**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?

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

**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.

**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.

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

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

**Хід роботи**

**1.** 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.** 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.** Алгоритм тернарного пошуку. 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 за ітерацію. 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 . Тернарний пошук може бути корисним у специфічних задачах, але для загальних випадків бінарний пошук є більш ефективним.

**4.** Будуємо графік залежності часу виконання алгоритмів пошуку.

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание

Рисунок 4.1 – Графік залежності 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.** 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.

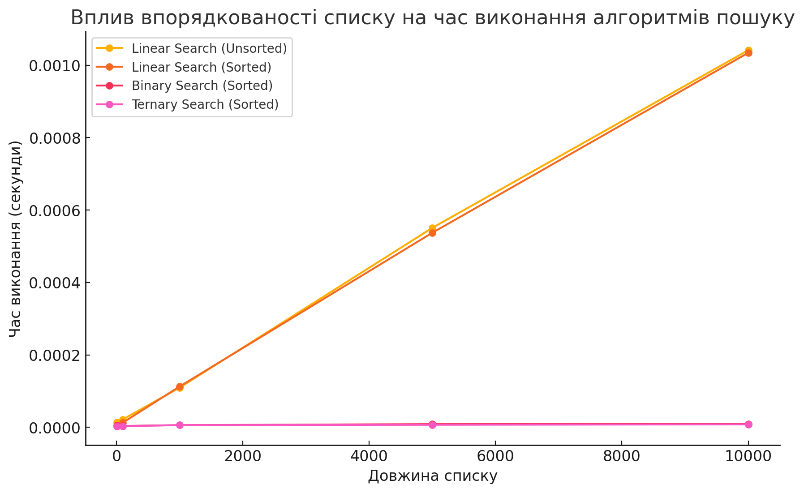


Рисунок 4.2 – Графік залежності часу виконання алгоритмів пошуку

- 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.** Кожен алгоритм пошуку має свої сильні сторони й підходить для різних типів задач. Нижче наведені практичні сценарії використання лінійного, бінарного та тернарного пошуку з обґрунтуванням вибору.

Лінійний пошук:

- сценарій. Пошук елемента в невідсортованих даних;

- приклад задачі. Пошук імені в списку зареєстрованих учасників, отриманому у випадковому порядку; пошук певного числа в потокових даних або невідсортованій базі даних;

- чому вибрати цей алгоритм. Дані не відсортовані, тому сортування потребуватиме додаткових ресурсів і часу;

- простота реалізації. Не вимагає додаткової пам'яті або структур;

- підходить для невеликих обсягів даних або коли сортування не потрібне після виконання завдання.

Бінарний пошук:

- сценарій. Пошук у відсортованих даних;

- приклад задачі. Пошук ISBN-коду в відсортованому списку книжок; перевірка, чи існує продукт з певною ціною в базі даних цін; пошук значень у масиві, наприклад, для задач алгоритмів на числових рядах;

- чому вибрати цей алгоритм. Дані вже відсортовані або можуть бути відсортовані один раз і повторно використані для пошуку;

- час виконання значно кращий за лінійний пошук при великих обсягах;

- реалізація достатньо проста, а ефективність висока.

Тернарний пошук:

- сценарій. оптимізація пошуку у відсортованих даних або задачах, які потребують максимального поділу;

- приклад задачі. Оптимізація пошуку глобального мінімуму/максимуму в унімо-дальних функціях; пошук елементів у відсортованих даних, коли кількість порівнянь має бути мінімізована (наприклад, у специфічних апаратних обмеженнях);

- чому вибрати цей алгоритм. Працює подібно до бінарного пошуку, але використовує три розділи, що може бути ефективнішим у ряді специфічних випадків;

- особливо корисний у задачах оптимізації, де пошук йде по діапазону чисел або значень функцій.

**Контрольні питання**

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

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

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

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

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

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

**1.** **Алгоритм пошуку** — це послідовність дій (правил), що визначає, як знайти потрібний елемент (або підтвердити його відсутність) у наборі даних. Цей набір може бути списком, масивом, графом, деревом або іншою структурою даних.

У комп'ютерних науках алгоритми пошуку є фундаментальними через такі причини:

- основна операція з даними. Більшість задач, пов’язаних із обробкою інформації;

- оптимізація продуктивності. Ефективний пошук — ключ до швидкої обробки даних, особливо у великих масштабах;

- різноманітність задач. Алгоритми пошуку використовуються в багатьох галузях;

- використання у складніших алгоритмах. Пошук є базовою операцією для багатьох інших алгоритмів.

**2.** Оцінка ефективності алгоритмів пошуку базується на кількох ключових критеріях. Вибір відповідного алгоритму залежить від завдання, типу даних і умов, у яких він використовується. Основні критерії:

- часова складність;

- просторова складність;

- стійкість до змін у вхідних даних;

- складність реалізації;

- гнучкість та універсальність;

- масштабованість;

- час виконання для різних операцій.

**3.** Лінійний пошук (або послідовний пошук) — це простий алгоритм пошуку, який перевіряє кожен елемент списку чи масиву, порівнюючи його із заданим значенням, доки не знайде відповідний елемент або не досягне кінця списку.

Робота лінійного пошуку:

- почати з першого елемента списку;

- порівняти поточний елемент зі значенням, яке потрібно знайти;

- якщо поточний елемент дорівнює значенню пошуку, повернути його індекс (або елемент);

- якщо поточний елемент не дорівнює значенню пошуку, перейти до наступного елемента;

- продовжувати до кінця списку;

- якщо елемент не знайдений, повернути результат, що елемент відсутній.

**4.** Для успішного застосування бінарного пошуку повинні бути виконані певні умови:

- дані мають бути відсортовані;

- дані мають підтримувати довільний доступ (random access);

- дані мають бути статичними під час пошуку;

- задане значення порівнюване з елементами масиву;

- крайні значення в межах масиву;

- відсутність дублікатів (для точного пошуку індекса).

**5.** Порівняння бінарного пошуку з іншими алгоритмами пошуку:

Таблиця 4.1

|  |  |  |
| --- | --- | --- |
| 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.** 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.

Таблиця 4.2 – Основні відмінності між бінарним та тернарним пошуком

|  |  |  |
| --- | --- | --- |
| **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). |