Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Кафедра інтелектуальних інформаційних систем

Алгоритми та складність

Завдання № 8

**Алгоритми пошуку шаблону в рядку**

Виконала студентка 2-го курсу

Групи К-28

Панченко Тетяна Андріївна

2020

**Завдання**

Реалізувати алгоритми пошуку зразка в текстовому рядку: наївний, Хорспула, Боєра-Мура, КПМ та Рабіна-Карпа і порівняйте їх ефективність. Виконайте пошук зразків різної довжини: випадкового бінарного зразка у випадковому бінарному тексті та випадкового слова у природному тексті на цій мові.

**Наївний алгоритм**

**Ідея алгоритму:**

Нехай довжина тексту - N, довжина шаблону - M  
1. нехай ціле i ← 1

1. якщо і-й символ рядка тексту співпадає з першим символом рядка-шаблону

порівняти другі символи і так далі  
інакше і← і+1, повторити п.2

Якщо і+довжина шаблону > довжина слова, шаблон не знайдено, повертаємо -1.  
Якщо х порівнянь підряд були вдалими, то алгоритм завершується успішно (де х - довжина шаблону)

**Складність:**

Найгірший випадок: текст виду {aaaa….aaab} і шаблон виду {aa…ab}.

Так як перевіряється кожен символ тексту і при співпадінні перевіряється увесь рядок шаблону, складність Θ((N-M+1)\*M).

**Переваги алгоритму:**

Легкість реалізації.

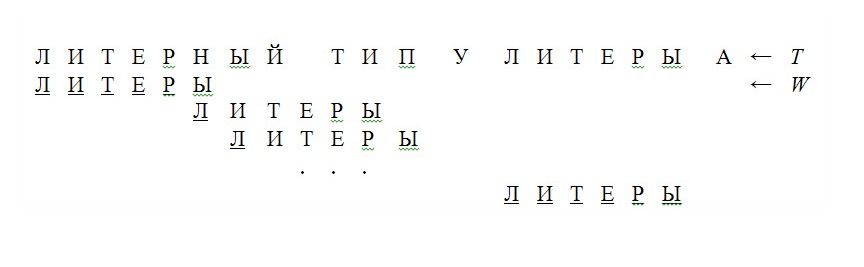
**Недоліки алгоритму:**

Після невдалої перевірки на співпадіння зсув обнуляється, тому виконуються зайві перевірки символів, які вже переглядались. Інформація про текст, що отримується при перевірці поточного зсуву, ніяк не використовується при наступних перевірках.

**Алгоритм Д. Кнута, Д. Мориса і В. Пратта (КМП)**

**Ідея КМП-пошуку**

При кожному неспівпаданні двох символів тексту і шаблону, зсув відбувається на усю пройдену відстань, так як менші зсуви не приведуть до повного збігу.

Приклад: порівнюються лише підкреслені символи  


**Переваги алгоритму:**

Завдяки талиці часткових збігів алгоритм потребує всього близько N+M порівнянь навіть в найгіршому випадку, де N - довжина тексту, M - довжина шаблону

**Недоліки алгоритму:**

Алгоритм дійсно дає виграш у швидкості лише тоді, коли невдалому порівнянню передувала певна кількість співпадінь. Лише в такому випадку рядок зсувається більш, ніж на одиницю. На жаль, співпадання зустрічаються значно рідше, ніж неспівпадання, тому зазвичай виграш від алгоритму досить незначний.

**Складність:**

Через те, що обидві складові алгоритму(побудова таблиці та сам пошук) мають складності, відповідно, O(k) і O(n), складність всього алгоритму становить O(N + M), де M - довжина шаблону, а N - довжина тексту

**Алгоритм Р. Боєра и Д. Мура (БМ-пошук)**

**Ідея БМ-пошуку**

Наївний алгоритм переміщує шаблон по тексту один за іншим. Алгоритм KMP виконує попередню обробку шаблону, щоб його можна було змістити більш ніж на одиницю. Алгоритм Бойера Мура виконує попередню обробку з тієї ж причини. Порівняння символів починається з кінця шаблону, тобто справа наліво. За допомогою евристичної процедури обчислюється зсув вправо і знову відбувається порівняння символів, починаючи з кінця шаблону.

Цей метод не лише покращує обробку найгіршого випадку, але й дає виграш у проміжних ситуаціях.

**Переваги алгоритму:**

Ціною деякої кількості [попередніх обчислень](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D0%BE%D1%80" \o "Препроцесор) над шаблоном, він порівнюється з вихідним текстом не у всіх позиціях — частина перевірок пропускаються як такі, що не дадуть результату. Тому майже завжди БМ-пошук потребує значно менше N порівнянь. Цей алгоритм швидко працює у ситуаціях коли шаблон набагато коротший від тексту пошуку, або коли відбувається пошук в декількох документах. Зазвичай, чим довше шаблон, тим швидше працює алгоритм.

**Недоліки алгоритму:**

На великих алфавітах (наприклад, Юнікод) таблиця стоп-символів може займати багато пам'яті. На коротких текстах вигр\*аш не виправдає попередніх обчислень.

**Складність:**  В найкращому випадку (коли символ шаблону завжди попадає на непідхожий символ тексту) число порівнянь дорівнює N/M, в найгіршому випадку - О((N-M+1)\*M+ p), где p – потужність алфавіту.

**Алгоритм Хорспула**

Алгоритм Хорспула є спрощенням алгоритму Боєра-Мура.

В 1980 році Хорспул запропонував використовувати лише зсув по самому правому символу для обчислення зсуву в алгоритміці Боєра-Мура.

Отриманий Алгоритм має квадратичну складність в найгіршому випадку, але було доведено, що середнє число порівнянь на символ тексту знаходиться між 1/n і 2/(s+1)

**Ідея алгоритму**

1. Сканування відбувається зліва направо, порівняння як в наївному алгоритмі, поєднуються початок тексту і шаблону. Якщо всі символи шаблону співпали з накладеними символами рядку, значить, підрядок знайдено і пошук закінчено.

Якщо якийсь символ шаблону не співпадає с відповідним символом рядку, шаблон зміщається на декілька символів вправо. Зсув визначається евристикою.

1. Модифікована евристика стоп-символу.

Беремо символ тексту, який опинився над останнім символом шаблону (незалежно від того, де відбулось неспівпадіння). В цьому прикладі це “b”:

↓ стоп-символ

Текст a b a d b \* \* \* \*

Шаблон a b b a d

Наступна перевірка a b b a d

Зсуваємо шаблон так, щоб під стоп-символом виявилась буква «b» шаблону. Це реалізовується 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]), где lastpos — останнє входження символа в рядок, needle[a..b] — операция взяття підрядка.

Для шаблону «abbad» таблиця має наступник вигляд:

Символ a b (решта символів)

Зсув 1 2 5

Для символів, що не увійшли в шаблон, величина зсуву встановлюється рівною довжині шаблону - 5.

Останній символ шаблону при обчисленні таблиці зсувів не розглядається (бо є ризик зациклитись).

**Алгоритм Рабіна-Карпа (РК-пошук)**

**Теорія**

В алгоритмі Рабина-Карпа ми створимо хеш нашого шаблону, який ми шукаємо, і перевіряємо, чи відповідає зсувний хеш нашого тексту шаблоном чи ні. Якщо не відповідає, ми можемо гарантувати, що шаблону не існує в тексті. Якщо він відповідає, шаблон може бути присутнім в тексті. Давайте подивимося на приклад:

Нехай дано рядок s[0...n−1]. Тоді поліноміальним хешем(англ. polynomial hash) s називається число , де p — деяке просте число, а s[i] − код i-ого символу рядка s.

Тоді формула для  : 

Проблему переповнення при обчисленні хешу можна вирішити обрахунком хешу по модулю.

**Алгоритм**

Алгоритм проходить зліва направо по 1 символу і порівнює хеш-значення паттерну з поточним хеш-значенням «підтексту». Якщо співпадають хеш-значення, тільки тоді він починає порівнювати символи.

Вирахуємо значення хешу, порівняємо їх, якщо не співпадають, тоді переходимо до перевірки наступної частинки тексту і так, поки не знайдемо всі співпадіння.

Aлгоритм Рабина - Карпа використовується для перевірки плагіату, так само що він ефективний для багаторядкового пошуку і пошук по безліч шаблонів і що потрібно робити пре підготовку тексту (вирізати все знаки пунктуації). Так само, що можна спокійно використовувати свої хеш функції.

**Складність алгоритму:** Для цього алгоритму середня і найкраща складність це О(n). В найгіршому випадку має складність О(n\*m)

**Переваги алгоритму:**

Ціною деякої кількості [попередніх обчислень](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%B5%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D0%BE%D1%80" \o "Препроцесор) над шаблоном, він порівнюється з вихідним текстом не у всіх позиціях — частина перевірок пропускаються як такі, що не дадуть результату. Тому майже завжди БМ-пошук потребує значно менше N порівнянь. Цей алгоритм швидко працює у ситуаціях коли шаблон набагато коротший від тексту пошуку, або коли відбувається пошук в декількох документах. Зазвичай, чим довше шаблон, тим швидше працює алгоритм.

**Недоліки алгоритму:**

Можна підібрати вхідні дані так, що кількість порівнянь буде неприпустимо великою.

**Складність:**  Підрахунок хешей відбувається за O(m), кожна ітерація відбувається за О(1), в циклі їх всього n-m+1. Як результат, час роботи алгоритму О(n+m).

**Мова реалізації алгоритмів**

Java

**Основні модулі програми**

Наївний алгоритм:

static int naive(String str, String template)

Алгоритм Рябіна-Карпа:

private static int RK\_search(String str, String template)

Алгоритр КМП:

private static void computeKMPTable(String str, ArrayList<Integer> T)

private static int kmpSearch(String str, String template)

Алгоритм Хорспула:

static int Horspool(String source, String template)

Алгоритм Боєра-Мура:

private static void badCharHeuristic( String template, int[] badchar)

private static int BoyerMoore(String source, String template)

**Інтерфейс користувача**

Вхідні дані беруться з консолі, результат (позиція, де знайдено шаблон) виводиться туди ж.

**Тестові приклади**

String str1 = "THIS IS A TEST TEXT";  
String str2 = "TEXT";

Result: 15

String str1 = "abcdcbcaa";

String str2 = "cbc"

Result: 4

String str1 = "abcdbcaa";

String str2 = "cbc"

Result: -1

**Використані джерела**

//http://algolist.ru/search/esearch/horspool.php

//https://habr.com/ru/post/116725/

//https://github.com/veysiertekin/SortingAndPatternMatchingAlgorithms/blob/master/src/com/hw02/algorithm/Horspool.java

<http://algolist.ru/search/esearch/index.php>

<https://www.cs.helsinki.fi/u/tpkarkka/opetus/13s/spa/lecture05.pdf>

<https://riptutorial.com/ru/algorithm/example/24653/%D0%B2%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5-%D0%B2-%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC-%D1%80%D0%B0%D0%B1%D0%B8%D0%BD%D0%B0-%D0%BA%D0%B0%D1%80%D0%BF%D0%B0>

<https://ru.qwe.wiki/wiki/Rabin%E2%80%93Karp_algorithm>

https://neerc.ifmo.ru/wiki/index.php?title=%D0%9F%D0%BE%D0%B8%D1%81%D0%BA\_%D0%BF%D0%BE%D0%B4%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%B8\_%D0%B2\_%D1%81%D1%82%D1%80%D0%BE%D0%BA%D0%B5\_%D1%81\_%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%D0%BC\_%D1%85%D0%B5%D1%88%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F.\_%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC\_%D0%A0%D0%B0%D0%B1%D0%B8%D0%BD%D0%B0-%D0%9A%D0%B0%D1%80%D0%BF%D0%B0