mathematics

Article

An Improved Order-Preserving Pattern Matching Algorithm Using Fingerprints

Youngjoon Kim †, Youngho Kim † and Jeong Seop Sim *

Department of Computer Engineering, Inha University, Incheon 22212, Korea; yjkim12130@gmail.com (Y.K.); yhkim85@inha.ac.kr (Y.K.)
* Correspondence: jssim@inha.ac.kr
† These authors contributed equally to this work.

Abstract: Two strings of the same length are order isomorphic if their relative orders are the same. The order-preserving pattern matching problem is to find all substrings of text T that are order isomorphic to pattern P when T ($|T| = n$) and P ($|P| = m$) are given. An $O(mn + nq \log q + q!)$-time algorithm using the $O(m + q!)$ space for the order-preserving pattern matching problem has been proposed utilizing fingerprints of q-grams based on the factorial number system and the bad character heuristic. In this paper, we propose an $O(mn + 2^q)$-time algorithm using the $O(m + 2^q)$ space for the order-preserving pattern matching problem, but utilizing fingerprints of q-grams converted to binary numbers. A comparative experiment using three types of time series data demonstrates that the proposed algorithm is faster than existing algorithms because it reduces the number of order isomorphism tests.

Keywords: order isomorphism; order-preserving pattern matching; bad character heuristic; fingerprints

MSC: 68U05; 65Y04

Citation: Kim, Y.; Kim, Y.; Sim, J.S. An Improved Order-Preserving Pattern Matching Algorithm Using Fingerprints. *Mathematics* **2022**, *10*, 1954. https://doi.org/10.3390/math10121954

Academic Editor: Dominik Köppl

Received: 16 March 2022
Accepted: 2 June 2022
Published: 7 June 2022

1. Introduction

Two strings of the same length from an integer alphabet Σ are order isomorphic if their relative orders are the same. For example, strings $x = (10, 5, 7)$ and $y = (53, 23, 47)$ are order isomorphic because their relative orders are the same as $(3, 1, 2)$. The order-preserving pattern matching (OPPM) problem is to find all substrings of text T that are order isomorphic to pattern P when T ($|T| = n$) and P ($|P| = m$) over Σ are given. Order-preserving pattern matching can be used to analyze time series data such as stock indices, climate data, melodies, and so on [1].

Various algorithms for solving the OPPM problem have been proposed. An algorithm proposed in [1,2] solves the OPPM problem in $O(n + sort(m))$ time using the failure function of the Knuth–Morris–Pratt (KMP) algorithm [3]. An algorithm proposed in [4] solves the problem in $O(mn + nq \log q + q!)$ time using fingerprints for q-grams that consist of q consecutive characters based on the factorial number system [5,6]. An algorithm presented in [7] is executed in sublinear time on average using binary encoding. An algorithm proposed in [8] uses a skip-search approach [9] and the Intel streaming SIMD extensions (SSE) instruction sets [10]. An algorithm using packed string matching [11,12], the SSE, and advanced vector extensions (AVX) instruction sets [13,14] was proposed in [15]. OPPM in a tree and a directed acyclic graph instead of a simple string were investigated in [16]. In [17], the OPPM problem was solved using a filtering method with minimum (or maximum) values. By generating order-preserving suffix trees in $O(n\sqrt{\log n})$ time, an algorithm presented in [18] searches P in $O(m + occ)$ time, where occ is the number of substrings of T that are order isomorphic to P.

Our study makes the following contributions:

- We improve the time and space complexity required to compute the fingerprint. In [4], the fingerprint of a q-gram based on the factorial number system was computed in $O(q \log q)$ time using the $O(q!)$ space. The OPPM algorithm proposed in this paper converts the q-gram to a binary number and computes the corresponding fingerprint in $O(q)$ time using the $O(2^q)$ space.
- We propose a fast algorithm by reducing the number of order isomorphism tests. Algorithms using fingerprints quickly find candidate locations where a pattern may occur, and they test whether order isomorphism actually occurs at those locations. The algorithm proposed in this paper improves the actual execution time by reducing the number of order isomorphism tests using fingerprints for two q-grams.
- We compare the actual execution times of algorithms through various implementations. The execution times are measured by varying the sizes of q-grams for three types of real time series data. The results of implementations are analyzed under various experimental conditions.

The rest of this paper is organized as follows. In Section 2, we define the terms, and we review previous work. In Section 3, we discuss our new order-preserving pattern matching algorithm. In Section 4, we present experimental comparisons of the execution times between the algorithms presented in [4,7] versus the algorithm proposed in this study. Finally, we conclude the paper in Section 5.

2. Preliminaries

A set of strings of length m over integer alphabet Σ is denoted as Σ^m. The length of string x is denoted as $|x|$, the ith character of x as $x[i]$ $(0 \le i < |x|)$, and the substrings of x from i to j $x[i]x[i+1]\dots x[j]$ as $x[i\dots j]$ $(0 \le i \le j < |x|)$. If $i = 0$, $x[i\dots j]$ is called a prefix of x; if $j = |x| - 1$, it is called a suffix of x.

If $x[i] \le x[j] \Leftrightarrow y[i] \le y[j]$ $(0 \le i, j < |x|)$ for two strings x and y of the same length, then x and y are order isomorphic and denoted as $x \approx y$ [2]. The prefix representation of string x uses prefix table μ_x, which is defined as follows [1]:

$$\mu_x[i] = |\{j : x[j] \le x[i] \text{ for } 0 \le j < i\}|.$$

That is, $\mu_x[i]$ is the number of characters smaller than or equal to $x[i]$ in $x[0\dots i-1]$. Prefix table μ_x can be computed in $O(|x| \log |x|)$ time using an order-statistic tree. If $x \approx y$, then $\mu_x = \mu_y$ [1]. The nearest neighbor representation of x uses location tables $LMax_x$ and $LMin_x$, which are defined as follows [1,2]:

$$LMax_x[i] = \quad j \quad \text{if } x[j] = \max\{x[k] : x[k] \le x[i] \text{ for } 0 \le k < i\}, \text{ and}$$

$$LMin_x[i] = \quad j \quad \text{if } x[j] = \min\{x[k] : x[k] \ge x[i] \text{ for } 0 \le k < i\}.$$

That is, $LMax_x[i]$ is the location of the largest character j among the characters that are smaller than or equal to $x[i]$ in $x[0\dots i-1]$, and $LMin_x[i]$ is the location of the smallest character j among the characters that are larger than or equal to $x[i]$ in $x[0\dots i-1]$. If there are two or more such j's that satisfy this condition, the largest j among them is defined as $LMax_x[i]$ (or $LMin_x[i]$); if there is no such j, they are defined as -1. $LMax_x$ and $LMin_x$ can be computed in $O(|x| \log |x|)$ time using order-statistic trees and can be used to determine whether x and y are order isomorphic or not in $O(|x|)$ time [1,2]. Table 1 shows prefix table μ_x and location tables $LMax_x$ and $LMin_x$ for string $x = (5, 11, 18, 7, 3, 9)$.

Table 1. Prefix table μ_x and location tables $LMax_x$ and $LMin_x$ for $x = (5, 11, 18, 7, 3, 9)$.

i	**0**	**1**	**2**	**3**	**4**	**5**
$x[i]$	5	11	18	7	3	9
$\mu_x[i]$	0	1	2	1	0	3
$LMax_x[i]$	−1	0	1	0	−1	3
$LMin_x[i]$	−1	−1	−1	1	0	1

The order-preserving pattern matching problem is formally defined as follows.

Problem 1. *Order-preserving pattern matching problem.*
Input: *text T $(\in \Sigma^n)$ and pattern P $(\in \Sigma^m)$.*
Output: *every position i $(m-1 \leq i < n)$ of T where $T[i-m+1\ldots i] \approx P$.*

In [4], to apply the bad character heuristic of the Horspool algorithm [19] to OPPM, the notion of a q-gram and a fingerprint based on a factorial number system were used. A q-gram consists of q $(1 \leq q < m)$ consecutive characters, and fingerprint $f(x)$ for q-gram x converts x into one integer as follows [4]:

$$f(x) = \sum_{k=0}^{q-1} \mu_x[k] \cdot k!.$$

For example, when $q = 3$, prefix table μ_x of q-gram $x = (11, 83, 32)$ is $(0, 1, 1)$, and $f(x) = (0 \times 0!) + (1 \times 1!) + (1 \times 2!) = 3$. The algorithm in [4] consists of two phases, a preprocessing phase and a search phase. In the preprocessing phase, the shift table and location tables for P are computed. First, all elements of shift table D are initialized to maximum moving distance $m - q + 1$, and then, D is computed using the following equation:

$$t = \max\{i : \mu_P[i-q+1\ldots i] = \mu_x,\ q-1 \leq i < m-1\},$$
$$D[f(x)] = \min(m-q+1,\ m-t-1).$$

In the search phase, OPPM is performed using the bad character heuristic and the tables. In the worst case, the algorithm proposed in [4] runs in $O(mn + nq\log q + q!)$ time using the $O(m + q!)$ space.

3. New Order-Preserving Pattern Matching Algorithm

Our new OPPM algorithm runs faster and uses less space than the algorithm in [4]. Our algorithm also consists of two phases like the algorithm in [4]. The main differences are as follows. First, our algorithm uses a different fingerprint. It converts q-grams into binary strings and computes the fingerprints for the converted binary strings. Second, our algorithm uses two fingerprints of q-grams to reduce the number of order isomorphism tests. In the preprocessing phase, our algorithm converts pattern P into binary string P' using the method from [7]. It also computes the shift tables for two q-grams and the location tables for P. In the search phase, it finds all substrings of T that are order isomorphic to P using the fingerprints for the q-grams of the binary strings and the bad character heuristic.

3.1. Preprocessing Phase

For string x over Σ, binary string x' $(|x'| = |x| - 1)$ is defined as follows [7]:

$$x'[i] = \begin{cases} 1 & \text{if } x[i] < x[i+1], \\ 0 & \text{otherwise.} \end{cases}$$

Fingerprint $g(w)$ of q-gram w for binary string x' is defined as follows:

$$g(w) = \sum_{k=0}^{q-1} w[k] \cdot 2^{q-k-1}.$$

For example, when $x = (21, 69, 93, 77)$, binary string x' converted from x is $(1, 1, 0)$. When $q = 3$, $w = (1, 1, 0)$ and $g(w) = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 6$.

In the preprocessing phase, we compute binary string P' and location tables $LMax_P$ and $LMin_P$ for P. P' can be computed in $O(m)$ time using the $O(m)$ space by scanning P. Location tables $LMax_P$ and $LMin_P$ for P can be computed in $O(m \log m)$ time using the $O(m)$ space, as explained above. We also compute shift tables D_1 and D_2 for P'. For binary string x', we call $x'[|x'| - q \ldots |x'| - 1]$ and $x'[|x'| - 2q \ldots |x| - q - 1]$, respectively, the primary q-gram and the secondary q-gram of x'. For example, when $q = 3$, as shown in Figure 1, the primary q-gram and the secondary q-gram of P' are $P'[5 \ldots 7]$ and $P'[2 \ldots 4]$, respectively. First, all the elements of D_1 and D_2 are initialized to $m - q$ and $m - 2q$, respectively, which are the maximum distances that the pattern can move via the two q-grams. Then, D_1 and D_2 are computed using the following equations for P':

$$a_w = \max\{i : P'[i - q + 1 \ldots i] = w,\ q - 1 \le i < m - 2\},$$

$$D_1[g(w)] = \min(m - q,\ m - a_w - 1),$$

$$b_w = \max\{i : P'[i - q + 1 \ldots i] = w,\ q - 1 \le i < m - q - 2\},$$

$$D_2[g(w)] = \min(m - 2q,\ m - b_w - 1).$$

Note that a_w and b_w are the last positions of the substrings of P' that match q-gram w in $P'[0 \ldots m - 3]$ and $P'[0 \ldots m - q - 3]$, respectively. $D_1[g(w)]$ and $D_2[g(w)]$ store the distances that the pattern can move via the primary q-gram and the secondary q-gram, respectively.

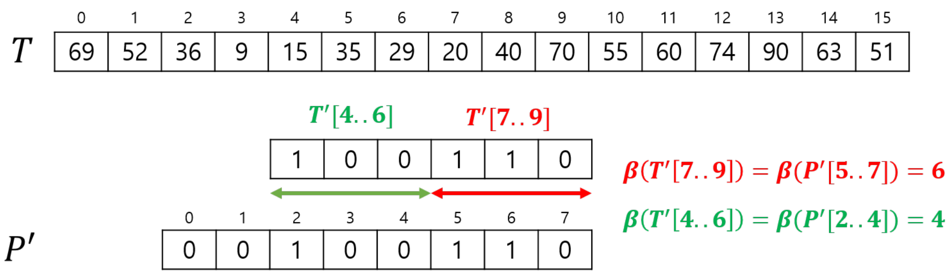

Figure 1. Order-preserving pattern matching using fingerprints of the primary q-gram and the secondary q-gram when $q = 3$.

Shift tables D_1 and D_2 can be computed in $O(2^q + m)$ time using the $O(2^q)$ space. Therefore, the preprocessing phase runs in $O(2^q + m \log m)$ time using the $O(2^q + m)$ space.

3.2. Search Phase

We denote the fingerprint of the primary q-gram of P' as p_1, and we denote the fingerprint of the secondary q-gram of P' as p_2. That is, $p_1 = g(P'[m - q - 1 \ldots m - 2])$ and $p_2 = g(P'[m - 2q - 1 \ldots m - q - 2])$. Furthermore, we denote the fingerprint of the primary q-gram of $T'[i - m + 1 \ldots i - 1]$ as t_1, and we denote the fingerprint of the secondary q-gram of $T'[i - m + 1 \ldots i - 1]$ as t_2. That is, $t_1 = g(T'[i - q \ldots i - 1])$ and $t_2 = g(T'[i - 2q \ldots i - q - 1])$. Algorithm 1 shows the pseudocode of our algorithm.

The search phase consists of $n - m + 1$ steps. In each step i $(m - 1 \le i < n)$, we check whether $T[i - m + 1 \ldots i]$ and P are order isomorphic. First, we check whether fingerprints p_1 and t_1 are the same (line 9 of Algorithm 1). If $p_1 \ne t_1$, we shift P forward by $D_1[t_1]$

increasing i by $D_1[t_1]$ (line 18 of Algorithm 1). If $p_1 = t_1$, we compare p_2 and t_2 (line 11 of Algorithm 1). If p_2 and t_2 are also the same, we test whether P and $T[i-m+1\ldots i]$ are order isomorphic using $LMax_P$ and $LMin_P$ in $O(m)$ time. If $T[i-m+1\ldots i] \approx P$, we report i as an occurrence. Meanwhile, if $T[i-m+1\ldots i] \approx P$, by the definition of the order isomorphism, $p_1 = t_1$ and $p_2 = t_2$. Therefore, if $p_1 \neq t_1$ or $p_2 \neq t_2$, $T[i-m+1\ldots i] \not\approx P$; hence, we can shift P forward by $\max(D_1[t_1],\ D_2[t_2])$, regardless of whether p_2 and t_2 are the same or not (line 15 of Algorithm 1). The search phase runs in $O(mn)$ time in the worst case because it might test order isomorphism in every step. Thus, the proposed algorithm solves the OPPM problem in $O(2^q + mn)$ time using the $O(2^q + m)$ space in total.

Algorithm 1 OPPM algorithm using fingerprints

1: **Input:** A text T of length n and a pattern P of length m.
2: **Output:** All positions of the substrings of T that are order isomorphic to P.
3: Compute P', D_1, D_2, $LMax_P$, and $LMin_P$
4: $p_1 \leftarrow g(P'[m-q-1\ldots m-2)]$
5: $p_2 \leftarrow g(P'[m-2q-1\ldots m-q-2)]$
6: $i \leftarrow m-1$
7: **while** $i < n$ **do**
8:     $t_1 \leftarrow g(T'[i-q\ldots i-1])$
9:     **if** $p_1 = t_1$ **then**
10:         $t_2 \leftarrow g(T'[i-2q\ldots i-q-1])$
11:         **if** $p_2 = t_2$ **then**
12:             **if** $T[i-m+1\ldots i] \approx P$ **then**
13:                 **print** i
14:             **end if**
15:         **end if**
16:         $i \leftarrow i + \max(D_1[t_1], D_2[t_2])$
17:     **else**
18:         $i \leftarrow i + D_1[t_1]$
19:     **end if**
20: **end while**

4. Experimental Results

The experimental environment was as follows. The operating system was Windows 10 (64-bit); the CPU was an Intel Core i7-6700 (3.4 GHz); the RAM was 32 GB; the development tool was Visual Studio 2015; the development language was C++. We used three types of time series data in the experiment: a power consumption index, particulate matter (PM2.5) levels, and the Dow Jones Index. The power consumption index consisted of measurement data on the average voltage per minute of a household in Sceaux, France, from 00:00 on 16 December 2006 to 22:00 on 2 December 2008 [20]. The PM2.5 levels were from data recorded in Beijing at one-hour intervals from 00:00 on 2 January 2010 to 22:00 on 9 October 2014 [21]. The Dow Jones Index data were the daily closing prices of the Dow Jones Industrial Average from 2 May 1885 to 12 April 2019 [22]. Lengths n of text T for the power consumption index, the PM2.5 levels, and the Dow Jones Index were generated as 10^6, 40,000, and 36,000, respectively. Pattern P was generated by extracting strings of lengths 7, 11, and 15 at random positions of T. For brevity, the power consumption index data are hereinafter referred to as VOLT, the particulate matter level data are referred to as PM2.5, and the Dow Jones Index data are indicated as DJIA. The algorithm proposed in [4] is referred to as OHq, and the algorithm based on SBNDM4 [23] and proposed in [7] is referred to as S4OPM. The algorithm proposed in this work was implemented in two versions. The first version was implemented as described in the previous section and is referred to as OHESq. The second version was implemented using only the fingerprint of the primary q-gram and is referred to as OHEq.

Table 2 compares the execution times of each algorithm, which are the sums for executing the algorithm for 1000 patterns and shows the sum of occurrences of all patterns in the text. In Table 2, the execution times of the fastest algorithms among the algorithms using q-grams for each m and q are in bold, and the execution times of the fastest algorithms regardless of q for each m are marked with an asterisk $(*)$. With VOLT, OHESq executed up to approximately 1.98-times faster than OHq $(m = 15,\ q = 6)$. With PM2.5, OHESq executed up to approximately 1.97-times faster than OHq or OHEq $(m = 15,\ q = 6)$. With DJIA, OHESq executed up to approximately 1.88-times faster than OHq or OHEq $(m = 15,\ q = 6)$. In all cases, OHESq executed at least 1.11-times faster than OHq, at least 1.19-times faster than OHEq, and at least 2.42-times faster than S4OPM.

Table 2. Comparison of the execution times and the number of occurrences for VOLT, PM2.5, and DJIA data (sums for 1000 patterns). Bold indicates the execution times of the fastest algorithms for each m and q, and the data marked with $*$ indicate the execution times of the fastest algorithms regardless of q for each m.

Data	m	Algorithm	Execution Time (Seconds)				Number of Occurrences
			$q = 3$	$q = 4$	$q = 5$	$q = 6$	
VOLT	7	OHq	2.756	2.338	3.692	6.835	992,773
		OHEq	3.052	2.886	3.778	6.822	
		OHESq	**2.098** $*$	·	·	·	
		S4OPM	5.258				
	11	OHq	2.129	1.301	1.612	2.232	2765
		OHEq	1.962	1.426	1.356	1.451	
		OHESq	**1.366**	**1.046**	**1.030** $*$	·	
		S4OPM	2.996				
	15	OHq	1.941	0.996	1.058	1.355	1001
		OHEq	2.080	1.176	1.003	0.923	
		OHESq	**1.184**	**0.791**	**0.671** $*$	**0.686**	
		S4OPM	2.513				
PM2.5	7	OHq	0.119	0.109	0.16	0.28	117,682
		OHEq	0.128	0.121	0.153	0.281	
		OHESq	**0.089** $*$	·	·	·	
		S4OPM	0.218				
	11	OHq	0.093	0.063	0.073	0.096	3613
		OHEq	0.081	0.063	0.058	0.06	
		OHESq	**0.057**	**0.048**	**0.043** $*$	·	
		S4OPM	0.122				
	15	OHq	0.084	0.048	0.049	0.059	1020
		OHEq	0.07	0.045	0.039	0.037	
		OHESq	**0.047**	**0.034**	**0.028** $*$	**0.030**	
		S4OPM	0.087				
DJIA	7	OHq	0.102	0.09	0.136	0.246	69,054
		OHEq	0.112	0.107	0.141	0.254	
		OHESq	**0.081** $*$	·	·	·	
		S4OPM	0.196				
	11	OHq	0.078	0.050	0.061	0.085	1381
		OHEq	0.073	0.053	0.049	0.055	
		OHESq	**0.051**	**0.042**	**0.038** $*$	·	
		S4OPM	0.112				
	15	OHq	0.072	0.038	0.040	0.049	1002
		OHEq	0.063	0.040	0.033	0.031	
		OHESq	**0.043**	**0.030**	**0.026** $*$	**0.026**	
		S4OPM	0.079				

Table 3 shows the average number of order isomorphism tests for each m and q of OHq, OHEq, and OHESq using the bad character heuristic. When comparing OHq and OHEq, OHq tested for order isomorphism fewer times than OHEq in all cases. This is because the fingerprints used in [4] based on the factorial number system have a smaller probability that two fingerprints are identical compared to the fingerprints used in this paper. Meanwhile, OHESq tested for order isomorphism fewer times than OHEq in all cases and fewer times than OHq in most cases. We show the execution times of the preprocessing phases and search phases of OHq, OHEq, and OHESq for 1000 patterns in Tables A1–A3.

Table 3. Comparison of the average numbers of order isomorphism tests for VOLT, PM2.5, and DJIA data.

Data	m	Algorithm	Average Number of Order Isomorphism Tests			
			$q = 3$	$q = 4$	$q = 5$	$q = 6$
VOLT	7	OHq	68,460	23,404	8186	3005
		OHEq	56,872	36,608	23,605	15,861
		OHESq	15,861	·	·	·
	11	OHq	51,311	21,718	3676	1037
		OHEq	36,729	18,356	10,381	6257
		OHESq	10,388	3393	1037	·
	15	OHq	47,110	9603	2440	662
		OHEq	31,112	13,148	6591	3587
		OHESq	36,065	8527	2336	725
PM2.5	7	OHq	3384	1523	711	367
		OHEq	2601	1638	1076	747
		OHESq	747	·	·	·
	11	OHq	2587	914	379	164
		OHEq	1737	902	511	306
		OHESq	464	162	55	·
	15	OHq	2244	658	263	114
		OHEq	1442	663	342	189
		OHESq	1480	376	115	39
DJIA	7	OHq	2492	930	366	156
		OHEq	2092	1319	855	577
		OHESq	577	·	·	·
	11	OHq	1870	518	171	60
		OHEq	1328	664	373	221
		OHESq	369	121	37	·
	15	OHq	1742	406	124	40
		OHEq	1146	487	243	132
		OHESq	1282	305	85	27

5. Conclusions

This study improved the time and space complexity of the previous work on the OPPM problem by utilizing fingerprints of q-grams converted to binary numbers. Experiments on three types of time series data showed our algorithm is faster than the previous work because we reduced the number of order isomorphism tests. We believe the execution times of OPPM algorithms are highly related to the characteristics of the data, such as permutation entropy. Therefore, classifying the criteria of data characteristics and identifying the data according to the criteria can be an important research tasks in the future.

Author Contributions: Y.K. (Youngjoon Kim) and Y.K. (Youngho Kim) designed and analyzed the algorithm. Y.K. (Youngjoon Kim) implemented and experimented with the algorithms and wrote the draft of the paper. Y.K. (Youngho Kim) and J.S.S. reviewed and revised the paper. J.S.S. analyzed the algorithm, provided algorithmic support, and was the project manager. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Genome Program for Fostering New Post-Genome Industry of the National Research Foundation (NRF) funded by the Korean Government (MSIP) (NRF-2014M3C9A3064706), by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean Government (MSIT) (2020-0-01389, Artificial Intelligence Convergence Research Center (Inha University)), and by INHA UNIVERSITY Research Grant.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare they have no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

OPPM	order-preserving pattern matching
KMP	Knuth–Morris–Pratt
SSE	streaming SIMD extensions
AVX	advanced vector extensions
VOLT	power consumption data
PM2.5	particulate matter data
DJIA	Dow Jones Index data

Appendix A

Table A1. Execution times of 1000 patterns for VOLT data.

Data	m	Algorithm	Phase	Execution Time (s)			
				$q = 3$	$q = 4$	$q = 5$	$q = 6$
VOLT	7	OHq	Prep.	0	0	0.002	0
			Search	2.756	2.338	3.690	6.835
			Total	2.756	2.338	3.692	6.835
		OHEq	Prep.	0.001	0	0	0
			Search	3.051	2.886	3.778	6.822
			Total	3.052	2.886	3.778	6.822
		OHESq	Prep.	0.001			
			Search	2.097	·	·	·
			Total	2.098			
	11	OHq	Prep.	0	0	0	0.003
			Search	2.129	1.301	1.612	2.229
			Total	2.129	1.301	1.612	2.232
		OHEq	Prep.	0.001	0.005	0	0.003
			Search	1.961	1.421	1.356	1.448
			Total	1.962	1.426	1.356	1.451
		OHESq	Prep.	0	0	0.002	
			Search	1.366	1.046	1.028	·
			Total	1.366	1.046	1.030	
	15	OHq	Prep.	0.001	0.001	0.001	0.003
			Search	1.940	0.995	1.057	1.352
			Total	1.941	0.996	1.058	1.355
		OHEq	Prep.	0.003	0.001	0.003	0.002
			Search	2.077	1.175	1.000	0.921
			Total	2.080	1.176	1.003	0.923
		OHESq	Prep.	0.002	0.002	0.001	0.004
			Search	1.182	0.789	0.670	0.682
			Total	1.184	0.791	0.671	0.686

Table A2. Execution times of 1000 patterns for PM2.5 data.

Data	m	Algorithm	Phase	Execution Time (s)			
				$q = 3$	$q = 4$	$q = 5$	$q = 6$
PM2.5	7	OHq	Prep.	0	0.002	0.001	0
			Search	0.119	0.107	0.159	0.280
			Total	0.119	0.109	0.160	0.280
		OHEq	Prep.	0	0	0.001	0
			Search	0.128	0.121	0.152	0.281
			Total	0.128	0.121	0.153	0.281
		OHESq	Prep.	0.001			
			Search	0.088	·	·	·
			Total	0.089			
	11	OHq	Prep.	0.001	0.001	0	0
			Search	0.092	0.062	0.073	0.096
			Total	0.093	0.063	0.073	0.096
		OHEq	Prep.	0.001	0.001	0.004	0
			Search	0.080	0.062	0.054	0.060
			Total	0.081	0.063	0.058	0.060
		OHESq	Prep.	0.004	0.001	0.001	
			Search	0.053	0.047	0.042	·
			Total	0.057	0.048	0.043	
	15	OHq	Prep.	0	0.001	0.001	0
			Search	0.084	0.047	0.048	0.059
			Total	0.084	0.048	0.049	0.059
		OHEq	Prep.	0.001	0.001	0	0.001
			Search	0.069	0.044	0.039	0.036
			Total	0.070	0.045	0.039	0.037
		OHESq	Prep.	0.001	0.004	0	0
			Search	0.046	0.030	0.028	0.030
			Total	0.047	0.034	0.028	0.030

Table A3. Execution times of 1000 patterns for DJIA data.

Data	m	Algorithm	Phase	Execution Time (s)			
				$q = 3$	$q = 4$	$q = 5$	$q = 6$
DJIA	7	OHq	Prep.	0.001	0	0	0.001
			Search	0.101	0.090	0.136	0.245
			Total	0.102	0.090	0.136	0.246
		OHEq	Prep.	0	0.001	0	0
			Search	0.112	0.106	0.141	0.254
			Total	0.112	0.107	0.141	0.254
		OHESq	Prep.	0.001			
			Search	0.080	·	·	·
			Total	0.081			
	11	OHq	Prep.	0.003	0.001	0.001	0.002
			Search	0.075	0.049	0.060	0.083
			Total	0.078	0.050	0.061	0.085
		OHEq	Prep.	0.001	0.001	0.001	0
			Search	0.072	0.052	0.048	0.055
			Total	0.073	0.053	0.049	0.055
		OHESq	Prep.	0.00	0.002	0	
			Search	0.050	0.040	0.038	·
			Total	0.051	0.042	0.038	

Table A3. *Cont.*

Data	m	Algorithm	Phase	Execution Time (s)			
				$q = 3$	$q = 4$	$q = 5$	$q = 6$
	15	OHq	Prep. Search Total	0.001 0.071 0.072	0.001 0.037 0.038	0 0.040 0.040	0.001 0.048 0.049
		OHEq	Prep. Search Total	0.001 0.062 0.063	0.002 0.038 0.040	0 0.033 0.033	0.001 0.030 0.031
		OHESq	Prep. Search Total	0.001 0.042 0.043	0.001 0.029 0.030	0 0.026 0.026	0 0.026 0.026

References

1. Kim, J.; Eades, P.; Fleischer, R.; Hong, S.; Iliopoulos, C.S.; Park, K.; Puglisi, S.J.; Tokuyama, T. Order-preserving matching. *Theor. Comput. Sci.* **2014**, *525*, 68–79. [CrossRef]
2. Kubica, M.; Kulczynski, T.; Radoszewski, J.; Rytter, W.; Walen, T. A linear time algorithm for consecutive permutation pattern matching. *Inf. Process. Lett.* **2013**, *113*, 430–433. [CrossRef]
3. Knuth, D.E.; Morris, J.H., Jr.; Pratt, V.R. Fast Pattern Matching in Strings. *SIAM J. Comput.* **1977**, *6*, 323–350. [CrossRef]
4. Cho, S.; Na, J.C.; Park, K.; Sim, J.S. A fast algorithm for order-preserving pattern matching. *Inf. Process. Lett.* **2015**, *115*, 397–402. [CrossRef]
5. Knuth, D.E. *The Art of Computer Programming, Volume II: Seminumerical Algorithms*, 3rd ed; Addison-Wesley: Boston, MA, USA, 1998.
6. Mares, M.; Straka, M. Linear-Time Ranking of Permutations. In Proceedings of the Algorithms—ESA 2007, 15th Annual European Symposium, Eilat, Israel, 8–10 October 2007; Lecture Notes in Computer Science; Arge, L., Hoffmann, M., Welzl, E., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4698, pp. 187–193. [CrossRef]
7. Chhabra, T.; Tarhio, J. A filtration method for order-preserving matching. *Inf. Process. Lett.* **2016**, *116*, 71–74. [CrossRef]
8. Cantone, D.; Faro, S.; Külekci, M.O. An Efficient Skip-Search Approach to the Order-Preserving Pattern Matching Problem. In Proceedings of the Prague Stringology Conference (PSC) 2015, Prague, Czech Republic, 24–26 August 2015; pp. 22–35.
9. Charras, C.; Lecrog, T.; Pehoushek, J.D. A very fast string matching algorithm for small alphabets and long patterns. In Proceedings of the Annual Symposium on Combinatorial Pattern Matching, Piscataway, NJ, USA, 20–22 July 1998; Springer: Berlin/Heidelberg, Germany, 1998; pp. 55–64.
10. Intel. Intel®64 and IA-32 architectures optimization reference manual. In *Order Number: 248966*; Intel: Santa Clara, CA, USA, 2011; Volume 25.
11. Faro, S.; Külekci, M.O. Fast Packed String Matching for Short Patterns. In Proceedings of the 15th Meeting on Algorithm Engineering and Experiments, ALENEX 2013, New Orleans, LA, USA, 7 January 2013; Sanders, P., Zeh, N., Eds.; SIAM: Philadelphia, PA, USA, 2013; pp. 113–121. [CrossRef]
12. Faro, S.; Külekci, M.O. Fast and flexible packed string matching. *J. Discret. Algorithms* **2014**, *28*, 61–72. [CrossRef]
13. Jeong, H.; Kim, S.; Lee, W.; Myung, S. Performance of SSE and AVX Instruction Sets. *CoRR* **2012**.
14. Intel. *Intel Architecture Instruction Set Extensions Programming Reference*; Intel Corp.: Santa Clara, CA, USA; 2004.
15. Chhabra, T.; Faro, S.; Külekci, M.O.; Tarhio, J. Engineering order-preserving pattern matching with SIMD parallelism. *Softw. Pract. Exp.* **2017**, *47*, 731–739. [CrossRef]
16. Nakamura, T.; Inenaga, S.; Bannai, H.; Takeda, M. Order Preserving Pattern Matching on Trees and DAGs. In Proceedings of the String Processing and Information Retrieval—24th International Symposium, SPIRE 2017, Palermo, Italy, 26–29 September 2017; Lecture Notes in Computer Science; Fici, G., Sciortino, M., Venturini, R., Eds.; Springer: Berlin/Heidelberg, Germany, 2017; Volume 10508, pp. 271–277. [CrossRef]
17. Na, J.C.; Lee, I. A Simple Heuristic for Order-Preserving Matching. *IEICE Trans. Inf. Syst.* **2019**, *102-D*, 502–504. [CrossRef]
18. Crochemore, M.; Iliopoulos, C.S.; Kociumaka, T.; Kubica, M.; Langiu, A.; Pissis, S.P.; Radoszewski, J.; Rytter, W.; Walen, T. Order-preserving indexing. *Theor. Comput. Sci.* **2016**, *638*, 122–135. [CrossRef]
19. Horspool, R.N. Practical Fast Searching in Strings. *Softw. Pract. Exp.* **1980**, *10*, 501–506. [CrossRef]
20. Hebrail, G.; Berard, A. Individual Household Electric Power Consumption Data Set. 2012. Available online: https://archive.ics.uci.edu/ml/datasets/individual+household+electric+power+consumption (accessed on 17 December 2021).
21. Chen, S.X. Beijing PM2.5 Data Set. 2017. Available online: https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data, (accessed on 17 December 2021).
22. Williamson, S. Daily Closing Values of the DJA in the United States, 1885 to Present, Measuring Worth. 2021. Available online: https://www.measuringworth.com/datasets/DJA/index.php (accessed on 17 December 2021).
23. Navarro, G.; Raffinot, M. *Flexible Pattern Matching in Strings—Practical Online Search Algorithms for Texts and Biological Sequences*; Cambridge University Press: Cambridge, UK, 2002. [CrossRef]