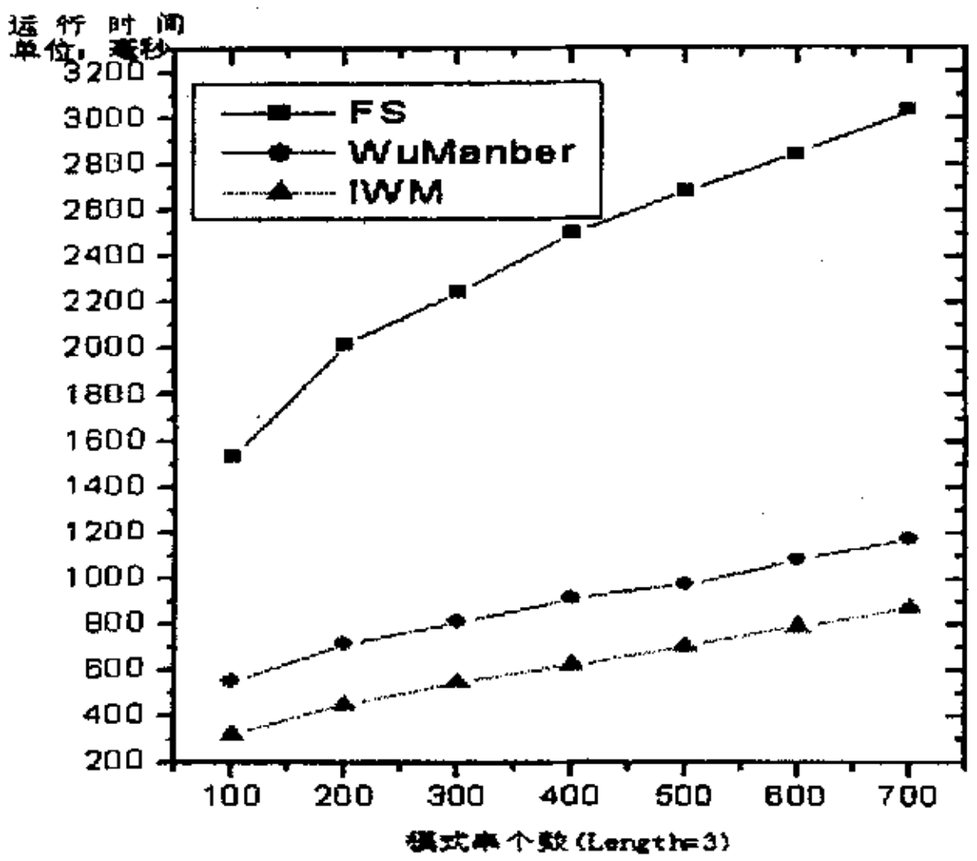
| 模式串长度(1000 个) | FS 算法 | Wu-Manber 算法 | IWM 算法 |
|---|---|---|---|
| 2 | 6650889 | 6777630 | 2982783 |
| 4 | 3885284 | 3147858 | 2231353 |
| 6 | 2725343 | 2345622 | 1666664 |
| 8 | 2009818 | 1910807 | 1389309 |
| 10 | 1451609 | 1113690 | 886989 |

(e) 图(d)所对应的尝试次数

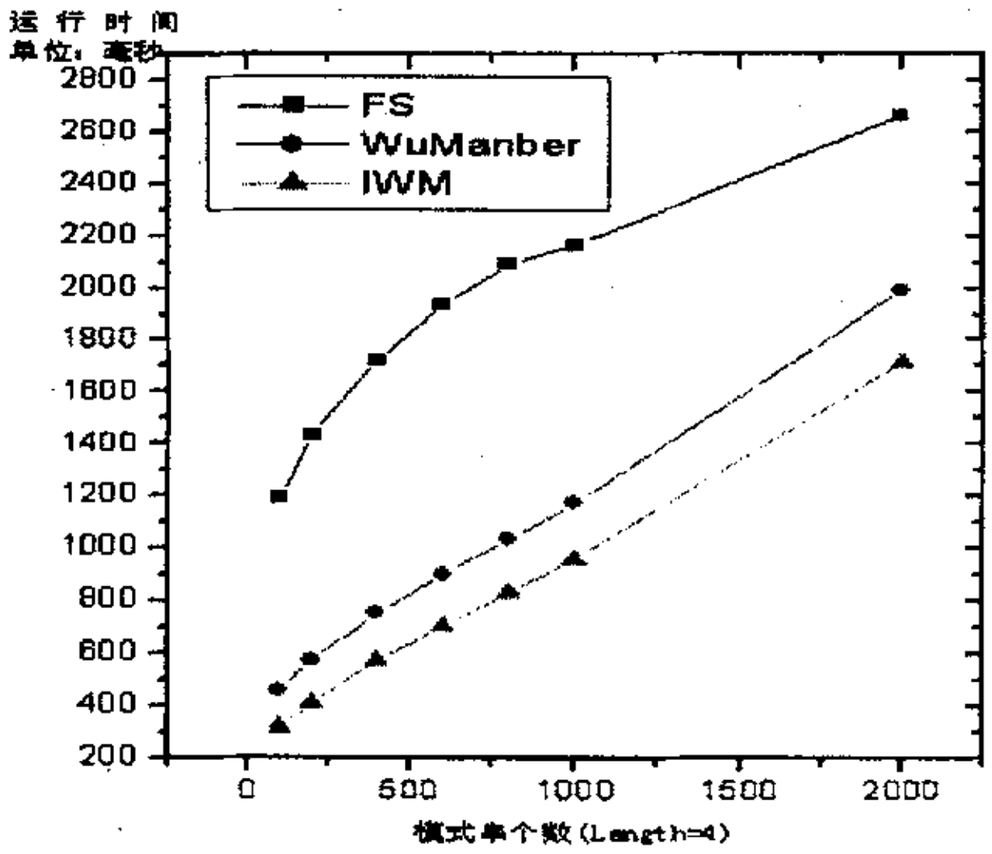
图 4.3.3.2.1 IWM 算法在中文语料下的实验结果（a～e）



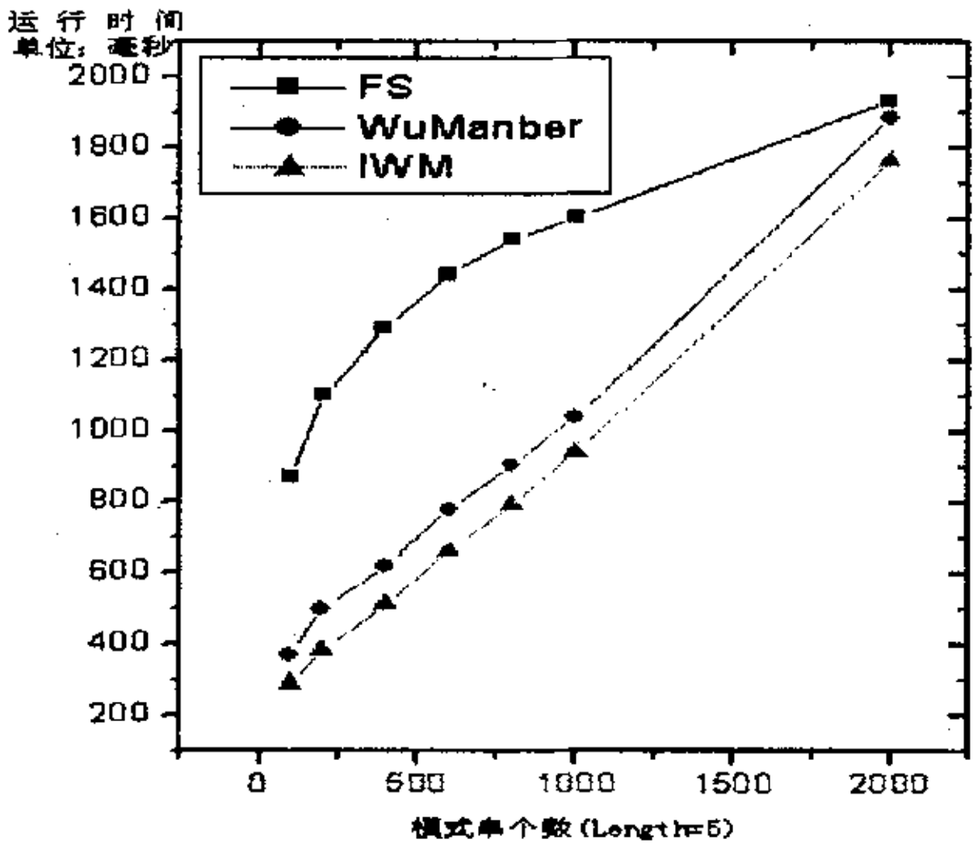

(a) 模式串长度为 2



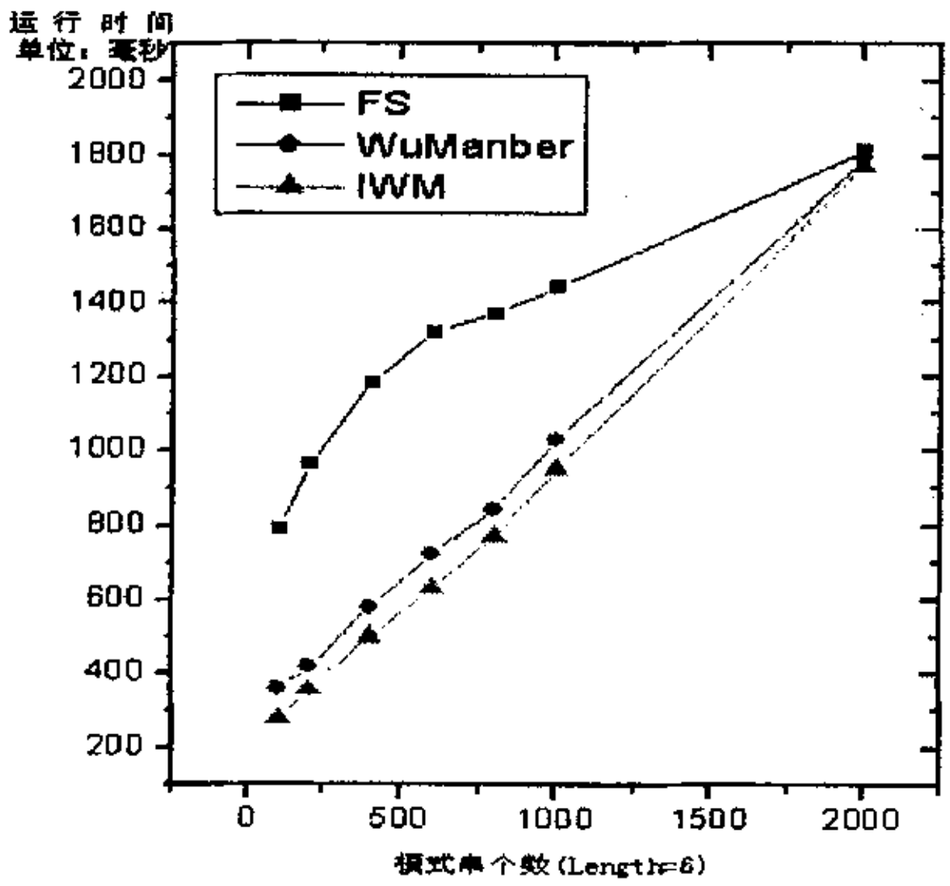

(b) 模式串长度为 3



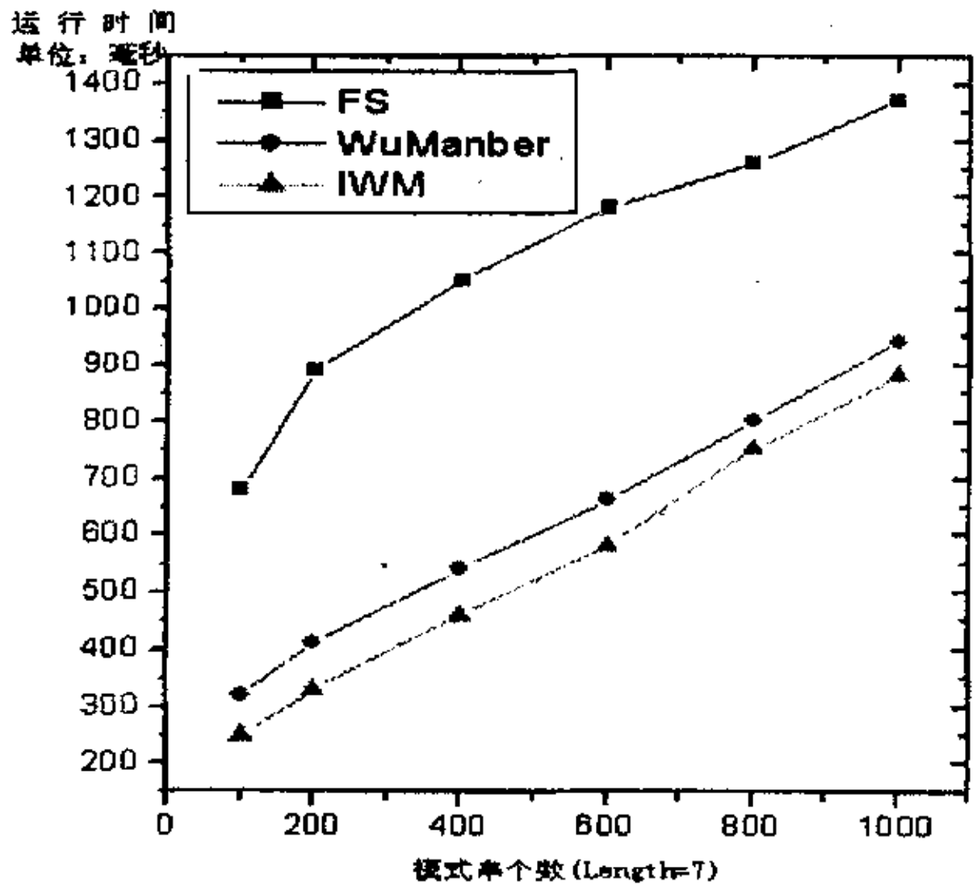

(c) 模式串长度为 4



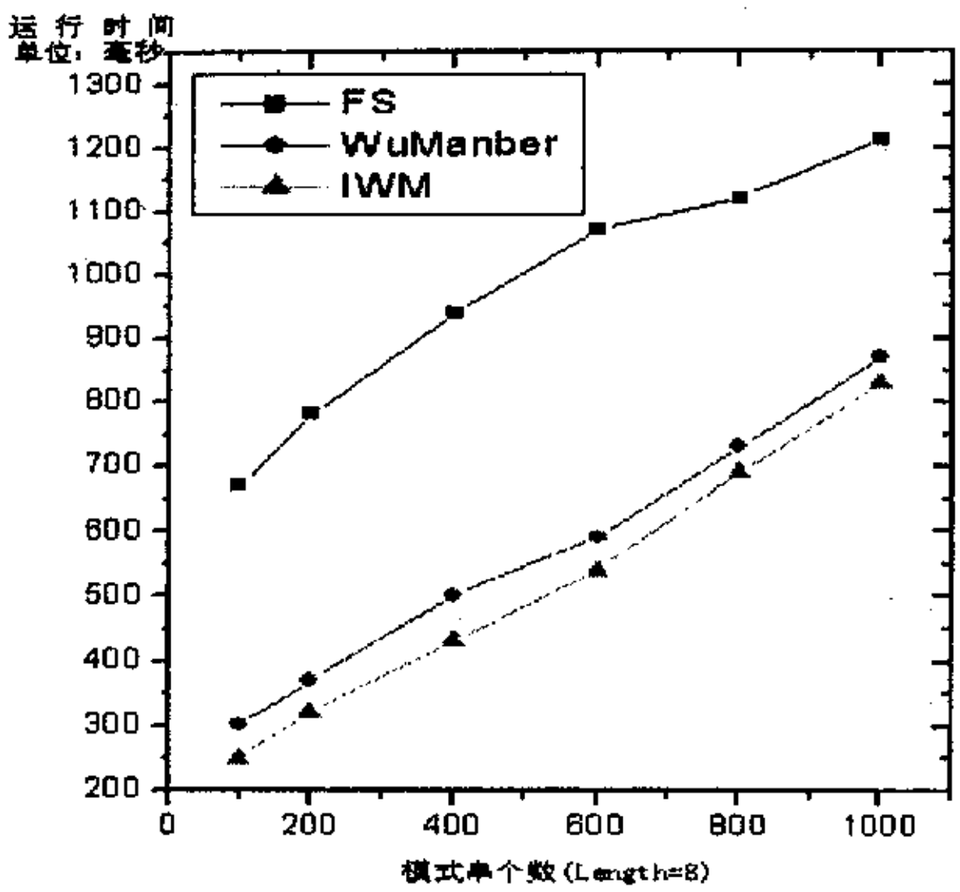

(d) 模式串长度为 5



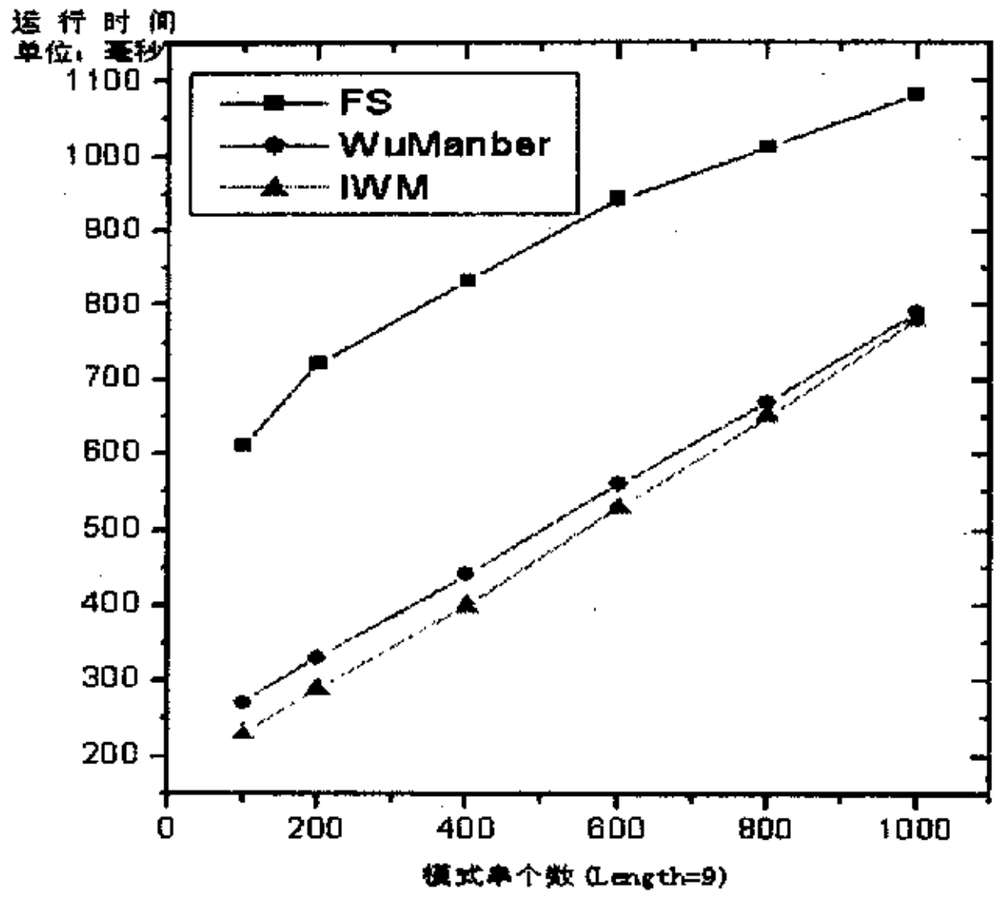

(e) 模式串长度为 6



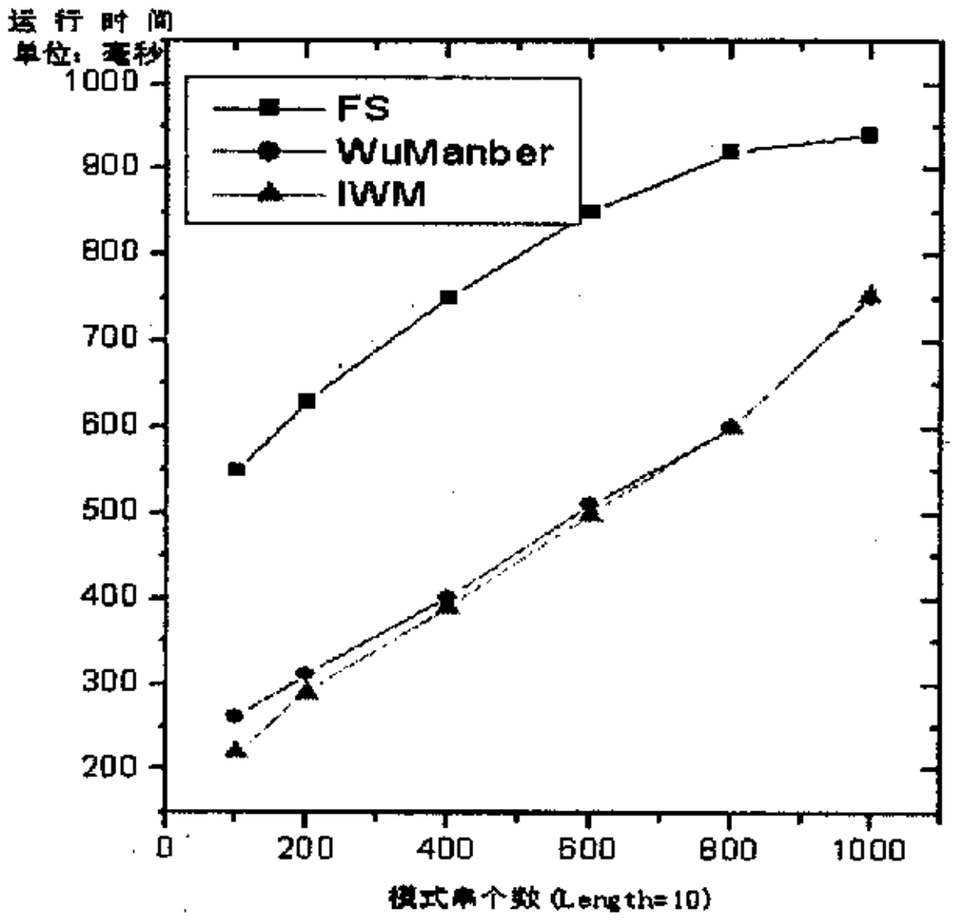

(f) 模式串长度为 7



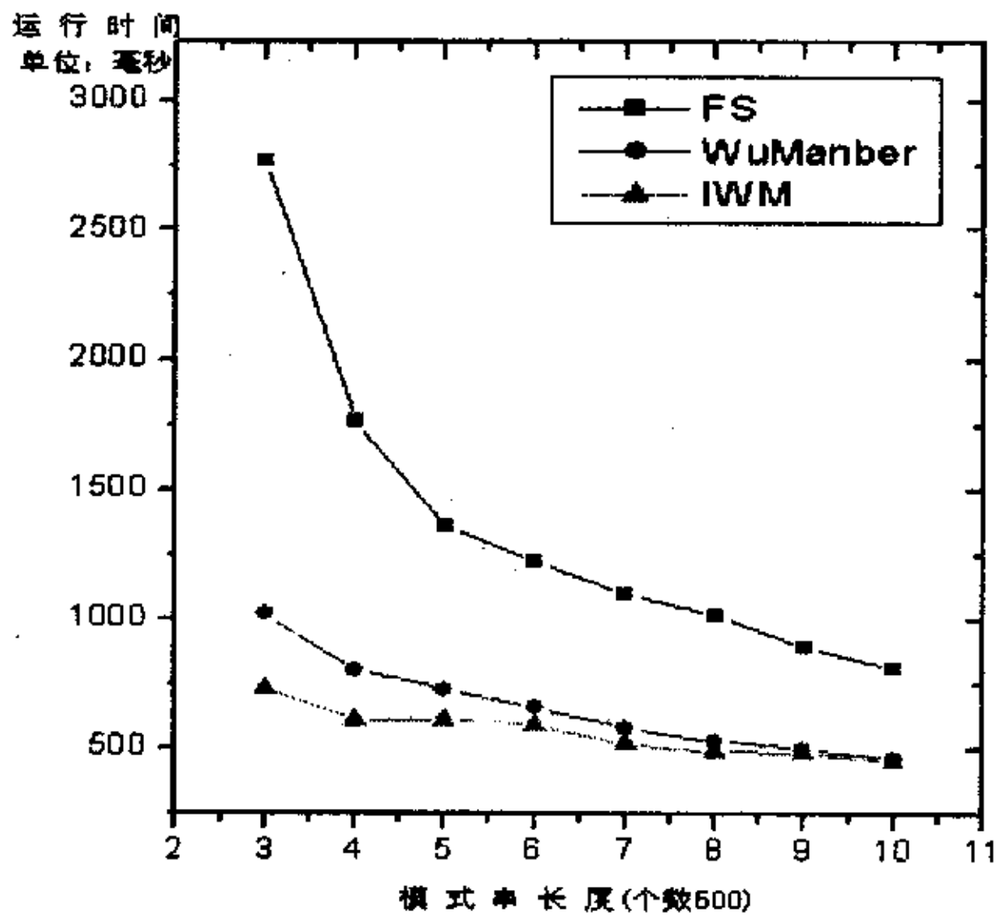

(g) 模式串长度为 8




(h) 模式串长度为 9




(i) 模式串长度为 10




(j) 模式串个数为 500

| 模式串长度（500 个） | FS 算法 | Wu-Manber 算法 | IWM 算法 |
|---|---|---|---|
| 3 | 29776115 | 18181628 | 11261745 |
| 4 | 18915789 | 13396737 | 9024638 |
| 5 | 14349477 | 10710630 | 7872216 |
| 6 | 12407106 | 8867307 | 6698593 |
| 7 | 10778824 | 7729182 | 5872855 |
| 8 | 9783996 | 6713764 | 5131800 |
| 9 | 8517665 | 5803461 | 4605562 |
| 10 | 7629250 | 5198632 | 4201089 |

(k) 图(j)所对应的尝试次数

图 4.3.3.2.2 IWM 算法在英文语料下的实验结果（a～k）

#### 4.3.3.3 分析比较

图 4.3.3.2.1 是 IWM 算法在中文语料下的实验结果，其中图(a～c)是当模式串长度分别为 4、6、8 时算法随模式串数量的变化情况，图(d)是当模式串数量为 1000 时算法随模式串长度的变化情况，表(e)是图(d)所对应的尝试次数的情况。从图(a～c)可以看出，随着模式串数量的增加，三种算法在特定模式串长度的情况下运行时间都有相应增大，IWM 算法的运行时间一直小于 Wu-Manber 算法和 FS 算法。从图(d)可以看出，当模式串数量为 1000 时，我们分别取长度为 2、4、6、8 及 10 以上的模式串（为画图方便，图中模式串长度 10 代表模式串长度大于等于 10 的情况），IWM 算法运行时间小于 Wu-Manber 算法和 FS 算法，当模式串长度为 2 时，IWM 算法的运行时间约为 Wu-Manber 算法的 46%。随着模式串长度的增加，IWM 算法与 Wu-Manber 算法的差距在进一步减小，当模式串长度为 10 时，IWM 算法的运行时间约为 Wu-Manber 算法的 80%。从图(e)可以看出，在模式串数量为 1000 时，随着模式串长度的增加，三种算法的尝试次数都逐渐减少，IWM 算法的尝试次数一直小于 Wu-Manber 算法和 FS 算法。

图 4.3.3.2.2 是 IWM 算法在英文语料下的实验结果，其中图(a～i)是当模式串长度分别为 2～10 时算法随模式串数量的变化情况，图(j)是当模式串数量为 500 时算法随模式串长度的变化情况，表(k)是图(j)所对应的尝试次数的情况。从图(a～i)可以看出，随着模式串数量的增加，三种算法在特定模式串长度的情况下运行时间都有相应增大，IWM 算法的运行时间一直小于 Wu-Manber 算法和 FS 算法。但是，与中文语料下测试结果不同的是：IWM 算法的优势在模式串长度较短时更为明显，当模式串长度为 2、模式串数量为 100 时，IWM 算法的运行时间约为 Wu-Manber 算法的 48%，而在模式串长度为 10、模式串数量为 100 时，IWM 算法的运行时间约为 Wu-Manber 算法的 84%，当在此条件下进一步增加模式串数量时，IWM 算法的运行时间与 Wu-Manber 算法大致相同。从图(j)可以看出，当模式串数量为 500 时，我们分别取模式串的长度为 3～10，IWM 算法的运行时间一直小于 Wu-Manber 算法和 FS 算法，并且随模式串长度的增加，IWM 算法与 Wu-Manber 算法的运行时间的差距在逐步缩小，当模式串长度为 3 时，IWM 算法的运行时间约为 Wu-Manber 算法的 72%，而当模式串长度为 10 时，IWM 算法的运行时间约为 Wu-Manber 算法的 98%。从图(e)可以看出，在模式串数量为 500 时，随着模式串长度的增加，三种算法的尝试次数都逐渐减少，IWM 算法的尝试次数一直小于 Wu-Manber 算法和 FS 算法。

从上面的实验结果可知，IWM 算法的运行时间小于 Wu-Manber 算法和 FS 算法，并且无论是在中文环境还是英文环境下，当模式串较短时算法的优越性表现的更为明显。首先，我们分析 IWM 算法比 Wu-Manber 算法的运行速度快的原

因。IWM 算法由于继承了 Wu-Manber 算法的字符块技术和散列技术，并结合了 QS 算法的跳跃思想，使得算法在保持 Wu-Manber 算法特性的同时获得了最大的跳跃距离 minlen+1。因此，IWM 算法的最大跳跃距离要比 Wu-Manber 算法大，从而使得扫描能够尽快结束。其次，还要分析 IWM 算法在模式串较短时优势更为明显的原因。我们知道 IWM 算法的最大跳跃距离比 Wu-Manber 算法的大，当模式串较短时，字符块出现在模式串中的概率较小，从而 IWM 算法中 SHIFT 表的平均跳跃距离也要大于 Wu-Manber 算法，但是当模式串较长时，情况发生了变化，字符块出现在模式串中的概率增大，使得 IWM 算法的平均跳跃距离与 Wu-Manber 算法大致相同。所以，与 Wu-Manber 算法相比，IWM 算法在模式串较短时的优势比模式串较长时更为明显。最后，分析 IWM 算法在中英文环境下的差异。从图 4.3.3.2.1 和图 4.3.3.2.2 不难看出，与 Wu-Manber 算法相比，IWM 算法在中文环境下的优势更为明显。这主要是由于中文大字符集的特点所定。在大字符集的情况下，IWM 算法获得的最大跳跃距离和平均跳跃距离要大于在小字符集时的情况，从而能够尽早结束匹配过程。

从以上的分析可知，IWM 算法总体来说表现出了良好的优势：运行时间在通常情况下小于 Wu-Manber 算法和 FS 算法，且这种优势不随模式串长度和数量的变化而变化；IWM 算法的尝试次数一直小于 Wu-Manber 算法和 FS 算法；IWM 算法比 Wu-Manber 算法更适合于中文大字符集的环境，符合网络内容监管的需求。

接下来，本文采用文献[45]中所用到的方法来评价 IWM 算法在中文环境下对于 Wu-Manber 算法的性能改进。已知中文词的出现频率（注：不是使用概率）依次为：单字词占 12.1%，双字词占 73.6%，三字词占 7.6%，四字词占 6.4%，多字词占 0.2%[51]。计算公式如下：

$$\frac{T_I}{T_W} = \sum_{i=1}^{5} \frac{Tii}{Twi} \times prop(i) \tag{4.3.3.3}$$

在式（4.3.3.3）中，$T_I$ 表示 IWM 算法的运行时间，$T_W$ 表示 Wu-Manber 算法的运行时间，prop(i)表示第 i 类中文词的出现频率。依据式(4.3.3.3)和图 4.3.3.2.1 中的图(d)计算得到：

$$\begin{aligned}\frac{T_I}{T_W} &= \frac{120}{260}\times 0.121 + \frac{100}{140}\times 0.736 + \frac{91}{120}\times 0.076 + \frac{70}{110}\times 0.064 + \frac{40}{50}\times 0.002 \\ &= 0.05585 + 0.52571 + 0.05763 + 0.04073 + 0.0016 \\ &= 0.68152\end{aligned}$$

由此我们可知，在平均情况下，IWM 算法在中文环境时的匹配时间约为 Wu-Manber 算法的 68.152%，因此 IWM 算法能够更好的完成多模式串的匹配任务，适合于网络内容监管的工作。

# 5 网络内容监管系统的集成实现

## 5.1 数据库的设计

在实际的工程项目中，为实现网络内容监管系统我们在数据库内设计了六张表，分别存放网站信息、监管操作信息、网站监管信息、非法和可疑信息、敏感信息类别以及关键字信息。本节我们主要介绍与串匹配算法有关的敏感信息类别和关键字信息这两张表。

敏感信息类别表存放敏感信息的类别及相关信息，其 ID 号依次为 1、2、4、8……，即 2 的（n－1）次方，n 为自然数，而关键字信息表存放在关键字过滤模块中所要用到的关键字信息，每个关键字所对应的敏感信息类别号为可能属于的所有敏感信息类别的 ID 号之和。举例来说，假设数据库中有法轮功类别和反动类别，其类别的 ID 分别为 1、2，关键字有“圆满”、“公投”、“法轮功”，其中关键字“法轮功”可能属于法轮功类别，也可能属于反动类别，因此，在关键字信息表中关键字“法轮功”所对应的敏感信息类别号为法轮功类别 ID 与反动类别 ID 之和。如下所示：

| ID | 信息类别 | …… |
|---|---|---|
| 1 | 法轮功 | …… |
| 2 | 反动 | …… |
| …… | …… | …… |

a) 敏感信息类别表

| ID | 关键字 | 对应的信息类别 | …… |
|---|---|---|---|
| 1 | 圆满 | 1 | …… |
| 2 | 法轮功 | 3 | …… |
| 3 | 公投 | 2 | …… |
| …… | …… | …… | …… |

b) 关键字信息表

图 5.1 网络内容监管系统中与串匹配相关的数据库表设计

这样的设计能带来两个好处：一是避免了不同敏感信息类别重复定义同一关键字的现象，为关键字过滤模块提供了便利，提高了该模块的运行效率；二是为后继的语义分析模块提供了分析依据，确定出了某一信息可能所属的信息类别，让语义分析模块在可能的信息类别中进行判断，提高了信息识别率和系统的效率。

在实际的系统中，用户人员可以设定敏感信息类别和关键字的信息。考虑到敏感信息类别和关键字会受时间、政治形势等诸多因素的影响，在执行监管任务时用户人员可以选择所要监管的敏感信息类别和关键字。

## 5.2 IQS 算法和 IWM 算法的集成

IQS 算法和 IWM 算法以类的形式进行封装。考虑到系统的可移植性和升级问题，我们用标准 C++进行代码编写，并以动态链接库的形式提供给系统进行调用。单模式串匹配 IQS 算法的集成较为简单，我们主要介绍多模式串匹配 IWM 算法在系统中的集成。

IWM 算法集成在敏感信息识别模块里的关键字过滤模块中。由于所要匹配的关键字在某一时期来说相对稳定，为最大限度的提高匹配速度，可以将 IWM

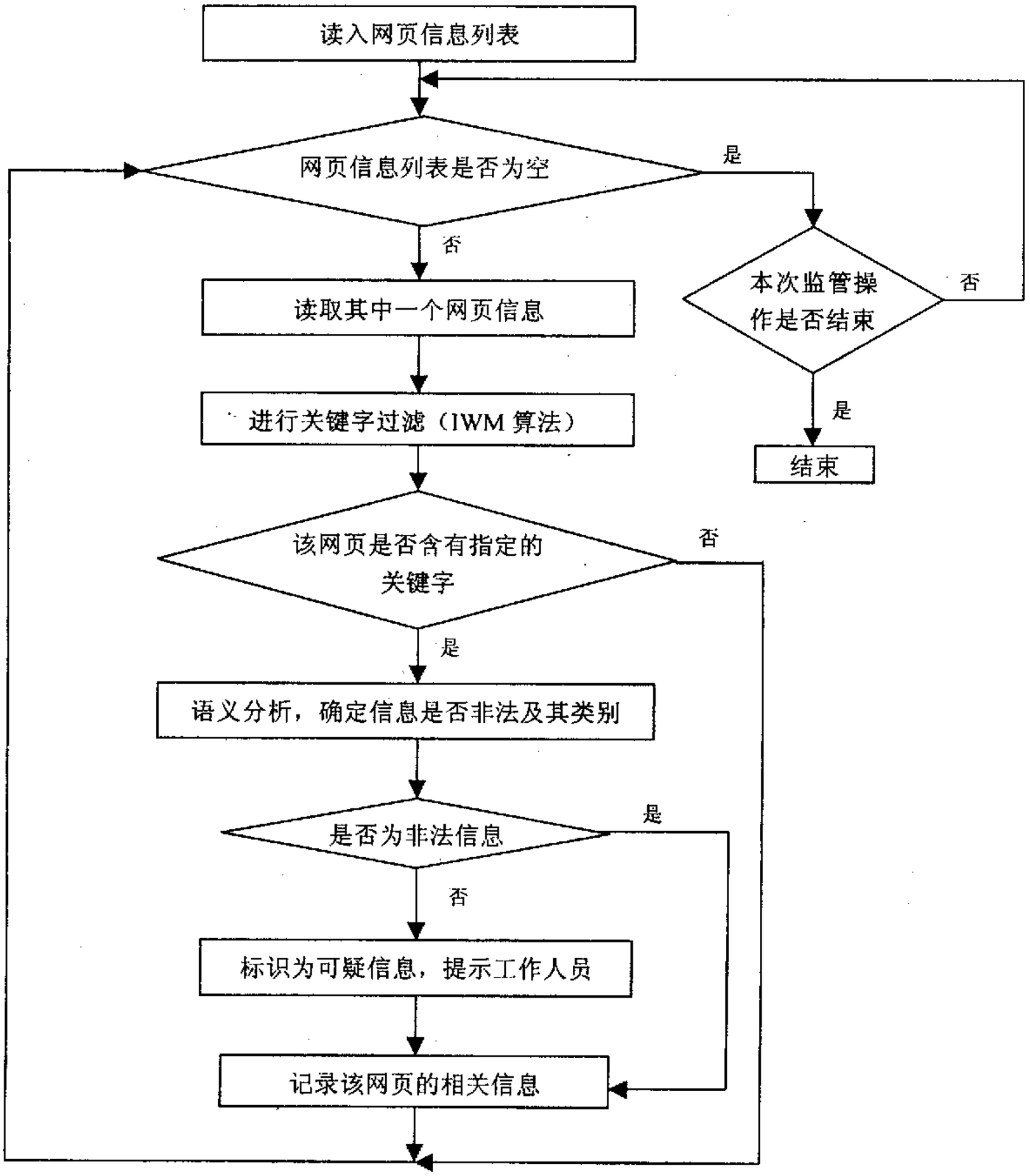



图 5.2 IWM 算法在敏感信息识别模块中的集成实现

算法对于多个关键字的预处理结果映射为文件形式，当匹配开始时，由系统调入即可。IWM 算法在敏感信息识别模块中的集成如图 5.2 所示。

另外，在 IWM 算法的集成过程中，还要考虑到和其它模块的接口定义。在 5.1 节中，我们已经介绍了为实现串匹配算法应用于关键字过滤模块而设计的相关数据库表。在敏感信息识别工作开始之前，主控模块从数据库中读取相关的敏感信息类别表和关键字信息表，以及实时获取信息采集模块所采集到的网页信息，并传送给关键字过滤模块，关键字过滤模块根据处理的结果再传送给语义识别模块。敏感信息识别模块也要将处理的结果信息传送给主控模块，包括非法信息和可疑信息的基本信息内容、对应的敏感信息类别、URL 等。 在此期间，当网页信息列表为空时，可能有两种情况：第一种情况是敏感信息识别模块已处理完信息采集模块所采集的全部信息，此时信息采集模块已停止工作；第二种情况是由于各种原因，信息采集模块的采集速度低于敏感信息识别模块的处理速度，此时敏感信息模块处于等待状态。

## 5.3 系统集成实现

### 5.3.1 接口定义

在系统集成阶段，主要考虑主控模块如何协调信息采集模块和信息处理模块共同工作，其模块间的参数传递如图 5.3.1 所示。

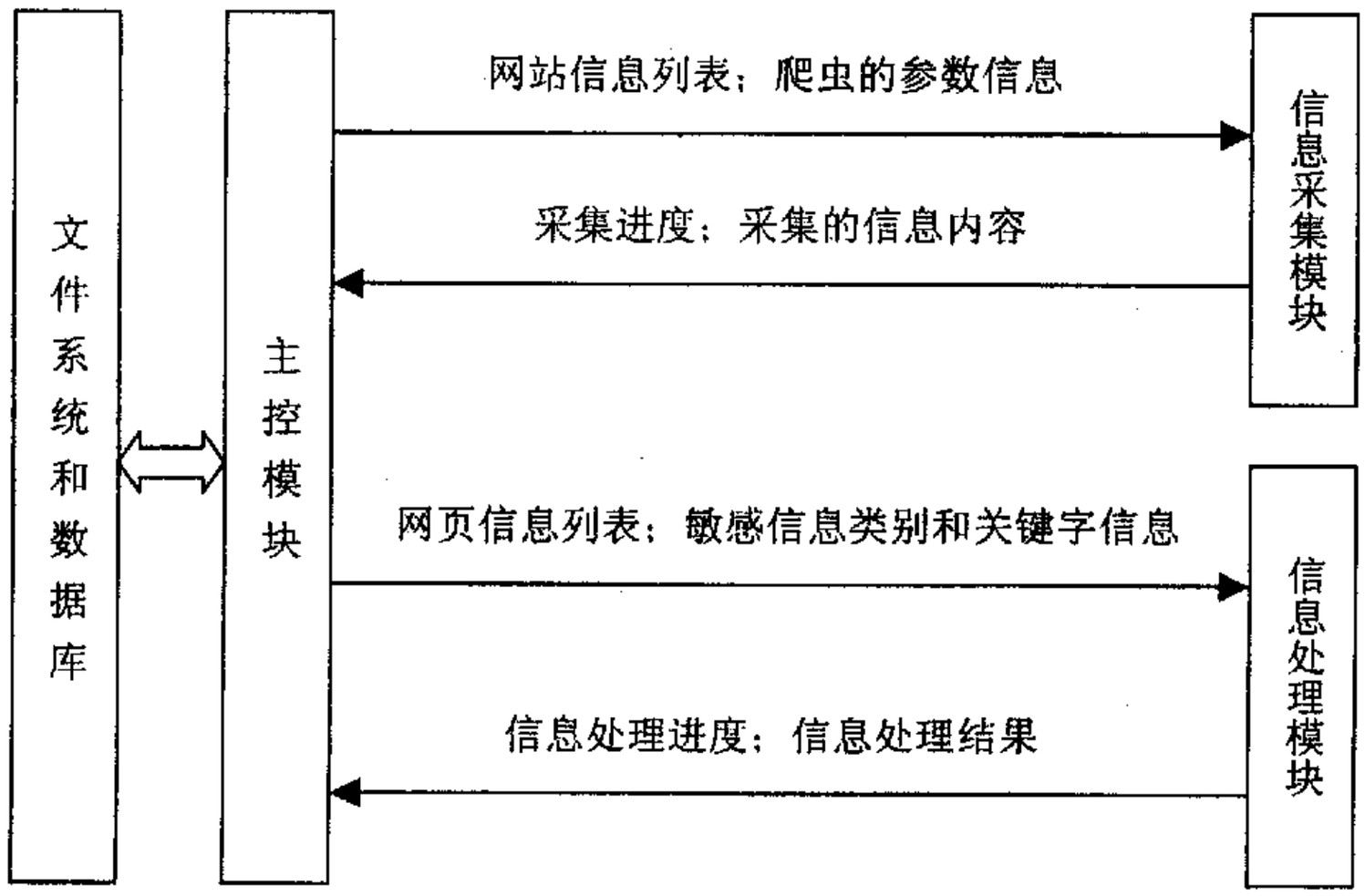



图 5.3.1 网络内容监管系统中模块间的参数传递

主控模块在监管过程中主要的工作是：启动信息采集模块开始采集网络数据，当第一条网页信息采集回来后，开启信息处理模块，对该网页信息进行处理分析，以此判断是否为非法信息。在此过程中，主要通过网站信息列表和网页信息列表进行信息互联。在实际的系统中，主控模块为在图形用户界面（GUI）上实时的显示监管信息，要实时访问信息采集模块获取当前时刻的信息采集情况。同时，也要实时访问信息处理模块获取当前时刻对网页信息的处理情况。根据上述传递的参数，我们可以以此定义出模块的接口。

### 5.3.2 系统流程

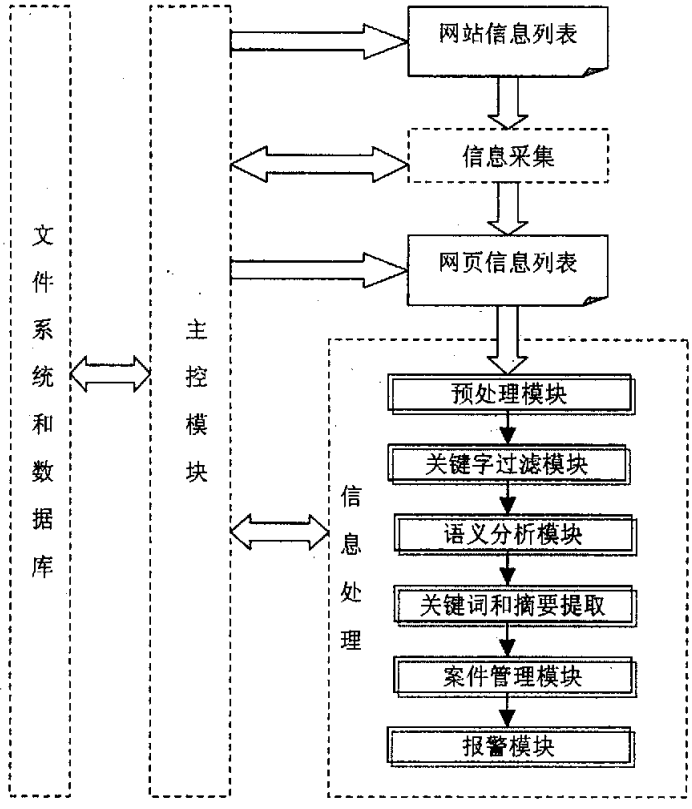



图 5.3.2 网络内容监管系统流程图

在新闻出版总署网络内容监管系统中，我们将对文件系统和数据库的全部操作让主控模块来完成。当信息采集模块把网站信息列表里的某一网站进行信息采集结束时，通知主控模块填写数据库，记录对该网站的监管信息；当网站信息列

表中的网站全部监管结束时，通知主控模块在数据库中记录对本次监管的操作信息。在采集过程中，网页将临时存放在文件系统中，与之相关的信息填写在网页信息列表中。当信息处理模块判定某一信息为合法时，将删除相应的临时文件；当为非法或可疑时，整合相应的信息内容并通知主控模块，主控模块以此填写数据库并管理相应的文件系统。

## 5.4 系统主要功能介绍

本节主要介绍新闻出版总署网络内容监管系统的一些主要功能。图 5.4.1 为系统运行的主界面。界面左边为一树型结构，按功能分为：查看、监管、设置和

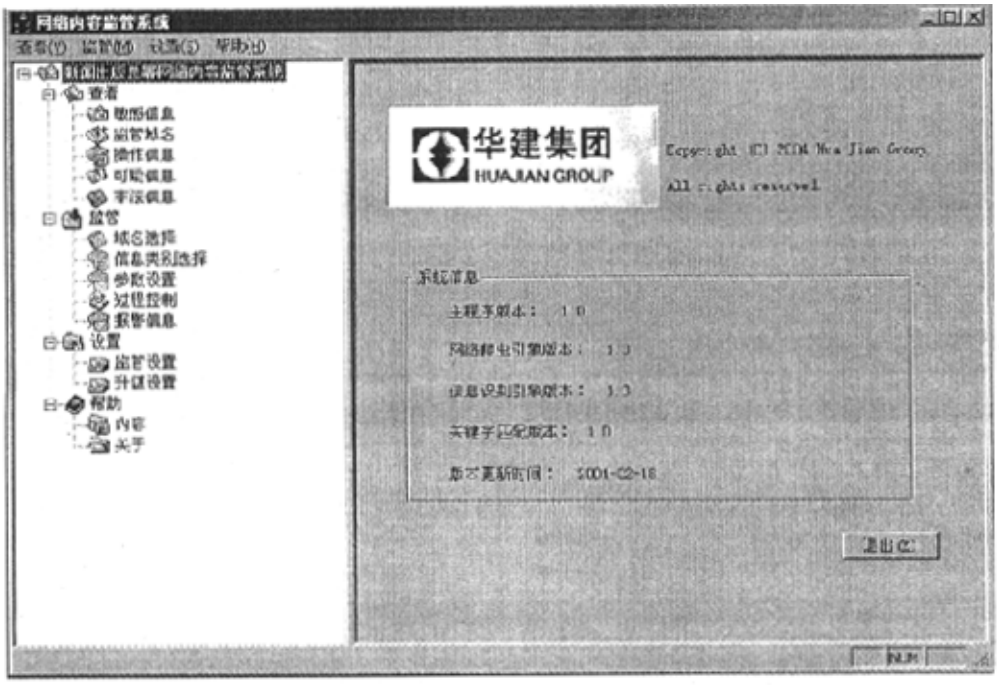



图 5.4.1 网络内容监管系统主界面

帮助。查看主要是查阅敏感信息、监管域名、历史操作、可疑信息和非法信息。监管主要是监管工作的流程，按照次序是域名选择、信息类别选择、参数设置、过程控制，当监管工作开始后，报警信息一栏显示本次监管所发现的非法信息和可疑信息。设置主要分为监管设置和升级设置，而帮助则显示帮助内容和关于该系统的一些信息。主界面的右边显示该系统的一些版本信息。

网络内容监管系统的信息采集和信息处理分别如图 5.4.2 和 5.4.3 所示。

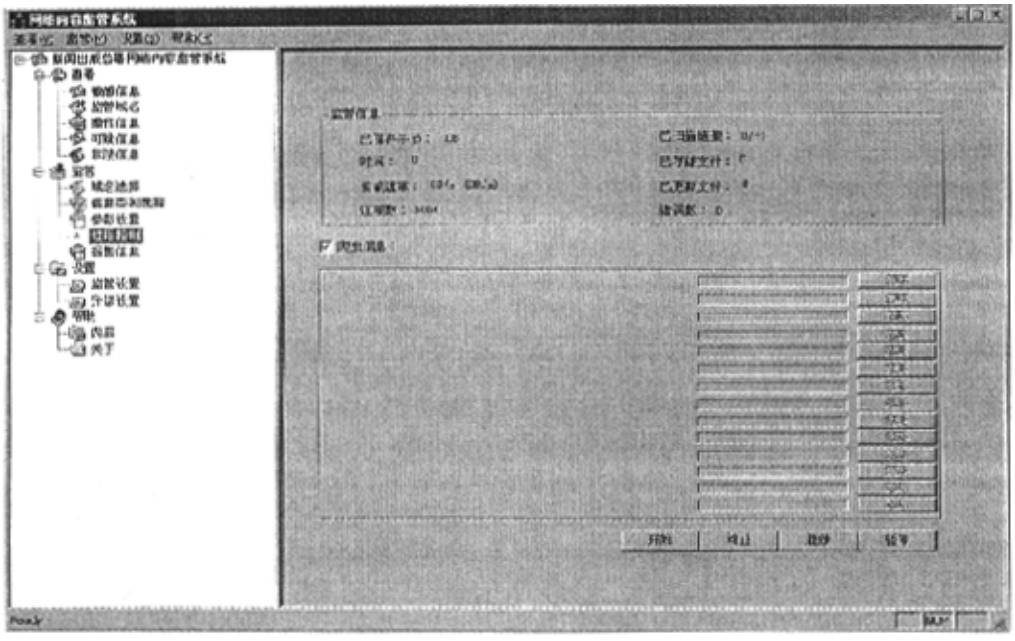


图 5.4.2 网络内容监管系统信息采集界面

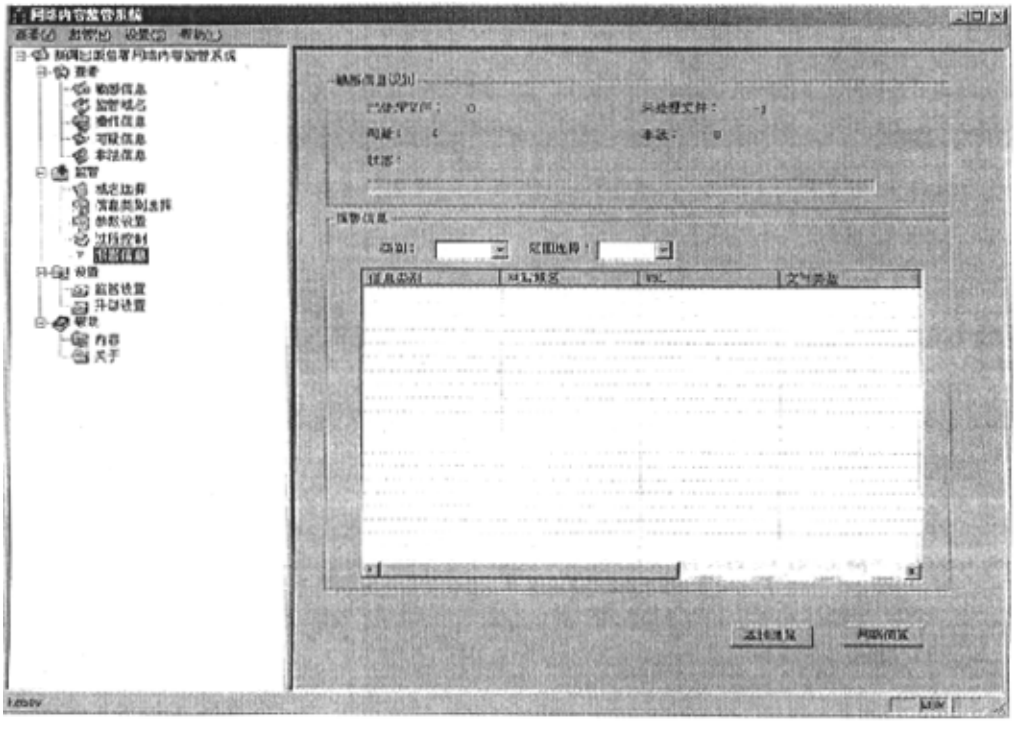


图 5.4.3 网络内容监管系统信息处理界面

## 5.5 系统测试

新闻出版总署网络内容监管系统在 Windows2000 环境下实现，开发环境为 Microsoft Visual C++6.0，核心代码采用 C++编写，可方便移植到 Linux 平台，数据库使用 Microsoft SQL Server 2000 数据库。测试环境：PentiumIII870MHz，256MB 内存，Windows 2000 Advanced Server。

郑重声明：在测试阶段，本文模拟了一个含有法轮功非法信息的网站，所涉及到的非法信息和测试网站及其域名纯属学术研究，绝无其它用意。

一、测试结果分析

该测试网站的所有页面为 HTML 格式。测试通过指定该网站的域名进行信息采集，然后再进行信息处理。鉴于新闻出版总署高速带宽（100Mbps）的实际运行环境，测试在局域网（10Mbps）内进行。图 5.5.1 为信息处理中的运行结果，信息处理和信息采集的测试结果分别如图 5.5.2 和 5.5.3 所示。结果显示该网站的链接总数为 691 个，实际链接数 659 个，在这 659 个网页链接中含有指定关键字的网页有 293 个，其中非法信息数为 6 个，且这 6 个非法信息属于法轮功类别和反动类别，实际传输速率 7705KiB/s，信息采集时间为 1 分 7 秒钟，信息处理模块大约在信息采集完毕后 6 秒钟结束，信息的采集和处理总时间约为 1 分 13 秒。

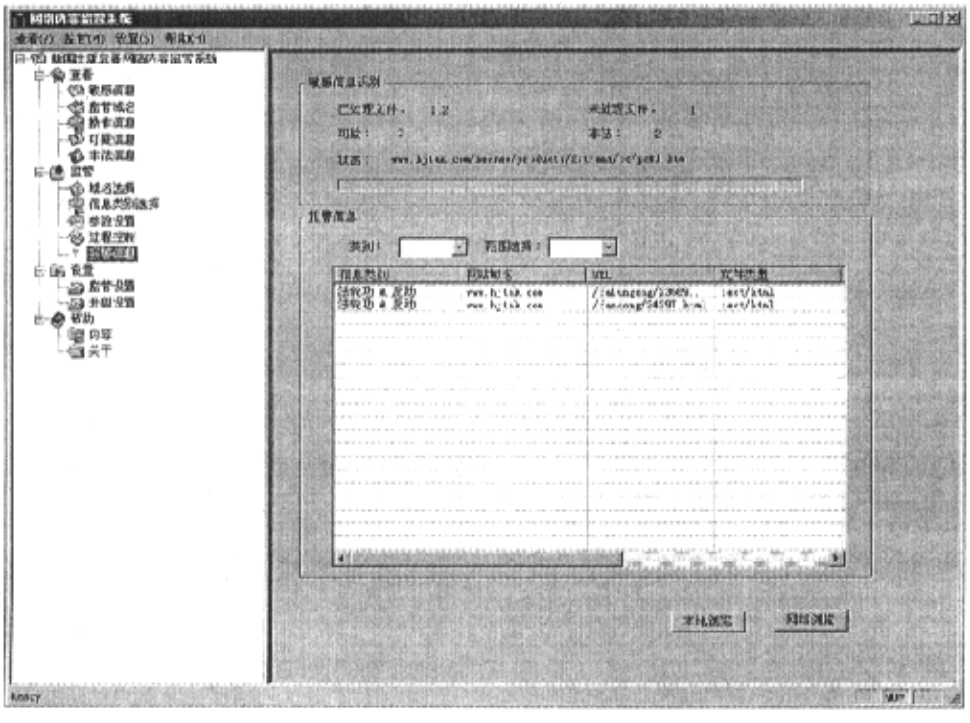


图 5.5.1 信息处理中的运行结果

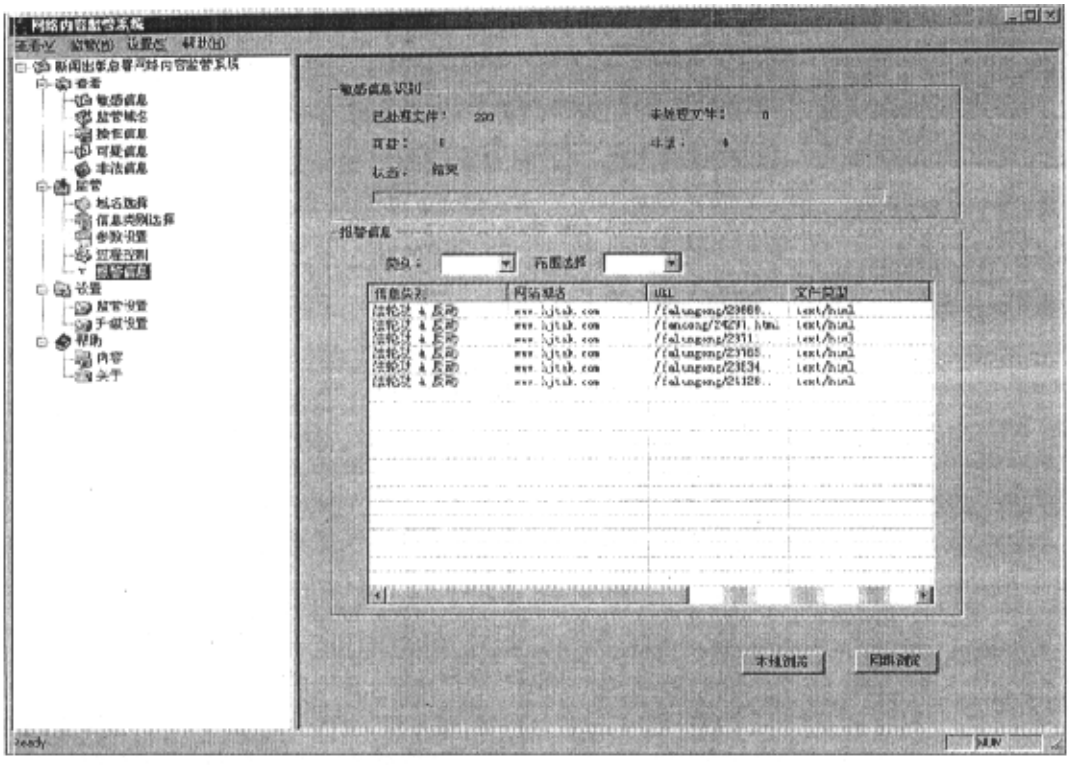


图 5.5.2 信息处理测试结果

在设计该测试网站时，我们考虑到以下情况：

1) 相对于网络上的海量数据，非法信息的数量较少。

2) 在含有敏感信息类别的关键字的信息内容中，非法信息的数量相对也较少。

3) 网络中存在诸多异常情况，例如无效链接等。

因此，在考虑到以上情况之后，上述实验结果，即 691 个链接中有效链接数为 659 个，含有指定关键字的网页链接只有 293 个，其中非法信息为 6 个，是符合实际运行情况的。

在实际应用中，系统的运行情况较为复杂，这主要由网站的性质、敏感信息在该网站中的分布和实际的运行环境所决定。考虑到新闻出版总署实际运行环境的带宽为 100Mbps，我们将对本系统的测试放在局域网中进行。测试结果表明，在对含有 691 个链接的网站进行测试时，信息采集和处理的总时间约为 1 分 13 秒，系统在运行速度方面已符合了新闻出版总署的基本需求，从信息识别的准确率来看，预先用来测试的非法信息网页已全部被识别出来，同时，其它 287 个含有指定关键字但不属于非法信息的网页没有出现误报现象。

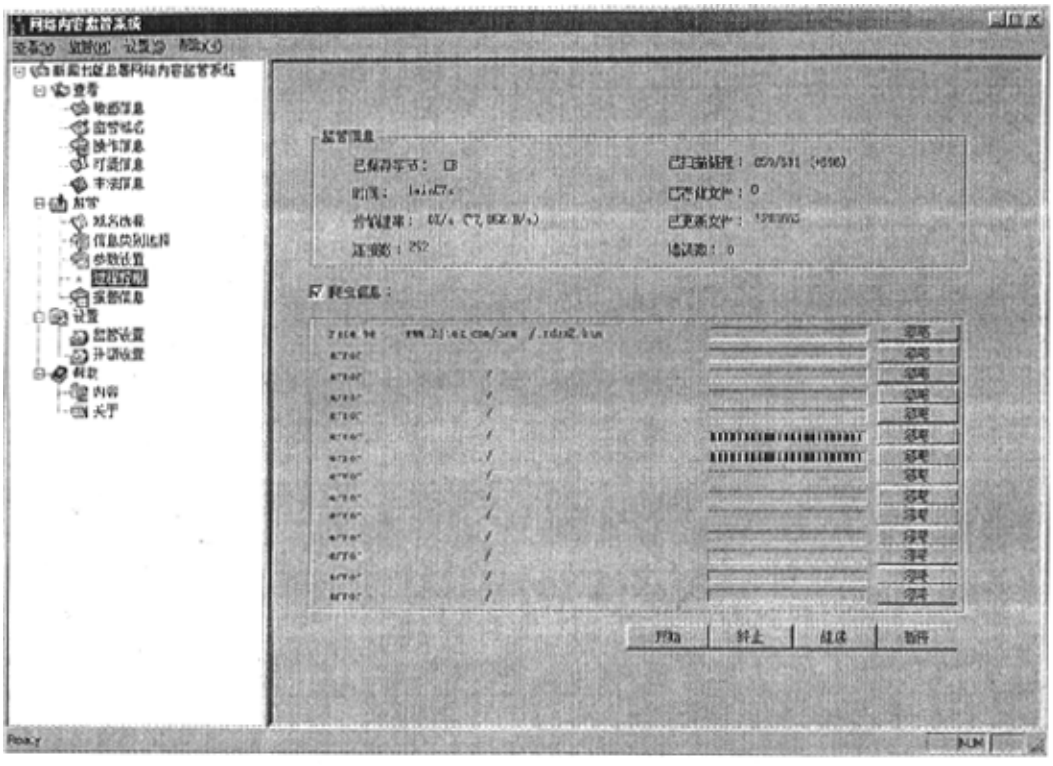


图 5.5.3 信息采集测试结果

二、评价

新闻出版总署网络内容监管系统由中科院华建集团和中科院计算机语言信息工程研究中心开发设计，采用自身强大的中文信息处理研究成果，设计出的系统实用性强，能够满足新闻出版总署的需求。测试结果表明其运行速度和信息识别的准确率等各项指标都已达到预先目标。鉴于网络内容监管在现实应用中具有很重要的意义，在设计阶段已考虑到今后的发展趋势，例如该系统可以作为个人的搜索引擎，搜索用户所关心的内容；也可以将该系统应用于工商、安全等部门领域，监管特定的信息内容。因此，新闻出版总署网络内容监管系统的实现为推出关于网络内容监管方面的产品奠定了坚实的基础。

# 6 结束语

## 6.1 本文总结

随着网络化的普及，越来越多的人们意识到网络中的非法信息已经严重影响了人们的正常生活和工作、青少年的健康成长以及国家的安定团结。本文首先对网络内容监管的研究现状进行了分析，阐述了网络内容监管的重要性，并且探讨了网络内容监管的相关技术。本文以新闻出版总署网络内容监管系统的实际项目为背景，在对现有的网络内容监管系统的分析之后，采用层次分类的方法，即在信息识别过程中，先将网络信息进行关键字过滤，然后再将含指定关键字的信息内容传送给语义分析模块进行处理。这样的设计会减轻语义分析模块的压力，提高系统的效率，并且更适应于非法信息多变的特点。

鉴于以上的分析，串匹配算法成为系统性能的瓶颈，这是因为关键字过滤模块通过串匹配算法来实现，因此采用一个高效的串匹配算法对于提高网络内容监管系统的性能是至关重要的。本文在分析了现有主要的几个串匹配算法（包括单模式串匹配算法和多模式串匹配算法）之后，通过对需求的仔细分析，设计实现了一个基于 QS 算法的单模式串匹配算法（IQS 算法）和一个基于 Wu-Manber 算法的多模式串匹配算法（IWM 算法）。IQS 算法主要用于工作人员的日后查阅，而 IWM 算法主要用于网络内容监管中关键字的过滤工作。之后，本文对 IQS 算法和 IWM 算法进行了实验比较分析，结果表明两个算法都获得了较好的效果。

然后，本文将 IQS 算法和 IWM 算法集成于网络内容监管系统，并集成实现于网络内容监管系统。最后对该系统进行了测试，实验结果表明，系统具有较快的运行速度和较好的信息识别率，其它性能指标均已达到预定要求，满足实用化的需求。

## 6.2 今后的研究方向

当前，已有越来越多的科研机构、企业和国家关注网络内容监管，网络内容监管不仅关系到个人，更对网络的健康发展、国家的安定团结有着举足轻重的作用。为此，关于网络内容监管方面本文今后还有以下两个研究方向。

1)　并行串匹配算法的研究

由于计算机的快速发展、网络技术的应用以及海量数据处理的需要，使得对计算机运算速度的要求也越来越高。但传统的计算机是串行结构，每一时刻只能按一条命令对一个数据进行操作。为克服此限制，人们将并行技术引入到计算机的结构设计中，以此来提高计算机的运算速度和吞吐量。在国内外，对并行串算

法的研究十分广泛[52,53]，尤其是在生物信息学领域[54]。研究并行串匹配算法对于处理海量数据具有重要的实际意义，因此可以作为本课题今后的一个研究方向。

2) 网格环境下网络内容监管的研究

网格是继 Internet、Web 之后的网络应用的第三次浪潮。传统 Internet 实现了计算机硬件的连通，Web 实现了网页的连通，而网格试图实现互联网上应用层面的互联互通，这就意味着所有资源的全面连通，包括计算资源、存储资源、通信资源、软件资源、信息资源、知识资源等等[55]。因此，在网格环境下研究网络内容监管将更有深远意义。

# 致　谢

两年的硕士学习转瞬即逝。

首先感谢母校南京理工大学。

真诚感谢导师黄河燕研究员。黄老师渊博的知识、严谨的作风、开拓的思想使我受益匪浅。感谢黄老师在北京做课题的一年时间里，为我们提供了一个优越的学术环境，并且在本文的选题和论点的论证过程中给予的帮助和指点。

衷心感谢南京理工大学的王树梅教授。她无微不至的关怀和帮助，给我创造了良好的学习环境，使我在南京学习的一年时间里打下了扎实的理论基础，在本文的写作过程中给予我许多帮助和指点。

感谢研究生在读期间的各位任课老师。他们的悉心授课为本文的写作打下了坚实的基础。同时，他们兢兢业业、严谨的工作态度和强烈的责任心都是我终身学习的榜样。

我还要感谢同在中科院计算机语言信息工程研究中心实习的师兄代六玲博士。在实习阶段，师兄如兄长般的关心和帮助我，本文的写作过程中也给我提出了诸多建议和意见。

感谢我的师兄、项目负责人郭琰，师兄对项目认真负责的态度和一丝不苟的精神是我永远学习的榜样。

感谢黄卫东高级工程师和徐海英在本文完成过程中给予的帮助。

感谢丁会平、鲁耀杰、王彬、李宁、龙丽君、李辉六位同门在长达两年的时间内给予的生活上和学术上的帮助，跟他们相处，让我感觉到大家庭的温暖。

感谢我的好友李贯峰、杜威、马婧和丁学科，感谢他们在论文期间给予的帮助和鼓励。

最后，我要感谢我的父亲母亲，感谢他们一直以来对我的支持与鼓励，他们是我前进的动力，在我的心目中，他们永远是最伟大的人。

感谢所有关心我的学长和朋友们。

# 参考文献

[1] 中华人民共和国公安部. http://www.mps.gov.cn/webPage/showfaguisecond.asp?IDitem=363&nameitem=公共信息网络安全监察常用法律法规.

[2] http://www.epic.org/free_speech/copa/.

[3] 赵俊玲. 网络信息过滤在美国公共图书馆中的应用. 图书馆理论与实践，2004，1：84-85，94.

[4] A Guide to the RIP v1.0. http://www.stand.org.uk/ripnotes/.

[5] 中国互联网络信息中心（CNNIC）. 2003 年中国互联网络信息资源数量调查报告. http://www.cnnic.net.cn.

[6] 美国联邦调查局. http://www.fbi.gov/congress/congress00/kerr090600.htm.

[7] ECHELON: America's Secret Global Surveillance Network. http://fly.hiwaay.net/~pspoole/echelon.html.

[8] http://archive.aclu.org/echelonwatch/networks.html.

[9] 刘琦，李建华. 网络内容安全监管系统的框架及其关键技术. 计算机工程，2003，29(2)：287－289.

[10]程圣宇，白英杰，肖瀛，芦东昕. 高速网络内容监控若干关键技术. 计算机应用，2003，23：365－367.

[11]G. Navarro. A guided tour to approximate string matching. ACM Computing Surveys, 2001, 33(1): 31–88.

[12]P.D. Michailidis, K.G. Margaritis. Parallel Implementations for String Matching Problem on a Cluster of Distributed Workstations. Neural Parallel and Scientific Computations, 2002, 10: 287-312.

[13]Aoe, J. Computer Algorithms: String Pattern Matching Strategies. Los Alamitos, California: IEEE Computer Society Press, 1994.

[14]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

[15]Mike Fisk, George Varghese. Applying Fast String Matching to Intrusion

Detection. http://public.lanl.gov/mfisk/papers/setmatch-raid.pdf.

[16]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.吉林大学学报(信息科学版)，2003，21(2)：167-171.

[17]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

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

[19]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.

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

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

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

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

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

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

[26]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.

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

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

[29]Karp R.M., Rabin M.O. Efficient randomized pattern-matching algorithms. IBM

J. RES. DEVELOP, 1987, 31(2): 249-260.

[30] D.M. Sunday. A very fast substring search algorithm. Communications of the ACM, 1990, 33(8): 132-142.

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

[32] 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.

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

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

[35] 中国互联网络信息中心（CNNIC）. 中国互联网络发展状况统计报告（2004年1月）. http://www.cnnic.net.cn.

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

[37] Hancart C. On Simon's string searching algorithm. Information Processing Letters, 1993, 47(2): 95-99.

[38] A. Apostolico and M. Crochemore. Optimal canonization of all substrings of a string. Information and Computation, 1991, 95(1): 76-95.

[39] Colussi L. Correctness and efficiency of the pattern matching algorithms. Information and Computation, 1991, 95(2): 225-251.

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

[41] 中华人民共和国新闻出版总署. http://www.gapp.gov.cn/Templates/zsjs.htm.

[42] LECROQ T. Experimental Results on String Matching Algorithms. Software - Practice & Experience, 1995, 25(7): 727-765.

[43] P.D. Michailidis, K.G. Margaritis. On-line String Matching Algorithms: Survey and Experimental Results. http://macedonia.uom.gr/~panosm/pdfs/jpaper001.pdf.

[44]http://www.research.att.com/~lewis/reuters21578.html.

[45]沈洲，王永成，刘功申. 改进的中文字串多模式匹配算法. 情报学报，2002，21(1)：27-32.

[46]许一震，王永成，沈洲. 一种快速的多模式字符串匹配算法. 上海交通大学学报，2002，36(4)：516－520.

[47]王永成，沈洲，许一震. 改进的多模式匹配算法. 计算机研究与发展. 2002，39(1)：55－60.

[48]沈洲，王永成，许一震. 一种面向中文的快速字串多模式匹配算法. 上海交通大学学报，2001，35(9)：1285－1289.

[49]吴楠，朱怀宏，夏黎春. 一种应用于现代网络搜索引擎的快速串匹配算法. 计算机与现代化，2003，11：7－11.

[50]张鑫，潭建龙，程学旗. 一种改进的 Wu-Manber 多关键字匹配算法. 计算机应用，2003, 23(7)：29－31.

[51]王永成. 中文信息处理技术及其基础. 上海交通大学出版社，1990，30－31.

[52]Moussouni F, L avault C. Distributed string matching algorithm on the N-cube. Proceedings of the 1st European Conference on Parallelism (EuroPar'96), Lecture Notes in Computer Science 1123 Springer-Verlag, 1996, 832-835.

[53]陈国良，林洁，顾乃杰. 分布式存储的并行串匹配算法的设计与分析. 软件学报，2000，11(6)：771-778.

[54]中科院计算所生物信息室. http://www.bioinfo.org.cn/orientation.htm.

[55]中国网格. http://www.chinagrid.com/data/grid-06.htm.

# 附　录

## 附录 A　IQS 算法程序代码

```
#define ASIZE 256
#include<TIME.H>

void preQsBc(const unsigned char *pat, int m, int iqsBc[])
{
    int i;
    for (i = 0; i < ASIZE; ++i)
        iqsBc[i] = m + 1;
    for (i = 0; i < m; ++i)
        iqsBc[pat[i]] = m - i;
}

void preQsBc1(const unsigned char *pat, int m, int iqsBc1[])
{
    int i;
    for (i = 0; i < ASIZE; ++i)
        iqsBc1[i] = m + 1;
    for (i = m-1; i >=0; i--)
        iqsBc1[pat[i]] = i+1;
}

void IQS_Match(register unsigned char *pat, register int m, register unsigned char *txt, long n)
{
    register int i, j=0, k=n-m;
    register int iqsBc[ASIZE], iqsBc1[ASIZE];
    register int Attempts=0, Matches=0;    //尝试次数，匹配次数
    register long duration=0;      //运行时间
    clock_t start, finish;

    /* 预处理 */
    preQsBc(pat,m,iqsBc);
    preQsBc1(pat,m,iqsBc1);

    start = clock();
    /* 开始匹配 */
    while (j<=k) {
        Attempts++;          //记录尝试次数
        for(i=0; i<m && pat[i] == txt[i+j]; i++);
```

```
        if (i == m)
            Matches++;          //记录匹配次数
        j += iqsBc[txt[j + m]];         //自左向右扫描

        Attempts++;
        for(i=0; i<m && pat[i]==txt[i+k]; i++);
        if (i == m)
            Matches++;
        k -= iqsBc1[txt[k - 1]];        //自右向左扫描
    }
    finish = clock();
    duration += (finish - start);       //记录匹配时间
}
```

## 附录 B　IWM 算法程序代码

1. 文件 IWM.h

```
#define MAXPAT   40  //单个模式串的最大长度
#define ASIZE   256    //ASCII 表的大小
#define MAXSHIFT 32768   //SHIFT 表长度最大值
#define MAXHASH 32768   //HASH 表长度最大值
#define MAX_NUM_PAT      10000  //最大模式串个数
#define MAXPATFILE 100000    //模式串集合的最大长度
#define HBITS   5 //左移大小

#include<VECTOR>
#include<STRING>
#include<TIME.H>

using namespace std;

class IWM
{
private:
    int B;          //字符块大小 B=2
    clock_t start, finish;      //记录匹配的开始时间和结束时间
    unsigned char buf[MAXPATFILE];
    unsigned char pat_spool[MAXPATFILE];

public:
    int minlen;          //最短模式串长度
    unsigned char *patt[MAX_NUM_PAT];     //每个模式串指针
    unsigned char pat_len[MAX_NUM_PAT];  //每个模式串大小
    unsigned char SHIFT_QS[MAXSHIFT];          //跳转表
    struct Pat_List {
```

```
        int index;
        struct Pat_List *next;
    } *HASH[MAXHASH], *pt, *qt;        //HASH 表所指向的模式串链表
    register int Attempts, Matches;    //尝试次数，匹配次数
    register long duration;            //运行时间

protected:
    //对每个模式串进行预处理
    void Pat_Preprocess(int pat_index, const unsigned char *Pattern);
public:
    IWM();
    ~IWM();
    //加载模式串，并对每个模式串调用 Pat_Preprocess 函数进行预处理
    int Load_Patterns(const vector<string> x);
    void Match(register unsigned char *text, long len);    //匹配过程
};
```

2. 文件 IWM.cpp

```
#include "IWM.h"

IWM::IWM()
{
    B = 2;
    Matches = Attempts = duration = minlen = 0;
    for(int i=0; i<MAXSHIFT; i++)   //初试化 HASH 表
        HASH[i] = 0;
}

IWM::~IWM()
{
    for(int i=0; i<MAXSHIFT; i++) {        //释放内存
        pt = HASH[i];
        while(pt!=0) {
            qt = pt->next;
            free(pt);
            pt = qt;
        }
    }
}

int IWM::Load_Patterns(const vector<string> x)
{
    int length = 0, p, i, num_pat;
    unsigned char *pat_ptr = pat_spool;
```

```
for(int j=0; j<x.size(); j++) {
        unsigned char *p = (unsigned char*)x[j].c_str();
        for(int k =0; k<x[j].length(); k++) {
                buf[length] = p[k];
                length++;
        }
        buf[length] ='\n';
        length++;
}

i = 0;       p = 0;
while(i<length) {
        patt[p] = pat_ptr;
        while((*pat_ptr = buf[i++])!='\n')
                pat_ptr++;
        *pat_ptr++ = 0;
        p++;
}

if(p>MAX_NUM_PAT) //模式串数目过多
        return 0;

num_pat = p;
minlen = MAXPAT;
for(i=0 ; i<num_pat; i++) {    // 判定最短模式串长度
        p = strlen((char*)patt[i]);
        pat_len[i] = p;
        if (p!=0 && p<minlen)
                minlen = p;
}
if (minlen == 0)        //模式串为空
        return 0;

for(i=0; i<MAXSHIFT; i++)
        SHIFT_QS[i] = minlen+1;    //最大跳跃距离

for(i=0; i<num_pat; i++)//对每个模式串进行预处理
        Pat_Preprocess(i, patt[i]);

        return 1;
}

void IWM::Pat_Preprocess(int pat_index, const unsigned char *Pattern)
```

```
{
    int i, m, hash;

    m = minlen;
    //设置在模式串第 1 至第 len 个字符中紧邻两个字符的最右出现位置的跳跃值
    for(i=0; i<=m-2; i++) {
        hash = (Pattern[i]<<HBITS)+Pattern[i+1];
        if (SHIFT_QS[hash] > m-i-1)
            SHIFT_QS[hash] = m-i-1;
    }
    //设置当长度为 2 的字符块有最大后缀时的跳跃值，其最大后缀长为 1
    for(int j = 0; j<ASIZE; j++) {
        hash = (j<<HBITS) + Pattern[0];
        if(SHIFT_QS[hash] > m)
            SHIFT_QS[hash] = m;
    }

    /* 建立 HASH 表索引 */
    hash = (Pattern[m-1]<<HBITS) + Pattern[m-2];
    qt = (struct Pat_List*)malloc(sizeof(struct Pat_List));
    qt->index = pat_index;
    pt = HASH[hash];
    qt->next = pt;
    HASH[hash] = qt;
        return;
}

void IWM::Match(register unsigned char *text, long len)
{
    register unsigned char *textend;
    register unsigned hash, hash_skip, htemp, i;
    register int m1;
    register unsigned char *px, *qx;
    register struct Pat_List *p;
    int pat_index, m = minlen;

    start = clock();
    /* 开始匹配 */
    textend = text + len;
    m1 = m - 1;
    text = text + m1;
    while (text<=textend)    {
        Attempts++;          //记录尝试次数
```

```
htemp = (*text)<<HBITS;
hash = htemp + (*(text - 1));  //查取 HASH 表的哈希值
hash_skip = htemp + (*(text + 1)); //查取 SHIFT 表的哈希值
p = HASH[hash];
while(p) {
    pat_index = p->index;
    p = p->next;
    px = patt[pat_index];
    qx = text - m1;
    j = 0;
    while(px[i]==qx[i])
        i++;
    if(pat_len[pat_index]<=i) {   //发生匹配
        Matches++;   //记录匹配次数
        if(text > textend)
            return;
    }
  } // end of while(p)
  text += SHIFT_QS[hash_skip];   //选择下一个可能匹配的入口位置
} // end of while(text<=textend)
finish = clock();
duration = finish - start; //记录匹配时间
return;
}
```

# 基于层次分类的网络内容监管系统中串匹配算法的设计与实现

作者： 章张
学位授予单位： 南京理工大学
被引用次数： 3次

## 本文读者也读过(10条)

1. 龙丽君 网络内容监管系统中基于局部信息的语义倾向性识别算法[学位论文]2004
2. 方勇.周安民.刘嘉勇.张志国.张雪峰 基于内容的网络应用监控系统研究与设计[会议论文]-2002
3. 吕金锁 探针技术在网络信息监控系统中的应用[学位论文]2007
4. 黄中清 基于报文内容的网络信息审计监控系统[学位论文]2004
5. 刘功申.王永成.胡佩华 大字符集语言单模式匹配算法[期刊论文]-上海交通大学学报2003,37(6)
6. 曾传璜.段智宏.Zeng Chuanhuang.Duan Zhihong 一种改进的QS串匹配算法[期刊论文]-计算机与数字工程2010,38(7)
7. 巫喜红.凌捷.WU Xi-hong.LING Jie 单模式精确匹配算法研究[期刊论文]-合肥工业大学学报（自然科学版）2007,30(7)
8. 李雪梅.代六玲.童新海.王雄.Li Xuemei.Dai Liuling.Tong Xinhai.Wang Xiong 对QS串匹配算法的一种改进[期刊论文]-计算机应用与软件2006,23(3)
9. 龚丽 基于语义网的内容监管技术的研究[学位论文]2003
10. 施敏 上海重要信息系统安全监管制度的研究[学位论文]2007

## 引证文献(3条)

1. 李雄伟.王希武.王盼卿 基于模式串匹配的Ethernet协议识别算法研究[期刊论文]-计算机工程与应用 2007(29)
2. 贺茹.李生琦 基于语言本体库的中文信息检索系统的研究[期刊论文]-价值工程 2007(10)
3. 薛传庆.韩明畅.金伟信 入侵检测系统中BM算法的改进[期刊论文]-计算机技术与发展 2011(6)