МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

Навчалньно-науковий інститут електричної інженерії та інформаційних технологій

Кафедра автоматизації та інформаційних систем

**ПРАКТИЧНА РОБОТА**

Виконав: студент групи КІ-23-1

Черниш В’ячеслав Олександрович

Перевірив:

Сидоренко Валерій Миколайович

м. Кременчук

2024 рік

**Практична робота № 4**

**Тема. Алгоритми пошуку та їх складність.**

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

**Хід роботи**

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

Асимптотична складність алгоритму лінійного пошуку:

- найгірший випадок. Лінійний пошук перебирає всі елементи масиву один за одним, доки не знайде потрібний елемент або не переконається, що його немає. У найгіршому випадку елемент знаходиться в кінці масиву, або його немає в масиві взагалі. Тоді потрібно перевірити всі *n* елементів. Складність: *O*(*n*);

- найкращий випадок. У найкращому випадку шуканий елемент знаходиться на першій позиції масиву, тому виконується лише одна перевірка. Складність: *O*(1).

Покращення алгоритму лінійного пошуку:

- підготувати дані (якщо можливо);

- застосувати експоненційний пошук для частково впорядкованих даних;

- інтерактивний підхід до оптимізації (знаючи ймовірність пошуку певних елементів);

- індексація або хешування;

- використання оптимізованих структур даних.

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

Алгоритм бінарного пошуку працює на впорядкованому масиві, розділяючи його навпіл на кожній ітерації.

Найгірший випадок.У найгіршому випадку шуканий елемент або знаходиться в кінці серії поділів, або його взагалі немає в масиві.Кожна ітерація скорочує розмір масиву вдвічі. Тобто, кількість ітерацій дорівнює кількості поділів *n*, поки масив не зменшиться до одного елемента.Максимальна кількість ітерацій *k* визначається як: **.** Отже складність у найгіршому випадку: .

Найкращий випадок. У найкращому випадку елемент одразу знаходиться посередині масиву на першій ітерації. Потрібна лише одна перевірка. Складність у найкращому випадку .

Алгоритм бінарного пошуку ефективний завдяки зменшенню області пошуку вдвічі на кожній ітерації, що значно вигідніше, ніж лінійний пошук, особливо на великих масивах.

**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 використовується на впорядкованих масивах і поділяє масив на три частини на кожній ітерації. Це схоже на бінарний пошук, але порівняння виконується з двома елементами, що ділять масив на три частини.

Псевдокод алгоритму тернарного пошуку:

- вхід: впорядкований масив A*A* довжиною n*n*, елемент x*x*, який потрібно знайти;

- вихід: індекс елемента x*x* (або −1−1, якщо елемента немає);

- процес:

function ternarySearch(A, left, right, x):

while left ≤ right:

mid1 = left + (right - left) / 3

mid2 = right - (right - left) / 3

if A[mid1] == x:

return mid1

if A[mid2] == x:

return mid2

if x < A[mid1]:

right = mid1 - 1

else if x > A[mid2]:

left = mid2 + 1

else:

left = mid1 + 1

right = mid2 - 1

return -1

Оцінка асимптотичної складності:

- найгірший випадок. Масив ділиться на три частини на кожній ітерації. На *k*-му кроці залишиться: . Зупинка відбудеться, коли масив зменшиться до одного елемента: ⇒ . У тернальному пошуку складність у найгіршому випадку , але з більшою основою логарифма порівняно з бінарним пошуком;

- найкращий випадок. Якщо шуканий елемент знаходиться в одній із двох точок поділу (mid1 або mid2), пошук завершується за одну ітерацію. Складність: .

Порівняння Тернального та бінарного пошуку:

- кількість ітерацій. Бінарний пошук ділить масив навпіл, тобто: . Тернальний пошук ділить масив на три частини: . Оскільки , тернарний пошук потребує більше ітерацій, ніж бінарний;

- кількість порівнянь. У бінарному пошуку - 1 порівняння за ітерацію. У тернарному пошук - 2 порівняння за ітерацію;

- обґрунтування оптимальності. Бінарний пошук має меншу кількість ітерацій та порівнянь, тому він оптимальніший з точки зору кількості операцій. Тернарний пошук не дає переваг у впорядкованих масивах і працює повільніше через додаткові порівняння.

Бінарний пошук є оптимальнішим за тернарний, оскільки потребує менше ітерацій і виконує менше порівнянь, зберігаючи таку саму асимптотичну складність .

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

Будуємо графік залежності часу виконання алгоритмів пошуку.

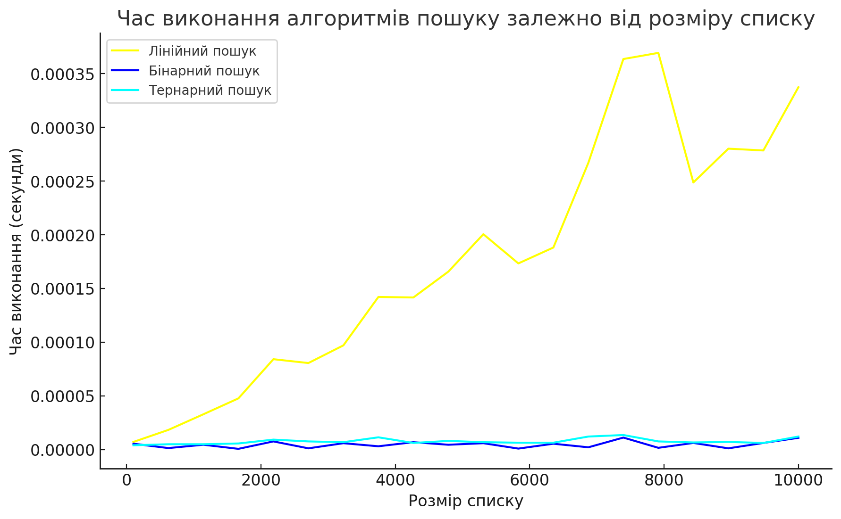


Рисунок 1 – залежность лінійного, бінарного та тернарного алгоритмів пошуку від розміру вхідного списку

Аналіз результатів:

- лінійний пошук. Має найгіршу ефективність, оскільки час виконання лінійно збільшується зі зростанням розміру списку. Найкраще підходить лише для невеликих списків;

- бінарний пошук. Значно швидше за лінійний пошук навіть для невеликих списків. Час виконання збільшується логарифмічно розміру списку;

- тернарний пошук. Дуже схожий на бінарний пошук, але трохи повільніший через додаткові порівняння на кожній ітерації. Також має логарифмічну залежність від розміру списку.

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

**.** Вплив відсортованості списку.

Відсортований список:

- лінійний пошук. Не отримує значної переваги, оскільки перевіряє кожен елемент незалежно від впорядкованості;

- бінарний і тернарний пошук. Працюють дуже ефективно, зменшуючи кількість елементів для перевірки на кожному кроці.

Невідсортований список:

- лінійний пошук. Єдиний алгоритм, який працює коректно;

- бінарний і тернарний пошук. Не можуть використовувати невідсортований список без попереднього впорядкування.

Аналіз впливу відсортованості списку на час виконання кожного алгоритму.

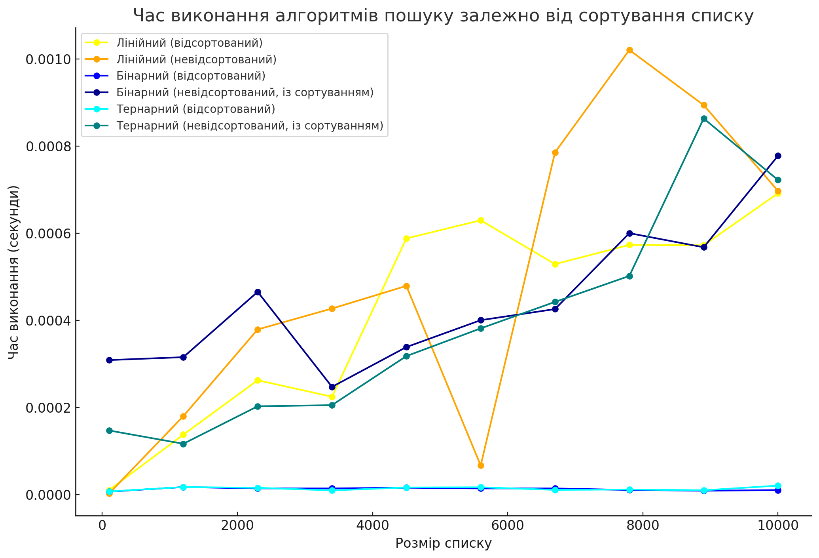


Рисунок 2 – Графік залежності часу виконання алгоритмів пошуку

- лінійний пошук (невідсортований список). Має найгірший час, оскільки перебирає всі елементи списку, незалежно від його сортування;

- лінійний пошук (відсортований список). Не відрізняється від невідсортованого, бо принцип роботи однаковий;

- бінарний і тернарний пошуки: значно швидші, але працюють лише на відсортованих списках. Тернарний пошук зазвичай трохи повільніший за бінарний через додаткові порівняння на кожному кроці.

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

Кожен алгоритм пошуку має свої переваги й підходить для різних завдань.Лінійний пошук:

**Лінійний пошук**:

**Сценарій**: Пошук у невідсортованих даних.

**Приклад**: Виявлення імені в випадковому списку учасників; знаходження певного числа в потокових даних або несортованій базі.

**Обґрунтування вибору**: Дані не впорядковані, а сортування вимагатиме додаткових ресурсів і часу. Алгоритм простий у реалізації, не потребує додаткової пам’яті чи підготовки.

**Коли використовується**: Підходить для невеликих обсягів даних або для одноразового пошуку без необхідності сортування.

**Бінарний пошук**:

**Сценарій**: Пошук у відсортованих даних.

**Приклад**: Знаходження ISBN у відсортованому списку книжок; перевірка наявності товару з певною ціною в базі; пошук у числових рядах для алгоритмічних задач.

**Обґрунтування вибору**: Ефективний при роботі з упорядкованими даними, особливо коли вони використовуються багаторазово. Складність алгоритму O(logn) значно перевершує лінійний пошук для великих списків.

**Коли використовується**: Ідеальний для великих масивів, які вже впорядковані або можуть бути відсортовані заздалегідь.

**Тернарний пошук**:

**Сценарій**: Пошук у відсортованих даних або задачах оптимізації.

**Приклад**: Пошук глобального мінімуму/максимуму в унімодальних функціях; пошук у відсортованих даних у випадках, коли мінімізація порівнянь критично важлива (наприклад, через апаратні обмеження).

**Обґрунтування вибору**: Використовує три розділи замість двох, як у бінарному пошуку. Це корисно в задачах оптимізації або специфічних умовах, де поділ на три частини підвищує ефективність.

**Коли використовується**: Найбільше підходить для задач оптимізації, де потрібно працювати з діапазонами значень або функцій.

**Контрольні питання**

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

Алгоритм пошуку це процедура знаходження елемента в структурі даних. Він важливий для ефективного доступу до даних у багатьох програмах, таких як бази даних, пошукові системи тощо.

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

Критерії ефективності: час виконання (асимптотична складність), використання пам’яті, здатність працювати з різними типами даних.

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

Лінійний пошук: проходить по кожному елементу списку від початку до кінця, доки не знайде потрібний елемент або не перегляне всі елементи.

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

Для успішного застосування бінарного пошуку повинні бути виконані певні умови:

1. дані мають бути відсортовані;
2. дані мають підтримувати довільний доступ (random access);
3. дані мають бути статичними під час пошуку;
4. задане значення порівнюване з елементами масиву;
5. крайні значення в межах масиву;
6. відсутність дублікатів (для точного пошуку індекса).

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

|  |  |  |
| --- | --- | --- |
| Алгоритм пошуку | Переваги | Недоліки |
| Бінарний пошук | Швидкий для великих відсортованих масивів. | Потрібна сортування, не підходить для динамічних даних. |
| Лінійний пошук | Простий, працює з будь-якими даними. | Повільний для великих даних, потрібно перевіряти всі елементи. |
| Тернальний пошук | Кращий для дуже специфічних випадків, наприклад, пошук унікальних елементів. | Менш відомий, вимагає спеціальних умов. |

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

Тернарний пошук — це алгоритм пошуку, схожий на бінарний пошук, але він розділяє пошуковий простір на три частини (замість двох), що дозволяє зменшити кількість порівнянь на кожному етапі. Цей пошук також працює тільки з відсортованими даними і зазвичай застосовується до масивів чи списків.

|  |  |  |
| --- | --- | --- |
| Критерій | Бінарний пошук | Тернарний пошук |
| Кількість поділів | Розділяє масив на дві частини. | Розділяє масив на три частини. |
| Порівняння на кожному кроці | Один елемент (середній). | Два елементи (перший і другий середній). |
| Кількість кроків | Зменшується вдвічі на кожному кроці. | Зменшується на третину на кожному кроці. |
| Простота реалізації | Легший для реалізації. | Складніший в реалізації через наявність двох середніх елементів. |
| Використання | Широко використовується для відсортованих даних. | Зазвичай застосовується для специфічних задач, де кожен крок має сенс для поділу на три частини (наприклад, для оптимізації деяких функцій). |