**Київський національний університет імена Тараса Шевченка  
Факультет комп’ютерних наук та кібернетики**

**Алгоритми та складність  
Лабораторна робота №8**

**Алгоритми знаходження підрядка**

**Звіт**

**Підготував:**студент групи К-29  
Дацюк Віталій Олегович

**Київ-2019**

1. **Постановка 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.

1. **Опис наївного алгоритму.**

1. I=1,

2. порівняти I-й символ масива S1 с першим символом масива S2,

3. співпадіння → порівняти другі символи і так далі,

4. неспівпадіння → I:=I+1 и перехід на пункт 2.

Найгіршим випадком буде:

S1[] = "AAAAAAAAAAAAAAAAAB";

S2[] = "AAAAB";

**Складність :** O(n\*m)

1. **Опис алгоритму Хорспула.**

Спочатку будується таблиця зміщень для шуканого шаблону. 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 знайдена, 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.

Кажуть, що така модифікація алгоритму Бойєр-Мура справляється з випадковими текстами краще, ніж сам вихідний алгоритм. Однак, оцінка складності в гіршому випадку у нього все-таки гірше. Але на відміну від вихідного алгоритму, тут не потрібно ніяких складних обчислень - будується тільки таблиця зсувів, що не так складно для реалізації.

**Складність :** O(n\*m)

1. **Опис алгоритму Боєра-Мура**

Алгоритм заснований на трьох ідеях.

1. Сканування зліва направо, порівняння справа наліво. Поєднується початок тексту (рядки) і шаблону, перевірка починається з останнього символу шаблону. Якщо символи збігаються, проводиться порівняння передостаннього символу шаблону і т. д. Якщо всі символи шаблону збіглися з накладеними символами рядка, значить, підрядок знайдений, і пошук закінчено.

Якщо ж якийсь символ шаблону не збігається з відповідним символом рядка, шаблон зсувається на кілька символів вправо, і перевірка знову починається з останнього символу.

Ці «декілька», згадані в попередньому абзаці, обчислюються за двома евристиками.

2. Евристика стоп-символу. Припустимо, що ми проводимо пошук слова «колокол». Перша ж буква не збіглася — «к» (назвемо цю букву стоп-символом). Тоді можна зсунути шаблон вправо до останньої його букви «к».

Стрічка: \* \* \* \* \* \* к \* \* \* \* \*\*

Шаблон: к о л о к о л

Наступний крок: к о л о к о л

Якщо стоп-символу в шаблоні взагалі немає, шаблон зміщується за цей стоп-символ.

Стрічка: \* \* \* \* \* а л \* \* \* \* \* \* \* \*

Шаблон: к о л о к о л

Наступний крок: к о л о к о л

В даному випадку стоп-символ — «а», і шаблон зсувається так, щоб він виявився прямо за цією буквою. В алгоритмі Бойера-Мура евристика стоп-символу взагалі не дивиться на співпавший суфікс (див. Нижче), так що перша буква шаблону («к») опиниться під «л», і буде проведена одна завідома холоста перевірка.

Якщо стоп-символ «к» опинився за іншою буквою «к», евристика стоп-символу не працює.

Стрічка: \* \* \* \* к к о л \* \* \* \* \*

Шаблон: к о л о к о л

Наступний крок: к о л о к о л ?????

У таких ситуаціях виручає третя ідея АБМ — евристика співпавшого суфікса.

**Евристика співпавшого суфікса.**

Якщо при порівнянні рядка і шаблону збіглося один або більше символів, шаблон зсувається в залежності від того, який суфікс збігся.

Стрічка: \* \* т о к о л \* \* \* \* \*

Шаблон: к о л о к о л

Наступний крок: к о л о к о л

В даному випадку збігся суфікс «ок», і шаблон зсувається вправо до найближчого «окол». Якщо підрядка «окол» в шаблоні більше немає, але він починається на «кол», зрушується до «кол», і т. д.

Обидві евристики вимагають попередніх обчислень — залежно від шаблону пошуку заповнюються дві таблиці. Таблиця стоп-символів за розміром відповідає алфавіту (наприклад, якщо алфавіт складається з 256 символів, то її довжина 256); таблиця суфіксів — шуканому шаблону. Саме через це алгоритм Бойера-Мура не враховує співпавший суфікс і неспівпавший символ одночасно — це вимагало б занадто багато попередніх обчислень.

Опишемо докладніше обидві таблиці.

**Таблиця стоп-символів**

У таблиці стоп-символів вказується остання позиція в needle (виключаючи останню букву) кожного з символів алфавіту. Для всіх символів, що не увійшли вneedle, пишемо 0 (для нумерації з 0 — відповідно, −1). Наприклад, якщо needle=«abcdadcd», таблиця стоп-символів буде виглядати так.

Символ a b c d [всі інші]

Остання позиція 5 2 7 6 0

Зверніть увагу, для стоп-символу «d» остання позиція буде 6, а не 8 — остання буква не враховується. Це відома помилка, що приводить до неоптимальності. Для АБМ вона не фатальна («витягує» евристика суфікса), але фатальна для спрощеної версії АБМ — алгоритму Хорспула.

Якщо розбіжність сталася на позиції i, а стоп-символ c, то зсув буде i-StopTable[c].

**Таблиця суфіксів**

Для кожного можливого суфікса S шаблону needle вказати найменшу величину, на яку потрібно зрушити вправо шаблон, щоб він знову збігся з S. Якщо такий зсув неможливий, ставиться |needle| (в обох системах нумерації). Наприклад, для того жneedle=«abcdadcd» буде:

Суфікс [пустий] d cd dcd ... abcdadcd

Зсув 1 2 4 8 ... 8

Ілюстрація

було ? ?d ?cd ?dcd ... abcdadcd

стало abcdadcd abcdadcd abcdadcd abcdadcd ... abcdadcd

Якщо шаблон починається і закінчується однією і тією ж комбінацією букв, |needle| взагалі не з'явиться в таблиці. Наприклад, для needle=«колокол» для всіх суфіксів (крім, звичайно, порожнього) зсув буде дорівнювати 4.

Суфікс [пустий] л ол ... олокол колокол

Зсув 1 4 4 ... 4 4

Ілюстрація

було ? ?л ?ол ... ?олокол колокол

стало колокол колокол колокол ... колокол колокол

Існує швидкий алгоритм обчислення таблиці суфіксів. Цей алгоритм використовує префікс-функцію рядка.

m = length(needle)

pi[] = префікс-функція(needle)

pi1[] = префікс-функція(звернення(needle))

for j=0..m

suffshift[j] = m - pi[m]

for i=1..m

j = m - pi1[i]

suffshift[j] = min(suffshift[j], i - pi1[i])

Тут suffshift[0] відповідає всій стрічці яка збіглася; suffshift[m] — пустому суфіксу. Оскільки префікс-функція обчислюється за *O*(|*needle*|) операцій, обчислювальна складність цього кроку також дорівнює *O*(|*needle*|).

**Складність:** O(n+m)

1. **Опис алгоритму Кнута-Морріса-Пратта**

Алгоритм повинен знайти початковий індекс m рядка W[] в рядку S[].

Найпростіший алгоритм пробігає по всьому рядку S[m], де m — індекс. Якщо індекс m досягне кінця рядка, то W[] не знайдено і алгоритм поверне результат «fail». На кожній позиції перевіряється рівність елемента на позиції m з S[] й елемента на першій позиції з W[], тобто S[m] =? W[0]. Якщо вони рівні, то алгоритм перевіряє наступні відповідні елементи в рядках за індексом i. Алгоритм перевіряє всі вирази S[m+i] =? W[i]. Якщо всі елементи з W знайдені, то алгоритм поверне позицію m.

Зазвичай, пробна перевірка відкидає можливість збігу. Якщо рядки складаються з рівномірно розподілених елементів, то шанс, що перші елементи дорівнюють один одному, буде 1 до 26. Отже, в більшості випадків пробна перевірка відкидатиме початкові елементи. Шанс, що перші два елементи будуть рівними, дорівнює 1 до 262 (тобто, 1 до 676). Тобто, якщо елементи рівномірно розподілені, очікувана складність пошуку в рядку S[] довжини k буде порядку k порівнянь або O(k). Якщо S[] має 1.000.000.000 елементів і W[] має 1000 елементів, то пошук рядка займе приблизно 1.000.000.000 порівнянь.

Проте очікувана продуктивність не є гарантованою. Якщо рядки не випадкові, то на кожному кроці m може знадобитися багато порівнянь. В найгіршому випадку два рядки збігаються майже за всіма літерами. Якщо рядок S[] має 1.000.000.000 елементів, що рівні A і рядок W[] складається з 999 елементів A і останній елемент B. Тоді найпростіший алгоритм на кожному кроці виконуватиме 1000 перевірок, а всіх перевірок буде 1 трильйон. Отже, якщо довжина W[] — n, то в найгіршому випадку складність дорівнюватиме O(k⋅n).

Алгоритм КМП має кращий показник продуктивності у найгіршому випадку. КМП витрачає небагато часу (за порядком розміру W[], O(n)) на попереднє обчислення таблиці, і потім використовує таблицю для швидкого пошуку рядка за час O(k).

З іншого боку, на відміну від попередньо розглянутого простого алгоритму, алгоритм КМП використовує інформацію про попередні порівняння. У прикладі, що наведений вище, коли KMП зустрічає незбіг на 1000-ному елементі (i = 999), тобто S[m+999] ≠ W[999], КМП знатиме, що 999 позицій вже перевірено. КМП використовує ці знання у попередньо обчисленій таблиці і додаткових змінних. Коли KMП знаходить незбіг, з таблиці визначається, наскільки збільшиться змінна m.

**Складність:** O(n+m)

1. **Опис алгоритму Рабіна-Карпа**

Для простоти припустимо, що алфавіт складається з десяткових цифр Σ = {0,1,…,9}. (В загальному випадку можна припустити, що кожний символ — це цифра в системі числення з основою d, де d = |Σ|.) Після цього, рядок з k символів, можна розглядати як число довжини k. Тобто символьний рядок «12345» відповідає числу 12345.

Для заданого зразка P[1..m] позначимо через p відповідне йому десяткове значення. Аналогічно, для заданого тексту T[1..n] позначимо через ts десяткове значення підрядка T[s+1..s+m] довжини m при s = 0,1,…,n-m. Очевидно, що ts=p тоді і тільки тоді, коли T[s+1..s+m]=P[1..m]; таким чином, s — допустимий зсув тоді і тільки тоді, коли ts=p.

Якщо значення p можна обчислити за Θ(m) а значення ts за сумарний час Θ(n-m+1), то усі допустимі зсуви можна було б знайти за час Θ(m) + Θ(n-m+1) = Θ(n) шляхом порівняння p з кожним з можливих ts. (Покищо до уваги не береться той факт, що величини p і ts можуть виявитись дуже великими.)

З допомогою схеми Горнера величину p можна обчислити за час Θ(m):

p=P[m]+10(P[m-1]+10(P[m-2]+…+10(P[2]+10P[1]))…)).

Значення t0 можна обчислити з масиву T[1..n] аналогічним способом за час Θ(m). В той же час, знаючи величину ts величину ts+1 можна обчислити за фіксований час:

ts+1 =10(ts -10m-1\*T[s+1])+T[s+m+1].

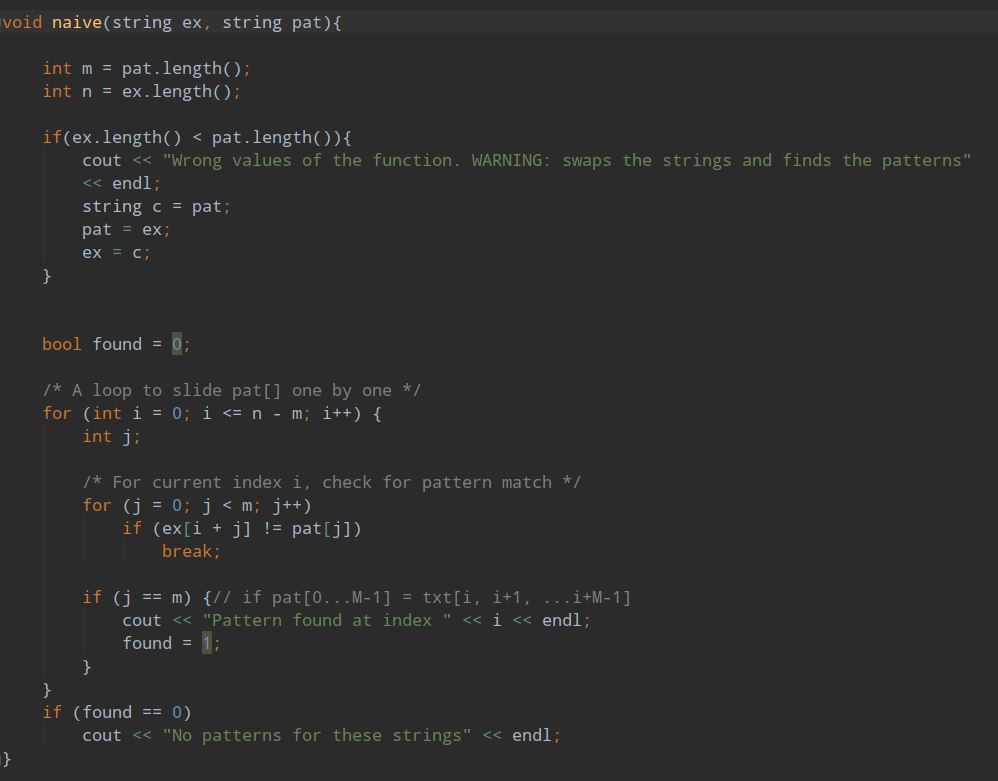
**Алгоритм полягає в наступному:**

* обчислити число p;
* обчислити всі ts;
* Для тих s для яких ts =p, виконати перевірку P[1..m] = T[s+1..s+m].

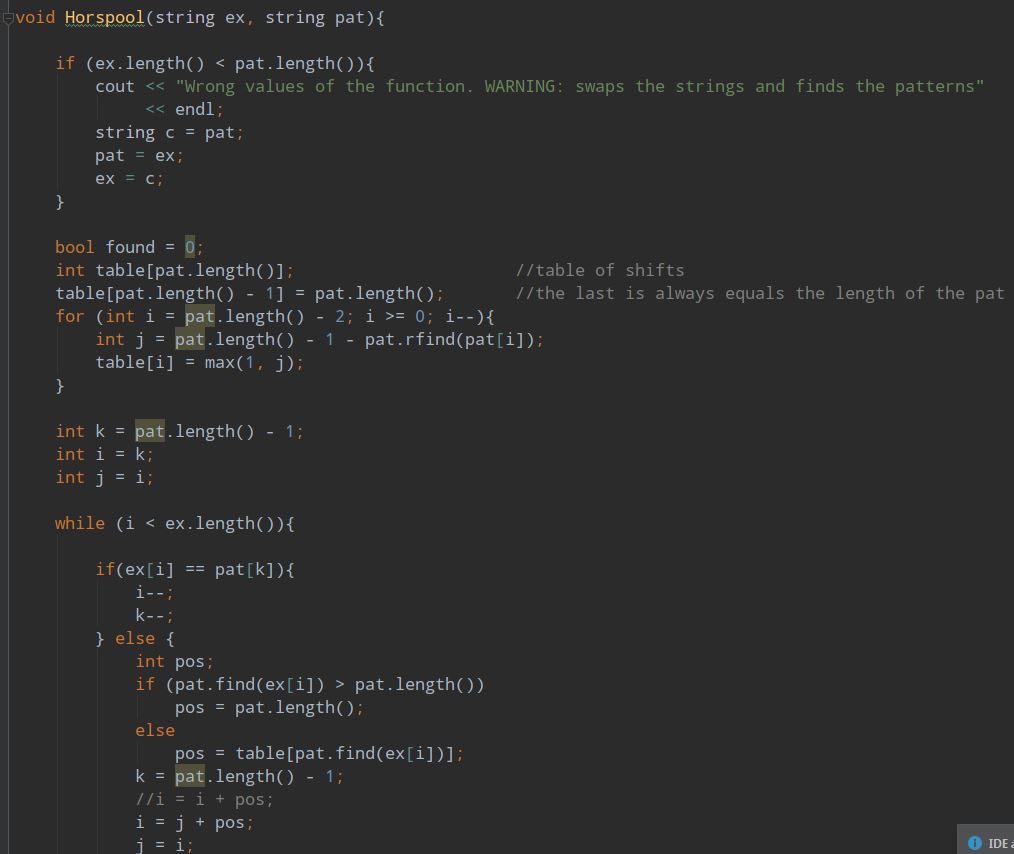
**Складність:** O(n+m)

1. **Основні модулі програми**

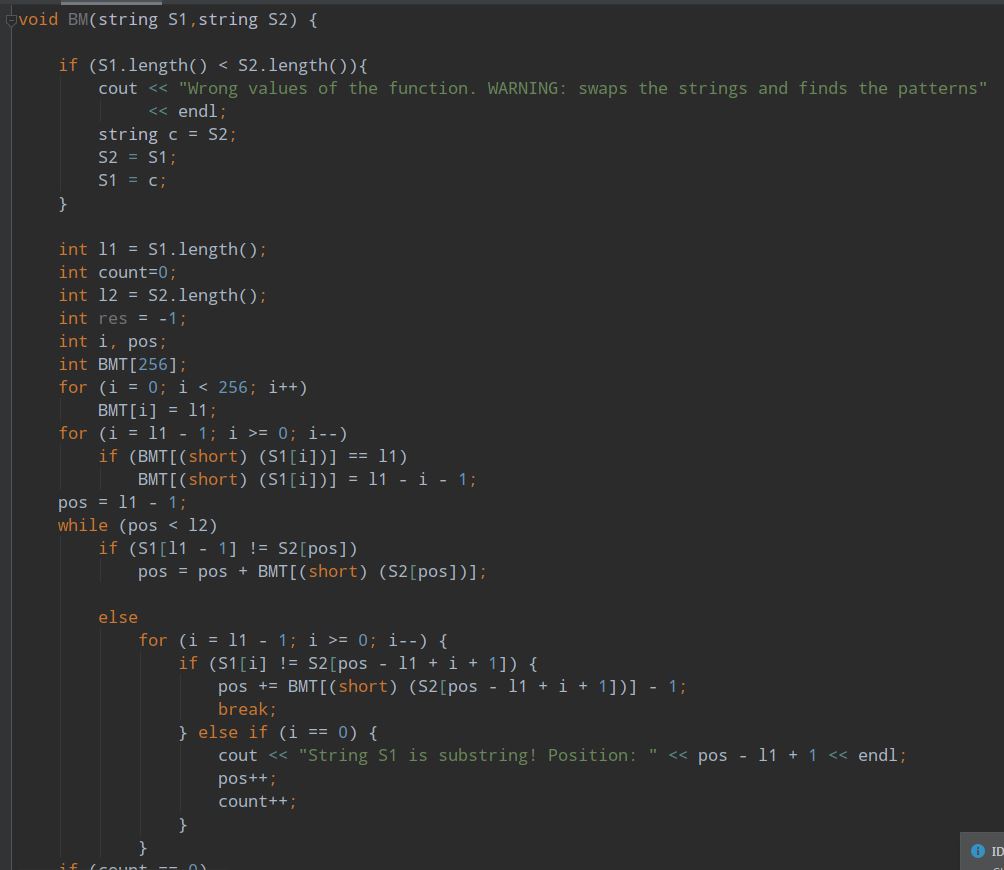
* **Наївний алгоритм**



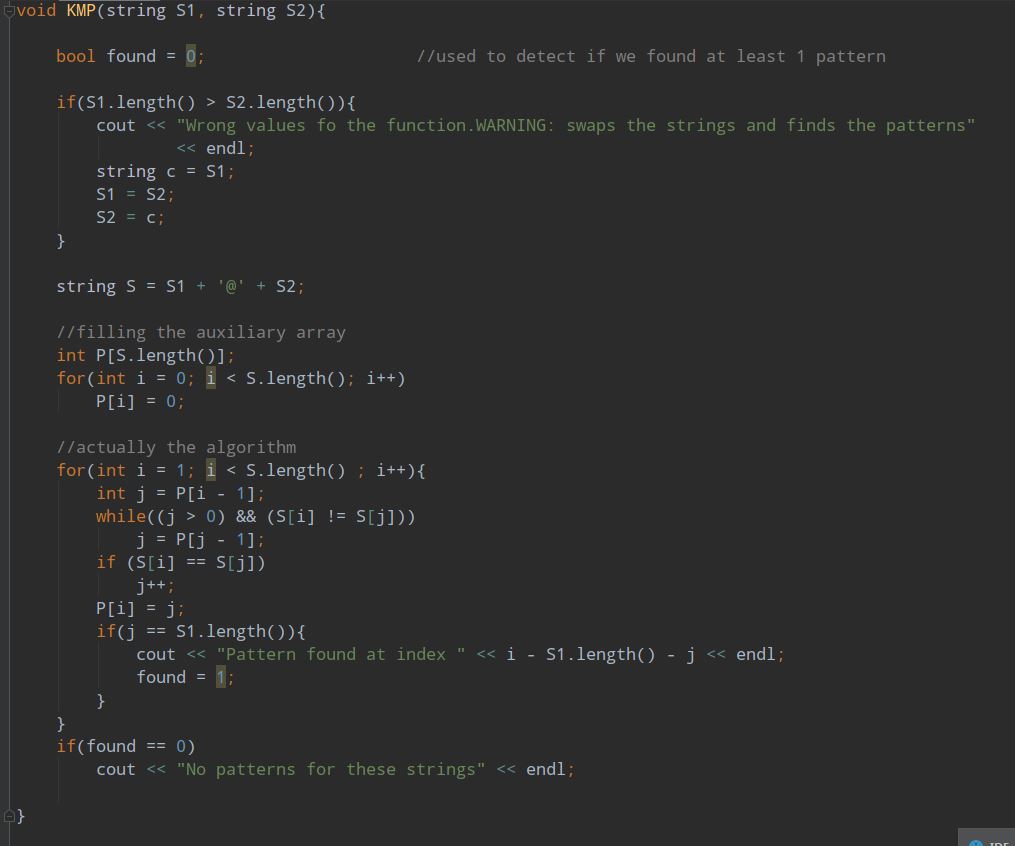
* **Алгоритм Хорспула**



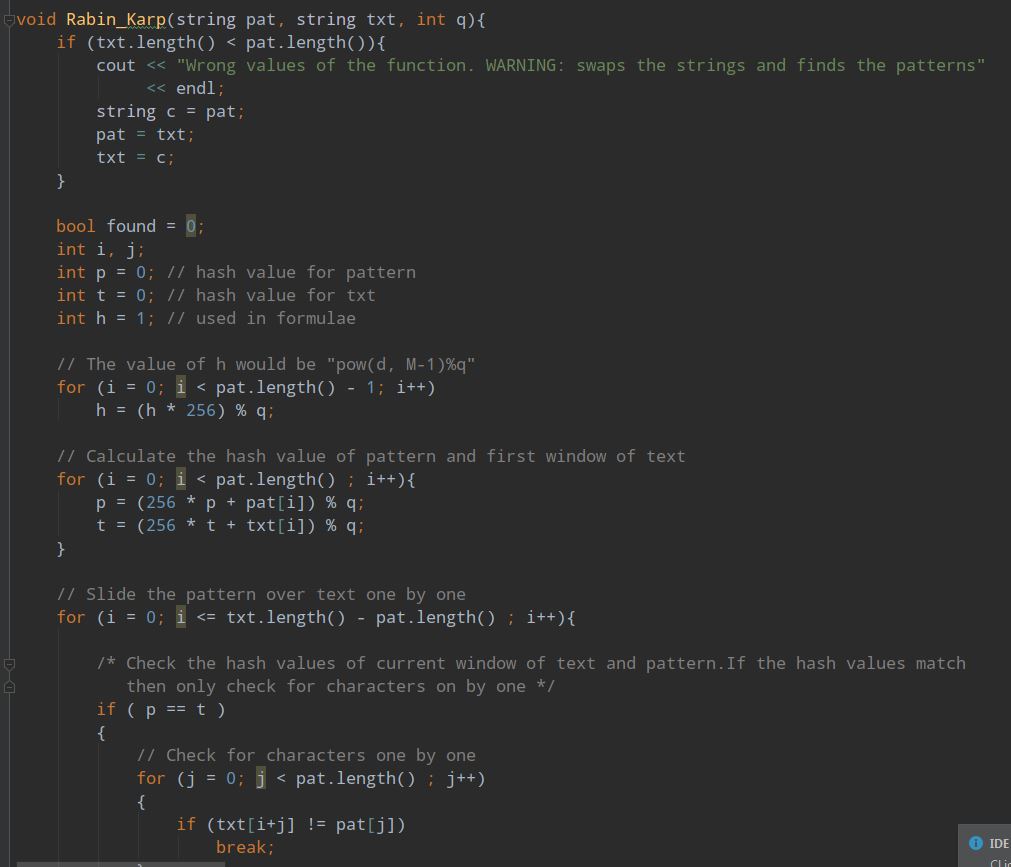
* **Алгоритм Бойера-Мура**

****

* **Алгоритм КПМ**



* **Алгоритм Рабіна-Карпа**



1. **Інтерфейс користувача**

Для інтерфейсу користувача буде достатньо базових бібліотек с++ та бібліотеки string.

1. **Тестові приклади**

Перший скріншот – приклад, в якому шукається “110111” в стрічці довжиною 10000 символів, кожен з яких 1 або нуль.

Другий скріншот – приклад, в якому шукається слово у тексті, природньому для цієї мови.

Числа відповідають кількості секунд, витрачених на відповідний алгоритм. Порядок алгоритмів:Наївний,КМП,Рабіна-Карпа,Хорспула та Бойена-Мура.

Як бачимо, найшвидшим алгоритмом для пошуку бінарної стрічки виявився алгоритм Хорспула, в той час коли найгіршим – наївний. Для пошуку слова на природній мові усі алгоритми показали себе однаково.

C:\Users\vetal\Desktop\1.JPG

C:\Users\vetal\Desktop\2.JPG

1. **Література**

* <https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A0%D0%B0%D0%B1%D1%96%D0%BD%D0%B0_%E2%80%94_%D0%9A%D0%B0%D1%80%D0%BF%D0%B0>
* <https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9A%D0%BD%D1%83%D1%82%D0%B0_%E2%80%94_%D0%9C%D0%BE%D1%80%D1%80%D1%96%D1%81%D0%B0_%E2%80%94_%D0%9F%D1%80%D0%B0%D1%82%D1%82%D0%B0>
* <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%D0%B9%D0%B5%D1%80%D0%B0_%D0%9C%D1%83%D1%80%D0%B0>
* <https://habr.com/ru/post/111449/>