arXiv:2303.05799v1 [cs.DS] 10 Mar 2023

最优哈希精确字符串匹配算法

Thierry Lecroq*

诺曼底鲁昂大学、诺曼底鲁昂国立应用科学学院、诺曼底勒阿弗尔大学、诺曼底大学、LITIS UR 4108、CNRS NormaSTIC FR 3638、IRIB，F-76000 鲁昂，法国

摘要

字符串匹配是指在文本中查找模式串所有出现位置的问题。我们提出了基于对 q-元组进行哈希的快速字符串匹配算法系列的改进版本。改进之处在于考虑最小值 q，使模式串的每个 q-元组都具有唯一的哈希值。这些新算法在大字母表上处理短模式时，比 HASH 系列的算法更快。

1 引言

字符串匹配问题是指在长度为 $y = y[0\,..\,n-1]$ 的文本中，查找长度为 m 的模式串 $x = x[0\,..\,m-1]$ 的一个或多个（通常是全部）出现位置。该问题在许多应用中都会遇到，例如信息检索、书目检索和分子生物学。人们对此进行了广泛研究，并设计了许多解决该问题的技术和算法（参见 [4, 8]）。本文关注这样一种问题：模式串预先给定，之后可以在不同文本中搜索。因此，可以对模式串进行预处理。

基本上，字符串匹配算法使用一个窗口扫描文本。窗口的大小等于模式串的长度。算法首先将窗口的左端与文本的左端对齐，然后检查窗口中是否出现模式串（这项具体操作称为一次尝试），并将窗口向右移动。算法重复这一过程，直到窗口的右端越过文本的右端

*法国鲁昂诺曼底大学科学与技术学院 CURIB，邮编 76821，蒙圣艾尼昂 Cedex，法国，Thierry.Lecroq@univ-rouen.fr，monge.univ-mlv.fr/~lecroq/lec_en.html

。各种不同的解决方案在比较窗口内容与模式串的方式、计算移动距离的方式，以及从一次尝试到下一次尝试所保存的信息方面各不相同。

不存在普适的算法，因为特定算法的效率主要取决于字母表的大小和模式串 [7] 的长度。根据 [8], ，这些算法可分为基于字符比较的算法、基于自动机的算法和基于位并行的算法。

在基于字符比较的算法中，HASH 系列字符串匹配算法 [12] 基本上通过对 q 元组 u 的每个 x 进行哈希，得到 $h(u)$，并计算 $h(u)$ 的移位函数，其值等于 $\min\{m-i \mid h(x[i\,..\,i+q-1] = h(u)\}$。这些算法的另一个特点是，它们在计算 $h(u)$ 时避免使用循环，因此 q 不是算法的参数；对于 q 的每个取值，都应有一个对应的算法。

自 2007年 发表以来，HASH 系列字符串匹配算法引起了广泛关注，至今仍被认为是最快的算法之一 [13,10, 3, 14, 2, 15] ，在某些特殊情况下甚至是最快的算法 [1, 16, 11]。然而，它有一个主要缺点：必须预先确定 q 元组的长度 q ，这意味着该算法无法处理长度小于 q 的模式串。本文提出了一些算法来解决这一问题：选择 q 元组的长度，使其为满足以下条件的最小 q ：模式串中不存在两个长度为 q 且哈希值相同的子串。这种方法有三项优势：第一，算法可以处理任意长度的模式串；第二，当 $q = 1$ 或 $q = 2$ 时，可以轻松使用完美哈希，并在检查候选项时避免一些字符比较；第三，在某些情况下，这些算法比原始 HASH 算法更快。

本文的组织结构如下：第 2 节介绍新算法，第 3 节展示实验结果，第 4 节给出结论。

2 新算法

HASH 系列字符串匹配算法考虑长度为 q 的子串。模式串 x 中长度为 q 的子串 u 使用函数 h 进行哈希，映射为介于 0 和 S 之间的整数值。随后，为每个哈希值定义一个移位值，如下所示。对于 $0 \le c \le S$:

$$shift[c] = \begin{cases} m-1-i & \text{with } i = \max\{0 \le j \le m-q+1 \mid h(x[j\,..\,j+q-1]) = c\} \\ m-q & \text{when such an } i \text{ does not exist} \end{cases}$$

其中， S 是哈希表的大小。

算法的搜索阶段包括读取长度为 u q的子串。如果$shift[h(u)] > 0$ ，则应用长度为$shift[h(u)]$的位移。否则，当

$shift[h(u)] = 0$ 时，将模式串 x 与文本进行朴素检查。在这种情况下，应用长度为sh 的位移，其中$sh = m - 1 - i$，且 $i = \max\{0 \le j \le m - q \mid h(x[j..j+q-1]) = h(x[m-q+1..m-1]\}$。

新算法的思路是：当模式串中不存在两个哈希值相同的长度为 q 的子串时，考虑长度为 q 的子串。利用模式串的后缀数组和最长公共前缀数组 [5, 6]，可以轻松计算出这一点。这给出了 q 的下界，随后必须对 x 的所有子串进行哈希，以检查哈希值是否各不相同。如果不是，则将 q 加一并重新检查这些值，直到 x 的所有 q- 元串的哈希值都各不相同。

当 $q = 1$ 时，可以轻松使用完美哈希，并采用大小为 256 的散列表；当 $q = 2$ 时，哈希表的大小会增至 65536。如此大小的表可以高效地分配在执行栈上。对于更大的 q，简单的完美哈希需要更大的哈希表，无法分配在执行栈上，因此需要在堆上显式分配和释放，这会显著降低算法的速度。

完美哈希的优势在于，如果 $h(y[j..j+q-1]) = 0$（对于某个$0 \le j \le n - q$），那么这意味着 $y[j..j+q-1] = x[m-q..m-1]$，因此检查阶段只需测试 $y[j+q-m..j-1] = x[0..m-q-1]$；而如果哈希并不完美，则测试变为 $y[j+q-m..j+q-1] = x$。

3 实验结果

为了评估新字符串匹配算法的效率，我们使用了 SMART 系统 [9]。我们在该系统的不同数据集上，使用不同算法进行了若干实验。

3.1 算法

我们测试了 6 种算法：

- HASH3 [12];
- HASH5 [12];
- HASH8 [12];
- OHASH1：对 $q = 1$ 进行完美哈希，对 $2 \le q \le 10$ 进行可能发生碰撞的哈希，并在 $q > 10$ 时使用 HASH8;

表 1：字母表大小为 8 时，短模式与随机文本的结果

m	2	4	6	8	10	12	14	16	18	20	22
HASH3	-	3.19	1.70	1.21	0.98	**0.83**	**0.72**	**0.67**	**0.61**	**0.57**	**0.54**
HASH5	-	-	3.38	1.78	**1.27**	0.98	0.82	0.72	0.64	**0.59**	**0.54**
HASH8	-	-	-	7.80	2.69	1.70	1.27	1.04	0.88	0.79	0.71
OHASH1	**3.54**	2.87	2.06	1.58	1.33	1.13	1.04	0.93	0.87	0.81	0.77
OHASH2	**3.48**	**2.53**	1.60	1.20	0.98	0.86	0.78	0.73	0.68	0.64	0.61
OHASH3	**3.48**	**2.53**	**1.59**	**1.18**	**0.97**	0.85	0.78	0.71	0.66	0.62	0.59

- OHASH2：对 $q = 1$ 和 $q = 2$ 进行完美哈希，对 $3 \leq q \leq 10$ 进行可能发生碰撞的哈希，并在 $q > 10$ 时使用 HASH8；
- OHASH3： $q = 1$ 和 $q = 2$的完美哈希， $3 \leq q \leq 10$ 的 HASH3，以及在 $q > 10$ 时的 HASH8。

这些算法均以统一的方式用 C语言 编写，以确保比较结果具有意义。由于需要为模式串构建后缀数组，而模式串相较于文本较短，因此其构建过程采用了朴素实现。我们使用的机器搭载一颗 2.4 GHz 的 Intel Xeon 处理器，运行 Ubuntu 14.04.5 LTS。OHASH算法的代码可在 github.com/lecroq/ohash 获取。

3.2 数据

我们给出了上述算法在不同类型文本上的运行时间实验结果：字母表大小为 8、16、32、64、128 和 250 的随机文本，自然语言文本（英语和意大利语）以及蛋白质序列。我们考察了短模式（长度为 2 到 22，包括端点）。

3.3 结果

短模式（长度小于或等于 22）的结果列于表 1 至表 9（最快的结果以粗体显示）以及图 1 至图 9 中。

实验结果表明，OHASH3 算法速度最快：

- 在字母表大小为 8 的随机文本上，对于 $2 \leq m \leq 10$；
- 在大小为 16 的字母表上生成的随机文本，针对 $2 \leq m \leq 14$；

表2：在大小为16的字母表上，短模式与随机文本的结果

m	2	4	6	8	10	12	14	16	18	20	22
HASH3	-	3.07	1.64	1.16	0.91	0.77	0.68	**0.60**	**0.55**	**0.51**	**0.48**
HASH5	-	-	3.44	1.78	1.24	0.98	0.81	0.70	0.62	0.57	0.52
HASH8	-	-	-	7.87	2.72	1.72	1.27	1.03	0.88	0.78	0.70
OHASH1	2.88	2.03	1.63	1.33	1.14	1.02	0.91	0.85	0.79	0.75	0.71
OHASH2	**2.85**	**1.93**	**1.38**	**1.07**	**0.88**	0.77	0.68	0.62	0.59	0.55	0.53
OHASH3	**2.85**	1.95	**1.38**	**1.06**	**0.88**	**0.75**	**0.67**	0.62	0.58	0.55	0.53

表3：大小为32的字母表上短模式与随机文本的结果

m	2	4	6	8	10	12	14	16	18	20	22
HASH3	-	3.02	1.60	1.13	0.89	**0.74**	**0.64**	**0.58**	**0.53**	**0.49**	**0.46**
HASH5	-	-	3.37	1.79	1.24	0.97	0.80	0.70	0.63	0.57	0.53
HASH8	-	-	-	7.81	2.72	1.70	1.26	1.02	0.90	0.78	0.70
OHASH1	2.47	1.59	1.22	1.06	0.95	0.88	0.81	0.77	0.74	0.69	0.67
OHASH2	2.48	**1.55**	**1.16**	**0.97**	0.84	0.74	0.67	0.62	0.58	0.54	0.52
OHASH3	**2.46**	**1.55**	**1.16**	**0.97**	**0.83**	**0.74**	0.67	0.62	0.58	0.54	0.52

Tabl表 4：字母表大小为 64 时短模式与随机文本的结果

m	2	4	6	8	10	12	14	16	18	20	22
HASH3	-	3.00	1.58	1.11	0.86	**0.73**	**0.64**	**0.57**	**0.52**	**0.48**	**0.45**
HASH5	-	-	3.35	1.79	1.23	0.96	0.81	0.72	0.62	0.57	0.53
HASH8	-	-	-	7.78	2.69	1.70	1.27	1.04	0.88	0.79	0.70
OHASH1	2.33	**1.34**	**1.03**	**0.88**	**0.78**	0.76	0.71	0.69	0.65	0.64	0.63
OHASH2	**2.32**	**1.38**	1.07	0.92	0.81	0.77	0.70	0.66	0.62	0.59	0.56
OHASH3	**2.32**	**1.38**	1.07	0.92	0.81	0.77	0.70	0.66	0.62	0.59	0.56

表 5：字母表大小为 128 时，短模式与随机文本的结果

m	2	4	6	8	10	12	14	16	18	20	22
HASH3	-	2.98	1.57	1.10	0.88	0.73	0.64	**0.58**	**0.52**	**0.48**	**0.44**
HASH5	-	-	3.35	1.76	1.25	0.97	0.81	0.71	0.63	0.57	0.52
HASH8	-	-	-	7.82	2.71	1.69	1.27	1.03	0.87	0.77	0.69
OHASH1	**2.21**	**1.26**	**0.92**	**0.79**	**0.68**	**0.64**	**0.61**	0.60	0.57	0.56	0.56
OHASH2	2.22	1.30	0.96	0.86	0.75	0.72	0.68	0.66	0.63	0.62	0.60
OHASH3	2.22	1.30	0.96	0.86	0.75	0.72	0.68	0.66	0.63	0.62	0.60

表6：字母表大小为250时短模式和随机文本的结果

m	2	4	6	8	10	12	14	16	18	20	22
HASH3	-	3.01	1.58	1.10	0.86	0.72	0.63	0.57	0.52	**0.48**	**0.45**
HASH5	-	-	3.35	1.77	1.22	0.96	0.80	0.71	0.63	0.57	0.52
HASH8	-	-	-	7.86	2.68	1.68	1.25	1.04	0.88	0.79	0.71
OHASH1	**2.17**	**1.22**	**0.86**	**0.71**	**0.62**	**0.56**	**0.53**	**0.52**	**0.50**	0.51	0.49
OHASH2	2.17	1.29	0.92	0.82	0.72	0.70	0.67	0.70	0.68	0.72	0.70
OHASH3	2.18	1.29	0.92	0.83	0.72	0.70	0.68	0.69	0.68	0.71	0.70

表7：短模式和英文文本的结果

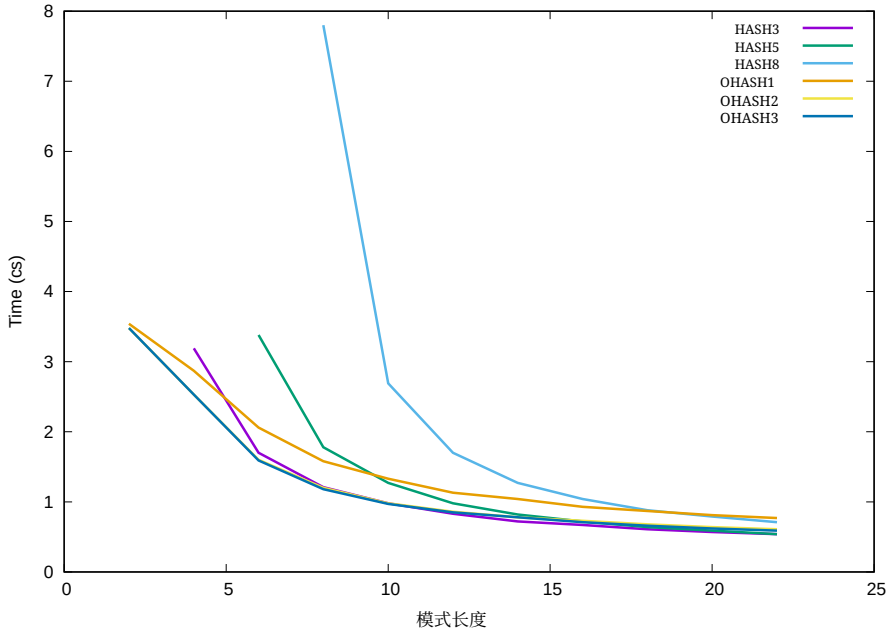
m	2	4	6	8	10	12	14	16	18	20	22
HASH3	-	3.06	1.63	1.14	**0.90**	**0.76**	**0.66**	**0.59**	**0.54**	**0.50**	**0.47**
HASH5	-	-	3.40	1.79	1.24	0.98	0.81	0.71	0.63	0.58	0.53
HASH8	-	-	-	7.81	2.70	1.70	1.26	1.03	0.88	0.79	0.71
OHASH1	2.85	1.80	1.45	1.21	1.01	0.89	0.80	0.75	0.70	0.68	0.65
OHASH2	**2.85**	**1.79**	1.39	1.12	0.92	0.80	0.71	0.65	0.61	0.58	0.55
OHASH3	2.88	1.80	**1.38**	**1.11**	0.92	0.79	0.70	0.64	0.60	0.56	0.53

表8：短模式和意大利文文本的结果

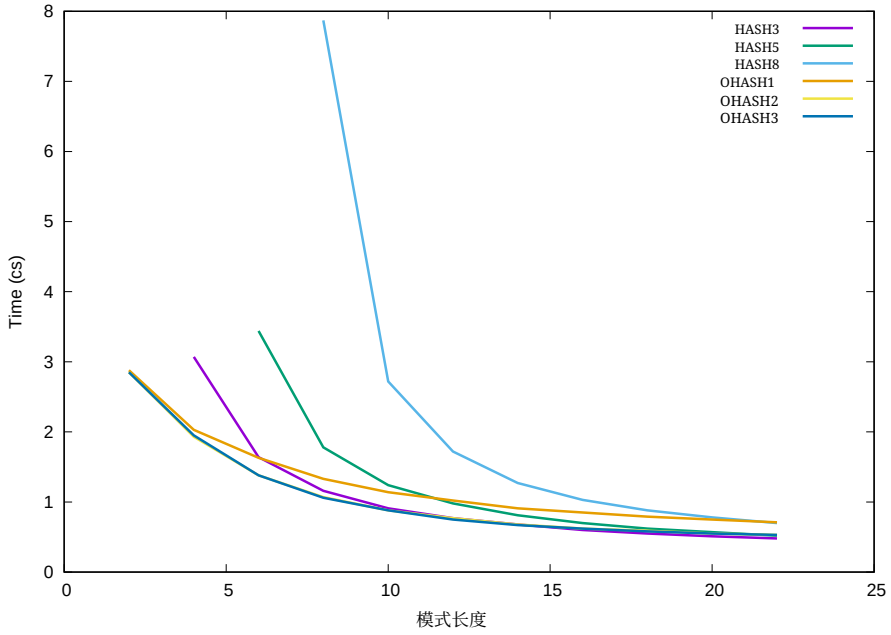
m	2	4	6	8	10	12	14	16	18	20	22
HASH3	-	3.03	1.66	1.15	**0.91**	**0.76**	**0.66**	**0.59**	**0.54**	**0.50**	**0.47**
HASH5	-	-	3.44	1.82	1.25	0.98	0.82	0.71	0.64	0.58	0.54
HASH8	-	-	-	7.92	2.74	1.71	1.28	1.04	0.89	0.78	0.72
OHASH1	**3.01**	**2.18**	1.77	1.36	1.11	0.97	0.87	0.81	0.76	0.71	0.69
OHASH2	3.04	2.17	1.72	1.28	1.02	0.88	0.78	0.70	0.65	0.61	0.58
OHASH3	**3.04**	**2.07**	**1.46**	**1.13**	0.93	0.80	0.70	0.64	0.59	0.55	0.51

表9：短模式和蛋白质序列的结果

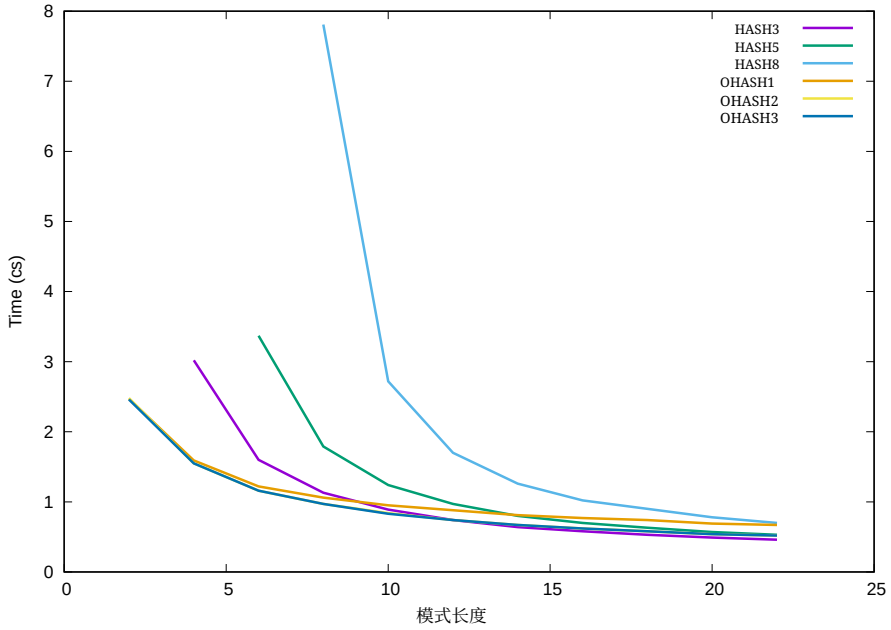
m	2	4	6	8	10	12	14	16	18	20	22
HASH3	-	3.03	1.60	1.13	0.89	**0.75**	**0.66**	**0.59**	**0.54**	**0.51**	**0.46**
HASH5	-	-	3.36	1.76	1.23	0.96	0.81	0.70	0.63	0.57	0.52
HASH8	-	-	-	7.81	2.70	1.70	1.27	1.03	0.89	0.80	0.71
OHASH1	2.84	2.10	1.60	1.29	1.12	0.96	0.88	0.81	0.76	0.73	0.69
OHASH2	**2.82**	**2.00**	1.43	1.10	0.91	0.77	0.71	0.63	0.60	0.56	0.53
OHASH3	**2.82**	**2.00**	**1.39**	**1.06**	**0.87**	**0.75**	0.70	0.62	0.59	0.54	0.51



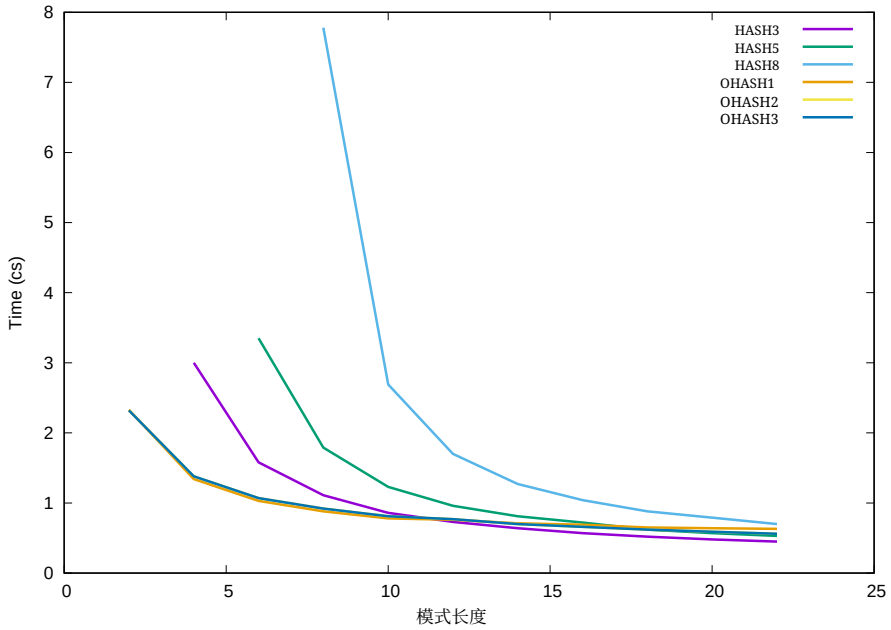
Fig图1：字母表大小为8时，短模式与随机文本的结果



Figu图 2：字母表大小为 1 时短模式和随机文本的结果



Figu图 3：字母表大小为 3 时，短模式与随机文本的结果 2



Figu图 4：字母表大小为 6 时，短模式和随机文本的结果 4

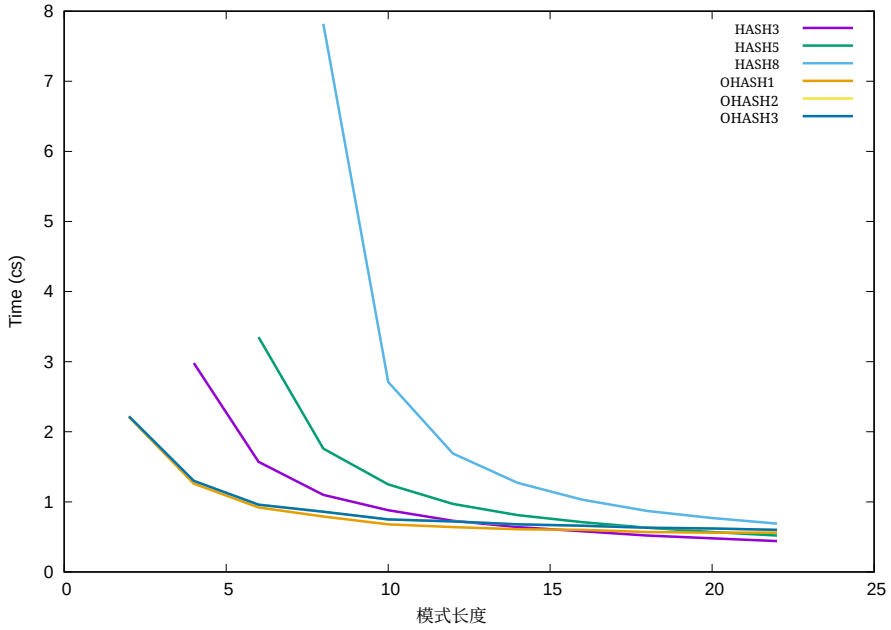


图 5：字母表大小为 12 时，短模式和随机文本的结果 8

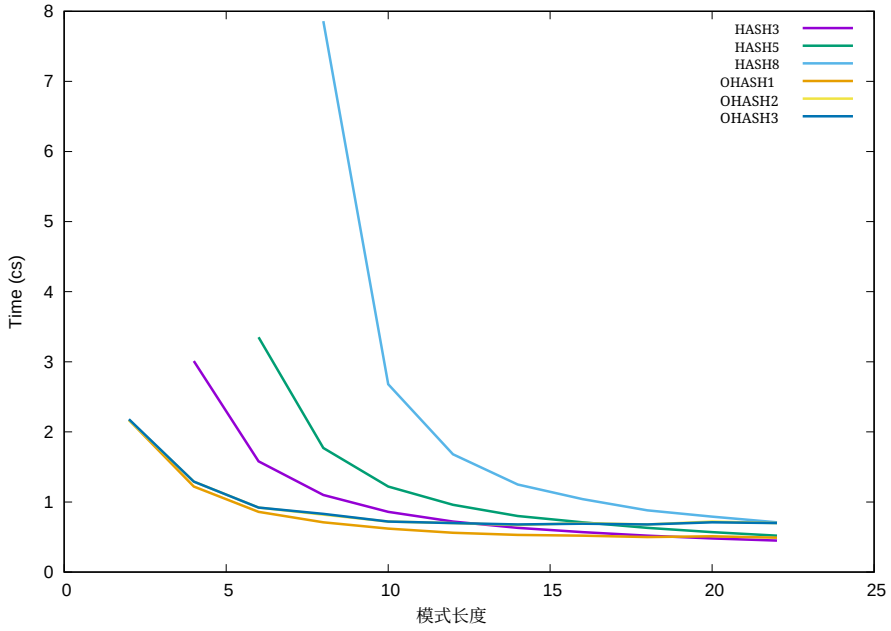


图 6：字母表大小为 25 时短模式与随机文本的结果 0

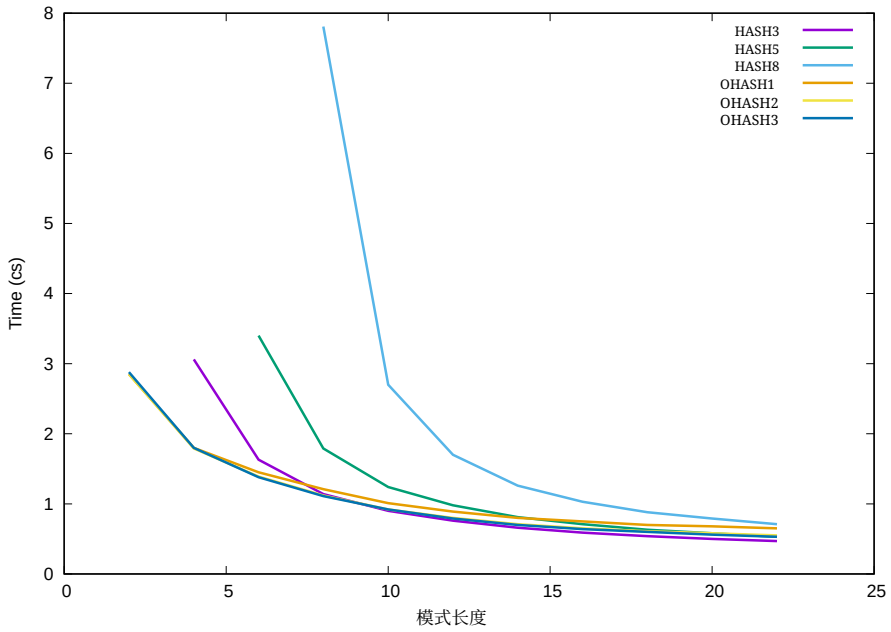


图7：英文文本中短模式的结果

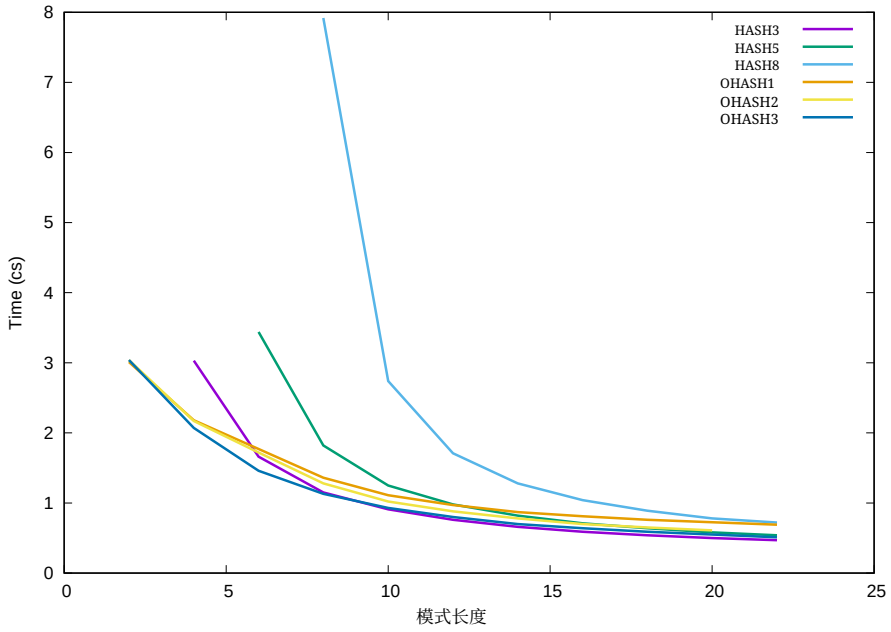


图 8：意大利文文本中短模式的结果

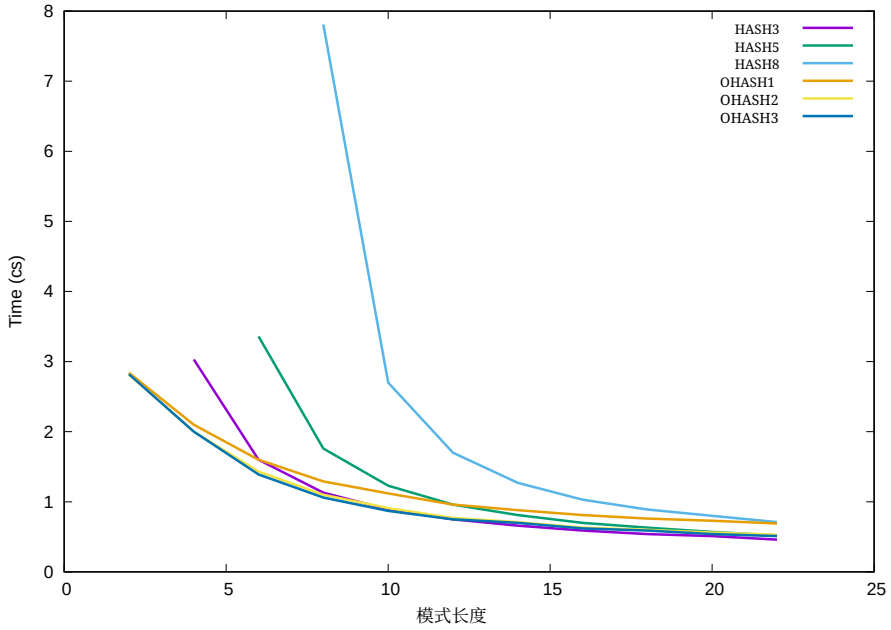


图 9：蛋白质序列上短模式的结果

- 在字母表大小为 32 的随机文本上，针对 $2 \leq m \leq 12$;
- 在英文文本上，针对 $2 \leq m \leq 8$;
- 在意大利文文本上，针对 $2 \leq m \leq 8$;
- 在蛋白质序列上，针对 $2 \leq m \leq 12$;

而算法 OHASH1 是最快的:

- 对于字母表大小为 64 的随机文本， $2 \leq m \leq 10$;
- 对于字母表大小为 128 的随机文本， $2 \leq m \leq 14$;
- 对于字母表大小为 250 的随机文本， $2 \leq m \leq 18$。

4 结论

本文介绍了 HASH 系列精确字符串匹配算法 [12] 的简单但速度极快的改进算法。对于长度为 2 到 18、字母表大小至少为 8 的短模式，这些新算法速度很快。这些数值大致对应于

在自然语言文本中搜索单词的情况。不过，与 HASH 系列算法相比，性能提升相对较小；然而，在绿色信息技术时代，对于查找模式这类重复性极高的任务，即使只有小幅提升，累积起来也能带来显著的整体收益。

本文是一项初步研究。至少还需要开展更多实验，以了解 q 的期望值如何随模式长度和字母表大小变化，从而调整在 OHASH 和 HASH 子程序之间进行选择的策略。对于 q 大于或等于 3 的情况，高效使用完美哈希也可能改进这些算法。

参考文献

[1] A. M. Al-Ssulami。带枢轴的混合字符串匹配算法。《信息科学杂志》，41(1):82–88，2015年。[2] A. M. Al-Ssulami、A. M. Azmi、H. Mathkour 和 H. Aboalsamh。LsHASHq：一种利用更长 q-gram 移位的字符串匹配算法。《信息处理与管理》，59(5):103057，2022年。[3] A. M. Al-Ssulami 和 H. Mathkour。基于哈希和位并行的快速字符串匹配。《信息处理快报》，123:51–55，2017年。[4] C. Charras 和 T. Lecroq。精确字符串匹配算法手册。King's College Publications，2004年。[5] M. Crochemore、C. Hancart 和 T. Lecroq。字符串算法。剑桥大学出版社，2007年。[6] M. Crochemore、T. Lecroq 和 W. Rytter。文本算法中的*125*个问题。剑桥大学出版社，2021年。[7] S. Faro 和 T. Lecroq。精确字符串匹配问题：全面的实验评估。报告 arXiv:1012.2547，2010年。[8] S. Faro 和 T. Lecroq。精确在线字符串匹配问题：最新成果综述。《*ACM*计算调查》，45(2):13，2013年。[9] S. Faro、T. Lecroq、S. Borzi、S. D. Mauro 和 A. Maggio。字符串匹配算法研究工具。载于 J. Holub 和 J. Zdàrek 编，*2016* 年布拉格字符串学会议论文集（*PSC 2016*），第99–113页，捷克共和国布拉格，2016年。

[10] A. A. Karcioglu 和 H. Bulut。利用 DNA 序列的完美哈希改进 hash-q 精确字符串匹配算法。《计算机生物学与医学》，131:104292，2021年。[11] S. Kobayashi、D. Hendrian、R. Yoshinaka 和 A. Shinohara。基于 q-gram 出现位置距离的快速线性时间字符串匹配算法。$CoRR$，abs/2002.08004，2020年。[12] T. Lecroq。快速精确字符串匹配算法。《信息处理快报》，102(6):229–235，2007年。[13] T. H. Nguyen。一种精确模式匹配的新方法。《计算机科学与控制论学报》，35(3):197–216，2019年8月。[14] H. Peltola 和 J. Tarhio。带前瞻的字符串匹配。《离散应用数学》，163:352–360，2014年。字符串学算法。[15] M. Tahir、M. Sardaraz 和 A. A. Ikram。EPMA：用于 DNA 序列的高效模式匹配算法。《专家系统与应用》，80:162–170，2017年。[16] I. O. Zavadskyi。一类具有多个相邻搜索窗口的精确模式匹配算法。载于 J. Holub 和 J. Žďárek 编，*2017* 年布拉格字符串学会议论文集，第152–166页，布拉格捷克技术大学，捷克共和国，2017年。