Міністерство освіти і науки України

Національний технічний університет

«Харківський політехнічний інститут»

Навчально-науковий інститут комп’ютерних наук та інформаційних технологій

ЗВІТ

про виконання лабораторної роботи №5

Варіант 7

за дисципліною

«Алгоритми і структури даних»

Виконав:

студент групи КН-320Б

Миргород В.І.

Перевірила:

старший викладач

Мошко Є.О.

Харків 2022

**Лабораторна робота №5**

**Тема лабораторної роботи:** Алгоритми пошуку (лінійний пошук, бінарний  
пошук, пошук з бар'єром, пошук підрядка в рядку, алгоритм Кнута-Морріса-Пратта, алгоритм Бойера-Мура, алгоритм Рабина – Карпа).

**Мета:** ознайомитися із основними алгоритмами пошуку даних та  
особливостями їх програмної реалізації. Набути практичних навичок по роботі зі алгоритмами пошуку даних.

**Порядок виконання роботи:**  
1. Визначити масив, в якому буде виконуватися пошук. Використовувати  
масив, створений в роботі №4.  
2. Розробити функції лінійного пошуку в масиві, бінарного пошуку в масиві.  
Для забезпечення бінарного пошуку в масиві використовувати найбільш ефективний  
алгоритм сортування, визначений при виконанні лабораторної роботі No4:  
− ввести інформацію в масив з файлу;  
− виконати лінійний пошук в масиві;  
− виконати пошук з бар'єром;  
− упорядкувати елементи масиву функцією сортування і виконати бінарний  
пошук.  
3. Розробити функції для пошуку підрядка в рядку:  
− виконати прямий пошук підрядка;  
− виконати алгоритм Кнута-Морріса-Пратта;  
− виконати алгоритм Бойера-Мура;  
− виконати алгоритм Рабина – Карпа.  
4. Дослідити складність алгоритмів. Провести асимптотичний аналіз алгоритмів пошуку та зробити висновки.  
− для порівняння алгоритмів пошуку виконати наступні кроки;  
− створити таблицю асимптотичних оцінок трудомісткості алгоритмів в  
кращому, середньому, гіршому випадках;

− розставити лічильники операцій у функціях пошуку;  
− провести експеримент, визначити середню кількість операцій для різних  
алгоритмів, побудувати графіки;  
− створити таблиці і представити графіки експериментальних оцінок  
алгоритмів.  
Для всіх випадків перевірити варіанти успішного і неуспішного пошуку.  
Зробити висновки. Зберегти файл з текстом програми для подальших робіт.

**Аналіз завдання:**

Розробити реалізацію алгоритмів пошуку елемента в масиві, а також пошуку підрядка в рядку або тексті. Провести випробування алгоритмів, впевнитись у правильності їх роботи. Провести пошук при різному розташуванні шуканого елемента (найкращій, середній, найгірший). Порахувати кількість операцій, зробити оцінку складності алгоритмів. Згідно з результатами, скласти таблиці та провести їх аналіз. Визначити найкращі сторони кожного з алгоритмів.

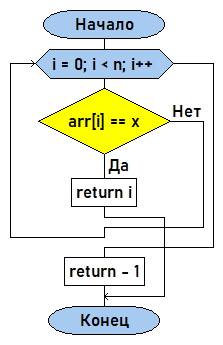
Структура вхідних та вихідних даних:

Для алгоритмів пошуку в масиві: масив генерується або зчитується з файлу.

Для пошуку підрядка: рядок (текст) зчитується з файлу.

**Лінійний пошук**

Алгоритм лінійного пошуку елемента в масиві є найпростішим, але в багатьох випадках він потребує максимум часу для роботи. Під час роботи алгоритму елементи масиву послідовно порівнюються з ключем.



int LinearSearch(int\* arr, int n, int x) {

for (int i = 0; i < n; i++) {

if (arr[i] == x)

return i;

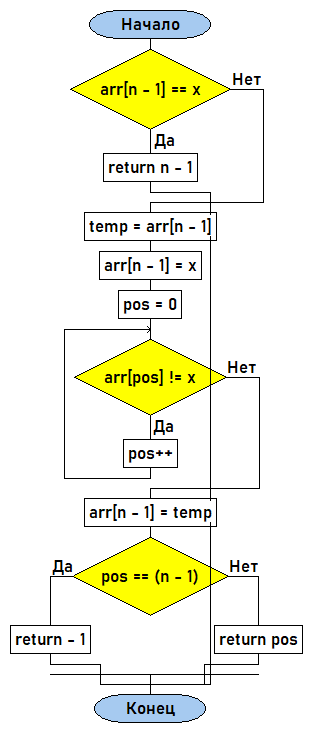
}

return -1;

}

**Лінійний пошук з бар’єром**

Принцип такого пошуку дуже схожий з лінійним пошуком. Різниця в тому, що перед початком роботи в кінець алгоритму додається елемент, що є ключем пошуку. Це дозволяє видалити з алгоритму перевірку на кінець масиву. Тобто на кожному кроці потрібно робити на 1 порівняння менше, що прискорює роботу.



int LinearSearchBarrier(int\* arr, int n, int x) {

if (arr[n - 1] == x) {

return n - 1;

}

int temp = arr[n - 1];

arr[n - 1] = x;

int pos = 0;

while (arr[pos] != x) {

pos++;

}

arr[n - 1] = temp;

if (pos == (n - 1))

return -1;

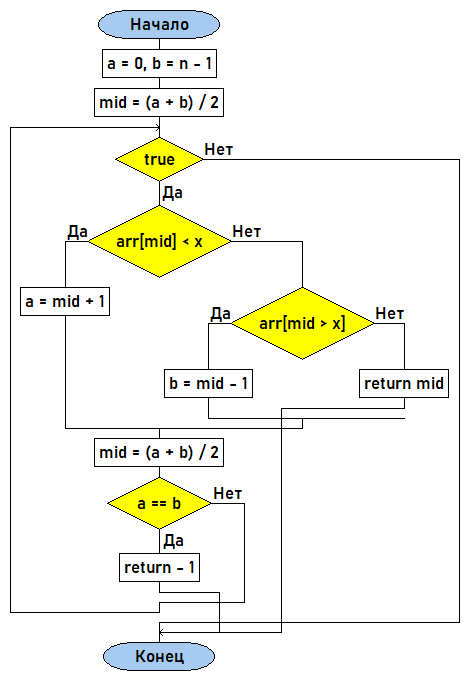
else

return pos;

}

**Бінарний пошук**

Бінарний пошук в масиві працює краще за попередні. Але вимагає відсортованого масиву. Алгоритм на кожному кроці ділить масив на 2 частини, обирає з них одну, в залежності від значення середнього елемента. Далі продовжується, поки елемент не буде знайдено.



int BinarySearch(int\* arr, int n, int x) {

int a = 0, b = n - 1;

int mid = (a + b) / 2;

while (true) {

if (arr[mid] < x)

a = mid + 1;

else if (arr[mid > x])

b = mid - 1;

else

return mid;

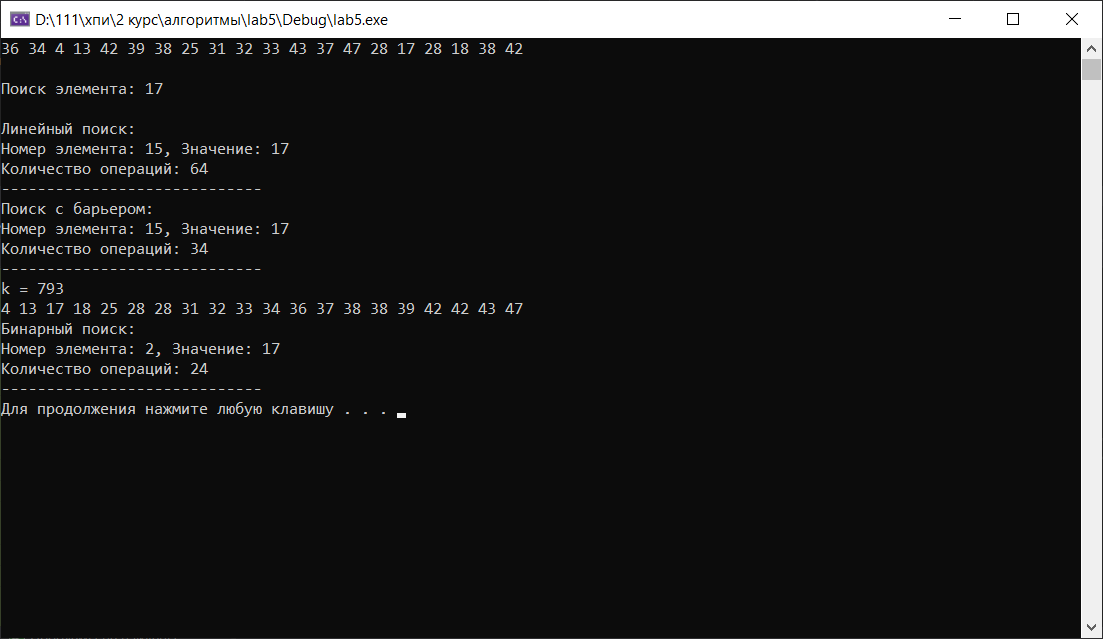
mid = (a + b) / 2;

if (a == b)

return -1;

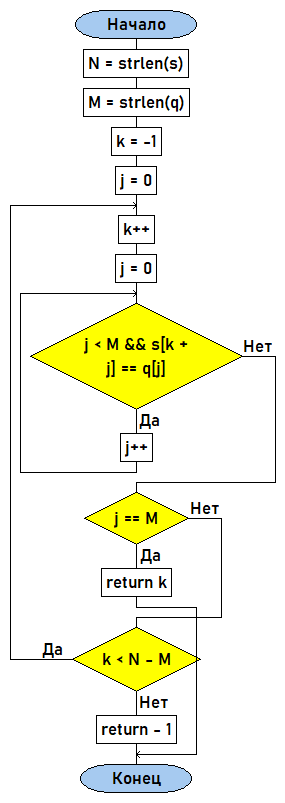
}

}



**Прямий пошук**

Алгоритм прямого пошуку виконує послідовне порівняння символів підрядка із символами рядка, поки не знайде потрібний підрядок, або поки не закінчиться рядок.



int SimpleSearch(char s[], char q[]) {

int N = strlen(s);

int M = strlen(q);

int k = -1;

int j = 0;

do {

k++;

j = 0;

while (j < M && s[k + j] == q[j])

j++;

if (j == M)

return k;

} while (k < N - M);

return -1;

}

При будь-якому розташуванні елементів, масив сортується за однакову кількість часу. Швидкість сортування – O(w\*n), де w – кількість біт для зберігання елемента масиву. Досить швидко сортує масиви великого розміру, але потребує багато додаткової пам’яті.

**Алгоритм Кнута-Морріса-Пратта**

Алгоритм заснований на тому, щоб на кожному кроці зсуватися не на один символ, а на якесь змінне значення. При чому на кожному кроці це значення має бути максимальним. Для роботи алгоритму будується префікс-функція, за допомогою якої відбуваються подальші порівняння.

int KMPSearch(char s[], char p[])

{

int i, j, N, M;

N = strlen(s);

M = strlen(p);

int\* temp = new int[M];

temp[0] = 0;

for (i = 1, j = 0; i < M; i++)

{

while (j > 0 && p[j] != p[i])

j = temp[j - 1];

if (p[j] == p[i])

j++;

temp[i] = j;

}

for (i = 0, j = 0; i < N; i++)

{

while (j > 0 && p[j] != s[i])

j = temp[j - 1];

if (p[j] == s[i])

j++;

if (j == M)

{

delete[] temp;

return i - j + 1;

}

}

delete[] temp;

return -1;

}

**Алгоритм Бойера-Мура**

Для роботи цього алгоритма будується додатковий масив зміщень для символів підрядка. Далі, використовуючи значення зміщень, відбувається порівняння з рядком, починаючи з кінця. Якщо символ не співпадає, підрядок зсувається вправо і дія повторюється.

int BMSearch(char str[], char substr[]) {

int n = strlen(str);

int m = strlen(substr);

int i, pos;

int BMT[256];

for (int i = 0; i < 255; i++)

BMT[i] = m;

for (int i = m - 1; i >= 0; i--)

if (BMT[(short)(substr[i])] == m)

BMT[(short)(substr[i])] = m - i - 1;

pos = m - 1;

while (pos < n) {

if (substr[m - 1] != str[pos])

pos = pos + BMT[(short)(str[pos])];

else

for (i = m - 2; i >= 0; i--) {

if (substr[i] != str[pos - m + i + 1]) {

pos += BMT[(short)(str[pos - m + i + 1])] - 1;

break;

}

else

if (i == 0)

return pos - m + 1;

cout << "\t" << i << endl;

}

}

if (pos > n - m)

return -1;

return pos;

}

**Алгоритм Рабіна-Карпа**

Цей алгоритм дозволяє знаходити підрядок в рядку, або слово в тексті. При чому при пошуку в тексті метод знаходить всі входження потрібного рядка. У роботі застосовує значення хеш-функції рядка. Дозволяє знаходити будь-яку строку однакового розміру за один є той же час, незалежно від розміру k.

int RabinKarp(string& text, string& pattern, int q)

{

int m = pattern.length();

int n = text.length();

int p = 0, t = 0, h = 1, d = 26;

for (int i = 0; i < m - 1; i++)

{

h = (h \* d) % q;

}

for (int i = 0; i < m; i++)

{

p = (d \* p + pattern[i]) % q;

}

for (int i = 0; i <= n - m; i++)

{

if (p == t)

{

int flag = 0;

for (int j = 0; j < m; j++)

{

if (text[i + j] != pattern[j])

{

flag = 1;

break;

}

}

if (flag == 0)

{

return i;

}

}

if (i < n - m)

{

t = (d \* (t - text[i] \* h) + text[i + m]) % q;

if (t < 0)

{

t = (t + q);

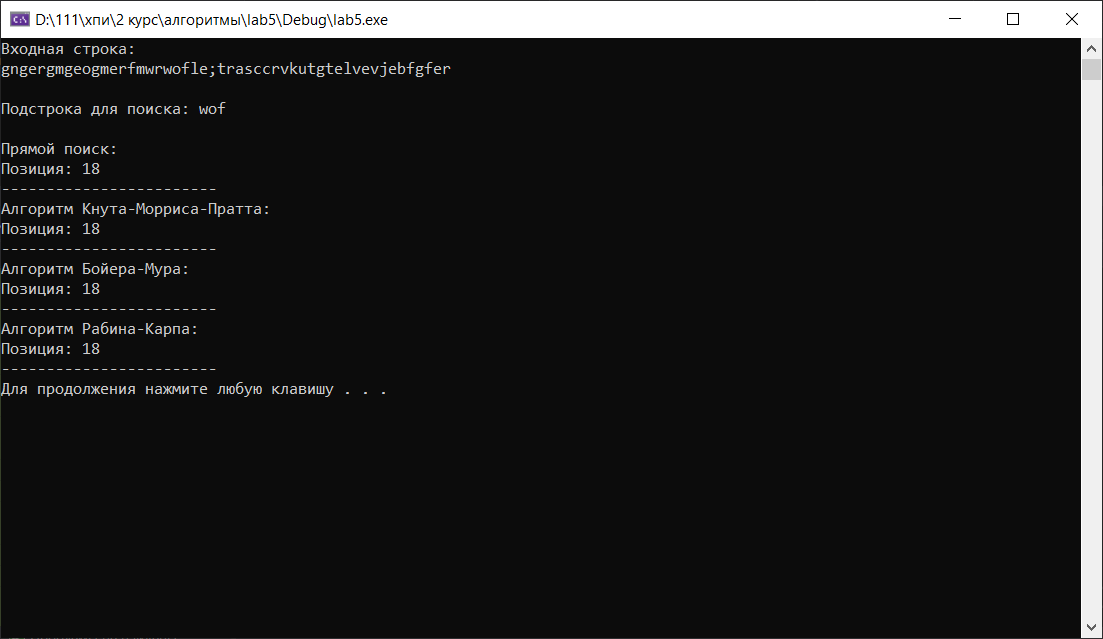
}

}

}

}

Знаходження підрядка за допомогою різних алгоритмів:



**Аналіз алгоритмів**

**Кількість операцій при пошуку (масив зі 100 елементів)**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Найкращий | Середній | Найгірший |
| Лінійний пошук | 8 | 140 | 534 |
| Пошук з бар'ером | 6 | 98 | 362 |
| Бінарний пошук | 10 | 38 | 136 |

Складність алгоритмів:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Найкращий | Середній | Найгірший |
| Лінійний пошук | O(1) | O(n) | O(n) |
| Пошук з бар'ером | O(1) | O(n) | O(n) |
| Бінарний пошук | O(1) | O(logn) | O(logn) |

**Кількість операцій при пошуку (рядок з 50 елементів)**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Найкращий | Середній | Найгірший |
| Прямий пошук | 21 | 136 | 224 |
| Кнута-Морріса-Пратта | 31 | 190 | 322 |
| Бойера-Мура | 284 | 296 | 302 |
| Рабіна-Карпа | 657 | 657 | 661 |

**Висновки:**

Для знаходження потрібного значення у масиві застосовуються такі алгоритми: лінійний, з бар’єром, бінарний пошук. Алгоритм пошуку з бар’єром є модифікацією лінійного пошуку, тому працює швидше. Оптимальним є метод бінарного пошуку, бо він потребує значно менше операцій для знаходження елемента, особливо при великих розмірах масиву. Але такий метод потребує, щоб масив було відсортовано перед пошуком.

Для знаходження підрядка в рядку найшвидшим вважається алгоритм Бойєра-Мура. Для його використання потрібно лише побудувати таблицю зміщень для символів. Для знахождення шаблона в тексті найкраще підходить метод Рабіна-Карпа. За його допомогою можна знаходити багато входжень слова або рядка в текст, а також він дозволяє швидко працювати з великими текстами. У роботі використовує хешування рядків і підрядків, для подальшого порівняння.