МІНІСТРЕСТВО ОСВІТИ Й НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

“Харківський Політехнічний Інститут”

Кафедра управління проєктами в інформаційних технологіях

Звіт з лабораторної роботи №6

“Однозв’язні та двозв’язні списки й робота з ними”

з дисципліни

“Алгоритми та структури даних”

Варіант №5

Перевірив: ст. викл. каф. УПІТ Мошко Є.О.

Виконав: ст. гр. КН-1223г Ставицький А.А.

Харків – 2024

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

**Мета:** ознайомитися із основними алгоритмами пошуку даних

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 Nº4.

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:

1. ﻿﻿ввести інформацію в масив з файлу;
2. ﻿﻿виконати лінійний пошук в масиві;
3. ﻿﻿виконати пошук з барʼєром;
4. ﻿﻿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;

5. Розставити лічильники операцій у функціях пошуку;

6. Провести експеримент, визначити середню кількість операцій для різних алгоритмів, побудувати графіки;

7. 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.

Заборонено використовувати готові реалізації структур даних (наприклад,

STL).

**Алгоритм лінійного пошуку**

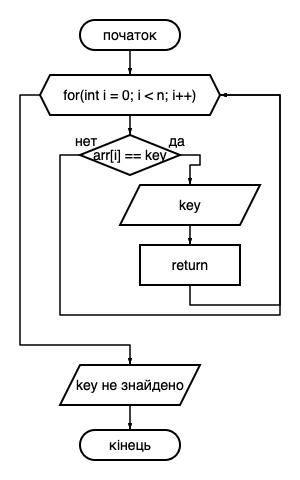
****

Рисунок 1 – Блок-схема для алгоритму лінійного пошуку

Функція linearSearch виконує лінійний пошук елемента в масиві. Вона приймає три параметри: масив цілих чисел arr[], кількість елементів в масиві n та шуканий елемент key. За допомогою циклу for функція проходить через всі елементи масиву і на кожній ітерації порівнює поточний елемент з шуканим значенням. Якщо знайдено збіг, функція виводить повідомлення про успішний пошук і позицію елемента в масиві, при цьому позиція виводиться з урахуванням індексації, що починається з 1. Якщо ж елемент не знайдений після перевірки всіх елементів, функція виводить повідомлення про те, що елемент не знайдено.



**Алгоритм лінійного пошуку бар’єром**

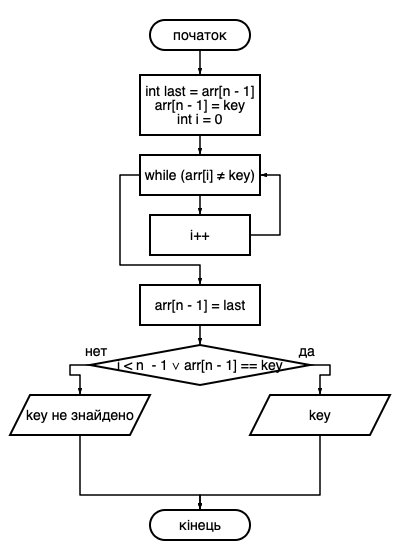
****

Рисунок 2 – Блок-схема до алгоритму лінійного пошуку бар’єром

Функція searchWithBarrier реалізує пошук з бар’єром у масиві. Вона приймає чотири параметри: масив цілих чисел arr[], кількість елементів у масиві n, шуканий елемент key та змінну result, яка містить результат пошуку. Спочатку функція зберігає останній елемент масиву у змінній last і замінює останній елемент на шуканий елемент key, створюючи таким чином бар’єр для пошуку. Потім функція за допомогою циклу while проходить по масиву, порівнюючи кожен елемент з шуканим значенням. Як тільки шуканий елемент знайдений, цикл завершується. Після завершення циклу функція відновлює останній елемент масиву в його початкове значення. Якщо шуканий елемент знаходиться до останнього елемента масиву, виводиться повідомлення про успішний пошук і його позицію (з урахуванням індексації з 1). Якщо елемент не знайдений, виводиться повідомлення про невдачу і значення result встановлюється на -1. Ця функція дозволяє здійснити пошук елемента з використанням бар’єра, що полегшує пошук в деяких випадках, коли шуканий елемент може бути розташований в кінці масиву.



**Алгоритм бінарного пошуку**

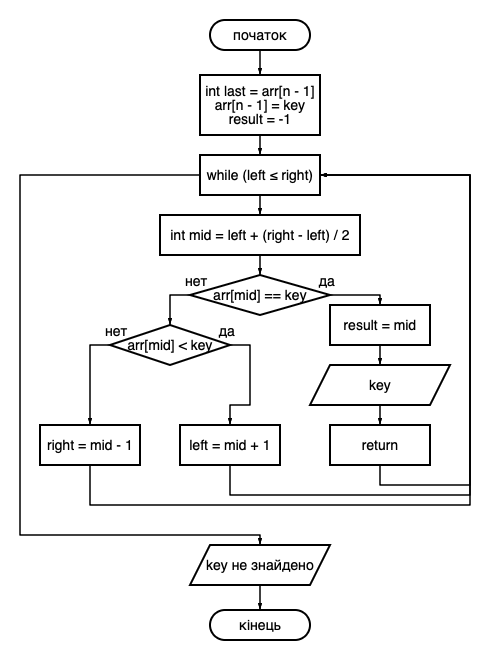
****

Рисунок 3 – Блок-схема до алгоритму бінарного пошуку

Функція binarySearch реалізує алгоритм бінарного пошуку в відсортованому масиві. Вона приймає чотири параметри: масив цілих чисел arr[], кількість елементів у масиві n, шуканий елемент key та змінну result, яка містить результат пошуку. Алгоритм бінарного пошуку починається з того, що встановлюються два індекси: left, який вказує на перший елемент масиву, і right, який вказує на останній елемент масиву. Змінна result ініціалізується значенням -1, яке використовується для позначення того, що елемент не знайдений. В циклі while, поки left менше або рівне right, обчислюється середній індекс масиву mid за формулою mid = left + (right - left) / 2. Потім перевіряється, чи дорівнює елемент на позиції mid шуканому значенню key. Якщо це так, то значення result встановлюється в індекс mid, і виводиться повідомлення про успішний пошук разом з позицією елемента (з урахуванням індексації з 1). Пошук завершується. Якщо ж елемент на позиції mid менший за key, то пошук продовжується в правій половині масиву, тому індекс left встановлюється рівним mid + 1. Якщо елемент на позиції mid більший за key, то пошук продовжується в лівій половині масиву, і індекс right зменшується до mid - 1. Якщо цикл завершується без знаходження елемента, виводиться повідомлення про те, що елемент не знайдений, і значення result залишиться рівним -1.



**Алгоритм прямого пошуку підрядка в рядку**

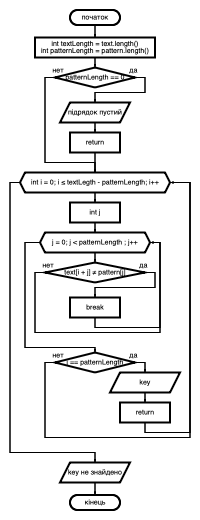
****

Рисунок 4 – Блок-схема до алгоритму прямого пошука пірядка в рядку

Функція directSubstringSearch шукає підрядок у тексті шляхом прямого порівняння. Вона приймає два параметри: текст, в якому потрібно знайти підрядок, та сам підрядок. Спочатку функція перевіряє, чи не є підрядок порожнім. Якщо підрядок порожній, виводиться повідомлення “Підрядок пустий!” і функція завершується. Потім функція перебирає всі можливі позиції в тексті, де може починатися підрядок, від 0 до textLength - patternLength. Для кожної позиції вона порівнює символи тексту з відповідними символами підрядка. Якщо символи не збігаються, перевірка для цієї позиції припиняється. Якщо всі символи підрядка співпали з символами тексту, функція виводить повідомлення про успішний пошук і вказує позицію підрядка, де він був знайдений. Якщо підрядок не знайдений, виводиться повідомлення про невдачу.



**Алгоритм Кнута-Морріса-Пратта**

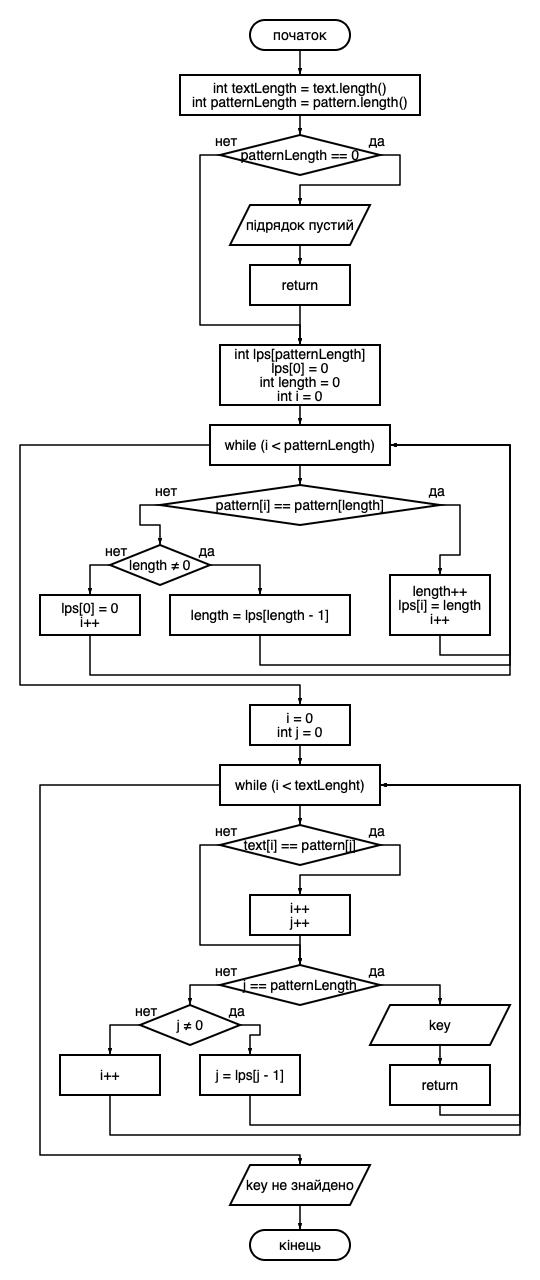
****

Рисунок 5 – Блок-схема до алгоритму Кнута-Морріса-Пратта

Функція knutMorrisPrattSearch реалізує алгоритм Кнута-Морріса-Пратта для пошуку підрядка в тексті. Спочатку вона перевіряє, чи не є підрядок порожнім. Якщо підрядок порожній, виводиться повідомлення “Підрядок пустий!” і функція завершується. Далі створюється масив lps (longest prefix suffix), який буде використовуватися для запам’ятовування довжини найбільшого префікса, що одночасно є суфіксом для кожної позиції підрядка. Алгоритм починає заповнювати цей масив шляхом порівняння символів підрядка. Якщо символи співпадають, збільшується довжина префікса, і значення в масиві оновлюється. Якщо символи не співпадають, то за допомогою масиву lps визначається, як потрібно змінити позицію в підрядку, щоб продовжити пошук без зайвих порівнянь. Після заповнення масиву lps, функція переходить до основного пошуку підрядка в тексті. Вона порівнює символи тексту з символами підрядка, і якщо вони співпадають, збільшується індекс обох рядків. Якщо підрядок знайдено, виводиться повідомлення про його успішний пошук і його позицію в тексті. Якщо на певному кроці символи не співпали, функція використовує масив lps для визначення, куди слід перемістити індекс підрядка, щоб уникнути зайвих порівнянь. Якщо пошук завершується без успіху, виводиться повідомлення про невдачу: “Підрядок не знайдено в тексті”.



Алгоритм Бойера-Мура

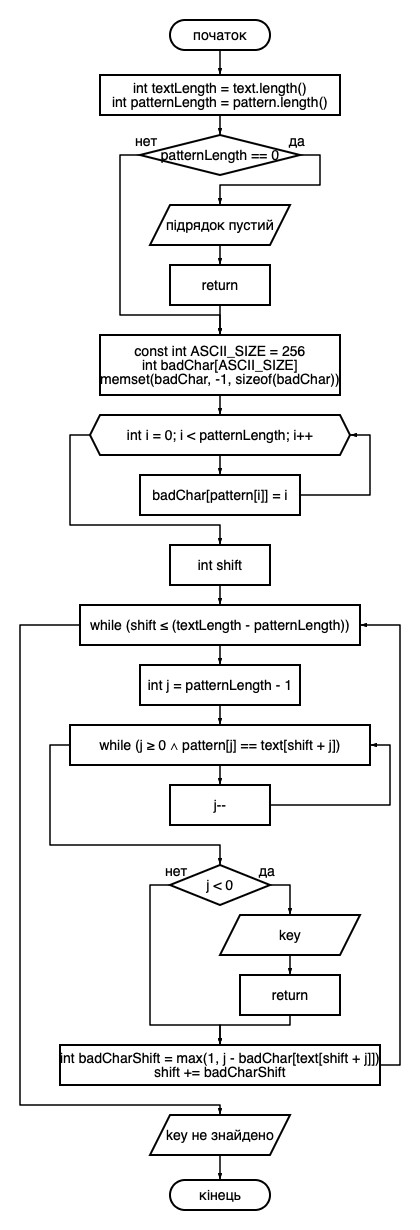


Рисунок 6 – Блок-схема до Бойера-Мура

Функція boyerMooreSearch реалізує алгоритм Бойєра-Мура для пошуку підрядка в тексті. Вона перевіряє, чи не є підрядок порожнім, і якщо так, виводить повідомлення “Підрядок пустий!” та завершує виконання. Далі, створюється масив badChar розміром 256 для збереження індексів останнього входження кожного символу з тексту в підрядок. Масив ініціалізується значенням -1 для всіх символів, а потім для кожного символу в підрядку оновлюється відповідний індекс у масиві badChar. Після цього алгоритм починає пошук підрядка в тексті. Початковий зсув встановлюється на нуль, а потім здійснюється порівняння символів підрядка з текстом, починаючи з останнього символу підрядка (правая частина підрядка). Якщо символи співпадають, індекс підрядка і тексту зменшуються. Якщо виявляється, що весь підрядок знайдений (індекс j стає меншим за нуль), виводиться повідомлення про успішний пошук і позицію підрядка в тексті. Якщо ж знайдений символ не збігається, обчислюється зсув для наступного порівняння. Зсув визначається на основі таблиці badChar так, щоб пропустити вже перевірені частини тексту та підрядка, використовуючи найкращий зсув для поточного символу. Якщо пошук завершується без успіху (підрядок не знайдено в тексті), виводиться повідомлення: “Підрядок не знайдено в тексті”.



**Алгоритм Рабина-Карпа**

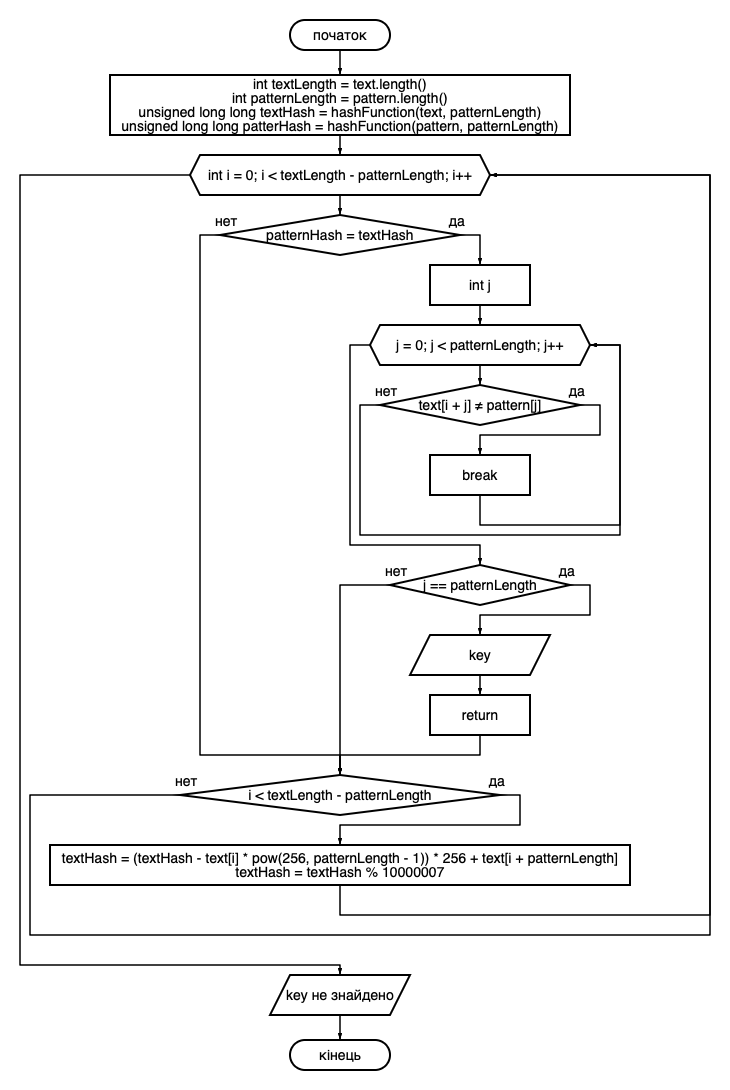
****

Рисунок 7 – Блок-схема до алгоритму Рабина-Карпа

Функція rabinKarpSearch реалізує алгоритм Рабіна-Карпа для пошуку підрядка в тексті, використовуючи хеш-функцію для оптимізації перевірки відповідності. Спочатку обчислюються хеші підрядка та першої частини тексту, яка має ту ж довжину, що і підрядок. Хеш-функція тут використовується для швидкого порівняння фрагментів тексту. Далі алгоритм виконує ітерацію по тексту, перевіряючи, чи збігається хеш поточного фрагмента тексту із хешем підрядка. Якщо хеші збігаються, виконується покрокове порівняння символів цього фрагмента тексту з символами підрядка для підтвердження збігу. У разі повного збігу виводиться повідомлення із зазначенням позиції, на якій знайдено підрядок, і функція завершує виконання. Якщо збігу немає, обчислюється хеш наступного фрагмента тексту за допомогою оновлення попереднього хешу, що значно скорочує час обчислення. У новому хеші символ, який більше не входить до фрагмента, віднімається, а символ, що додається, враховується. Для уникнення переповнення застосовується модуль 1000000007. Якщо пошук завершується без виявлення підрядка, виводиться повідомлення: “Підрядок не знайдено в тексті”.



**Асимптотний аналіз алгоритмів пошуку у масиві**

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Найкращий випадок | Середній випадок | Найгірший випадок |
| Лінійний пошук | O(1) | O(N) | O(N) |
| Пошук з бар’єром | O(1) | O(N) | O(N) |
| Бінарний пошук | O(1) | O(log N) | O(log N) |

Таблиця 1 – Асимптотичні оцінки трудомісткості алгоритмів

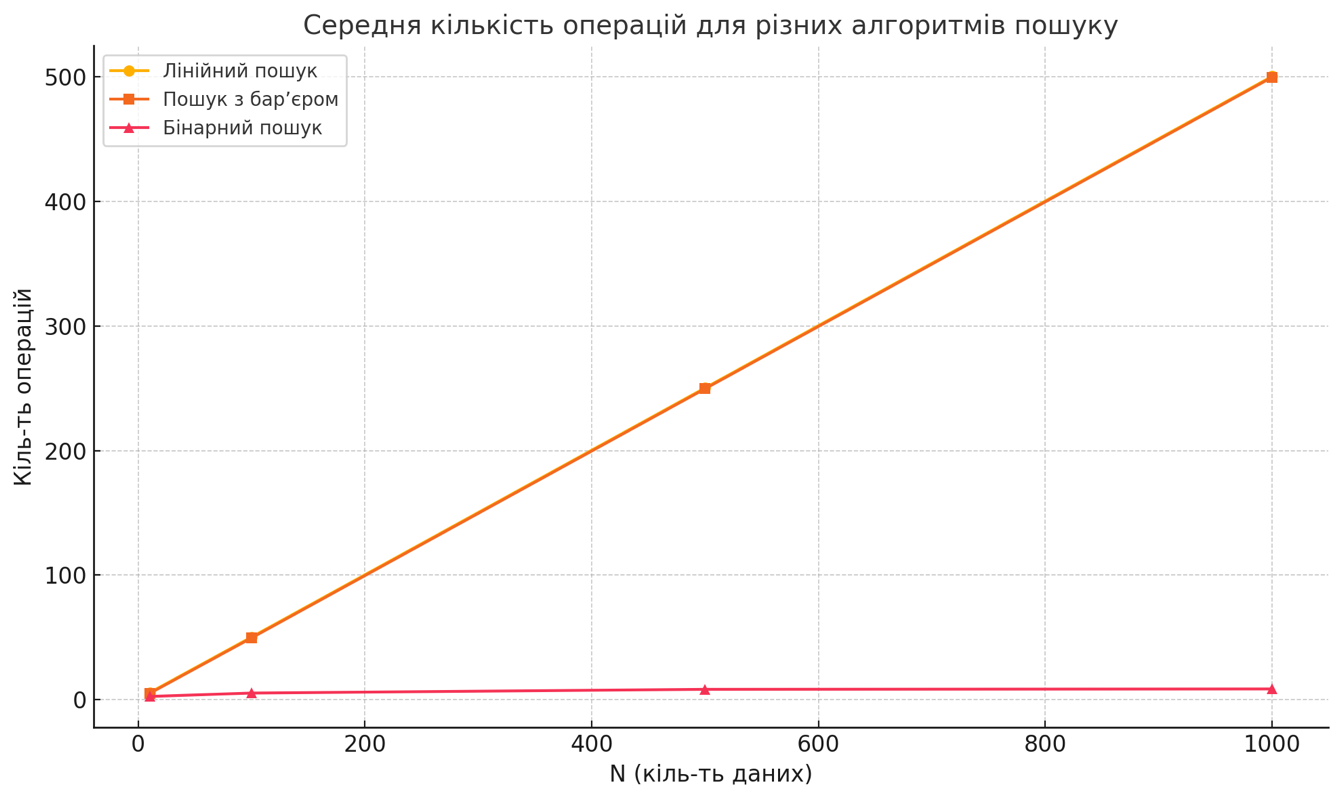
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Місце елемента | Алгоритм | N=10 | N=100 | N=500 | N=1000 |
| На початку | Лінійний пошук | 1 | 1 | 1 | 1 |
| Пошук з бар’єром | 1 | 1 | 1 | 1 |
| Бінарний пошук | 3 | 6 | 8 | 9 |
| У середині | Лінійний пошук | 6 | 51 | 251 | 501 |
| Пошук з бар’єром | 5 | 50 | 250 | 500 |
| Бінарний пошук | 3 | 6 | 8 | 9 |
| В кінці | Лінійний пошук | 10 | 100 | 500 | 1000 |
| Пошук з бар’єром | 9 | 99 | 499 | 999 |
| Бінарний пошук | 4 | 7 | 9 | 10 |
| Середня кіл-ть операцій | Лінійний пошук | 5,6 | 50,3 | 250,3 | 500,3 |
| Пошук з бар’єром | 5 | 49,6 | 249,6 | 499,6 |
| Бінарний пошук | 2,6 | 5,3 | 8,3 | 8,6 |

Таблиця 2 – Кіл-ть операцій при виконанні алгоритмів

Графік 1 – Середня кіл-ть операцій при N для лінійного пошуку

Графік 2 – Середня кіл-ть операцій при N для лінійного пошуку з бар’єром

Графік 3 – Середня кіл-ть операцій при N для бінарного пошуку

****

Графік 4 - Середня кіл-ть операцій при N для всіх алгоритмів

**Асимптотичний аналіз алгоритмів пошуку у рядку**

|  |  |  |  |
| --- | --- | --- | --- |
| Алгоритм | Найкращий випадок | Середній випадок | Найгірший випадок |
| Прямий пошук | O(N) | O(M\*N) | O((N-M+1)\*M) |
| Кнута-Морріса-Пратта | O(N+M) | O(N+M) | O(N+M) |
| Бойера-Мура | O(N/M) | O(N+M) | O(N\*M) |
| Рабина-Карпа | O(N+M) | O(N+M) | O(N\*M) |

Таблиця 3 – Асимптотичні оцінки трудомісткості алгоритмів

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Алгоритм | | N=10 | N=30 | N=50 | N=100 |
| Підрядок є в рядку | Прямий пошук | 22 | 74 | 126 | 256 |
| Кнута-Морріса-Пратта | 18 | 49 | 81 | 161 |
| Бойера-Мура | 6 | 18 | 30 | 60 |
| Рабина-Карпа | 8 | 28 | 48 | 98 |
| Підрядку немає в рядку | Прямий пошук | 17 | 56 | 96 | 196 |
| Кнута-Морріса-Пратта | 22 | 61 | 101 | 201 |
| Бойера-Мура | 0 | 1 | 2 | 5 |
| Рабина-Карпа | 8 | 28 | 48 | 98 |
| Середня кіл-ть операцій | Прямий пошук | 19 | 65 | 111 | 226 |
| Кнута-Морріса-Пратта | 10 | 54,5 | 91 | 181 |
| Бойера-Мура | 3 | 9,5 | 16 | 32,5 |
| Рабина-Карпа | 8 | 28 | 48 | 98 |

Таблиця 4 – Кіл-ть операцій при виконанні алгоритмів

Графік 5 – Середня кіл-ть операцій при N для прямого пошуку

Графік 6 – Середня кіл-ть операцій при N для алгоритму Кнута-Морріса-Пратта

Графік 7 – Середня кіл-ть операцій при N для алгоритму Бойера-Мура

Графік 9 – Середня кіл-ть операцій при N для алгоритмів

**Висновки:** Під час виконання лабораторної я досліджував різні алгоритми пошуку як для масивів чисел, так і для рядків. Зокрема, я реалізував і порівняв ефективність лінійного пошуку, пошуку з бар’єром, бінарного пошуку, а також чотирьох алгоритмів пошуку підрядків: простого пошуку, Кнута-Морріса-Пратта, Бойєра-Мура і Рабіна-Карпа. Кожен з цих алгоритмів має свої переваги і недоліки в залежності від типу даних і контексту використання.

1. Лінійний пошук працює шляхом перевірки кожного елемента масиву або рядка по черзі. Він має складність O(n), де n — кількість елементів у масиві або довжина рядка. Це простий і універсальний алгоритм, але його ефективність погіршується при збільшенні розміру масиву чи рядка.

2. Пошук з бар’єром є вдосконаленою версією лінійного пошуку, де останній елемент масиву чи рядка дублюється в кінці, щоб прискорити пошук. Цей алгоритм працює ефективніше за лінійний пошук, особливо для випадків, коли шукані елементи розташовані ближче до кінця масиву. Його складність також O(n), але з меншими операціями порівняння у порівнянні з лінійним пошуком.

3. Бінарний пошук є дуже ефективним для впорядкованих масивів, оскільки на кожному кроці зменшує розмір пошукової області вдвічі. Його складність становить O(log n), що робить його значно швидшим за лінійний пошук для великих масивів, однак для цього масив має бути відсортованим.

4. Для пошуку підрядка я розглянув чотири алгоритми:

• Простий пошук підрядка здійснює порівняння кожного символу тексту з символами підрядка. Його складність O(n \* m), де n — довжина тексту, а m — довжина підрядка. Це базовий алгоритм, але для великих текстів він може бути неефективним.

• Алгоритм Кнута-Морріса-Пратта (KMP) використовує попереднє обчислення префіксних функцій для прискорення пошуку підрядка в тексті. Його складність становить O(n + m), що робить його значно швидшим за простий пошук, особливо для текстів середнього та великого розміру.

• Алгоритм Бойєра-Мура працює на основі пропуску певних частин тексту, що дозволяє скоротити кількість порівнянь. Його складність у найгіршому випадку O(n \* m), але на практиці цей алгоритм часто значно швидший завдяки своїй стратегії пропуску, особливо для текстів, які містять багато повторюваних символів.

• Алгоритм Рабіна-Карпа використовує хешування для швидкого порівняння підрядків тексту з шуканим патерном. Його середня складність становить O(n + m), але в найгіршому випадку (при великій кількості колізій) може досягати O(n \* m). Цей алгоритм ефективний при пошуку всіх входжень підрядка в тексті.

Висновок:

1. Для пошуку в масивах чисел бінарний пошук є найефективнішим, якщо масив відсортований. Він має значно кращу складність O(log n) порівняно з лінійними методами.

2. Для пошуку підрядків алгоритм Кнута-Морріса-Пратта (KMP) є найкращим вибором у більшості випадків завдяки лінійній складності O(n + m), хоча алгоритм Бойєра-Мура також може показати хороші результати для специфічних випадків, особливо для текстів з великою кількістю повторюваних символів.

3. Алгоритм Рабіна-Карпа корисний у випадках, коли потрібно знайти всі входження підрядка в тексті, але його ефективність залежить від якості хешування та ймовірності колізій.

Таким чином, вибір алгоритму залежить від конкретних вимог до задачі та характеристик даних. Для загальних випадків найкращими варіантами є бінарний пошук для впорядкованих масивів та алгоритм Кнута-Морріса-Пратта для пошуку підрядків.