Звіт

Практична робота №4

Варіант 9

Тема: Алгоритми пошуку та їх складність

Постановка завдань:

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.

1. Лінійний пошук

Асимптотична складність:

- Найгірший випадок: O(n) , де n — кількість елементів у списку.

- Найкращий випадок: O(1) , коли елемент знаходиться на першій позиції.

Покращення лінійного пошуку:

- Використання сентинельного елементу.

- Паралельний пошук на багатоядерних процесорах.

Бінарний пошук

Асимптотична складність:

- Найгірший випадок: O( log n) , де n — кількість елементів у списку.

- Найкращий випадок: O(1) , коли елемент знаходиться на середній позиції.

Тернарний пошук

def ternary\_search(arr, x, left, right):

if left > right:

return -1

one\_third = left + (right - left) // 3

two\_third = right - (right - left) // 3

if arr[one\_third] == x:

return one\_third

if arr[two\_third] == x:

return two\_third

if x < arr[one\_third]:

return ternary\_search(arr, x, left, one\_third - 1)

elif x > arr[two\_third]:

return ternary\_search(arr, x, two\_third + 1, right)

else:

return ternary\_search(arr, x, one\_third + 1, two\_third - 1)

Асимптотична складність:

- Найгірший випадок: O(log\_3 n)

- Найкращий випадок: O(1)

Порівняння:

- Бінарний пошук ефективніший, оскільки кількість порівнянь на крок менша.

Експериментальне дослідження

import time

import numpy as np

import matplotlib.pyplot as plt

def linear\_search(arr, x):

for i in range(len(arr)):

if arr[i] == x:

return i

return -1

def bin\_search(arr, x, left, right):

if left > right:

return -1

middle = (left + right) // 2

if arr[middle] == x:

return middle

elif arr[middle] < x:

return bin\_search(arr, x, middle + 1, right)

else:

return bin\_search(arr, x, left, middle - 1)

def ternary\_search(arr, x, left, right):

if left > right:

return -1

one\_third = left + (right - left) // 3

two\_third = right - (right - left) // 3

if arr[one\_third] == x:

return one\_third

if arr[two\_third] == x:

return two\_third

if x < arr[one\_third]:

return ternary\_search(arr, x, left, one\_third - 1)

elif x > arr[two\_third]:

return ternary\_search(arr, x, two\_third + 1, right)

else:

return ternary\_search(arr, x, one\_third + 1, two\_third - 1)

sizes = [10, 100, 1000, 10000, 100000]

linear\_times = []

binary\_times = []

ternary\_times = []

for size in sizes:

arr = list(range(size))

x = size - 1 # шуканий елемент в кінці списку для найгіршого випадку

# Лінійний пошук

start\_time = time.time()

linear\_search(arr, x)

linear\_times.append(time.time() - start\_time)

# Бінарний пошук

start\_time = time.time()

bin\_search(arr, x, 0, len(arr) - 1)

binary\_times.append(time.time() - start\_time)

# Тернарний пошук

start\_time = time.time()

ternary\_search(arr, x, 0, len(arr) - 1)

ternary\_times.append(time.time() - start\_time)

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

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

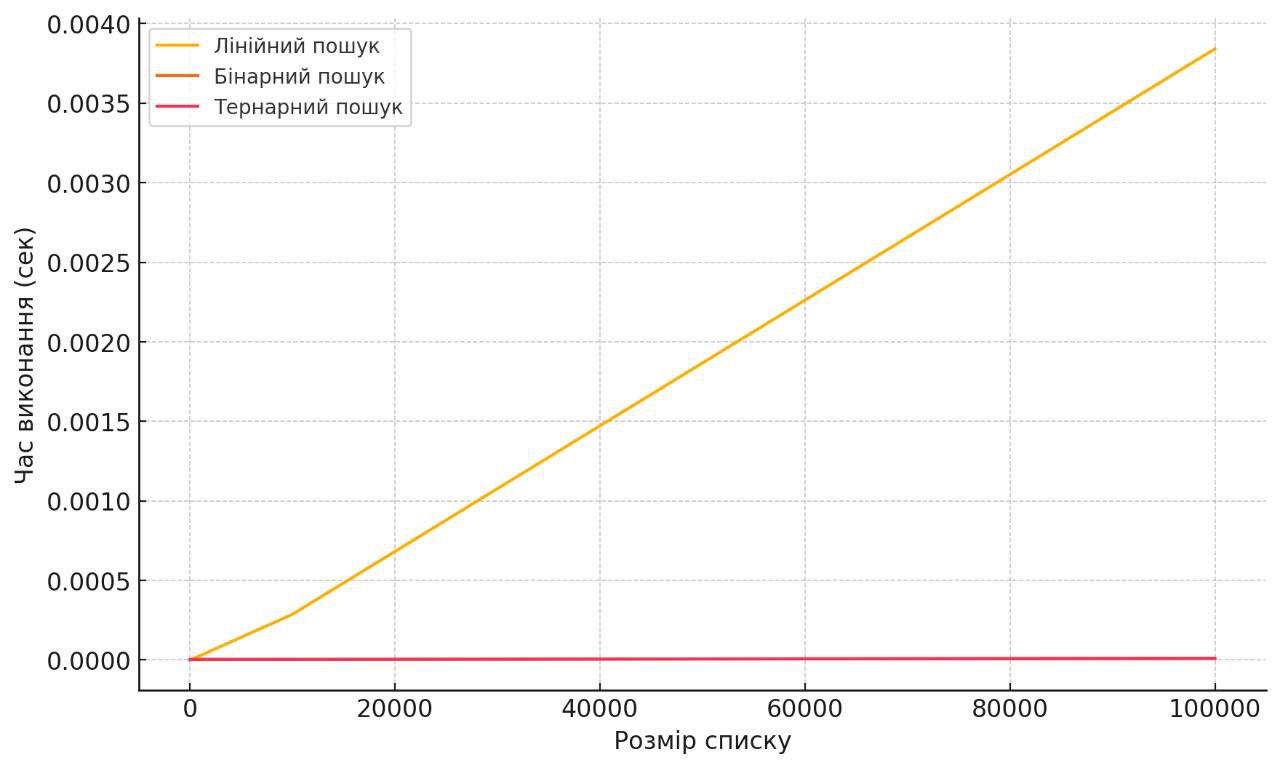
plt.plot(sizes, ternary\_times, label="Тернарний пошук")

plt.xlabel('Розмір списку')

plt.ylabel('Час виконання (сек)')

plt.legend()

plt.show()



Вплив відсортованості списку

Відсортований список:

- Лінійний пошук: немає переваг.

- Бінарний і тернарний пошук: необхідний.

Не відсортований список:

- Лінійний пошук: працює без змін.

- Бінарний і тернарний пошук: необхідне попереднє сортування O(n log n) .

Сценарії використання

Лінійний пошук:

- Використовується, коли списки малі або невідсортовані.

Бінарний пошук:

- Використовується, коли списки великі та відсортовані. Наприклад, у базах даних для швидкого пошуку за індексами.

Тернарний пошук:

- Менш використовується на практиці, але може бути корисним в специфічних задачах з особливими вимогами.

Контрольні питання

1. Алгоритм пошуку важливий для ефективного знаходження елементів у масивах і базах даних.

2. Критерії: час виконання, асимптотична складність, пам'ять.

3. Лінійний пошук перевіряє кожен елемент послідовно.

4. Для бінарного пошуку список має бути відсортований.

5. Переваги бінарного пошуку: швидкість для відсортованих списків. Недоліки: не працює з невідсортованими списками.

6. Тернарний пошук ділить список на три частини, але зазвичай менш ефективний через більше порівнянь.