

Matlab 环境下素数筛选算法的分析及比较

张琦¹, 许勇²

(1. 西北大学软件学院, 陕西 西安 710127;

2. 安徽师范大学数学计算机科学学院, 安徽 芜湖 241000)

摘要:利用 Matlab 对矩阵科学运算的支持, 在 Matlab 环境下对埃拉托斯特尼筛法、Dirichlet 定理衍生的素数筛法、辛答拉姆筛法和基于奇合数分解式的素数筛法进行算法实现和初步优化, 并测试其性能, 研究发现在计算大数范围的素数表时, 算法之间的性能差异明显。通过对这些算法的比较和评价, 分析各个算法的优缺点。研究表明, 对于不同环境要求不同的待解问题需要选取合适的素数筛选算法, 因此, 文中结论具有一定的指导意义和实际参考价值。

关键词: Matlab; 素数; 筛法

中图分类号: TP301.6

文献标识码: A

文章编号: 1673-629X(2009)03-0095-04

Analysis and Comparison of Several Sieve Methods on Prime Number Searching with Matlab

ZHANG Qi¹, XU Yong²

(1. College of Software, Northwest University, Xi'an 710127, China;

2. College of Mathematics & Computer Science, Anhui Normal University, Wuhu 241000, China)

Abstract: With the support of matrix computing of Matlab, several computer algorithms of prime number searching sieve methods with Matlab are implemented and some improvement of the algorithms are also be done. After researching and testing the performance of the programs, obvious difference among these algorithms is discovered when the test data is large. By analysis and comparison, the advantages and drawbacks of these algorithms are given. The research proves that different environments and different related problems need the appropriate sieve method to get the prime number distribution. So the research result of this paper has some significance and practical values.

Key words: matlab; primenumber; sieve method

0 引言

素数的性质及其分布是数论研究的核心问题之一, 目前人类尚未找出素数的分布规律。但随着计算技术的快速发展, 在一个大数范围内快速筛选出素数已经成为可能。素数的筛选方法很多, 如埃拉托斯特尼筛法、Dirichlet 定理衍生的素数筛法、辛答拉姆筛法、基于奇合数的分解式的素数筛法等, 此类方法在计算机算法中均有实现, 但对这些算法的实现讨论并不多见。文中选择 Matlab 环境, 针对目前应用的上述素数筛选算法进行分析及比较。

1 几种常见的素数筛选算法及其实现

1) 埃拉托斯特尼筛法^[1]。

简称埃氏筛或爱氏筛, 是一种由埃及数学家埃拉托斯特尼所提出的一种简单检定素数的算法。他在寻找整数 N 以内的素数时, 先将 $2-N$ 的各数写在纸上: 在 2 上面做上标记, 然后划去 2 的其他倍数; 第一个既未标记又没有被划去的数是 3, 将它标记, 再划去 3 的其他倍数; 现在既未标记又没有被划去的第一个数是 5, 将它标记, 并划去 5 的其他倍数……依此类推, 一直到所有小于或等于 N 的各数都做上标记或划去为止。这时, 做上标记的以及未划去的那些数正好就是小于 N 的素数。

算法实现: 先设定一个 $1 \times N$ 的单位向量(假设要筛选的范围为 N), 向量中的零代表非素数, 非零(一般是 1)代表素数, 筛选时将准备筛去的非素数在行向量中对应值设为 0, 筛选后剩下的行向量中值为 1 的数

收稿日期: 2008-06-15

基金项目: 安徽省自然科学基金重点项目(2005kj009ZD)

作者简介: 张琦(1987-), 男, 安徽芜湖人, 研究方向为程序和算法设计; 许勇, 教授, 博士, 硕士生导师, 主要从事计算机网络、网络安全、网络教育方面的研究。

即为素数。优化后的实现如下:

(1) 首先筛选能够被 2 除的数(因大于 2 的素数都是奇数)。

(2) 每次筛除某个素数的倍数时,因乘数小于该素数的积都已经被作为小于该素数的积被筛除,因此倍数应该从该素数的平方开始。

2) Dirichlet 定理衍生的素数筛法。

因为 $6n \mp 1$ 中不含 2, 3 素数, $6n$ 合数中不含 2, 3, 5 素数, 所以自 11, 13, 17, 19, 23, 29, 31, 37 起, 以这 8 个素数分别为首项, 公差均为 30, 可构成 8 个等差级数。每个等差级数称为一系, 故称八系, 八系中不含 2, 3, 5, 7 素数^[2]。

定理^[3] 设 $X, X_1, X_2 = 7, 11, 13, 17, 19, 23, 29, 31; m, m_1, m_2 = 0, 1, 2, \dots$, 则

(1) ≥ 7 的素数集合是

$$\{X + 30m\} \setminus \{(X_1 + 30m_1)(X_2 + 30m_2)\}$$

(2) 全体素数集合是

$$\{2, 3, 5\} \cup \{X + 30m\} \setminus \{(X_1 + 30m_1)(X_2 + 30m_2)\}$$

算法实现:

(1) 先筛选出 $X + 30m$ 的集合($X = 7, 11, 13, 17, 19, 23, 29, 31; m = 0, 1, 2, \dots$), 这个集合不包括 2, 3, 5, 但可能包含能被分解为形如 $(X_1 + 30m_1)(X_2 + 30m_2)$, $X_1, X_2 = 7, 11, 13, 17, 19, 23, 29, 31; m_1, m_2 = 0, 1, 2, \dots$ 的非素数。

(2) 继续对上面的集合进一步筛选。定理可以展开为便于编程的八个素数差集的形式^[3], 如 $\{7 + 30m\} \setminus \{(X_1 + 30m_1)(X_2 + 30m_2)\}$, X_1, X_2 可以分别取 A1 到 A4 各组数值, $m, m_1, m_2 = 0, 1, 2, \dots$ 。

$$A1: X_1 = 7, X_2 = 31 \quad A2: X_1 = 11, X_2 = 17$$

$$A3: X_1 = 13, X_2 = 19 \quad A4: X_1 = 23, X_2 = 29$$

对另外七系也可得出对应的取值, 在这里不全部列出。

该算法的优化方案如下:

a. 预先将上述八系数差集取值组合算式的最小值求出并按从大到小排序, 根据 N 的值和组合的最小值决定对其中的不超过 N 值的子集式进行计算, 减少了 1000 以内的不必要的乘法运算。

b. 若子集式中的 X_1, X_2 取值相同, 则 m_2, m_1 的取值范围为 $m_2 = m_1, m_1 + 1, \dots, m_1 = 0, m_1 = 0, 1, 2, \dots$ 。避免因两者相同而造成重复乘法和赋值运算。

3) 辛答拉姆筛法^[4]。

构造一个数表(见表 1) 使第一行为首项为 4、公差为 3 的数列 4, 7, 10, ..., 同时第一列也为此数列, 并且自第二行起每一行的公差依次是 5, 7, 9, 11, ...

表 1 辛答拉姆数表

4	7	10	13	16	19	22	...
7	12	17	22	27	32	37	...
10	17	24	31	38	45	52	...
13	22	31	40	49	58	67	...
16	27	40	49	60	71	82	...
...

该表有一个很好的性质: 若 k 出现在这个表中, 则 $2k + 1$ 不是素数, 若 k 没有出现在这个表中, 则 $2k + 1$ 是素数。如下表:

表中没有出现的数 n	1	2	3	5	6	8	9
$2n + 1$ 都是素数	3	5	7	11	13	17	19

由此可以得出一个新的素数算法。

算法实现:

(1) 建立一个 $1 \times N$ 的单位行向量。

(2) 建立 N 以内的辛答拉姆数表, 在行向量中把表中的元素设为 0。

(3) 通过系统函数找出向量中的非零元素, 也就是表中未出现过的元素。

(4) 对表中未出现的元素进行 2 倍加 1 操作, 最后得出 N 范围内的素数表。

4) 基于奇合数的分解式的素数筛法^[5]。

设 N 为全体正整数(自然数)集, $N1$ 为奇数集, $N2$ 为偶数集, M 为奇合数集, P 为素数集, $M1$ 为除个位数为 5 以外的奇合数的集合, $M2$ 为个位数为 5 的奇合数的集合, 于是合数集便为 $N2 \cup M$, 素数集 $P = \{2\} \cup (N1 - M \cup \{1\})$ 。奇合数 $m \in M1$ 的构成及分布规律是

$$m \in \bigcup_{i=1}^{10} F_i, F_i = \{f_i\} \quad i = 1, 2, \dots, 10$$

素数 $p \in P - \{2, 3, 5, 7\}$ 的性质与分布满足条件:

$$m \notin \bigcup_{i=1}^{10} F_i, F_i = \{f_i\} \quad i = 1, 2, \dots$$

其中

$$f1 = f1(x, y) = (10x + 3)(10y + 7);$$

$$f2 = f2(x, y) = (10x + 9)(10y + 9);$$

$$f3 = f3(x, y) = (10x + 11)(10y + 11);$$

$$f4 = f4(x, y) = (10x + 3)(10y + 11);$$

$$f5 = f5(x, y) = (10x + 7)(10y + 9);$$

$$f6 = f6(x, y) = (10x + 3)(10y + 9);$$

$$f7 = f7(x, y) = (10x + 7)(10y + 11);$$

$$f8 = f8(x, y) = (10x + 3)(10y + 3);$$

$$f9 = f9(x, y) = (10x + 7)(10y + 7);$$

$$f10 = f10(x, y) = (10x + 9)(10y + 11)$$

这里 $x, y \in \{0\} \cup N$ 。

算法实现:

(1) 依据上述奇合数算式求出 N 范围内的所有奇

合数。

(2) 列出小于 N 的个位数为 1, 3, 7, 9 的奇数列 {3, 7, 9, 11, 13, 17, 19, 21, 23, 27, 29, ...}。

(3) 从上述数列中划掉(1)中求出的所有奇合数, 再添上 2, 5, 即得到小于 N 的全部素数。

2 算法性能比较

针对上述四种算法, 编写了 Matlab 程序对筛法实现, 并对这些程序进行了计算 1 千, 1 万, 100 万, 1000 万和 1 亿范围内的素数表的性能测试。另外添加了 Matlab 自带的系统函数 `isprime(n)` 作为比较算法。表 2 中 `prime1` ~ 4 为上述的四种算法, `prime5` 为系统函数, 低于计时函数精度的时间均为 0.00E+00, 对于计算时间过长的算法不进行更大数据的测试。性能测试包括记录时间消耗、循环次数和乘法次数(见表 3~4, 图 1~2)。

3 算法分析

从算法测试的结果可以看出, 各算法均能够完成 1000 万以内的素数表计算, 但只有 Dirichlet 定理衍生的素数筛法可以在 1 亿的计算中得到令人满意的结果, 计算 1 亿内的素数表仅用时 7.6 秒。

Dirichlet 定理衍生的素数筛法、辛

答拉姆筛法和基于奇合数的分解式的素数筛法在 N 增大时循环次数和运行时间接近线性增长, 而埃拉托斯特尼筛法, 计算时间和循环次数的增长速率大于 N 的增长速率。

(1) Dirichlet 定理衍生的素数筛法和基于奇合数的分解式的素数筛法两个算法都为两步筛选, 初筛中依照大的特征筛选出只含较少非素数的集合, 精筛中进一步筛掉初筛集合中的非素数。两个算法的理论基础不同, 前者初筛使用八系数表, 精筛时用差集取值组合算式获得非素数, 由于差集式公差为 30, 使得筛选能够快速增长到所需的大数, 循环的次数较少。而后者从奇合数入手, 初筛中留下了个位数为 1, 3, 7, 9 的数, 在精筛中用推导出的 10 个奇合数公式对初筛数集

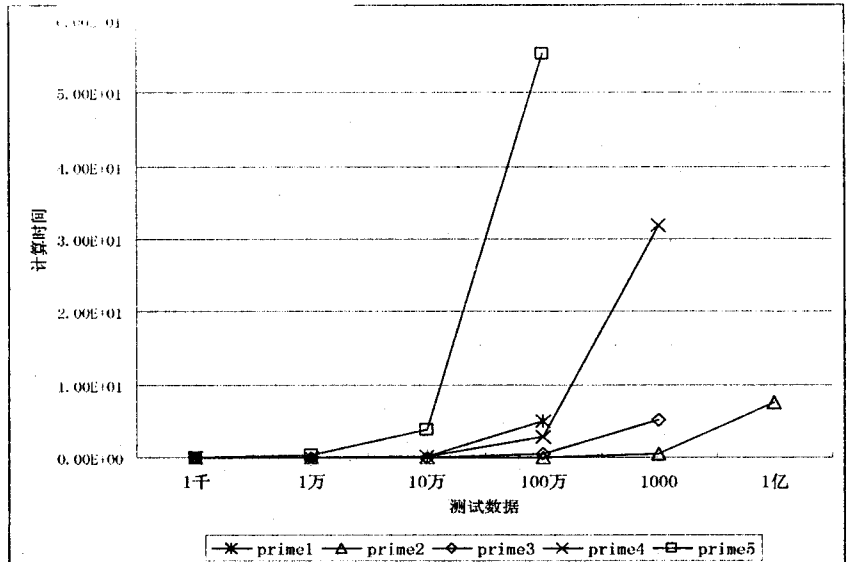


图1 算法时间消耗比较

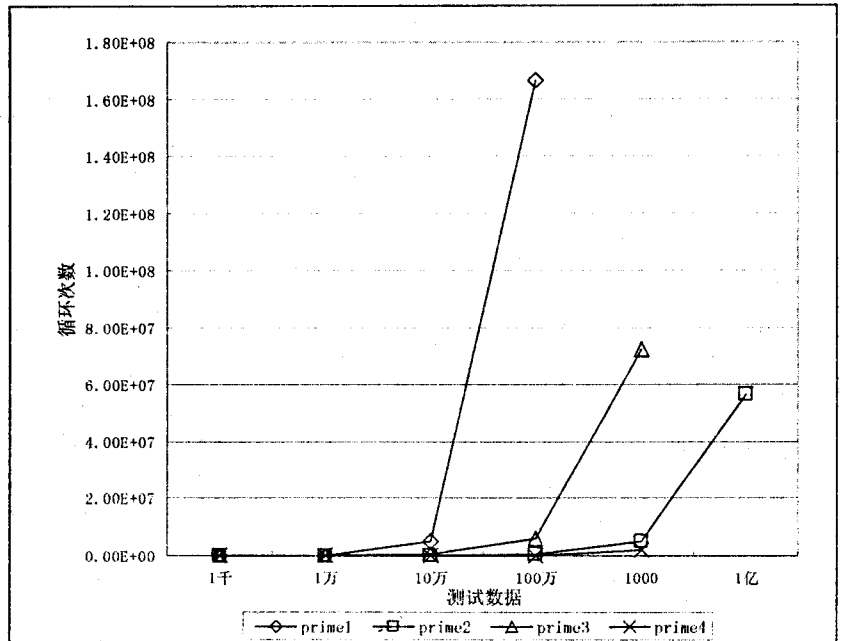


图2 各算法循环次数比较

应用, 不需要像前者中分八种情况分别筛选, 循环次数比前者更少, 但测试结果证明 Dirichlet 定理衍生的素数筛法的性能高于基于奇合数的分解式的素数筛法, 分析算法可以看出后者进行的乘法次数高于前者, 以表 3 中一千万的测试数据为例, 基于奇合数的分解式的素数筛法的乘法次数是 Dirichlet 定理衍生的素数筛法的 3.6 倍, 多进行了 38946210 次乘法运算, 计算时间前者为后者的 60 倍。可见基于奇合数的分解式的素数筛法的乘法运算是降低其性能的主要因素, Dirichlet 定理衍生的素数筛法虽然有略多的循环次数, 但由于乘法运算相对较少, 性能突出。

(2) 埃拉托斯特尼筛法是容易实现并具有较好性能的一种筛法, 但因为筛选条件较为宽松, 筛选效率较

表 2 各算法的时间消耗(单位:秒)

数据 算法名	1000	10000	100000	1000000	10000000	100000000
prime1	0.00E+00	0.00E+00	2.04E-01	5.05E+00	1.34E+02	>10 分钟
prime2	0.00E+00	0.00E+00	0.00E+00	4.65E-02	5.54E-01	7.60E+00
prime3	0.00E+00	0.00E+00	3.15E-02	4.32E-01	5.16E+00	>10 分钟
prime4	0.00E+00	1.60E-02	2.50E-01	2.84E+00	3.18E+01	>10 分钟
prime5	3.15E-02	2.97E-01	3.84E+00	5.55E+01	1.01E+03	不算

表 3 各算法循环次数

测试数据 算法名	1000	10000	100000	1000000	10000000	100000000
prime1	4.78E+03	1.62E+05	5.22E+06	1.66E+08	5.27E+09	不算
prime2	2.05E+02	2.49E+03	3.25E+04	4.05E+05	4.87E+06	5.69E+07
prime3	2.75E+03	3.85E+04	4.97E+05	6.10E+06	7.24E+07	不算
prime4	1.20E+02	1.40E+03	1.62E+04	1.85E+05	2.09E+06	不算

表 4 Dirichlet 定理衍生的素数筛法和基于奇合数的分解式的素数筛法的乘法次数

测试数据 算法名	1000	10000	100000	1000000	10000000	100000000
prime2	6.15E+02	7.46E+03	9.74E+04	1.22E+06	1.46E+07	1.71E+08
prime4	2.66E+03	3.29E+04	3.97E+05	4.66E+06	5.36E+07	不算

表 5 几种素数筛选算法评价

算法名	优点	缺点	适用范围
埃拉托斯特尼筛法	古老筛法,容易实现,100 万内性能良好	100 万以上性能较差,优势不明显	100 万内的素数筛选
Dirichlet 定理衍生的素数筛法	性能优越,可以快速计算 1 亿内的素数	判定子集较多,程序较复杂,不容易实现	1 亿的素数筛选(目前只做到 1 亿的测试)
辛答拉姆筛法	容易实现,性能优良,快速计算 1000 万内的素数	不能高效的筛选 1 亿或更大范围内的素数	1000 万内的素数筛选
基于奇合数的分解式的素数筛法	性能优良,可以计算 1000 万内的素数	实现复杂,性能低于辛答拉姆筛法和 Dirichlet 定理衍生的素数筛法	1000 万内的素数筛选
系统函数 isprime(n)素数筛法	系统函数,直接使用,可以作为判定函数和小范围内的素数筛选	不能进行大数范围内的素数筛选	10 万内的素数筛选和素数的判定

低,不适合进行大数的素数筛选。辛答拉姆筛法另辟蹊径,同样提供了一种简易高效的素数筛选。尤其在 Matlab 环境中,将辛达拉姆数表、单位行向量和系统函数结合使用,可以降低算法实现的复杂性并提高算法性能^[6]。

4 结束语

从原理、算法实现、性能测试到算法分析,对四个素数筛法进行了实现、比较和评价,对在不同环境要求下选取合适的素数筛选算法有着一定的指导意义。利用计算机的高速运算能力和 Matlab 对数学运算的支持性可以极大地提升数学应用中的计算规模。通过把数学中的素数筛法转化为计算机算法,可以快速计算出大数范围内的素数表。文中提供了算法的简单实现和初级优化策略供读者参考,希望通过素数理论上的

创新和对现有算法的进一步优化,创造出更优性能的大数范围内的素数筛选算法。

参考文献:

[1] 杜瑞庆,夏万林. 埃拉托斯特尼筛法及改进(C++语言)[J]. 中国科技信息, 2006(18):152-156.

[2] 尚德庆,王际昭. Dirichlet 定理衍生的素数筛法[J]. 平原大学学报,1997(2):47-49.

[3] 胡代. 寻找素数的一种筛法[J]. 深圳教育学院学报, 1999,4(1):83-85.

[4] 李维超. 辛达拉姆筛法的推广[J]. 数学通报,2001(3):38-39.

[5] 侯绍胜,王顺庆. 奇合数的分解公式,素数分布及筛法[J]. 西北民族学院学报:自然科学版,2002,23(2):1-6.

[6] Dunten B. A space-efficient fast prime number sieve[J]. Information Processing Letters, 1996,59(2):79-84.