Міністерство освіти і науки України

Кременчуцький національний університет   
імені Михайла Остроградського

Навчально-науковий інститут електричної інженерії   
та інформаційних технологій

Кафедра автоматизації та інформаційних систем

НаВчальна дисципліна  
«**аЛГОРИТМИ І СТРУКТУРИ ДАННИХ**»

Звіт

З ПРАКТИЧНОЇ роботи №4

Виконала

студентка групи КН-24-1

Процко П.П.

Перевірив

доцент кафедри КІЕ

Сидоренко В. М.

Кременчук 2025

|  |  |
| --- | --- |
| Тема: | Алгоритми пошуку та їх складність. |
| Мета: | опанувати основні алгоритми сортування та навчитись методам  аналізу їх асимптотичної складності. |

Хід роботи

Постановка завдання

Виконати індивідуальне завдання. Завдання полягає у розв’язанні всіх наведених задач.

Розв’язання задач

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 алгоритму лінійного пошуку:

* Найгірший випадок: Елемент відсутній у масиві або розташований в самому кінці. Алгоритм перевіряє всі n елементів. Отже, асимптотична складність: O(n).
* Найкращий випадок: Елемент знаходиться на першій позиції. Алгоритм виконує лише одну перевірку. Отже, складність: O(1).

Як покращити лінійний пошук?

* Використання відсортованого масиву: Якщо масив відсортований і ми шукаємо елемент, який менший за поточний, можна зупинити пошук раніше (але це працює лише для відсортованих даних).
* Паралелізація: Розбиття масиву на частини та пошук у кількох потоках може зменшити реальний час виконання на великих наборах даних, хоча асимптотична складність залишиться O(n).

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

* Найгірший випадок: Елемент відсутній або знаходиться в крайній точці після всіх поділів. Кількість кроків дорівнює , оскільки масив ділиться навпіл. Отже, складність: .
* Найкращий випадок: Елемент знаходиться в середині масиву на першому кроці. Складність: O(1)

1. 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  
def ternary\_search(arr, target):  
 left, right = 0, len(arr) - 1  
 while left <= right:  
 third = (right - left) // 3  
 mid1 = left + third  
 mid2 = right - third  
  
 if arr[mid1] == target:  
 return mid1  
 if arr[mid2] == target:  
 return mid2  
  
 if target < arr[mid1]:  
 right = mid1 - 1  
 elif target > arr[mid2]:  
 left = mid2 + 1  
 else:  
 left = mid1 + 1  
 right = mid2 - 1  
 return -1

* Найгірший випадок: На кожному кроці масив зменшується до від попереднього розміру. Рівняння рекурентності: . Розв’язок дає оскільки .
* Найкращий випадок: Елемент знаходиться в одній із точок поділу на першому кроці. Складність: O(1).

Порівняння бінарного та тернарного пошуку:

* Бінарний пошук: , де .
* Тернарний пошук: , де .
* Константа в більша, ніж у , але на кожному кроці тернарний пошук виконує більше порівнянь (2 замість 1). У результаті бінарний пошук зазвичай ефективніший, бо зменшує простір пошуку швидше з меншою кількістю операцій.

Висновок: Бінарний пошук є оптимальнішим, оскільки за абсолютним значенням, а кількість порівнянь на крок менша.

1. 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 (рис. Рисунок 1).

import time  
import numpy as np  
import matplotlib.pyplot as plt  
  
# Лінійний пошук  
def linear\_search(arr, target):  
 for i in range(len(arr)):  
 if arr[i] == target:  
 return i  
 return -1  
  
# 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  
  
# Тернарний пошук  
def ternary\_search(arr, target):  
 left, right = 0, len(arr) - 1  
 while left <= right:  
 third = (right - left) // 3  
 mid1 = left + third  
 mid2 = right - third  
  
 if arr[mid1] == target:  
 return mid1  
 if arr[mid2] == target:  
 return mid2  
  
 if target < arr[mid1]:  
 right = mid1 - 1  
 elif target > arr[mid2]:  
 left = mid2 + 1  
 else:  
 left = mid1 + 1  
 right = mid2 - 1  
 return -1  
  
# Розміри масивів для тесту  
n\_values = [10, 100, 1000, 10\_000, 100\_000, 1\_000\_000]  
times\_linear = []  
times\_binary = []  
times\_ternary = []  
  
# Проведення експерименту  
for n in n\_values:  
 arr = sorted(np.random.randint(0, 10\*\*7, size=n))  
 target = arr[-1] # останній елемент (гірший випадок для лінійного пошуку)  
  
 # Лінійний пошук  
 start = time.time()  
 linear\_search(arr, target)  
 times\_linear.append(time.time() - start)  
  
 # Бінарний пошук  
 start = time.time()  
 binary\_search(arr, target)  
 times\_binary.append(time.time() - start)  
  
 # Тернарний пошук  
 start = time.time()  
 ternary\_search(arr, target)  
 times\_ternary.append(time.time() - start)  
  
# Побудова графіків  
plt.figure(figsize=(10, 5))  
plt.plot(n\_values, times\_linear, label="Лінійний пошук", marker='o')  
plt.plot(n\_values, times\_binary, label="Бінарний пошук", marker='s')  
plt.plot(n\_values, times\_ternary, label="Тернарний пошук", marker='^')  
plt.xlabel("Розмір списку (n)")  
plt.ylabel("Час виконання (секунди)")  
plt.title("Порівняння часу пошуку")  
plt.legend()  
plt.grid(True)  
plt.xscale("log")  
plt.yscale("log")  
plt.show()

Зображення, що містить текст, ряд, схема, Графік

Вміст, створений ШІ, може бути неправильним.

Рисунок 1 – Графік залежності часу від розміру

Висновки:

* Найповільніший — лінійний пошук (час зростає лінійно).
* Бінарний і тернарний пошук набагато швидші (логарифмічна складність).
* Тернарний пошук трохи повільніший за бінарний через більше порівнянь, але перевага мізерна, і незначна при великих обсягах даних.

Лінійний пошук:

* Найгірший: O(n).
* Найкращий: O(1)

Бінарний пошук:

* Найгірший:
* Найкращий: O(1)

Тернарний пошук:

* Найгірший:
* Найкращий: O(1)

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

* Лінійний пошук: Працює однаково на відсортованих і невпорядкованих списках — O(n).
* Бінарний пошук: Вимагає відсортованість. На невпорядкованих списках не працює коректно. Час сортування додається, якщо список невпорядкований.
* Тернарний пошук: Аналогічно бінарному, вимагає відсортованість.

Аналіз: Для невпорядкованих даних лінійний пошук є єдиним варіантом. Для відсортованих — бінарний і тернарний значно швидші.

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

* Лінійний пошук: Маленькі списки (<100 елементів), невпорядковані дані, однократний пошук.
* Бінарний пошук: Великі відсортовані списки (наприклад, індекси баз даних), де потрібна висока швидкість.
* Тернарний пошук: Рідко використовується, але може бути корисним у задачах оптимізації унімодальних функцій (наприклад, пошук максимуму).

Відповіді на контрольні питання

Контрольні питання

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

Це метод знаходження певного елемента в наборі даних. Важливий у комп'ютерних науках для обробки даних, оптимізації запитів і підвищення ефективності програм.

1. Які основні критерії оцінки ефективності алгоритмів пошуку?

Часова складність, просторова складність, адаптивність до типу даних, вимоги до попередньої обробки (наприклад, сортування).

1. Що таке лінійний пошук, і як він працює?

Лінійний пошук перевіряє кожен елемент масиву послідовно, доки не знайде ціль або не дійде до кінця.

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

Масив має бути відсортованим у порядку зростання або спадання.

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

* Переваги: Швидкість, простота реалізації.
* Недоліки: Вимагає сортування, не працює з динамічними даними без додаткових структур.

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

Тернарний пошук ділить масив на три частини замість двох. Відмінність від бінарного — більше порівнянь на крок, але менша глибина рекурсії.