**Київський національний університет імені Тараса Шевченка**

**Факультет комп'ютерних наук та кібернетики**

**Алгоритми і складність**

**Лабораторна робота 8**

**Звіт**

**Підготував:**

студент групи К-29

Григорович Олег Андрійович

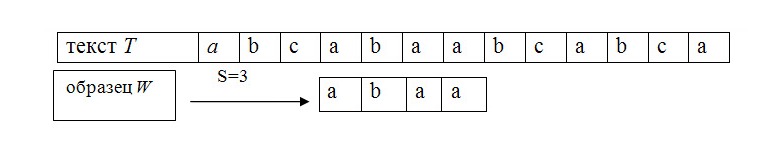
**Київ-2019**

1. **Постановка задачі пошуку в рядку**

Часто доводиться стикатися зі специфічним пошуком, так званим пошуком рядка (пошуком в рядку). Нехай є деякий текст Т і слово (або образ) W. Необхідно знайти перше входження цього слова в зазначеному тексті. Ця дія типова для будь-яких систем обробки текстів. (Елементи масивів Т і W - символи деякого кінцевого алфавіту - наприклад, {0, 1}, або {a, ..., z}, або {а, ..., я}.)

Найбільш типовим додатком такого завдання є документальний пошук: заданий фонд документів, що складаються з послідовності бібліографічних посилань, кожне посилання супроводжується «дескриптором», що вказує тему відповідного посилання. Треба знайти деякі ключові слова, що зустрічаються серед дескрипторів. Міг би мати місце, наприклад, запит «QT» і «C++». Такий запит можна трактувати наступним чином: чи існують статті, які мають дескрипторами «QT» і «C++».

Пошук рядка формально визначається наступним чином. Нехай заданий масив Т з N елементів і масив W з M елементів, причому 0 <M≤N. Пошук рядка виявляє перше входження W в Т, результатом будемо вважати індекс i, який вказує на перший з початку рядка (з початку масиву Т) збіг з образом (словом).

Приклад. Потрібно знайти всі входження зразка W = abaa в текст T = abcabaabcabca.

Образ входить в текст лише один раз з зсувом 3, індекс 4

1. **Алгоритм прямого пошуку**
2. ***Ідея алгоритму:***

1. I = 1,

2. порівняти I-й символ масиву T з першим символом масиву W,

3. збіг → порівняти другі символи і так далі,

4. розбіжність → I: = I + 1 і перехід на пункт 2,

**2. Умова закінчення алгоритму:**

1. поспіль М порівнянь вдалі,

2. I + M> N, тобто слово, не знайдено.

**3. Складність алгоритму:**

Найгірший випадок. Нехай масив T → {AAA ... .AAAB}, довжина │T│ = N, зразок W → {A ... .AB}, довжина │W│ = M. Очевидно, що для виявлення збігу в кінці рядка потрібно виробити близько N \* M порівнянь, тобто O (N \* M).

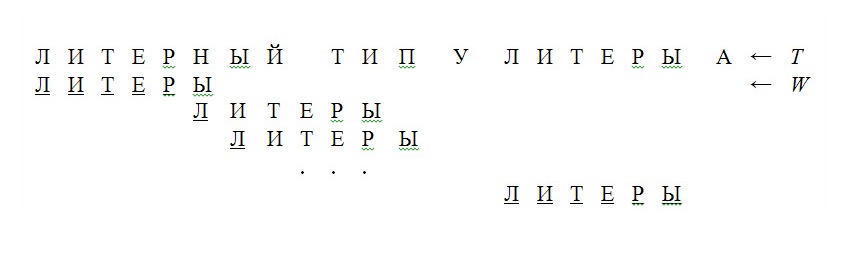
**4. Недоліки алгоритму:**

1. висока складність - O (N \* M), в гіршому випадку - Θ ((N-M + 1) \* M);

2. після розбіжності перегляд завжди починається з першого символу зразка і тому може включати символи T, які раніше вже були видимими (якщо рядок читається з вторинної пам'яті, то такі повернення займають багато часу);

3. інформація про текст T, що отримується при перевірці даного зсуву S, ніяк не використовується при перевірці наступних зрушень.

**2. Алгоритм Д. Кнута, Д. Моріса і В. Пратта (КМП-пошук)**  
  
  
Алгоритм КМП-пошуку фактично вимагає лише близько N порівнянь навіть в найгіршому випадку.  
1. Приклад.  
(Символи, які зазнали порівняння, підкреслені.)



Після часткового збігу початкової частини образу W з відповідними символами рядка Т ми фактично знаємо пройдену частину рядка і може «вирахувати» деякі відомості (на основі самого способу W), за допомогою яких потім швидко просунемося по тексту.

**1. Ідея КМП-пошуку**

при кожному розбіжності двох символів тексту і способу образ зсувається на все пройдене відстань, так як менші зрушення не можуть привести до повного збігу.

**2. Особливості КМП-пошуку:**

1. потрібно близько (N + M) порівнянь символів для отримання результату;

2. схема КМП-пошуку дає справжній виграш тільки тоді, коли невдачі передувало деяке число збігів. Лише в цьому випадку образ зсувається більш ніж на одиницю. На жаль збіги зустрічаються значно рідше ніж розбіжності. Тому виграш від КМП-пошуку в більшості випадків текстів дуже незначний.

**6. Алгоритм Р. Боуер і Д. Мура (БМ-пошук)**

На практиці алгоритм БМ-пошуку найбільш ефективний, якщо зразок W довгий, а потужність алфавіту досить велика.

**1. Ідея БМ-пошуку**

порівняння символів починається з кінця зразка, а не з початку, тобто порівняння окремих символів відбувається справа наліво. Потім за допомогою деякої евристичної процедури обчислюється величина зсуву вправо s. І знову проводиться порівняння символів, починаючи з кінця зразка.

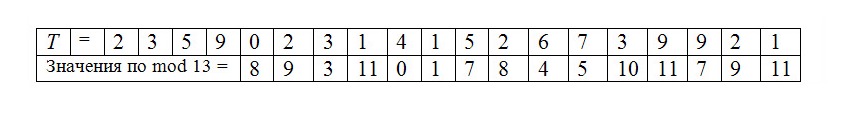
Цей метод не тільки покращує обробку найгіршого випадку, але і дає виграш в проміжних ситуаціях.

У більшості випадків, крім спеціально побудованих прикладів, БМ-пошук вимагає значно менше N порівнянь. У самих же сприятливих обставин, коли останній символ зразка завжди потрапляє на неспівпадаючий символ тексту, число порівнянь одно (N / M), в гіршому ж випадку - О ((N-M + 1) \* M + p), де p - потужність алфавіту .

7. Алгоритм Рабіна-Карпа (РК-пошук)

Нехай алфавіт D = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, тобто кожен символ в алфавіті є d-кова цифра, де d = │D│.

Приклад. Нехай зразок має вигляд W = 3 1 4 1 5

Обчислюємо значення чисел з вікна довжини | W | = 5 по mod q, q - просте число.

23590 (mod 13) = 8, 35902 (mod 13) = 9, 59023 (mod 13) = 9, ...

k1 = 314157 (mod 13) - входження зразка,

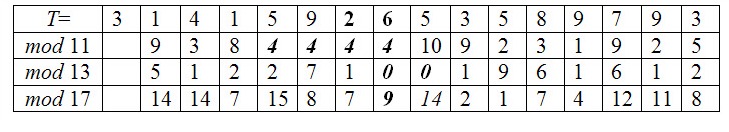
k2 = 673997 (mod 13) - неодружене спрацьовування.

З рівності ki = kj (mod q) не слід, що ki = kj (наприклад, 31415 = 67399 (mod 13), але це не означає, що 31415 = 67399). Якщо ki = kj (mod q), то ще треба перевірити, чи збігаються рядки W [1 ... m] і T [s + 1 ... s + m] насправді.

Якщо просте число q досить велике, то додаткові витрати на аналіз холостих спрацьовувань будуть невеликі.

У гіршому випадку час роботи алгоритму РК - Θ ((N-M + 1) \* M), в середньому ж він працює досить швидко - за час О (N + M).

Приклад: Скільки холостих спрацьовувань k зробить алгоритм РК, якщо

q = 11, 13, 17. Нехай W = {2 6}

26 mod 11 = 4 → k = 3 холостих спрацьовування,

26 mod 13 = 0 → k = 1 холостих спрацьовування,

26 mod 17 = 9 → k = 0 холостих спрацьовувань.

Очевидно, що кількість холостих спрацьовувань k є функцією від величини простого числа q (якщо функція обробки зразка mod q) і, в загальному випадку, від виду функції для обробки зразка W і тексту Т.

1. **Алгоритм Бойєра - Мура – Хорспула**

Алгоритм Бойєра - Мура - Хорспула - алгоритм пошуку підрядка в рядку, спрощений варіант алгоритму Бойєр - Мура.К того ж, вимагає багатьох попередніх обчислень евристика співпала суфікса опускається.

**опис алгоритму**

Алгоритм є модифікацією алгоритму Бойєра - Мура. Ідея алгоритму така.

1. Сканування зліва направо, порівняння в режимі «чорного ящика». Як і в примітивному алгоритмі, поєднується початок тексту і шаблону, проводиться порівняння звичайною процедурою «порівняти ділянки пам'яті». Якщо все символи шаблону збіглися з накладеними символами рядки, значить, подстрока знайдена, і пошук закінчено.

Якщо ж якийсь символ шаблону не збігається з відповідним символом рядка, шаблон зсувається на кілька символів вправо. Ці «кілька» вибираються відповідно до такої евристикою.

2. Змінена евристика стоп-символу. Беремо символ тексту, який опинився над останнім символом шаблону (незалежно від того, де трапилося розбіжність!). На малюнку це «b».

                             ↓ стоп-символ

Текст a b a d b \* \* \* \*

Шаблон a b b a d

Наступна перевірка a b b a d

Зрушуємо шаблон так, щоб під стоп-символом виявилася буква «b» шаблону. Це реалізується за допомогою таблиці зсувів: для кожного символу алфавіту зберігаємо максимально можливий зсув, що не пропускає стоп-символ. Тобто (при нумерації рядків з 1): shift (c) = | needle | -lastpos (c, needle [1 .. | needle | -1]), де lastpos - останнє входження символу в рядок, needle [a .. b] - операція взяття підрядка.

Для шаблону «abbad» таблиця має такий вигляд.

Символ a b (всі решта)

зміщення 1 2 5

Для символів, що не увійшли в шаблон, величина зсуву встановлюється рівною довжині шаблона - 5. Останній символ шаблону при обчисленні таблиці зсувів не розглядається (загрожує зацикленням).

Таблицю зручніше розраховувати, проходячи по всім символам шаблона.