Алгоритми та складність

I семестр

Лекція 10

Інші підходи до пошуку підрядків

- В основі вже розглянутих методів пошуку підрядка в рядку лежить порівняння.
- Їх головною елементарною операцією є безпосереднє порівняння двох символів.
- Існують алгоритми, що використовують інші принципи: застосування бітових та арифметичних операцій.
- Ці алгоритми намагаються не зменшити кількість кроків пошуку, а мінімізувати обчислення на кожному з них.

Алгоритм Shift-And

- Також відомий як алгоритм Shift-Or, bitap-алгоритм, двійковий алгоритм пошуку підрядка, алгоритм Baeza-Yates – Gonnet.
- Ідея полягає у використанні швидких бітових операцій при порівнянні.
- Метод ефективний при пошуку невеликих зразків (довжиною з типове англійське слово).

- Нехай маємо зразок P довжини n та текст T довжини m.
- Для обчислень розглядається двійкова (з 0 та 1) матриця M розміром $n \cdot (m+1)$.

Алгоритм Shift-And

- В матриці M індекс i змінюється від 1 до n (довжина шаблону), індекс j – від 0 до m (довжина тексту).
- $M(i,j) = 1 \Leftrightarrow P[1..i] = T[(j-i+1)..j]$
- Тобто перші i символів зразка P співпадають з i символами тексту T, закінчуючи позицією j.
- Решта елементів матриці M нулі.
- Іншими словами, одиниці в рядку i показують всі місця в тексті, де закінчуються копії P[1..i], одиниці в стовпці j відповідно показують всі префікси зразка, що закінчуються в позиції j рядка T.

Алгоритм Shift-And

- Нехай T = «CALIFORNIA», P = «FOR»

			C	A	L	I	F	O	R	N	I	A
		0	1	2	3	4	5	6	7	8	9	10
F	1	0	0	0	0	0	1	0	0	0	0	0
O	2	0	0	0	0	0	0	1	0	0	0	0
R	3	0	0	0	0	0	0	0	1	0	0	0

- $M(n,j) = 1 \Leftrightarrow$ входження P закінчується в позиції j тексту.
- Отримання одиниці в останньому рядку означає розв’язання задачі про точне співпадіння.

Для побудови матриці M потрібно ввести ще дві операції.

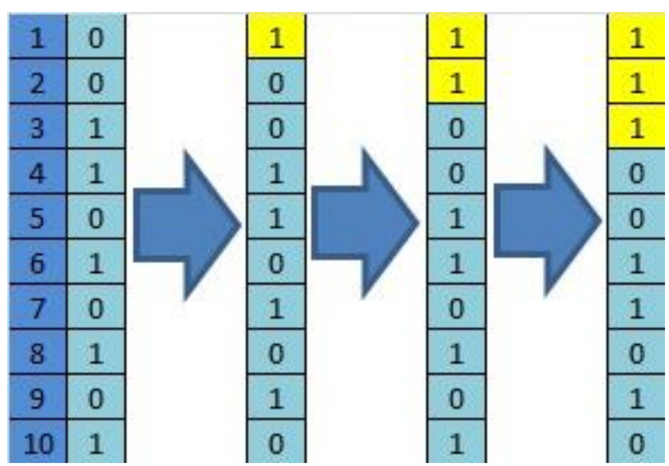
Алгоритм Shift-And

- Введемо характеристичні вектори-маски U(x) для шаблону для кожного символу x, що входить до тексту.
- Одиниці будуть стояти в тих позиціях P, де є символ x.
- Наприклад, при P = «ABACDEAB» отримаємо зокрема вектор U(A) = 10100010:

		1	2	3	4	5	6	7	8
P	=	A	B	A	C	D	E	A	B
U(A)	=	1	0	1	0	0	0	1	0

Алгоритм Shift-And

- Також вводиться операція Bit-Shift(j) – вектор, отриманий зсувом стовпчика j на позицію вниз та дописуванням згори 1; старе значення в позиції n втрачається.
- Іншими словами, вектор Bit-Shift(j) складається з 1, до якої приписано перші (n–1) бітів стовпчика j.
- Приклад трьох послідовних застосувань Bit-Shift до стовпця 0011010101:



Алгоритм Shift-And

- Матриця M обчислюється по стовпцях.
- Нульовий стовпчик містить всі нулі.
- Кожен наступний стовпчик j отримується через попередній та вектор U для символа T(j):

$$M(j) = \text{Bit-Shift}(j–1) \text{ AND } U(T(j))$$

- Тут AND – побітове логічне множення.
- Зберігати всю матрицю не потрібно, достатньо двох стовпчиків.

Алгоритм Shift-And

Приклад: нехай дано шаблон P = «ABAAC» та текст T = «XABXABAAXA».

- M(0) = (0,0,0,0,0)
- U(T(1)) = U(X) = (0,0,0,0,0)
- Shift-And(0) = (1,0,0,0,0)
- M(1) = (1,0,0,0,0) AND (0,0,0,0,0) = (0,0,0,0,0)

			X	A	B	X	A	B	A	A	X	A
		0	1	2	3	4	5	6	7	8	9	10
A	1	0	0									
B	2	0	0									
A	3	0	0									
A	4	0	0									
C	5	0	0									

Алгоритм Shift-And

P = «ABAAC», T = «XABXABAAXA»

- U(T(2)) = U(A) = (1,0,1,1,0)
- Shift-And(1) = (1,0,0,0,0)
- M(2) = (1,0,0,0,0) AND (1,0,1,1,0) = (1,0,0,0,0)

			X	A	B	X	A	B	A	A	X	A
		0	1	2	3	4	5	6	7	8	9	10
A	1	0	0	1								
B	2	0	0	0								
A	3	0	0	0								
A	4	0	0	0								
C	5	0	0	0								

Алгоритм Shift-And

P = «ABAAC», T = «XABXABAAXA»

- U(T(3)) = U(B) = (0,1,0,0,0)
- Shift-And(2) = (1,1,0,0,0)
- M(3) = (1,1,0,0,0) AND (0,1,0,0,0) = (0,1,0,0,0)

			X	A	B	X	A	B	A	A	X	A
		0	1	2	3	4	5	6	7	8	9	10
A	1	0	0	1	0							
B	2	0	0	0	1							
A	3	0	0	0	0							
A	4	0	0	0	0							
C	5	0	0	0	0							

Алгоритм Shift-And

P = «ABAAC», T = «XABXABAAXA»

Врешті-решт отримуємо:

- U(T(10)) = U(A) = (1,0,1,1,0)
- Shift-And(9) = (1,0,0,0,0)
- M(10) = (1,0,0,0,0) AND (1,0,1,1,0) = (1,0,0,0,0)

			X	A	B	X	A	B	A	A	X	A
		0	1	2	3	4	5	6	7	8	9	10
A	1	0	0	1	0	0	1	0	1	1	0	1
B	2	0	0	0	1	0	0	1	0	0	0	0
A	3	0	0	0	0	0	0	0	1	0	0	0
A	4	0	0	0	0	0	0	0	0	1	0	0
C	5	0	0	0	0	0	0	0	0	0	0	0

- Як видно, повних входжень зразка у текст не було знайдено.

Алгоритм Shift-And

- Для довільного $i>1$ справедливо $M(i,j) = 1 \Leftrightarrow P[1..(i-1)] = T[(j-i+1)..(j-1)]$ та $P[i] = T[j]$.
- Перша умова вірна при $M(i-1,j-1) = 1$.
- Друга умова вірна, якщо i-й біт вектора U для символа T[j] дорівнює 1.
- Після зсуву стовпчика $j-1$, елемент $M(i-1,j-1)$ множиться на i-й елемент вектора U(T[j]).
- Отже, елементи матриці M обчислюються коректно.

Алгоритм Shift-And

- Алгоритм має складність O(m*n).
- Побудова масиву U вимагає O(|Σ|*n) операцій та пам’яті. (|Σ| – довжина алфавіту.)
- Дуже швидкий, якщо довжина зразка n не перевищує довжину машинного слова та алфавіт невеликий: оцінки O(m) та O(n+|Σ|) відповідно.
- Не чутливий до поганих вхідних даних.
- Однак не має явної переваги над іншими алгоритмами.
- Легко модифікується для наближеного та множинного пошуку.

Алгоритм Shift-Or

- Варіант підходу Shift-And.
- Оскільки при реальній операції зсуву на вільному місці з'являється 0, ролі 0 та 1 в алгоритмі можна поміняти.
- «Сигнальним» значенням стає 0.
- Операція Bit-Shift' записує вгорі 0.
- W(T(j)) = not U(T(j))
- Тоді значення стовпчика обчислюватиметься як

$$M'(j) = \text{Bit-Shift}'(j-1) \text{ OR } W(T(j))$$

- Формула коректна, оскільки є запереченням виразу, що використовувався для Shift-And.

Наближений пошук рядків

- Наближений (нечіткий) пошук рядків (approximate string matching, fuzzy string searching).
- Допускається «неточне» входження зразка – з кількістю «помилок», що не перевищує задану.
- «Помилка» – неспівпадіння, вставка чи пропуск символа, у порівнянні з шаблоном.
- Алгоритми наближеного пошуку використовуються при перевірці орфографії, в пошукових системах, в біоінформатиці (визначення схожості послідовностей ДНК).

Наближений пошук рядків

- Вводиться «міра близькості» слів – відстань редагування.
- Оцінюється мінімальна кількість примітивних операцій, необхідних для перетворення одного слова в інше.
- Можуть розглядатися різні набори примітивних операцій, яким може призначатися різна вага. Серед них:

Заміна: co**a**t → co**s**t

Вставка: cot → co**a**t

Видалення: co**a**t → cot

Транспозиція: co**st** → co**ts**

Наближений пошук рядків

- Найпростіший варіант – оцінка кількості відмінних відповідних позицій в словах однієї довжини.
- Це – відстань Геммінга (Hamming).
- Приклади відстаней:

"ka**rol**in" та "ka**thr**in": 3

"k**a**r**ol**in" та "k**e**r**st**in": 3

10**1**1**1**01 та 10**0**1**0**01: 2

2**17**3**8**96 та 2**23**3**7**96: 3

Наближений пошук рядків

- Найчастіше під відстанню редагування мають на увазі відстань Левенштейна.
- Найприродніша з метрик; містить примітивні операції заміни, вставки та видалення символу.
- Наприклад, відстань між словами "kitten" та "sitting" дорівнює 3:

kitten => **s**itten (заміна)

sitt**e**n => sitt**i**n (заміна)

sittin => sittin**g** (вставка)

- Якщо додатково допускається перестановка сусідніх символів, отримуємо відстань Дамерау-Левенштейна.

Модифікація Shift-And

- Для наближеного пошуку без індексації найчастіше використовують модифікації алгоритму Shift-And.
- Таку варіацію як один з алгоритмів містить, зокрема, пошукова утиліта agrep.
- Алгоритм працює з відстанню Левенштейна.
- Розглядається сімейство двійкових матриць M^k:

$M^k(i,j) = 1 \Leftrightarrow$ не менше (i–k) з перших i символів зразка P співпадають з i символами у відрізку тексту T, що закінчується позицією j.

- Параметр k визначає допустиму кількість неспівпадінь.

Модифікація Shift-And

- M^0 – матриця, що відповідає масиву M первинного методу.
- Умова $M^k(n,j) = 1$ означає наявність входження зразка P в T, що закінчується в позиції j та має не більше k неспівпадінь.
- Ефективність методу залежить від значення k: чим воно більше, тим повільніше працює алгоритм.
- Для малих k метод дуже швидкий.
- Для кожного наступного символу тексту послідовно обчислюються значення стовпців $M^0(j)$, $M^1(j)$, …, $M^k(j)$.

Модифікація Shift-And

- Обчислення значень $M^0(j)$ залишається незмінним.
- Для різних типів неспівпадінь обраховується своє сімейство $M^1(j),\ldots,M^k(j)$, за своїми умовами.
- Для вставки символу:

$$M^k_{\text{встав}}(j) = (\text{Bit-Shift}(M^k_{\text{встав}}(j-1)) \text{ AND } U(T(j))) \text{ OR } M^{k-1}_{\text{встав}}(j-1)$$

- Для видалення символу:

$$M^k_{\text{видал}}(j) = (\text{Bit-Shift}(M^k_{\text{видал}}(j-1)) \text{ AND } U(T(j))) \text{ OR } \text{Bit-Shift}(M^{k-1}_{\text{видал}}(j))$$

Модифікація Shift-And

- Для заміни символу:

$$M^{k}_{зам}(j) = (\text{Bit-Shift}(M^{k}_{зам}(j-1)) \text{ AND } U(T(j)))$$
$$\text{OR } M^{k-1}_{зам}(j-1)$$

- Перші частини виразів описують ситуацію вже наявності k неточностей при співпадінні поточного символу в тексті з шаблоном.
- Другі частини описують випадок виявлення помилки.
- Результуюче значення по всіх типах неспівпадінь береться через побітовий логічний OR умов:

$$M^{k}(j) = M^{k}_{встав}(j) \text{ OR } M^{k}_{видал}(j) \text{ OR } M^{k}_{зам}(j)$$

Деякі означення теорії чисел

- $d|a$ – d ділить a; a кратне d.
- Якщо $d|a$ та $d \geq 0$, то d є *дільником* a.
- 0 ділиться на всі цілі числа.
- Кожне число a ділиться на *тривіальні дільники* 1 та a. Нетривіальні дільники також називають *множниками*.
- *Просте число*: ціле $a>1$, що має лише тривіальні дільники.
- Ціле $a>1$, яке не є простим – *складене*.
- Число 1 (*одиниця*) не є ні простим, ні складеним.
- Число 0 та всі від'ємні цілі також не є ні простими, ні складеними.

Деякі означення теорії чисел

- Для довільного цілого a та довільного додатного цілого n існує єдина пара цілих q та r, що $0 \le r < n$ та $a = qn + r$.
- $q = [a/n]$ – *частка* від ділення, $r = a \bmod n$ – *остача* від ділення. Тобто $n|a \Leftrightarrow a \bmod n = 0$.
- Відповідно до їх остач по модулю n, всі числа можна розбити на n *класів еквівалентності*, які містять число a:

$$[a]_n = \{a + kn : k \in \mathbb{Z}\}$$

- Наприклад, $[3]_7 = \{\ldots, -11, -4, 3, 10, 17, \ldots\}$
- Інші варіанти позначення цієї ж множини: $[-4]_7$, $[10]_7$.

Деякі означення теорії чисел

- *Множина* всіх таких *класів еквівалентності*:

$$\mathbb{Z}_n = \{[a]_n : 0 \le a \le n-1\}$$

- a $\in [b]_n$ означає a ≡ b (mod n), тобто

(a – b) mod n = 0, або ж

a mod n = b mod n.

- Альтернативний запис

$\mathbf{Z}_n = \{0, 1, \ldots, n-1\}$

підкреслює, що кожен клас представлений найменшим невід'ємним елементом: 0 позначає $[0]_n$, 1 позначає $[1]_n$ і т.д.

- З іншого боку, $-1 \in [n-1]_n$, бо $-1 \equiv n-1 \pmod{n}$.

Деякі означення теорії чисел

- Правила робота з остачами:
  - $(a + b) \bmod n = ((a \bmod n) + (b \bmod n)) \bmod n$
  - $(a - b) \bmod n = ((a \bmod n) - (b \bmod n)) \bmod n$
  - $(a \cdot b) \bmod n = ((a \bmod n) \cdot (b \bmod n)) \bmod n$
- Для всіх цілих x, y: з $d|a$ та $d|b$ => $d|(ax + by)$
- *Найбільший спільний дільник* gcd(a,b) двох цілих чисел a та b, не рівних 0 одночасно, – найбільший зі спільних дільників a та b.
- Якщо a та b – довільні цілі, не рівні 0 одночасно, то величина gcd(a,b) дорівнює найменшому додатному елементу множини $\{ax + by \mid x,y \in \mathbf{Z}\}$ лінійних комбінацій чисел a та b.

Алгоритм Рабіна-Карпа (Rabin-Karp)

- Нехай $\Sigma = \{0,1,2,\ldots,9\}$, тобто кожен символ є десятковою цифрою.
- Для загального випадку можна припустити, що кожен символ – цифра в системі числення з основою $d=|\Sigma|$.
- Розглянемо рядок з k послідовних символів як k-цифрове десяткове число.
- Позначимо p – десяткове значення, що відповідає заданому шаблону.
- Так само t_s – десяткове значення підрядків T[(s+1)..(s+m)] довжини m при s=0,1,…,(n–m).

Алгоритм Рабіна-Карпа

- $t_s = p \Leftrightarrow T[(s+1)..(s+m)] = P[1..m]$.
- Тоді s є допустимим зсувом $\Leftrightarrow t_s = p$.
- Якщо значення p можна обчислити за $\Theta(m)$, а всі значення t_s за загальний час $\Theta(n-m+1)$, то значення всіх допустимих зсувів обчислюється порівнянням p з кожним із t_s за час $\Theta(m)+\Theta(n-m+1) = \Theta(n)$.
- Запис $\Theta(n-m+1)$ символізує, що при n = m час роботи буде $\Theta(1)$.
- Величину p справді можна обчислити за $\Theta(m)$ (схема Горнера):

$$p = P[m] + 10\left(P[m-1] + 10(P[m-2] + \cdots + 10(P[2] + 10P[1])\cdots)\right).$$

Алгоритм Рабіна-Карпа

- Аналогічно Θ(m) обчислюється t_0 з T[1..m].
- Величину t_{s+1} можна обчислити, маючи t_s, за константний час, використавши співвідношення

$$t_{s+1} = 10(t_s - 10^{m-1}T[s+1]) + T[s+m+1] \ .$$

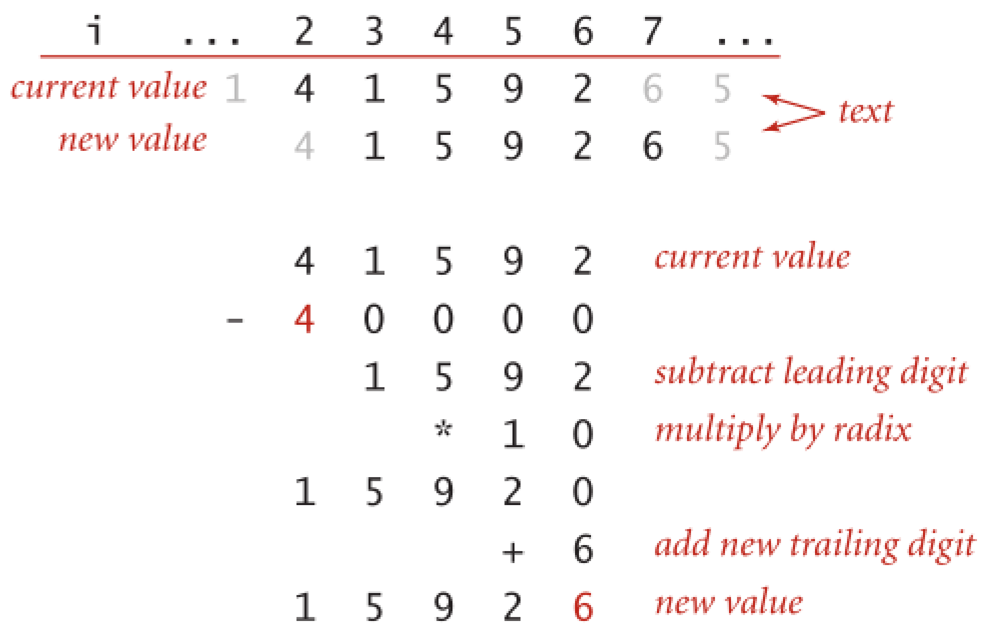
- З величини t_s видалається старша цифра, результат домножується на 10 і додається відповідна молодша цифра.

Алгоритм Рабіна-Карпа

- Наприклад, t_s=31415, m=5 і нова молодша цифра 2:

$$t_{s+1} = 10(31415 - 10000 \cdot 3) + 2$$
$$= 14152 \ .$$

- Або так:



Алгоритм Рабіна-Карпа

- Якщо константа 10^{m-1} вже попередньо обчислена за час O(m), то на обчислення кожного t_{s+1} піде фіксована кількість арифметичних операцій.
- Таким чином визначаються t_1 , t_2 , …, t_{n-m} за час $\Theta(n-m)$.
- Отже, всі входження зразка P[1..m] в текст T[1..n] можна знайти, витративши на передобробку час $\Theta(m)$, а на фазу порівняння $\Theta(n-m+1)$.
- А якщо значення t_s та p виявляться занадто великими?
- Можна проводити обчислення по модулю!

Алгоритм Рабіна-Карпа

- Обчислимо значення p по модулю деякого q за час $\Theta(m)$ та значення усіх t_s по цьому ж модулю – сумарно за час $\Theta(n–m+1)$.
- Якщо взяти q таким простим числом, що 10q поміщається в машинне слово, то обчислення виконаються з використанням арифметики одинарної точності (для розглянутого випадку).
- В загальному випадку для d-символьного алфавіту {0,1,…, (d – 1)} вибирається q так, щоб значення dq поміщалося в машинне слово.
- Рекурентне співвідношення для t_{s+1} слід поправити, щоб воно працювало по модулю q.

Алгоритм Рабіна-Карпа

- Вигляд оновленого рекурентного співвідношення:

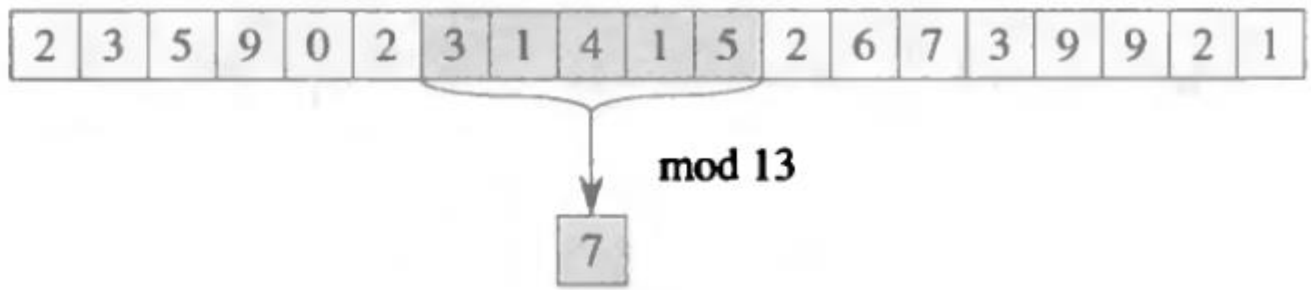
$$t_{s+1} = (d(t_s - T[s+1]h) + T[s+m+1]) \bmod q$$

- Тут $h \equiv d^{m-1} \pmod q$ – значення цифри 1 в старшому розряді вікна розміром m цифр.
- Однак з $t_s \equiv p \pmod q$ не випливає $t_s \equiv p$!
- Але відсутність рівності по модулю гарантує $t_s \neq p$, що дає недопустимість зсуву s.
- Тому перевірка $t_s \equiv p \pmod q$ буде евристичним тестом, що допоможе виключити недопустимі зсуви s.
- У разі виконання співвідношення зсув s необхідно додатково перевірити на допустимість.

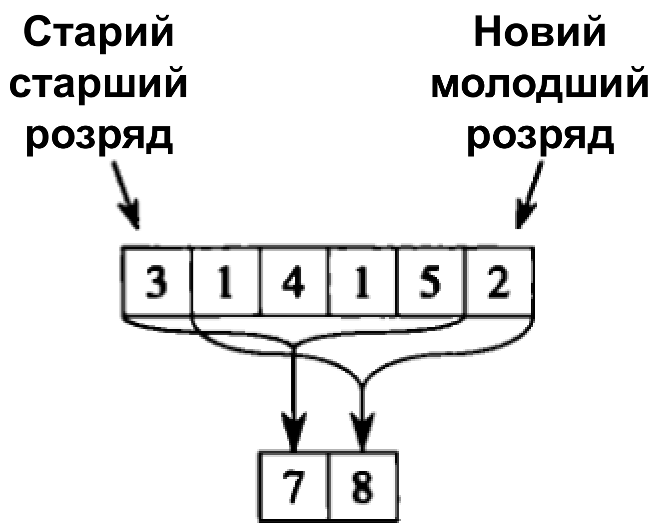
Алгоритм Рабіна-Карпа

- Ситуація, при якій виконується $t_s \equiv p \pmod{q}$, але зсув s є недопустимим, називається *хибним співпадінням*.
- Відрізнити хибне співпадіння від справжнього входження можна безпосередньою перевіркою $P[1..m] = T[(s+1)..(s+m)]$.
- При підборі достатньо великого q висока ймовірність рідкості випадків хибних співпадінь, тому вартість додаткової перевірки виявиться низькою.
- По суті, алгоритм виконує пошук зразка в тексті з використанням хешування.

Алгоритм Рабіна-Карпа



Результат обчислення чисельного значення вікна довжини 5 по модулю 13



Старий старший розряд **Зміщення** **Новий молодший розряд**

$$
\begin{aligned}
14152 &\equiv (31415 - 3 \cdot 10000) \cdot 10 + 2 \pmod{13} \\
&\equiv (7 - 3 \cdot 3) \cdot 10 + 2 \pmod{13} \\
&\equiv 8 \pmod{13}
\end{aligned}
$$

Обчислення чисельного значення поточного вікна за відомим попереднім з обчислень по модулю

Алгоритм Рабіна-Карпа

Пошук зразка «31415» в заданому тексті методом Рабіна-Карпа. Обчислені всі чисельні значення вікон довжини 5. Всі обчислення ведуться за модулем 13.

Приклад наявності істинного входження та хибного співпадіння.

Алгоритм Рабіна-Карпа

Пошук зразка «26535»

i	0	1	2	3	4	
	2	6	5	3	5	
0	2					% 997 = 2
1	2	6				% 997 = (2*10 + 6) % 997 = 26
2	2	6	5			% 997 = (26*10 + 5) % 997 = 265
3	2	6	5	3		% 997 = (265*10 + 3) % 997 = 659
4	2	6	5	3	5	% 997 = (659*10 + 5) % 997 = 613

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
	3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3	
0	3																% 997 = 3
1	3	1															% 997 = (3*10 + 1) % 997 = 31
2	3	1	4														% 997 = (31*10 + 4) % 997 = 314
3	3	1	4	1													% 997 = (314*10 + 1) % 997 = 150
4	3	1	4	1	5												% 997 = (150*10 + 5) % 997 = 508
5		1	4	1	5	9											% 997 = ((508 + 3*(997 - 30))*10 + 9) % 997 = 201
6			4	1	5	9	2										% 997 = ((201 + 1*(997 - 30))*10 + 2) % 997 = 715
7				1	5	9	2	6									% 997 = ((715 + 4*(997 - 30))*10 + 6) % 997 = 971
8					5	9	2	6	5								% 997 = ((971 + 1*(997 - 30))*10 + 5) % 997 = 442
9						9	2	6	5	3							% 997 = ((442 + 5*(997 - 30))*10 + 3) % 997 = 929
10							2	6	5	3	5						% 997 = ((929 + 9*(997 - 30))*10 + 5) % 997 = *match* 613

Алгоритм Рабіна-Карпа

$Rabin\_Karp\_Matcher(T, P, d, q)$

$n \leftarrow length[T]$
$m \leftarrow length[P]$
$h \leftarrow d^{m-1} \mod q$
$p \leftarrow 0$
$t_0 \leftarrow 0$
for $i \leftarrow 1$ to m //Попередня обробка
    do $p \leftarrow (dp + P[i]) \mod q$
        $t_0 \leftarrow (dt_0 + T[i]) \mod q$
for $s \leftarrow 0$ to $n - m$ //Перевірка
    do if $p = t_s$
        then if $P[1..m] = T[s+1..s+m]$
            then print «Зразок знайдено зі зсувом» s
    if $s < n - m$
        then $t_{s+1} \leftarrow (d(t_s - T[s+1]h) + T[s+m+1) \mod q$

- Опис алгоритму. В якості вхідних даних виступають текст T, зразок P, основа системи числення d (яку часто вибирають як |Σ|) та просте число q.

Алгоритм Рабіна-Карпа

- В порівнянні з іншими алгоритмами пошуку підрядка повільно працює в найгіршому випадку.
- Гарно адаптується для множинного пошуку рядків і при цьому матиме зручнішу реалізацію серед інших альтернатив.
- Адаптується також до пошуку в розмірності 2D і більше (наприклад, підматриці в матриці).
- Популярне застосування – перевірка на плагіат.