Вісник Київського національного університету імені Тараса Шевченка Серія: фізико-математичні науки

2023, 1

Bulletin of Taras Shevchenko National University of Kyiv Series: Physics & Mathematics

https://doi.org/10.17721/1812-5409.2023/1.11

Зуєв А.М.[1], *аспірант*

Пошук рядка в тексті. Актуальні досягнення та дослідження

[1] Київський національний університет імені Тараса Шевченка, 83000, м. Київ, пр-т. Глушкова 4д,
e-mail: ant0n3uev@gmail.com

Anton Zuiev[1], *postgraduate*

Exact pattern matching. Current achievements and research.

[1]Taras Shevchenko National University of Kyiv, 83000, Kyiv, 4d Glushkova str.,
e-mail: ant0n3uev@gmail.com

Задача пошуку рядка в тексті є дуже важливою задачею програмування. На різних алгоритмах розв'язання даної задачі базуються пошукові системи, системи контролю версій, текстові редактори, аналізатори ДНК та багато інших систем. Дана робота розглядає, не претендуючи на повноту, набір алгоритмів що є актуальними для розв'язання цієї задачі для різної специфіки вхідних даних та пояснює закономірності у зміні швидкостей роботи алгоритмів в залежності від специфіки вхідних даних.

Ключові слова: пошук підрядка в рядку, точний пошук підрядка в рядку, алгоритми на рядках, обробка текстів.

The problem of exact pattern matching is an essential programming problem. Different algorithms that solve this problem are core elements of search engines, version control systems, text editors, DNA analyzers, and many others. For simplification reasons articles usually denote pattern as P or p and pattern length as M or m. Similarly, the text is usually denoted as T or t and its length - N or n. Alphabet is denoted Σ and its length - $|\Sigma|$. Based on these notations the problem of pattern matching can be written as follows:

Find all positions/ amount of i, such that $P[0...m] = T[i...i + m]$, or: Find all positions i in text for which substring starting at position i of the text of length m is equal to the pattern.

The main parameters of this problem are pattern length and alphabet size. The length of the text usually doesn't matter because, for any long enough text of a specific structure, the run time of the algorithm per character will be close to constant. Besides that, the specifics of the input data and text may also impact the performances of the algorithms. All of that makes the problem both very nuanced and interesting to investigate. This problem features a lot of different existing solutions developed over the course of the last 5 decades. The main part of the work provides short descriptions and analyses of a set of algorithms that are still relevant in the field. Besides that, some remarks are made on the topic of their theoretical regions of efficiency and how they depend on the specifics of the input. The results of the practical experimentation on the variety of randomly generated test data are provided. The conclusion provides some analysis of the received results and algorithms' class efficiency based on the input as well as a visual representation of the received results in a form of a table representing the most efficient algorithm for each pair of pattern length and alphabet size.

Key Words: pattern matching, exact pattern matching, string algorithms, text processing.

Сімейство алгоритмів пошуку підрядка в рядку з використанням суміжних вікон пошуку

Дана робота демонструє алгоритм MAW22, що базується на наступній ідеї використання декількох суміжних вікон пошуку які зміщуються одночасно доки одне з них не досягне потенційної точки збігу. Це дозволяє трохи оптимізувати час роботи, адже кожна ітерація циклу є досить затратною, а використання декількох вікон пошуку дозволяє зменшити кількість ітерацій (приблизно) у стільки разів, скільки вікон пошуку використано. Використання суміжних вікон дозволяє уникнути суттєвих ускладнень логіки усередині циклу, але натомість створює необхідність конструювання загальної таблиці зсувів. Розмір цієї таблиці складатиме $|\Sigma|w * l$, де $|\Sigma|$ - розмір алфавіту, w - кількістю суміжних вікон, l - довжина кожного з вікон. На жаль, збільшення будь-якого з параметрів призводитиме до суттєвого збільшення у споживанні пам'яті що, своєю чергою сповільнюватиме пошук. Результати, продемонстровані у ро-

боті показують високу ефективність алгоритму MAW22 (2 вікна пошуку, 2 символи для кожного з вікон) для алфавітів розміром 4-8 та довжин патернів 3-72. Дана робота є демонстрацією використання специфіки вхідних даних для отримання кращого для них алгоритму. Крім цього, робота демонструє метод для отримання стиснутої таблиці зсувів що дозволятиме використання даного підходу до алфавітів більшого розміру (алгоритм MAWP22).

Швидкий пошук підрядка в рядку з використанням бітового представлення символів

У цій статті обговорюються алгоритми, що базуються на ідеї пошуку максимального можливого зсуву для заданого вікна пошуку. Дослідження зосереджується на наступних методах покращення продуктивності: 1. Використання 1,5-байтового зчитування замість 1-байтового або 2-байтового зчитування 2. Використання прапорця наявності зсуву замість довжини зсуву 3. Мультивіконний пошук Перший метод передбачає модифікацію таблиці зсуву, що використовується в алгоритмі, яка визначає, наскільки далеко зсунути патерн у разі невідповідності з текстом. Пропонується використовувати 1,5-байтове зчитування, де два послідовні байти тексту використовуються для обчислення зсуву. Цей підхід більш ефективний, ніж використання 1-байтового зчитування, коли шаблон довгий і 1-байтовий зсув недостатній, але потребує менше пам'яті, ніж використання 2-байтового зчитування, при цьому втрата довжини зсуву є порівняно невеликою. Використовуючи бітову маску, 1,5-байтове читання може досягти продуктивності, подібної до 2-байтового читання, з меншим використанням пам'яті (що також може позитивно вплинути на швидкість виконання). Другий метод полягає в тому, щоб використати так званий "fast loop". Виконувати порівняння патерну з вікном пошуку необхідно лише у випадку якщо виконується деяка попередня передумова (для даних алгоритмів це збіг деякої кількості символів патерну з відповідними символами в тексті). У іншому випадку необхідності заходити у цикл порівняння немає. Також, замість того щоб зберігати довжину зсуву для кожної позиції буде зберігатися лише факт можливості зсуву на максимальну довжину. У випадку, коли часто можливість зсуву означає можливість максимального зсуву дана методика дозволить прискорити виконання алгоритму. Можна піти далі, та після перевірки можливості зсуву у позиції і також перевірити можливості зсуву у позиції (i-1). Така перевірка значно збільшує ймовірністю виконати зсув на максимальну довжину навіть у випадках якщо перша перевірка мала негативний результат. Третій метод використовує той факт що цикли є найбільш повільною частиною програми та намагається мінімізувати кількість їх запусків. Для цього замість пошуку однієї позиції для збігу шукається одразу декілька. Текст розбивається на декілька шматків, традиційно рівної довжини. Після цього, по кожному з цих шматків рухається вказівник, який перевіряє на потенціал збігу (використовуючи оптимізації з методів 1 та 2). Якщо хоч одна з позицій є позицією потенційного збігу то для усіх позицій будуть виконані додаткові перевірки, після чого рух усіх вікон пошуку відновлюється. Даний підхід приводить до отримання алгоритму RZk. Також наявна можливість виконувати перегляд тексту з кінця в початок що дозволяє дещо змінити імплементацію та отримати алгоритм Zk. У роботі представлені експериментальні результати (у тому числі для багатовіконних версій алгоритмів), що демонструють ефективність розроблених алгоритмів на випадкових шаблонах і текстах різного розміру. Вони показують що сукупність методів представлених у роботі разом утворюють досить ефективний алгоритм для специфічної області даних. Також, оскільки розроблений алгоритм оперує на нецілій кількості біт, його також природно застосувати до задачі пошуку підрядка в рядку для якої розмір символу алфавіту виражається кількістю біт не кратною 8. Дана область є порівняно малодослідженою, утім результати, показані у роботі суттєво перевищують у швидкості інші наявні підходи. Також робота розглядає алгоритм Bricks-wq, що використовує ідею побітового суміщення декількох останніх символів патерну, суміщення декількох останніх символів тексту та їх подальше порівняння для знаходження можливості зсуву. Даний алгоритм є ефективним у випадку, коли ймовірність зсуву по одному символу досить низька - а саме для малих алфавітів.

Бітопаралельні алгоритми пошуку для довгих патернів

Як зрозуміло з назви, робота присвячена розробці алгоритмів для довгих патернів. У роботі продемонстровано 3 різних алгоритми. BXS - BNDMq з розширеними зсувами (BNDMq with eXtended Shift). Патерн поділяється на $\lceil \frac{m}{w'} \rceil$ послідовних шматків, кожен з них довжини w‘ (крім, можливо, останнього). Після цього шматки накладаються одне на одного для отримання одного модифікованого (у роботі використовується термін “накладений”) патерну довжини w‘. При порівнянні модифікованого патерну з текстом збіг з символом будь-якого зі шматків, що утворюють модифікований патерн вважається збігом з відповідним символом модифікованого патерну. Далі відбувається ініціалізація масиву B таким же чином як і у BNDM, але для модифікованого патерну. А от при підрахунку D буде відбуватися циклічний зсув замість звичайного зсуву у BNDM. Це, з одного боку зменшить необхідну кількість біт для зберігання D, а з іншого призведе до потреби додавати додаткову перевірку чи не було уже пройдено m символів тексту. Утім практичні результати його використання є досить гарними. BQL Довгий BNDMq (BNDMq Long). BQL збільшує ефективний розмір алфавіту, використовуючи q-грами, що перекриваються, наприклад, при використанні 3-грамового патерну «ACCTGGT» обробляється як «ACC-CCT-CTG-TGG-GGT». Таким чином, ми ефективно шукаємо патерн із $m-q+1$ q-грам, що перекриваються. Подібно до BXS ми поділяємо шаблон q-грами на - $\lceil \frac{m-q+1}{w'} \rceil$ частин і накладаємо їх один на одного. Потім вектор B ініціалізуються для шаблону накладеної q-грами за принципами як у BNDM. На етапі пошуку ми використовуємо модифікацію Simplified BNDM, яка дозволяє нам завжди зрушувати на $m-q+1$, але все ще використовувати лише w‘ біт для бітових векторів B і D. Ми розбиваємо текст на неперекриваючісь вікна пошуку довжиною $m-q+1$ і в кожному вікні ми виконуємо BNDM справа наліво. Щоразу, коли встановлено старший біт у D, ми перевіряємо всі можливі зсуви на збіг шаблону з текстом. Коли вектор D стає нульовим, ми завжди зсуваємо на $m-q+1$. QF - Алгоритм фільтрування q-грам. m. На етапі препроцесингу для кожної фази i, $0 \leqslant i < q$, ми зберінаємо які q-грами входять до патерну у тій фазі, тобто починаються у позиції $i + jq$ для деякого j. Для зберігання цієї інформації вектор B буде ініціалізовано для кожної з q-грам, i-тий біт буде рівний 1 якщо q-грама є у патерні в фазі i. Під час пошуку ми читаємо послідовні q-грами у вікні пошуку та відстежуємо активні фази, тобто таких фаз, для яких усі зчитані q-грами знаходяться в патерні у цій фазі.

Швидкий пошук збігів на основі фільтра з використанням інструкцій SIMD

Дана робота присвячена використанню інструкцій SIMD для розробки алгоритму SSEF. Ці інструкції дозволяють виконувати декілька однакових операцій виконавши лише одну процесорну операцію. Робота фокусується на довгих патернах $m \geqslant 32$. Для них виконуються операції над 16 байтними числами, кожне з яких відповідатиме 16 символам з патерну або тексту. Спочатку для патерну відбувається передпідрахунок на базі послідовних 16 символів патерну. Після цього відбувається фаза пошуку у тексті, яка дозволяє виконати швидку перевірку на збіг для заданої позиції (насправді для множини послідовних позицій) тексту та зсунутися на довжину патерну. На машинах що підтримують вищезгадані інструкції даний алгоритм і справді показує себе у вищому степені ефективно.

Швидкий алгоритм пошуку підрядка в рядку (на основі хешування)

Ідея нового алгоритму полягає в розгляді підрядків довжини q. Підрядки такої довжини хешується за допомогою функції h для значень $0 \leqslant c \leqslant 255$. Етап передобробки цього алгоритму полягає в обчисленні зсуву для усіх можливих рядків довжини q.

$$\text{shift}[c] = \begin{cases} m-1-i \\ \text{with } i = \max\{0 \leqslant j \leqslant m-q+1 \mid \\ h(x[j..j+q-1]) = c\} \\ m-q \\ \text{якщо таке } i \text{ не існує.} \end{cases} \tag{1}$$

Фаза пошуку алгоритму полягає в читанні підрядків довжини q. Якщо $shift[h(B)] > 0$ тоді застосовується зсув довжини $shift[h(B)]$. Якщо $shift[h(B)] = 0$ то перевіряється збіг шаблону з підрядком, починаючи з даної позиції в тексті. У цьому випадку застосовується зсув

стандартної довжини, що базується на хеші патерну. Даний алгоритм дозволяє експериментувати з різними довжинами рядків для хешування. Дана робота показує як застосування хешування може бути ефективним для невеликих патернів та розмірів алфавіту Пошук підрядка з поглядом вперед Forward-SBNDM — це варіант алгоритму BNDM для розв'язання задачі пошуку підрядка в рядку. Forward-SBNDM перевіряє 2-грами в тексті так, що перший символ є останнім у вікні пошуку, а другий знаходиться одразу за вікном. У роботі [6] представлено узагальнення цього підходу, що полягає у перевірці кількох символів одразу за межами вікна пошуку. Крім того, у роботі представлено жадібний цикл пропуску для SBNDM2, що дозволяє виконати стрибки на відстань до 2m. Параметр q та кількість символів за межами вікна пошуку створюють 2 можливих параметри для пошуку оптимальної конфігурації алгоритму. У роботі ці імплементації показано у форматі FSB(q, f), де q- параметр аналогічний BNDMq, а f- кількість символів. В результаті отримано кілька нових варіацій SBNDMq.

Ефективні варіанти алгоритму зворотного пошуку з Оракулом - EBOM, FBOM

Ідея швидкого циклу була описана для алгоритму [1]. Швидкий цикл, який було використано у цій роботі було вперше представлено в алгоритмі Tuned-Boyer-Moore та зазвичай використовується у варіаціях алгоритму Боєра-Мура. Зазвичай швидкий цикл реалізується шляхом ітерації за евристикою поганого символу в циклі без інших перевірок для того, щоб швидко знайти позицію найправішого символу патерну у тексті. Робота пропонує застосувати ідею швидкого циклу до автоматних переходів. Це полягає у зсуві патерну вздовж тексту без додаткових перевірок доки не знайдено визначений перехід з останнього символу вікна пошуку. Запропонована варіація алгоритму намагається виконати два послідовні переходи для кожної ітерації швидкого циклу з метою знайти з більшою ймовірністю невизначений перехід. Це дозволяє отримати алгоритм EBOM. Крім цього алгоритм використовує наступне спостереження: коли під час порівняння шаблону зустрічається символ, що не відповідає символу з поточного вікна пошуку у тексті $t[s..s+m-1]$ шаблон завжди зсувається вправо принаймні на один символ, але ніколи не більше ніж на m символів. Таким чином, символ $t[s+m]$ завжди бере участь обрахунку наступного потенційного вікна пошуку. Таким чином символ що іде одразу після вікна пошуку також можна подавати на вхід автомата, виконавши деякі попередні модифікації його конструювання. Це, разом із попереднім спостереженням дозволяє отримати алгоритм FBOM.

Альтернативні методи бітопаралельного пошуку підрядка в рядку

Дана робота розглядає варіації алгоритму BNDM. Окрім TNDM, у роботі розглянуто три варіанти BNDM. Перший з алгоритмів є двосторонньою модифікацією BNDM. Його назва TNDM. Якщо символ тексту, вирівняний з кінцем шаблону не є збігом, BNDM сканує текст у зворотному напрямку на наявність незбігу в іншому місці шаблону. У цьому випадку TNDM зможе продовжити сканування вперед, тобто продовжуватиме перевірку тексту після вирівнювання. Другий з них має простіше обчислення зсуву, ніж BNDM. Цей алгоритм називається SBNDM. У SBNDM у випадку збігу зсув виконується як у BNDM. Але якщо T_hT_i не з'являються в патерні, ми пропускаємо перевірку префіксів і встановлюємо h + 1 як початок наступного вікна пошуку. Це зменшує середню довжину зсуву, але з іншого боку внутрішній цикл алгоритму стає простішим. Надані у роботі експерименти показують, що даний підхід справді є пришвидшенням порівняно з BNDM у більшості випадків. Третій пропонує техніку, для покращення швидкості роботи на довгих патернах. Цей алгоритм називається Long BNDM або LBNDM. LBNDM здатен робити довгі зсуви. Шаблон розділено на $\lfloor \frac{m}{k} \rfloor$ послідовних частин, кожна з яких складається з $k = \lfloor \frac{m-1}{w} \rfloor + 1$ символів. Залишкові символи залишаються на початку патерну. Цей поділ передбачає k підпослідовностей шаблону, у i-тий з яких входить i-й символ кожної частини. Ідея полягає в тому, щоб спочатку шукати накладений шаблон цих послідовностей, щоб перевіряти лише кожен k-й символ. Це фаза фільтрації виконується за стандартним алгоритмом BNDM. Кожен випадок збігу модифікованого шаблону є потенційним збігом з оригінальним шаблоном та потребує додаткової перевірки.

Швидкий пошук підрядка в рядку з використанням двох вікон пошуку

TSW являє собою один з перших та базових алгоритмів паралельного пошуку. Сама назва TSW - two sliding windows, гарно репрезентує, що його ідея базується на використанні двох вікон пошуку. Одне з вікон встановлюється на початку патерну, а інше в кінці, після чого обидва вікна рухаються до середини патерну та перевіряють на збіг. Ліве вікно перевіряє наявність збігу справа наліво, праве - зліва направо. Таким чином після знаходження частково або повного збігу виконується зсув та продовжується процес зсування вікон пошуку доки вони не зустрінуться. Цей алгоритм є досить простим, утім навіть зараз він здатний показувати пристойні результати.

Ефективний пошук підрядка в бінарному рядку

У цій статті розглядається задача пошуку підрядка в рядку, коли обидва рядки побудовані над двійковим алфавітом, і кожен символ патерну та тексту представлено 1 бітом. Таким чином простір, необхідний для представлення патерну та тексту, становить відповідно $\lceil \frac{n}{8} \rceil$ та $\lceil \frac{m}{8} \rceil$ байт. Ідея алгоритму q-Hash полягає у врахуванні підрядків патерну довжини q. Кожен підрядок w такої довжини q хешується за допомогою хеш-функції у ціле значення в межах від 0 до 255. Фаза пошуку алгоритму складається з читання для кожної позиції вікна пошуку s підрядку $w = t[s + m - q..s + m - 1]$ довжини q. Якщо $H_s(hash(w)) > 0$, тоді застосовується зсув довжини $H_s(hash(w))$. Інакше коли $H_s(hash(w)) = 0$ виконується перевірка на входження патерну в текст. У цьому випадку застосовується зсув довжини (m − 1 − i), де i - початкова позиція крайнього правого входження $p[j..j + q - 1]$ в патерн, для якого такого, що $hash(p[j..j + q - 1]) = hash(p[m - q + 1..m - 1])$.

Експериментальні результати

Тут наведено результати тестування більшості розглянутих алгоритмів на локальній машині. Також, багато алгоритмів мають різні версії, у цьому випадку було вибрано одну з них для репрезентації загальної поведінки алгоритму. Генерація даних для експерименту відбувалася наступним чином - генерується текст та патерн з використанням псевдовипадкових чисел, після цього до тексту 10 разів додається входження патерну у випадкову позицію та після кожного з додавань запускається кожен алгоритм пошуку. Наведені результати - середній час виконання у процесорних тіках з використанням функції QueryPerformanceCounter. Розмір тексту 500 000 символів. OS Windows 10. Процесор AMD Ryzen 5 5600H.

Табл. 1: $|\Sigma| = 2$

algo	2	4	8	16	32	64	128	256	512
MAW22	1643.7	1568.6	1568.27	1147.6	1632.27	1317.27	1273.2	1259.53	1306.27
Bricks43		901.13	851.67	712.67	920.2	923.53	1176.8	1036.67	1034.33
qf33		1254.07	1645.67	1476.27	3278.53	3734.13	6547.4	10703.5	19649.7
bxs4		913.67	668.33	404.33	205.13	253.8	674.73	67088	219410
hash3		1372.8	665.6	451.73	424.53	357.73	416.4	390.2	450.47
hash8			2595.13	296.93	109.53	51.47	27.73	23.33	44.47
fsbndmq41		875.2	662	376.8	192.6	193.13	197.33	195.8	205.07
fsbndmq61			350.33	195.07	139.87	149.67	153.07	139.87	164.93
ebom	1350	1433.07	1282.73	695.93	400.73	225.47	128.6	77.07	48.53
sbndm2	1316.0	1269	882.2	426.33	206.27	191.87	197.93	195.53	202.33
tsw	1157.2	1303.53	1431.4	1018	1269.27	1389.47	1144.67	1484.33	1367.4
z13-w5	**878.53**	1545.13	1635.13	1851.67	2336.13	2147.67	2314.4	2431.8	2029.67

Табл. 2: $|\Sigma| = 4$

algo	2	4	8	16	32	64	128	256	512
MAW22	1914	511.13	311.2	218.13	181.47	170.8	195.13	170.13	161
Bricks43		287.4	142.93	118.33	123.07	100.13	110.53	282.27	504.27
qf33		356.6	156.73	107.33	109.27	84.4	133.6	4839.73	18388.8
bxs4		568	123.73	62.47	41	86.07	71.4	137.27	588.6
hash3		1152.33	418.87	213.33	155.47	117.73	108	103.8	111.13
hash8			2602.4	294.8	112.07	52.33	27.53	21.33	22.2
fsbndmq41		355.6	117.67	64.2	41.93	41.47	42.27	41.53	43.53
fsbndmq61			181.47	61.47	28.67	28.87	28.87	29.47	30.4
ebom	519.87	485.73	445.8	283.93	168.07	95.07	57.93	34	23.27
sbndm2	552.53	471.6	405.8	219.07	115.27	109.13	108.87	112.93	110.67
tsw	1036.4	786.2	497.33	329.13	276.2	210.8	216.93	188.2	213.53
z13-w5	460.93	471.67	401.4	353.27	309.53	395.13	514.4	637.53	586

Табл. 3: $|\Sigma| = 8$

algo	2	4	8	16	32	64	128	256	512
Bricks43		160.07	59.87	33.2	21.27	16.93	14.8	15.47	16.47
qf33		161	62.67	30.27	19.47	15.53	12.67	13.27	10.53
bxs4		473.8	94.87	38.67	18.73	17.2	36.6	35.8	71
hash3		1018.53	348.6	171.33	95.07	65.33	50.8	46.27	41.53
hash8			2538.93	294	109.67	53.13	27.07	21.6	20.07
fsbndmq41		251.93	84.07	37.13	19.67	19.53	19.27	19.13	19.07
fsbndmq61			175.67	60.07	27.33	27.87	26.47	27.2	27
ebom	334.47	161.47	111.67	84.4	72.53	47.8	25.07	18.13	13.67
sbndm2	331.2	164.13	110.8	85.67	66.93	70.67	72.47	66.93	69.2
tsw	574.13	383.53	225.53	133.33	80.27	49.47	35.6	32.4	29.47
z13-w5	228.33	152.8	123.67	99.13	100.53	85.93	66.8	107.93	160.27

Табл. 4: $|\Sigma| = 16$

algo	2	4	8	16	32	64	128	256	512
Bricks43		150.13	51.13	23.8	11.6	7.67	5.53	3.67	3.13
qf33		159.93	62.27	29.87	18.67	14.73	11.53	11.33	9.87
bxs4		476.4	95.53	36.4	16.8	9	9.07	17	16.73
fsbndmq41		249.73	82.27	35.33	17	17	17	17.27	17.6
fsbndmq61			179.33	60.07	26.13	26.53	26.6	26.6	26.87
ebom	276.8	104.47	55.87	34.53	24.53	18	15.93	12.6	11.07
sbndm2	276.2	104.6	55.27	34.47	24.13	25.6	25.27	25.47	25.8
tsw	393.8	261.73	157.6	90.2	49.4	27.6	16.13	12.07	11.2
z13-w5	168.27	70.73	42.13	31.8	25.67	21.47	23.2	21.53	19.53

Табл. 5: $|\Sigma| = 32$

algo	2	4	8	16	32	64	128	256	512
qf33		153.47	60	31.4	18.2	13.4	10.6	10.87	10.27
bxs4		468.93	94	37	16.87	9.27	6	4.67	3.73
fsbndmq41		242.53	81.07	36.13	17	17	17.13	19.73	18.33
fsbndmq61			177.13	60.2	26.8	27	27.33	30.4	27.8
ebom	269.13	90.4	39.4	20	12.6	9.07	8.47	7.6	10.07
sbndm2	256.4	85.07	37.4	18.27	9.8	10.07	10.6	12.87	11.4
tsw	369.4	245.73	147.93	84.73	45.67	24.07	13	11.33	6.4
z13-w5	152.47	55.4	28.87	14.8	9.6	7.27	6.73	6.13	7

Табл. 6: $|\Sigma| = 64$

algo	2	4	8	16	32	64	128	256	512
qf33		154.13	73.73	29.33	18.6	13.53	11.07	12.33	8.67
bxs4		476.93	108.73	36.47	17.4	9.13	6.2	4.13	3.4
fsbndmq41		249.33	94.4	35.2	17	17.13	17.2	18.33	18.6
fsbndmq61			202.8	59.6	26.93	27.13	27	28	28.07
ebom	267.53	88.67	49.2	20.6	12.6	10.53	8.6	7.2	11.4
sbndm2	257.47	85.93	42.33	17.27	9.2	11.07	10.07	10.4	10.73
tsw	382.2	254.27	164.93	86.8	47.87	26.67	15.2	12.87	9.07
z13-w5	151.8	53.6	24	12.2	7.6	5.67	4.6	4.33	3.47

Табл. 7: $|\Sigma| = 128$

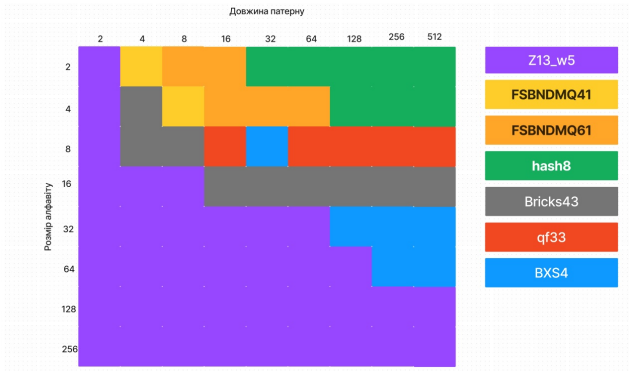
algo	2	4	8	16	32	64	128	256	512
qf33		158.27	60.8	29.8	17.87	13.67	11.47	10	10.8
bxs4		482.93	93.33	37.13	16.73	9.13	6.73	4.47	3.67
fsbndmq41		251.4	80.8	35.07	17.87	17	17.87	17.8	22.6
fsbndmq61			176.47	60.67	26.93	27.2	29.47	27.73	32.53
ebom	265.07	91.47	45.13	19.13	12.6	9.53	8.73	7.8	13.4
sbndm2	252.87	86.93	37.4	18.2	9.13	9.93	10.73	10.33	15.07
tsw	380.33	258.73	151.4	88.27	47.93	28.2	16.27	12.93	9.93
z13-w5	159.07	55	25	12.87	8	5.33	5.87	2.53	2.33

Табл. 8: $|\Sigma| = 256$

algo	2	4	8	16	32	64	128	256	512
qf33		154.47	57.93	30.27	19.47	14.27	18.4	19.53	10.53
bxs4		476.53	94.6	38.6	18.13	10.13	15	8.47	3.73
fsbndmq41		248.47	82.2	35.13	17.47	17.4	22.87	28.13	20.53
fsbndmq61			180.4	62.2	27.87	27.47	36	36	29.6
ebom	272.13	91.8	43.8	23.27	16.6	13	16.33	15.47	15.47
sbndm2	252.6	86.53	37.2	17.67	10.47	11.6	16.6	20.73	14.47
tsw	567.53	385.4	240.13	139.87	90.53	50.4	27.47	37.2	13
z13-w5	153	51.73	22.33	11	6.33	5.07	3.73	2.07	1.6

Висновок

Нижче наведено візуальну репрезентацію найкращих результатів на базі отриманих результатів. Варто зазначити, що список алгоритмів розглянутих у роботі є далеко не вичерпним, у порівнянні брали участь далеко не всі версії алгоритмів та результати можуть залежати від специфіки методів генерації вхідних даних. Для початку варто зазначити, що алгоритми дуже ефективні для патернів довжини 2 не були включені у порівняння тому для цієї довжини результати не є актуальними та включені виключно для ілюстративності. Утім, навіть даний результат дозволяє помітити певні тенденції залежності типу ефективних алгоритмів від параметрів вхідних даних. Правий верхній кут - довгий патерн короткий текст є зоною де домінують алгоритми на базі хешування, адже вони за даних умов по суті дозволяють виконувати зсуви суттєвої довжини, адже збіг хешів за даних умов порівняно малоймовірний за відсутності збігу відповідних множин символів. У регіоні середньо-коротких патернів та малих алфавітів домінують похідні алгоритми FSBNDM, адже вони дозволяють ефективно виконувати декілька порівнянь одночасно, які у випадку інших алгоритмів доводилося б порівнювати окремими операціями. Збільшуючи розміри текстів та довжини патернів ефективність множинного порівняння дещо знижується та алгоритми що використовують для початкового порівняння деяку функцію від підрядка патерну та відповідного підрядку тексту стають значно ефективнішими. У даній роботі це qf33, BXS4, Bricks43. Усі вищеперераховані алгоритми займають шматок нижче головної діагоналі. Для випадків коли розмір алфавіту є досить суттєвим та перевищує розмір патерну ефективність зсуву по множині символів майже не перевищує ефективності зсуву виконаного за одним символом, але витрачає більше часу на порівняння. Тому дану зону займатимуть алгоритми що ефективно комбінують підходи зсуву за одним символом та кілька вікон пошуку. У нашому випадку даним алгоритмом є розглянутий у роботі z13-w5.



Список використаних джерел

1. *Zavadskyi, I.O.* A family of exact pattern matching algorithms with multiple adjacent search windows. Proceedings of the Prague Stringology Conference, p.152–166. J. Holub and J. Zdarek, Eds. Czech Technical University in Prague, Czech Republic (2017)

2. *Zavadskyi, I.O* Fast Exact Pattern Matching by the Means of a Character Bit Representation. SN Computer Science. 3. Springer (2022)

3. *Durian, B., Peltola, H., Salmela, L., Tarhio, J. Bit-parallel search algorithms for long patterns.* 9th International Symposium on Experimental Algorithms.Ischia Island, Naples, Italy., pp. 129–140. Springer (2010)

4. *Kulekci, M.* Filter based fast matching of long patterns by using simd instructions. Proceedings of the Prague Stringology Conference, pp. 118–128. J. Holub and J. Zdarek, Eds. Czech Technical University in Prague, Czech Republic (2009)

5. *Lecroq, T.* Fast exact string matching algorithms. Information Processing Letters 102(6), 229–235 (2007)

6. *Peltola, H., Tarhio, J.* String matching with lookahead. Discrete Applied Mathematics 163, 352–360 (2014)

7. *Faro, S., Lecroq, T.* Efficient variants of the backward-oracle-matching algorithm. Proceedings of the Prague Stringology Conference, pp. 146–160. J. Holub and J. Zdarek, Eds. Czech Technical University in Prague, Czech Republic (2008). DOI 10.1142/S0129054109006991

8. *Peltola, H., Tarhio, J.* Alternative algorithms for bit-parallel string matching. pp.80–94 (2003)

9. *Hudaib, A., Al-khalid, R., Suleiman, D., Itriq, M.A.A., Al-Anani, A.* A fast pattern matching algorithm with two sliding windows (tsw). Journal of Computer Science 4(5), 393–401. DOI 10.3844/jcssp.2008.393.401

10. *Faro, S., Lecroq, T.* Efficient pattern matching on binary strings. 35th International Conference on Current Trends in Theory and Practice of Computer Science, p. Poster (2009)

11. *Zavadskyi, I.O* Fast exact pattern matching in a bitstream and 256-ary strings

References

1. ZAVADSKYI, I.O. *A family of exact pattern matching algorithms with multiple adjacent search windows.* Proceedings of the Prague Stringology Conference, p.152–166. J. Holub and J. Zdarek, Eds. Czech Technical University in Prague, Czech Republic (2017)

2. ZAVADSKYI, I.O *Fast exact pattern matching by the means of a character bit representation* SN Computer Science. 3. Springer (2022)

3. DURIAN, B., PELTOLA, H., SALMELA, L., TARHIO, J. *Bit-parallel search algorithms for long patterns.* 9th International Symposium on Experimental Algorithms.Ischia Island, Naples, Italy., pp. 129–140. Springer (2010)

4. KULEKCI, M. *Filter based fast matching of long patterns by using simd instructions.* Proceedings of the Prague Stringology Conference, pp. 118–128. J. Holub and J. Zdarek, Eds. Czech Technical University in Prague, Czech Republic (2009)

5. LECROQ, T. *Fast exact string matching algorithms.* Information Processing Letters 102(6), 229–235 (2007)

6. PELTOLA, H., TARHIO, J. *String matching with lookahead.* Discrete Applied Mathematics 163, 352–360 (2014)

7. FARO, S., LECROQ, T. *Efficient variants of the backward-oracle-matching algorithm.* Proceedings of the Prague Stringology Conference, pp. 146–160. J. Holub and J. Zdarek, Eds. Czech Technical University in Prague, Czech Republic (2008). DOI 10.1142/S0129054109006991

8. PELTOLA, H., TARHIO, J. *Alternative algorithms for bit-parallel string matching.* pp.80–94 (2003)

9. HUDAIB, A., AL-KHALID, R., SULEIMAN, D., ITRIQ, M.A.A., AL-ANANI, A. *A fast pattern matching algorithm with two sliding windows (tsw).* Journal of Computer Science 4(5), 393–401. DOI 10.3844/jcssp.2008.393.401

10. FARO, S., LECROQ, T. *Efficient pattern matching on binary strings.* 35th International Conference on Current Trends in Theory and Practice of Computer Science, p. Poster (2009)

11. ZAVADSKYI, I.O *Fast exact pattern matching in a bitstream and 256-ary strings*

Надійшла до редколегії 29.01.2023