УДК 519.72
https://doi.org/10.17721/1812-5409.2020/4.5

І.О. Завадський, д. ф.-м. н., доцент

Метод швидкого пошук патерна в потоці бітів

Київський національний університет імені Тараса Шевченка, 03680, м. Київ, пр-т. Глушкова 4д, e-mail: zava@ukr.net

Igor O. Zavadskyi, Associate Professor, Ph. D.

Fast pattern matching method for a bitstream

Taras Shevchenko National University of Kyiv, 03680, Kyiv, Glushkova av., 4d
e-mail: zava@ukr.net

Описано родину алгоритмів швидкого пошуку заданої послідовності бітів у бітовому потоці, який зберігається в пам'яті, умовно поділеній на байти та машинні слова. Ці алгоритми базуються на алгоритмах швидкого пошуку в тексті на 256-символьному алфавіті родини RZ. За швидкодією запропоновані алгоритми перевершують всі відомі аналоги для всіх довжин патерна від 20 до 500.
Ключові слова: пошук рядка, бітовий потік, алгоритм Фаро-Лекро, вікно пошуку.

A family of algorithms for fast pattern matching in a bitstream is described. A bitstream is assumed to be stored in a memory and divided into bytes and machine words. The proposed algorithms are based on RZ-family of algorithms for fast pattern matching in a 256-ary text and inherit such their properties as '1.5-byte read', double skip loop, right-to-left text processing as well as a special technique of multiple sliding windows. Beyond that, the new algorithms provide longer average shifts than the best known Binary Faro-Lecroq algorithm and more efficient technique of match checking. Applied to a text close to random, our algorithms outperform all known analogues for all pattern lengths from 20 up to 500.
Key words: pattern matching, Faro-Lecroq, fast search, bitstream, search window.

Статтю представив д. ф.-м. н., проф. Анісімов А.В.

1. Вступ

Задача пошуку заданої послідовності бітів (патерна) у бітовому потоці є важливим різновидом загальної задачі пошуку рядка в тексті. Вона становить інтерес, оскільки широкий спектр даних прийнято подавати в бінарній формі, наприклад зображення, відео, архівні дані тощо. Ця задача відрізняється від задачі пошуку рядка в тексті на двійковому алфавіті, оскільки в останньому випадку за допомогою однієї операції можна зчитати один символ тексту (що може мати лише 2 різних значення), а в потоці бітів одна операція зчитує машинне слово (як правило, від 8 до 64 бітів).

Базова ідея будь-якого нетривіального алгоритму пошуку даних у бінарному потоці полягає в тому, щоб уникати виконання витратних операцій з окремими бітами настільки довго, наскільки це можливо. Таким чином, пошук виконується на рівні байтів, тобто в тексті на 256-символьному алфавіті. І тільки коли підрядок-кандидат знайдено, на рівні окремих бітів перевіряється, чи справді має місце збіг бітового патерна з частиною тексту.

Отже, будь-який алгоритм пошуку в бітовому потоці базується на певному загальному алгоритмі пошуку рядка в тексті, застосованому до 256-символьного алфавіту. Так, найперший з відомих нетривіальних алгоритмів бітового пошуку [1] базується на загальному пошуковому методі Боєра-Мура, а в найшвидшому з відомих на сьогодні, алгоритмі Binary Faro-Lecroq [2], комбінується багаторядкова версія алгоритму BNDM зі спрощеною пошуковою стратегією алгоритму Коментц-Волтер. Описані далі алгоритми пошуку в бітовому потоці *RZ-bit* базуватимуться на алгоритмах пошуку в 256-розрядному тексті родини RZ-byte, описаних в [3]. Відмінними рисами алгоритмів RZ-byte є виконання зсувів на основі опрацювання бітів більш ніж одного, але менш ніж двох байтів, подвійний цикл пропуску та ін. У [3] пояснено, як ці техніки підвищують ефективність пошуку.

Надалі розглядатимемо процесорну архітектуру зі зворотним зчитуванням байтів машинного слова (little endianness) і використовуватимемо такі позначення:

$T[0...n-1]$ – вхідний текст;

$P[0...m]$ – шуканий рядок (патерн).

© І.О. Завадський, 2020

Як останній байт патерна $P[m]$, так і останній байт тексту $T[n–1]$ може бути неповним і доповнюється справа нулями, якщо довжина тексту чи патерну не є кратною 8. Інакше припускаємо, що $P[m]=0$, а $T[n–1]$ — повний байт. Також завжди $P[m+1]=T[n]=0$. Бітову довжину патерна позначимо через l, а масив бітів патерна — через $p[0...l–1]$. Під «вікном пошуку» розумітимемо послідовність l бітів тексту, щодо якої робиться припущення про збіг з патерном.

2. Пошуковий алгоритм

У розглянутих в [3] алгоритмах байтового пошуку RZ-Byte на кожній ітерації пошукового «циклу пропуску» вікно пошуку зміщується на якомога більшу кількість символів уліво. У бітових алгоритмах вважатимемо, що *pos* — це номер найлівішого повного байта, що охоплюється вікном пошуку. Загалом це вікно охоплює повністю байти $T[pos],…,T[pos+m–2]$, у той час як байти $T[pos–1]$, $T[pos+m–1]$ і, можливо, $T[pos+m]$ можуть містити лівий та правий «хвости» патерна.

Максимальний зсув вікна пошуку на $m–1$ байтів уліво можливий за виконання двох умов: пара байтів ($T[pos]$, $T[pos+1]$) не належить патерну та певний префікс цієї пари байтів довжиною більше 8 бітів не є суфіксом патерна. Таблиця зсувів Z заповнюється згідно цих умов на стадії передобчислень (алгоритм 3), та застосовується у фазі пошуку (рядки 5 і 8 алгоритму 1). Загалом блок псевдокоду в рядках 2–8 алгоритму 1 реалізує подвійний цикл пропуску, розглянутий у [3].

Після виходу з подвійного циклу пропуску ми перевіряємо, чи збігається патерн з вікном пошуку, зсунутим на q бітів уліво, $q = 0,…,7$. Ця перевірка виконується процедурою *CheckMatch*(*q,pos*), що викликається в рядку 11 алгоритму 1. Замість перевірки всіх 7 значень q, яка була б неефективною, ми зберігаємо в множині $\lambda[c]$ всі такі значення $q < 8$, що підрядок патерна $p[q…q+7]$ збігається з байтом c. Це означає, що всі такі можливі збіги, що байт тексту $T[pos]$ є найлівішим повним байтом, що охоплюється патерном, у рядках 10 та 11 алг. 1 будуть перевірені. Після цієї перевірки вказівник *pos* безпечно (тобто гарантовано без пропуску збігу) переміщується на 1 байт вліво після додавання в рядку 12 та віднімання в рядку 4 наступної ітерації основного циклу.

```
1  pos ← n – 1;
2  repeat
3     repeat
4        pos ← pos – m + 1;
5        while Z[word(T[pos], T[pos+1]) & mask]≠0 do
6           pos ← pos – m + 1;
7        pos ← pos + 1;
8     until Z[word(T[pos], T[pos+1]) & mask]≠0;
9     pos ← pos – 1;
10    foreach q ∈λ[T[pos]] do
11       CheckMatch(q,pos);
12    pos ← pos + m – 2;
13 until pos ≥ m – 1;
```

Алг. 1. Пошуковий цикл алгоритму *RZk-bit*

Тіло процедури CheckMatch наведено в алгоритмі 2. Потрібно перевірити, чи збігається з патерном фрагмент тексту, що починається з q молодших бітів байту $T[pos–1]$ та містить $l–q$ наступних бітів. Функція byte в рядку 2 обтинає двобайтне значення до молодшого байту. Мовою C її можна реалізувати через механізм зведення типів: (unsigned char). Загалом у циклі, наведеному в рядках 2–3, конструюється послідовність байтів на основі частин суміжних байтів тексту. А саме, беруться q молодших бітів байту $T[j]$ та $8 – q$ старших бітів байту $T[j+1]$. Сконструйовані байти порівнюються з байтами патерна. Якщо всі вони збігаються, то останній байт конструюється та порівнюється з $P[m]$ у рядку 5. Оскільки байт $P[m]$ неповний, сконструйований байт обтинається за допомогою маски *lastMask* лише до значущих бітів $P[m]$.

```
1  start ← j ← pos – 1;
2  while j – start < m AND
      byte((T[j] << (8 – q))|(T[j + 1] >> q)) = P[j – start] do
3     j ← j + 1
4  if  j – start = m  then
5     if ((T[j] << (8 – q))|(T[j + 1] >> q))&
                                  lastMask = P[m] then
6        output(start · k + q)
```

Алг. 2. Процедура CheckMatch (q,pos)

Зауважимо, що для коректного завершення роботи алгоритму 1 перед текстом треба дописати так званий «стоп-патерн», а умову в рядку 13 замінити на pos ≥ 0. Для спрощення відповідні операції в алгоритмі не наведено.

3. Передобчислення

Значення *lastMask*, як і інші значення й таблиці, що залишаються фіксованими протягом усієї фази пошуку, визначаються на стадії передобчислень, наведеній в алг. 3. Пояснимо, як конструюється пошукова таблиця Z у циклі в

рядках 7–15 цього алгоритму. Для кожного 16-бітного підрядка патерна, починаючи з позиції i, відповідний елемент масиву Z ініціалізується прапорцем 0, що позначає немаксимальний зсув (короткі суфікси патерна, $l - i \leq 8$, не опрацьовуються, оскільки їхні можливі збіги з префіксами вікна пошуку не впливають на можливість безпечного зсуву довжиною m–1). Під час пошуку перевіряється, чи цей 16-бітний рядок збігається з деяким підрядком тексту з урахуванням того, що рівність деяких бітів не перевіряється через накладання маски (див. рис. 1 у [3]). Функція bitWord, що викликається в рядку 8 алгоритму 3, переставляє біти, як показано на рис. 1,б) у [3], і повертає згаданий підрядок у компактній формі, зображеній у нижній частині цього рисунку. Якщо цей 16-бітний рядок вирівняно за межами двобайтного слова в пам'яті, то така перестановка стає тривіальною і виконується за допомогою операції зведення типів. Саме так працює функція word, яка викликається в алгоритмі 1. Проте в бітовому потоці згаданий підрядок може містити частини 3 послідовних байтів патерна і перестановка ускладнюється. Її виконання описано в коментарях до алгоритму 4.

```
1  mask ← 2^k − 1; m ←⌈ l/8 ⌉;
2  lastMask ← byte(0xFF << (8 − (l mod 8)));
3  foreach i ∈ [0;7] do
4    c ← (P[0] << i) | (P[1] >> (8 − i));
5    λ[c] ← λ[c]∪{i};
6  foreach i ∈ [0;2^k) do Z[i] ← 1;
7  foreach i ∈ [0;l−8) do
8    t ← bitWord(i,mask);
9    if l−i ≥ 16 then
10     Z[t] ← 0;
11   else
12     if l−i > 24−k then s ← 16−(l−i);        // рис. 1 (а)
13     else s ← k − 8;                          // рис. 1 (б)
14     foreach j ∈ [0;2^s) do
15       Z[ t | (j << 8)] ← 0;
```

Алг. 3. Фаза передобчислень алгоритму *RZk-bit*

Підрядок, відформатований функцією bitWord, присвоюється змінній t на кроці 8 алгоритму 3. Якщо $l - i \geq 16$, то весь цей підрядок належить патерну і $Z[t]$=0 (крок 10 алгоритму 3). Інакше підрядок обтинається на правій межі патерна й після перестановки bitWord всередині відформатованого підрядка може утворитися «прогалина» з несуттєвих бітів (рис. 1,а)). Довжина цієї прогалини обчислюється на кроках 12 (рис. 1, а)) або 13 (рис. 1, б)). На кроках 14 і 15 ця прогалина в рядку t заповнюється всіма можливими 2^s значеннями. У такий спосіб формується множина індексів усіх нульових елементів пошукової таблиці Z.

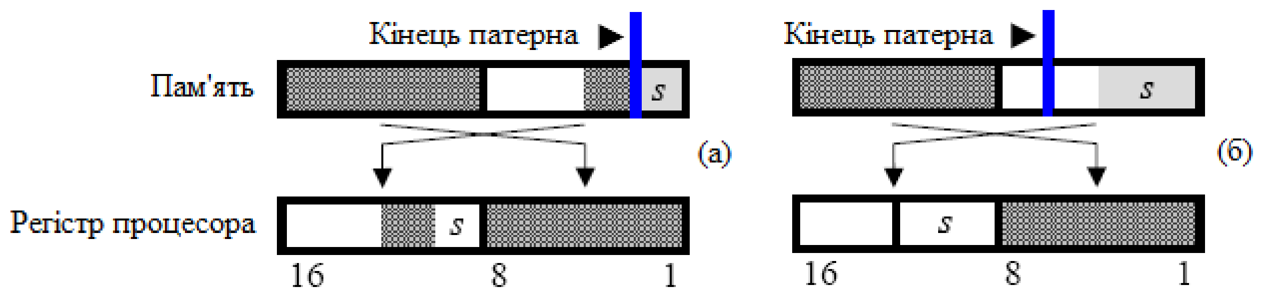

Рис. 1. Перестановка бітів bitWord на правій межі патерна: (а) патерн закінчується на суттєвих бітах правого байта; (б) паттерн закінчується на несуттєвих бітах. Біти, що не відкидаються після перестановки, зафарбовано темно-сірим. Біти, які залишилися б значущими після перестановки, якби не кінець патерна, зафарбовано світло-сірим.

```
1  r ← ⌊ i/8 ⌋;                                 // індекс найстаршого байту
2  s ← (P[r] << 16) | (P[r + 1] << 8) | P[r + 2];  // зчитування 3 байтів з пам'яті в прямому порядку
3  s ← s << (i mod 8);                          // зсув до лівої межі трибайтного слова
4  mask1 ← 0xFF0000;                            // маска для найстаршого байту
5  mask2 ← mask & 0xFF00;                       // маска для середнього байту
6  return ((s&mask1) >> 16) | (s&mask2);        // переміщення найстаршого байту в наймолодший
```

Алг. 4. Функція bitWord(i,mask)

4. Обчислювальні експерименти

Наведені в цій статті алгоритми було реалізовано мовою C++. Також було застосовано до пошукового методу відому методику *вікон, що ковзають* (sliding windows). Кількість таких вікон у табл. 1 вказано після літери w, а число

після літер RZ означає кількість бітів у масці *mask*. Для порівняння було взято найшвидший із відомих алгоритмів бітового пошуку — Binary Faro-Lecroq. Пошук виконувався у файлі обсягом 10 Мб, що являв собою архів англомовних текстів, створений одним із сучасних архіваторів. Результати усереднено за 500 пробігами кожного алгоритму на 40 різних комп'ютерах і нормовано таким чином, щоб значущість усіх комп'ютерів була однаковою. Час виконання найшвидших алгоритмів для кожної довжини патерна зафарбовано сірим.

Довжина патерна	20	40	60	80	100	200	300	400	500
RZ13-Bit	157,65	48,28	34,83	26,03	22,22	15,27	13,17	12,52	12,04
RZ14-Bit	148,76	44,52	32,22	23,74	20,37	13,90	12,14	11,49	11,15
RZ15-Bit	145,22	42,73	31,23	23,10	19,86	13,74	12,12	11,51	11,13
RZ13-Bit-w2	129,49	35,91	26,81	20,88	18,25	13,89	12,29	11,84	11,54
RZ14-Bit-w2	120,65	33,24	24,81	19,07	16,90	13,08	11,78	11,35	11,10
RZ15-Bit-w2	116,87	32,57	24,50	18,96	17,03	13,15	11,94	11,53	11,31
RZ13-Bit-w3	135,61	38,50	28,40	21,96	19,09	13,90	12,25	11,82	11,51
RZ14-Bit-w3	126,74	35,19	26,08	19,79	17,25	12,74	11,43	11,03	10,77
RZ15-Bit-w3	122,25	34,03	25,47	19,44	17,10	12,72	11,41	11,03	10,77
RZ16-Bit-w2	102,02	30,56	23,76	18,91	17,05	13,35	12,26	11,95	11,66
BFL	277,77	91,46	63,80	43,23	36,63	23,38	21,65	25,29	31,32
BFL/найкращий	2,72	2,99	2,68	2,28	2,16	1,83	1,89	2,29	2,68

Табл. 1. Час виконання алгоритмів у бітовому потоці (мілісекунди)

5. Висновки

Як видно з результатів експерименту, різні алгоритми родини RZ-bit є в 1,83 і більше разів швидшими за алг. BFL для всіх досліджених довжин патернів. Алгоритм BFL, своєю чергою, для таких довжин патернів є істотно (принаймні на 30%) швидшим за всі інші нетривіальні алгоритми пошуку патерна в бітовому потоці ([2]). Причинами такої переваги алгоритмів родини RZ-bit є насамперед переваги базових алгоритмів пошуку в 256-розрядних текстах, зазначені в [3], і, крім того, такі фактори:

- хоча максимальна довжина «довгого зсуву» в BFL становить $2m$–1, на практиці цей зсув часто дорівнює $2m$–3 або менше, в той час як будь-який максимальний зсув у двовіконній версії RZ-Bit становить $2m$–2, тобто є довшим у середньому;
- особливо велика різниця в часі виконання RZ-bit і BFL на коротких патернах свідчить про вищу ефективність техніки перевірки збігу в алгоритмах RZ-bit.

Список використаних джерел

1. Klein S. Accelerating Boyer Moore search on binary texts. / S. Klein, M.K. Ben-Nissan // Proceedings of International Conference on Implementation and Application of Automata, CIAA-07, 2007, p. 130–143.

2. Faro S. An efficient matching algorithm for encoded DNA sequences and binary strings / S. Faro, T. Lecroq // Proceedings of Combinatorial Pattern Matching, 2009, p. 106–115.

3. Завадський І.О. Пошук рядка в тексті з урахуванням обмежень на обсяг кеш-пам'яті / І.О. Завадський // Вісник КНУ ім. Т. Шевченка, серія: фіз.-мат. науки, вип. 3, 2019, с. 69–72.

References

1. KLEIN S., BEN-NISSAN M.K. (2007) Accelerating Boyer Moore search on binary texts. *Proceedings of International Conference on Implementation and Application of Automata*, CIAA-07, p. 130–143.

2. FARO S., LECROQ T. (2009) An efficient matching algorithm for encoded DNA sequences and binary strings. *Proceedings of International Conference on Implementation and Application of Automata*, p. 106–115.

3. ZAVADSKYI I.O. (2019) Pattern matching in text with respect to cache memory size. *Bulletin of Taras Shevchenko National University of Kyiv. Series Physics & Mathematics, no. 3, p. 69–72.*

Надійшло у редакції 4.12.2020