Київський національний університет імені Тараса Шевченка

Факультет комп’ютерних наук та кібернетики

Кафедра інформаційних систем

**Алгоритми та складність**

**Завдання №8**

“алгоритми пошуку зразка в текстовому рядку ”

Виконав:

студент групи К-29

Печкуров Віталій Михайлович

**Київ-2018**

**Умова завдання:**

Реалізуйте 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.

**Вирішення завдання:**

**Наївний алгоритм пошуку**

Алгоритм послідовного пошуку є стандартним алгоритмом для всіх стандартних бібліотек мов програмування, та багатьох текстових редакторів. Він є найпростішим і найбільш неефективним. Принцип його роботи полягає у послідовному проходженні по тексту з кроком в 1 символ і порівнянні всіх підслів з шуканим словом. При цьому виконується дуже [велика кількість порівнянь](http://refua.in.ua/konkurenciya.html), більшість з яких є зайвими.

**Алгоритм Рабіна-Карпа**

Алгоритм Рабіна є модифікацією алгоритму послідовного пошуку. Прохід по слову відбувається аналогічним способом. Різниця полягає тільки в тому, що при проході на словах фіксується деяка числова функція і тоді все завдання зводиться до порівняння отриманих чисел, що є швидше. Якщо значення цієї функції на слові, пошук якого виконується і фрагменті тексту різні, то збігу немає, а [якщо однакові](http://refua.in.ua/principova-shema-elektroprivodu-okremih-linij-konveyera-ris-2.html), то тільки тоді вже необхідно виконувати перевірку послідовного збігу по буквах. Даний алгоритм хоч і є ефективнішим та швидшим за послідовний алгоритм, але поступається по цим показникам алгоритму КМП та Бойєра-Мура.

**Алгоритм Бойєра-Мура**

Алгоритм Бойєра-Мура вважається найшвидшим і найефективнішим серед алгоритмів призначених для пошуку підрядка в рядку. При його використанні будується таблиця зміщень для шуканого слова (зразка). Далі поєднуються початок рядка і шуканого слова та розпочинається перевірка з останнього символу шуканого слова. Якщо він не співпадає, то зразок зміщується щодо рядка на величину, отриману з таблиці зміщень, і знову проводиться порівняння, починаючи з останнього символу зразка. Якщо ж символи збігаються, проводиться порівняння передостаннього символу і т.д. Якщо всі символи шуканого слова збіглися з накладеними символами рядка, то шукане слово знайдено і пошук завершено. Даний алгоритм є найшвидшим і найефективнішим для великих текстів. У випадку, коли пошук проводиться в невеликому тексті і розмір шуканого слова невеликий, то його ефективність трохи знижується і у такому випадку краще використовувати алгоритм КМП.

**Алгоритм Кнута-Морріса-Пратта (КМП)**

Алгоритм Кнута-Морріса-Пратта (КМП) є одним з найефективніших алгоритмів пошуку пiдрядка в рядку. Він виконує передобробку шуканого рядка, а саме, на його основі створюється префікс-функція. Наприклад, для пiдрядка abcWordabc таким підрядком є abc (також він одночасно є і префіксом, і суфіксом). Переваги використання префікс функції у тому, що можна відкинути свідомо невірні варіанти, тобто якщо при пошуку збіглося 4 символа, на наступній ні, то має сенс продовжувати перевірку не з другого, а з п’ятого символу. Час роботи даного алгоритму залежить [тільки від розміру тексту](http://refua.in.ua/analiz-uroku-z-informatiki.html), у якому виконується пошук. Даний алгоритм є дуже швидким, повільнішим лише за алгоритм Бойєра-Мура, однак алгоритм КМП є швидшим при невеликих розмірах текстів. Тому, алгоритм Кнута-Морріса-Пратта можна використовувати як універсальний, коли невідомі розміри текстів і фрагментів текстів, які потрібно знайти.

**Алгоритм Бойєра-Мура-Хорспула**

Алгоритм Бойєра-Мура-Хорспула - [алгоритм пошуку рядка](https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%BF%D0%BE%D1%88%D1%83%D0%BA%D1%83_%D1%80%D1%8F%D0%B4%D0%BA%D0%B0) - спрощений варіант [алгоритму HYPERLINK "https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC\_%D0%91%D0%BE%D1%8F%D1%80%D0%B0-%D0%9C%D1%83%D1%80%D0%B0"Бойєра HYPERLINK "https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC\_%D0%91%D0%BE%D1%8F%D1%80%D0%B0-%D0%9C%D1%83%D1%80%D0%B0"- HYPERLINK "https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC\_%D0%91%D0%BE%D1%8F%D1%80%D0%B0-%D0%9C%D1%83%D1%80%D0%B0"Мура](https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%91%D0%BE%D1%8F%D1%80%D0%B0-%D0%9C%D1%83%D1%80%D0%B0). АБМХ працює краще алгоритму Бойєра-Мура на випадкових текстах.

Спочатку будується таблиця зміщень для шуканого [шаблону](https://uk.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD). Поєднується початок тексту (рядки) і шаблона, перевірка починається з останнього символу шаблону.

Якщо останній символ шаблону і відповідний йому при накладенні символ рядка не збігаються, то зразок зрушується щодо рядка на величину, отриману з таблиці зміщень. Причому символ береться з рядка (а не з шаблону), відповідний зсув знаходиться в таблиці. Проводиться зрушення і знову починається перевірка з останнього символу.

Якщо ж символи збігаються, проводиться порівняння передостаннього символу шаблону і т. д. Якщо всі символи шаблону збіглися з накладеними символами рядки, значить, підрядок знайдено, і пошук закінчено. Якщо ж якийсь (не останній) символ шаблону не збігається з відповідним символом рядка, шаблон зсувається на один символ вправо, і перевірка знову починається з останнього символу. Весь алгоритм виконується до тих пір, поки або не буде знайдено входження шуканого зразка, або не буде досягнуто кінець рядка.

**Висновок:**

Таким чином, можна зробити висновок, що кожен з алгоритмів має як свої плюси так і мінуси. Так реалізація алгоритму може бути дуже простою, але і його ефективність буде доволі поганою. Тому, вибір того чи іншого алгоритму пошуку є доволі важливим і відповідальним завданням.

**Модулі програми**

* void Rabina\_Karpa()

Функція, де реалізований алгоритм Рабіна-Карпа

* vector<int> naïve(string p, string t)

Функція, де реалізований наївний алгоритм пошуку

* bool compare(string::const\_iterator a, const string::const\_iterator b, int num)

Функція для порівняння

* int Horspool(const string &t, const string &s)

Функція, де реалізований алгоритм Хорспула

* int KMP(const string &t, const string s)

Функція, де реалізований алгоритм Кнута-Морріса-Прата

* int Boyer-Moore(const string &t, const string &s)

Функція, де реалізований алгоритм Боєра-Мура