МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

Кафедра комп’ютерної інженерії та електроніки

Виконала: Групи

Пугачова Д. В. КІ-23-1

Кременчук 2024

Практична робота №4

Тема. Алгоритми пошуку та їх складність

Мета: опанувати основні алгоритми сортування та навчитись методам аналізу їх асимптотичної складності.

Завдання

**1. Оцінити асимптотичну складність алгоритму лінійного пошуку у 𝑂- нотації в найгіршому і в найкращому випадку. Як можна покращити алгоритм лінійного пошуку?**

***Оцінка асимптотичної складності лінійного пошуку***

***Лінійний пошук*** *— це алгоритм, який послідовно перевіряє кожен елемент у списку, щоб знайти шукане значення.*

***-Найгірший випадок:***

*У найгіршому випадку, коли шуканий елемент знаходиться в кінці списку або його взагалі немає, алгоритм мусить перевірити всі елементи списку. Якщо в списку є nnn елементів, то лінійний пошук здійснить nnn порівнянь.*

* ***Складність в найгіршому випадку****: O(n)*

***-Найкращий випадок:***

*У найкращому випадку, коли шуканий елемент знаходиться на самому початку списку, алгоритм знаходить його після першого порівняння.*

* ***Складність в найкращому випадку****: O(1)*

***Як покращити лінійний пошук?***

*Лінійний пошук сам по собі не можна значно прискорити, оскільки він перевіряє кожен елемент. Однак, є кілька способів, як покращити ефективність пошуку:*

1. ***Використання хеш-таблиць****: Якщо потрібно часто виконувати пошуки, можна застосувати* ***хеш-таблицю****, яка дозволяє знаходити елементи набагато швидше, за час O(1) у середньому випадку.*
2. ***Сортування та бінарний пошук****: Якщо список вже відсортований, можна використовувати* ***бінарний пошук****, який шукає елемент за O(log n), що набагато швидше за лінійний пошук. Однак для цього спочатку потрібно відсортувати список, що займає)O(n log n)) часу.*
3. ***Оптимізація для частих пошуків****: Якщо пошук виконується багато разів, можна застосувати різні стратегії, щоб прискорити пошук. Наприклад, можна використовувати структури даних, які дозволяють швидко знаходити елементи, або зберігати результати пошуку в спеціальних індексах.*

*У загальному випадку лінійний пошук залишається найпростішим варіантом для пошуку в неструктурованих списках, і його ефективність залежить від того, як часто та в яких умовах потрібно виконувати пошук.*

**2. Оцінити асимптотичну складність алгоритму бінарного пошуку у 𝑂- нотації в найгіршому і в найкращому випадку.**

***Оцінка асимптотичної складності алгоритму бінарного пошуку***

***Бінарний пошук*** *— це ефективний алгоритм пошуку елемента в відсортованому масиві або списку. Він працює за принципом ділення масиву на дві частини і порівняння шуканого елемента з середнім значенням масиву. Потім пошук продовжується лише в тій частині масиву, де може бути знайдений шуканий елемент.*

*-****Найгірший випадок:***

*У найгіршому випадку бінарний пошук продовжує ділити масив на половину до тих пір, поки не знайде елемент або не визначить, що його немає в масиві. Оскільки на кожному кроці масив ділиться на два, кількість порівнянь обмежена логарифмом від розміру масиву. Кількість таких поділів буде дорівнювати , де n — кількість елементів у масиві.*

* ***Складність у найгіршому випадку****: O(log n)*

***-Найкращий випадок:***

*У найкращому випадку шуканий елемент може бути знайдений відразу ж на першому кроці, тобто в середині масиву. Це трапляється, коли шуканий елемент є саме середнім елементом на першій перевірці.*

* ***Складність у найкращому випадку****: O(1)*

**3. Побудувати алгоритм тернарного пошуку і оцінити його асимптотичну складність алгоритму у 𝑂-нотації в найгіршому і в найкращому випадку. Який з алгоритмів є оптимальнішим: бінарний, чи тернарний? Обґрунтувати відповідь відповідними обчисленнями.**

***Тернарний пошук*** *є варіантом бінарного пошуку, але замість того, щоб ділити масив на дві частини, він ділить його на три частини. Алгоритм перевіряє два середніх елементи (взяті через поділ масиву на три рівні частини) і на основі порівняння шуканого елемента з ними визначає, у якій частині масиву слід продовжити пошук.*

***Псевдокод тернарного пошуку:***

*def ternary\_search(arr, left, right, target):*

*if right >= left:*

*mid1 = left + (right - left) // 3*

*mid2 = right - (right - left) // 3*

*# Якщо шуканий елемент знайдено*

*if arr[mid1] == target:*

*return mid1*

*if arr[mid2] == target:*

*return mid2*

*# Якщо шуканий елемент менший за mid1, шукаємо в лівій частині*

*if target < arr[mid1]:*

*return ternary\_search(arr, left, mid1 - 1, target)*

*# Якщо шуканий елемент більший за mid2, шукаємо в правій частині*

*elif target > arr[mid2]:*

*return ternary\_search(arr, mid2 + 1, right, target)*

*# Якщо елемент знаходиться між mid1 і mid2*

*else:*

*return ternary\_search(arr, mid1 + 1, mid2 - 1, target)*

*return -1 # Елемент не знайдений*

***Оцінка асимптотичної складності тернарного пошуку***

*-****Найгірший випадок:***

*У кожному кроці тернарний пошук розділяє масив на три частини, а не на дві, як це робить бінарний пошук. Це означає, що кожен поділ зменшує розмір масиву на третину, і на кожному етапі алгоритм буде виконувати три порівняння (одне для кожного з двох середніх елементів). Тому кількість рекурсивних кроків у найгіршому випадку можна оцінити як:*

* *Кількість поділів:*n*, де n — кількість елементів у масиві.*

*Таким чином, асимптотична складність тернарного пошуку в найгіршому випадку дорівнює:*

* ***Складність у найгіршому випадку****: O(*n*), що є аналогом O(log n), оскільки основні логарифмічні функції вважаються еквівалентними для різних основ.*

***-Найкращий випадок:***

*У найкращому випадку тернарний пошук знаходить шуканий елемент на першій перевірці, тобто на першому кроці алгоритм порівнює шуканий елемент з двома середніми елементами і знаходить його.*

* ***Складність у найкращому випадку****: O(1)*

***Порівняння бінарного та тернарного пошуку***

*-****Бінарний пошук:***

*Бінарний пошук розділяє масив на дві частини на кожному кроці. Кількість порівнянь на кожному кроці — одне (для середнього елемента).*

* ***Найгірший випадок****: O(*n*)*
* ***Найкращий випадок****: O(1)*

***-Тернарний пошук:***

*Тернарний пошук розділяє масив на три частини. Кількість порівнянь на кожному кроці — два (для двох середніх елементів).*

* ***Найгірший випадок****: O(*n*)*
* ***Найкращий випадок****: O(1)*

*Таким чином,* ***бінарний пошук*** *є більш ефективним за* ***тернарний пошук****, навіть попри те, що тернарний пошук розділяє масив на більше число частин.*

1. **Порівняти ефективність алгоритмів лінійного, бінарного та тернарного пошуку для різних розмірів вхідного списку. Для цього провести 30 експериментальне дослідження та побудувати графіки залежності часу виконання алгоритму від розміру вхідного списку.**

Щоб порівняти ефективність алгоритмів **лінійного**, **бінарного** та **тернарного пошуку** для різних розмірів вхідного списку, потрібно провести експериментальні дослідження. Ось загальний підхід до проведення такого дослідження:

**Кроки для дослідження**

1. **Створення алгоритмів пошуку**:
   * **Лінійний пошук**: Алгоритм перевіряє кожен елемент списку по черзі.
   * **Бінарний пошук**: Потрібно відсортувати список, а потім шукати елемент, порівнюючи його з середнім елементом і рекурсивно ділити список навпіл.
   * **Тернарний пошук**: Потрібно відсортувати список, і потім ділити список на три частини замість двох, порівнюючи елемент з двома середніми елементами.
2. **Тестування з різними розмірами вхідного списку**:
   * Потрібно генерувати списки різних розмірів (наприклад, від 10 до 1000 елементів).
   * Для кожного списку виконати всі три алгоритми пошуку (для фіксованого елемента) і заміряти час виконання.
3. **Вимірювання часу виконання**:
   * Для вимірювання часу виконання можна використовувати функції типу time() або timeit() в Python або відповідні методи в інших мовах.
4. **Побудова графіків**:
   * На осі X буде розмір вхідного списку, а на осі Y — час виконання алгоритму.

**Алгоритми**

**1. Лінійний пошук:**

def linear\_search(arr, target):

for i in range(len(arr)):

if arr[i] == target:

return i

return -1

**2. Бінарний пошук:**

def binary\_search(arr, target):

low, high = 0, len(arr) - 1

while low <= high:

mid = (low + high) // 2

if arr[mid] == target:

return mid

elif arr[mid] < target:

low = mid + 1

else:

high = mid - 1

return -1

**3. Тернарний пошук:**

def ternary\_search(arr, target):

low, high = 0, len(arr) - 1

while high >= low:

mid1 = low + (high - low) // 3

mid2 = high - (high - low) // 3

if arr[mid1] == target:

return mid1

elif arr[mid2] == target:

return mid2

elif target < arr[mid1]:

high = mid1 - 1

elif target > arr[mid2]:

low = mid2 + 1

else:

low = mid1 + 1

high = mid2 - 1

return -1

**Тестування:**

Згенеруємо випадкові списки різних розмірів і для кожного списку проведемо всі три пошуки, заміряючи час виконання:

import random

import time

# Генерація списку

def generate\_list(size):

return random.sample(range(size \* 10), size)

# Функція для вимірювання часу виконання пошуку

def time\_search(algorithm, arr, target):

start\_time = time.time()

algorithm(arr, target)

return time.time() - start\_time

# Тестування для різних розмірів списку

sizes = [10, 50, 100, 500, 1000, 5000, 10000]

linear\_times = []

binary\_times = []

ternary\_times = []

for size in sizes:

arr = sorted(generate\_list(size)) # Для бінарного та тернарного пошуку список має бути відсортованим

target = random.choice(arr) # Вибір випадкового елемента для пошуку

# Лінійний пошук

linear\_times.append(time\_search(linear\_search, arr, target))

# Бінарний пошук

binary\_times.append(time\_search(binary\_search, arr, target))

# Тернарний пошук

ternary\_times.append(time\_search(ternary\_search, arr, target))

# Побудова графіків (використовуємо matplotlib для візуалізації)

import matplotlib.pyplot as plt

plt.plot(sizes, linear\_times, label="Лінійний пошук")

plt.plot(sizes, binary\_times, label="Бінарний пошук")

plt.plot(sizes, ternary\_times, label="Тернарний пошук")

plt.xlabel('Розмір списку')

plt.ylabel('Час виконання (сек.)')

plt.title('Порівняння ефективності пошуку для різних алгоритмів')

plt.legend()

plt.show()

**Оцінка результатів:**

1. **Лінійний пошук**: Очікується, що час виконання зростатиме лінійно з ростом розміру списку, оскільки складність алгоритму — O(n)
2. **Бінарний пошук**: Оскільки складність бінарного пошуку — O(log n), час виконання буде зростати значно повільніше в порівнянні з лінійним пошуком, особливо для великих розмірів списку.
3. **Тернарний пошук**: Складність тернарного пошуку також O(log n), але через додаткові порівняння він може бути трохи повільнішим за бінарний пошук на великих розмірах списку.

**Висновок:**

* **Лінійний пошук** працює повільніше за інші алгоритми, особливо на великих списках.
* **Бінарний пошук** значно швидший на великих даних, оскільки його складність —O(log n).
* **Тернарний пошук** також має складність O(log n), але може бути менш ефективним за бінарний через додаткові операції порівняння.

**5. Порівняти алгоритми пошуку за їхньою здатністю працювати з відсортованими та не відсортованими списками. Провести аналіз впливу відсортованості списку на час виконання кожного алгоритму.**

**Порівняння алгоритмів пошуку**

Існує три основні алгоритми пошуку, про які йдеться в цьому порівнянні: **лінійний пошук**, **бінарний пошук** і **тернарний пошук**. Кожен з них має свої особливості в залежності від того, чи відсортований список, з яким працює алгоритм.

**1. Лінійний пошук**

**Як працює:** Лінійний пошук просто перевіряє кожен елемент списку по черзі, поки не знайде потрібний елемент або не пройде весь список.

* **Не відсортовані списки**: Лінійний пошук працює однаково ефективно для будь-якого списку, будь то відсортований чи ні, бо він просто перебирає всі елементи. Час його роботи не залежить від порядку елементів у списку.
* **Відсортовані списки**: Якщо список відсортований, це не впливає на ефективність лінійного пошуку. Алгоритм все одно перебирає елементи один за одним, тому складність залишається O(n).

**Підсумок:** Лінійний пошук хороший, коли список не дуже великий, або якщо список не відсортований. Однак він працює повільніше, коли розмір списку великий.

**2. Бінарний пошук**

**Як працює:** Бінарний пошук працює тільки з відсортованими списками. Він постійно ділить список навпіл, порівнюючи шукане значення з середнім елементом, і в залежності від результату порівняння, шукає далі в одній з половин списку.

* **Не відсортовані списки**: Для застосування бінарного пошуку список потрібно відсортувати. Тому для не відсортованих списків спочатку потрібно виконати сортування, що додасть додатковий час роботи алгоритму O(nlogn), а потім вже застосувати бінарний пошук, що дасть складність O(logn). В цілому, загальна складність для не відсортованих списків — це O(nlogn).
* **Відсортовані списки**: Бінарний пошук дуже ефективний, коли список вже відсортований. Тому час пошуку буде дуже швидким O(logn).

**Підсумок:** Бінарний пошук — це найкращий вибір для відсортованих списків, бо має дуже швидку складність O(logn), в той час як для не відсортованих списків потрібно спочатку сортувати, що робить його менш ефективним.

**3. Тернарний пошук**

**Як працює:** Тернарний пошук схожий на бінарний, але замість того, щоб ділити список на дві частини, він ділить його на три частини. Порівнює шукане значення з двома середніми елементами і далі працює в одній з трьох частин.

* **Не відсортовані списки**: Як і бінарний пошук, тернарний пошук вимагає попереднього сортування списку. Це означає, що для не відсортованих списків, як і в випадку з бінарним пошуком, необхідно спочатку відсортувати список. Загальна складність буде O(nlogn) для сортування і O(logn) для пошуку.
* **Відсортовані списки**: Для відсортованих списків тернарний пошук працює так само, як бінарний, тільки з додатковими порівняннями, тому він трохи повільніший за бінарний пошук.

**Підсумок:** Тернарний пошук менш ефективний за бінарний пошук, хоча і має складність O(logn). Додаткові порівняння роблять його трохи повільнішим, ніж бінарний пошук.

**Висновки:**

1. **Лінійний пошук** — це універсальний алгоритм, який не залежить від того, чи відсортований список. Однак він є менш ефективним, особливо для великих списків, оскільки має складність O(n).
2. **Бінарний і тернарний пошуки** — набагато швидші, ніж лінійний пошук, але тільки в **відсортованих списках**. Бінарний пошук є кращим за тернарний, оскільки він працює з меншими витратами на порівняння.
3. Для **не відсортованих списків** лінійний пошук залишається найкращим варіантом, оскільки бінарний і тернарний пошуки вимагають попереднього сортування списку, що збільшує загальну складність.

Отже, ефективність кожного алгоритму сильно залежить від того, чи є список відсортованим. Якщо список відсортований, бінарний пошук буде найшвидшим, а для не відсортованих списків лінійний пошук буде найпростішим і найшвидшим вибором.

**6. Розглянути сценарії використання кожного з алгоритмів пошуку у практичних задачах і обґрунтувати вибір кожного алгоритму в конкретному випадку.**

У практичних задачах вибір алгоритму пошуку залежить від ряду факторів, таких як розмір даних, тип даних (відсортовані чи не відсортовані), вимоги до швидкості виконання, частота оновлення даних тощо. Розглянемо конкретні сценарії, в яких доцільно використовувати **лінійний пошук**, **бінарний пошук** та **тернарний пошук**.

**1. Лінійний пошук**

**Сценарії використання:**

* **Не відсортовані дані**: Лінійний пошук є універсальним і підходить для пошуку в не відсортованих списках. Наприклад, у базах даних або великих масивах даних, які не мають чіткої структури або потребують швидкої перевірки на наявність елемента без попереднього сортування.
* **Малий обсяг даних**: Якщо розмір списку малий або число елементів незначне, то різниця в ефективності між лінійним та іншими алгоритмами може бути незначною, тому простота лінійного пошуку буде перевагою.
* **Часті зміни списку**: Лінійний пошук корисний, коли список часто змінюється, і сортування кожного разу є надмірним. Наприклад, у сценаріях, де дані постійно оновлюються або додаються нові елементи, і немає часу чи потреби для сортування перед пошуком.

**Приклад:**

* Пошук елемента в базі даних, де записи часто оновлюються, і база не сортується за жодним критерієм.

**2. Бінарний пошук**

**Сценарії використання:**

* **Відсортовані дані**: Бінарний пошук підходить для пошуку в відсортованих списках. Якщо дані вже відсортовані (наприклад, у базі даних або в масиві), бінарний пошук дозволяє значно прискорити пошук, порівняно з лінійним, оскільки складність бінарного пошуку становить O(logn), що значно швидше для великих обсягів даних.
* **Пошук в статичних або рідко змінюваних даних**: Якщо дані не часто змінюються, а пошук потрібен регулярно, бінарний пошук є оптимальним варіантом, оскільки один раз відсортований список дозволяє значно скоротити час пошуку.
* **Інтерфейси пошуку в великих колекціях даних**: Наприклад, в електронних таблицях або пошукових системах, де дані відсортовані і потрібно часто шукати елементи.

**Приклад:**

* Пошук книги в бібліотеці, де книги розташовані за алфавітним порядком, або пошук конкретного елемента у відсортованому списку товарів на сайті.

**3. Тернарний пошук**

**Сценарії використання:**

* **Відсортовані дані з великою кількістю елементів**: Тернарний пошук може бути корисним у відсортованих списках, коли треба зробити пошук, поділяючи дані на три частини замість двох, як це робить бінарний пошук. Однак через додаткові порівняння тернарний пошук може бути менш ефективним, ніж бінарний. Але в деяких випадках, наприклад, у великих базах даних або пошукових системах, коли важлива не тільки швидкість, але й оптимізація кількості порівнянь, тернарний пошук може бути корисним.
* **Інтерфейси для пошуку в структурованих даних**: Якщо список дуже великий і в ньому кілька тисяч або мільйонів елементів, тернарний пошук може бути кращим варіантом для розподілених систем або систем, де кількість порівнянь є важливим параметром.
* **Моделі, які потребують трьох груп порівнянь**: У специфічних сценаріях, коли потрібно робити поділ на три частини (наприклад, для пошуку в трирівневих структурах або в певних типах баз даних), тернарний пошук може бути доцільним.

**Приклад:**

* Пошук у трирівневих ієрархічних базах даних, де кожен рівень має кілька можливих варіантів для поділу даних.

**Порівняння сценаріїв використання:**

* **Лінійний пошук**: Краще застосовувати для **не відсортованих** даних або коли часто змінюється склад списку. Він простий і ефективний для малих даних, але менш ефективний для великих обсягів.
* **Бінарний пошук**: Ідеальний для **відсортованих** даних, де потрібно швидко знайти елемент. Це найшвидший варіант для **статичних списків**, де дані не змінюються або змінюються рідко.
* **Тернарний пошук**: Може бути корисним для специфічних задач, де потрібно поділити список на три частини, але для більшості звичайних випадків бінарний пошук буде кращим.

**Висновок**

1. **Лінійний пошук** є кращим для невеликих або не відсортованих даних, де сортування не є ефективним або взагалі не потрібне.
2. **Бінарний пошук** ідеально підходить для відсортованих даних, особливо коли потрібно часто виконувати пошук на великих наборах даних.
3. **Тернарний пошук** може бути корисним для специфічних задач, але зазвичай бінарний пошук буде працювати швидше через меншу кількість порівнянь.

**Контрольні питання**

1. **Що таке алгоритм пошуку і чому він важливий у контексті комп'ютерних наук?**

***Алгоритм пошуку*** *— це набір інструкцій, які використовуються для знаходження елементів у структурі даних, як-от масив, список або дерево. Алгоритми пошуку допомагають знайти конкретну інформацію в даних або визначити, чи існує потрібний елемент. Вони важливі в комп'ютерних науках, оскільки пошук є базовою операцією в багатьох програмах, базах даних, веб-пошукових системах, а також у програмуванні загалом.*

2. **Які основні критерії оцінки ефективності алгоритмів пошуку?**

*Основні критерії ефективності алгоритмів пошуку:*

* ***Часова складність****: Вимірює, скільки часу алгоритм потребує для виконання в залежності від розміру вхідних даних. Вимірюється в O-нотації.*
* ***Просторова складність****: Вимірює, скільки пам'яті використовує алгоритм під час виконання.*
* ***Універсальність****: Наскільки широко алгоритм можна застосовувати до різних типів даних і в різних ситуаціях.*
* ***Простота реалізації****: Наскільки легко і зрозуміло реалізувати алгоритм на практиці.*

3. **Що таке лінійний пошук, і як він працює?**

***Лінійний пошук*** *— це алгоритм пошуку, який перевіряє кожен елемент у списку або масиві один за одним, поки не знайде шуканий елемент або не завершить перевірку всіх елементів. Алгоритм працює за принципом порівняння кожного елемента з шуканим, починаючи з першого. Якщо елемент знайдено, алгоритм зупиняється. Якщо ж усі елементи перевірено, а шуканого немає, алгоритм завершується.*

***Часова складність****: O(n), де n — кількість елементів у списку.*

4. **Які умови повинні бути виконані для успішного застосування бінарного пошуку?**

*Для успішного застосування* ***бінарного пошуку*** *повинні бути виконані наступні умови:*

* *Масив (або інша структура даних) повинен бути* ***відсортованим****.*
* *Пошук відбувається шляхом розподілу масиву на дві частини, і пошук продовжується в тій частині, де може знаходитися шуканий елемент. Це дає змогу значно зменшити кількість перевірок.*

***Часова складність****: O(log n).*

5. **Які переваги та недоліки використання бінарного пошуку порівняно з іншими алгоритмами пошуку?**

***Переваги****:*

* ***Швидкість****: Бінарний пошук має значно кращу часову складність O(log n) порівняно з лінійним пошуком (O(n)), особливо для великих обсягів даних.*
* ***Ефективність****: При наявності відсортованих даних бінарний пошук є дуже ефективним.*

***Недоліки****:*

* ***Відсортованість даних****: Бінарний пошук працює лише з відсортованими даними, тому перед його використанням часто необхідно спочатку відсортувати дані, що може бути дорогим процесом для великих наборів даних.*
* ***Мінімальна підтримка для змінних даних****: Якщо дані часто змінюються, необхідно постійно підтримувати їх відсортованість, що може бути неефективно.*

6. **Що таке тернарний пошук, і в чому його відмінність від бінарного пошуку?**

***Тернарний пошук*** *— це варіант бінарного пошуку, при якому масив або список розділяється на три частини, а не на дві, як у бінарному пошуку. На кожному кроці алгоритм перевіряє два елементи (наприклад, в точках n3\frac{n}{3}3n​ і 2n3\frac{2n}{3}32n​), щоб визначити, в якій третині масиву продовжувати пошук.*

***Відмінності від бінарного пошуку****:*

* *Бінарний пошук розділяє масив на 2 частини, а тернарний — на 3.*
* *Бінарний пошук здійснює 1 порівняння на крок, а тернарний — 2 порівняння.*
* *Тернарний пошук може бути корисним для специфічних задач, де треба зменшити діапазон на 3 частини замість 2, але в загальному випадку він не дає значної переваги за ефективністю перед бінарним пошуком.*

***Часова складність****:*

* *Для* ***бінарного пошуку****: O(log n)*
* *Для* ***тернарного пошуку****: O(log₃ n)*

*Хоча тернарний пошук може зменшити кількість кроків, він зазвичай менш ефективний в плані часу, оскільки кожен крок вимагає додаткових порівнянь.*