Звіт

Практична робота №3

Варіант 9

Тема: Алгоритми сортування та їх складність. Порівняння алгоритмів сортування

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

Завдання

1 Вивчити самостійно і записати (будь-яким способом) алгоритм бульбашкового сортування. Оцінити асимптотику алгоритму сортування методом бульбашки в найгіршому і в найкращому випадку. Порівняти за цими показниками бульбашковий алгоритм з алгоритмом сортування вставлянням. Чому на практиці бульбашковий алгоритм виявляється менш ефективним у порівнянні з сортуванням методом зливанням?

2 Оцінити асимптотичну складність алгоритму сортування зливанням, скориставшись основною теоремою рекурсії.

3 Вивчити і записати (будь-яким способом) самостійно алгоритм швидкого сортування. Оцінити асимптотичну складність алгоритму швидкого сортування, скориставшись основною теоремою рекурсії.

1 Алгоритм бульбашкового сортування працює за наступним принципом:

1. Проходимо по масиву.
2. Порівнюємо кожен елемент з наступним.
3. Якщо елементи в неправильному порядку, міняємо їх місцями.
4. Повторюємо кроки 1-3, поки масив не буде відсортований.

Def bubble\_sort(arr):

N = len(arr)

For i in range(n):

Swapped = False

For j in range(0, n-i-1):

If arr[j] > arr[j+1]:

Arr[j], arr[j+1] = arr[j+1], arr[j]

Swapped = True

If not swapped:

Break

Оцінка асимптотики

Найгірший випадок

У найгіршому випадку (масив відсортований у зворотному порядку), кожен елемент порівнюється з усіма іншими, тому кількість операцій пропорційна n⋅(n−1)/2, де n — кількість елементів. Отже, асимптотика в найгіршому випадку:

O(n2)

Найкращий випадок

У найкращому випадку (масив вже відсортований), ми лише один раз пройдемо по масиву і побачимо, що жодних обмінів не потрібно, тобто це займе −1 порівнянь. Отже, асимптотика в найкращому випадку:

O(n)

Алгоритм сортування вставлянням

1. Починаючи з другого елемента, вибираємо його як ключ.
2. Порівнюємо ключ з попередніми елементами і переміщуємо його в потрібну позицію.
3. Повторюємо кроки 1-2 для всіх елементів.

Def bubble\_sort(arr):

N = len(arr)

For i in range(n):

Swapped = False

For j in range(0, n-i-1):

If arr[j] > arr[j+1]:

Arr[j], arr[j+1] = arr[j+1], arr[j]

Swapped = True

If not swapped:

Break

Оцінка асимптотики

Найгірший випадок

У найгіршому випадку (масив відсортований у зворотному порядку), кожен елемент порівнюється з усіма попередніми, тому кількість операцій пропорційна ⋅(n−1)/2. Отже, асимптотика в найгіршому випадку:

O(n2)

Найкращий випадок

У найкращому випадку (масив вже відсортований), ми робимо лише n−1 порівнянь без переміщення елементів. Отже, асимптотика в найкращому випадку:

O(n)

Порівняння алгоритмів

* Найгірший випадок:
* Бульбашкове сортування: O(n2)
* Сортування вставлянням: O(n2)
* Найкращий випадок:
* Бульбашкове сортування: O(n)
* Сортування вставлянням: O(n)

2 Для алгоритму сортування злиттям:

* А=2 (ми розбиваємо масив на 2 частини)
* B=2 (кожна частина має розмір 2/n
* F(n)=O(n) (злиття двох відсортованих частин займає лінійний час)

Розрахуємо logba:

Logba=log22=1

Тепер порівняємо f(n) з nlogba:

* F(n)=O(n)
* N^logba=n1=n

Отже, f(n)=Θ(n^logba)=Θ(n). За другим випадком теореми рекурсії, коли f(n) та n^logba є однаковими, маємо:

T(n)=Θ(nlogn)

3

Алгоритм швидкого сортування працює за наступним принципом:

1. Вибирається опорний елемент (pivot) з масиву.
2. Масив поділяється на дві частини: елементи, менші за опорний, і елементи, більші за опорний.
3. Рекурсивно застосовується швидке сортування до кожної частини.

Def quick\_sort(arr):

If len(arr) <= 1:

Return arr

Else:

Pivot = arr[len(arr) // 2]

Left = [x for x in arr if x < pivot]

Middle = [x for x in arr if x == pivot]

Right = [x for x in arr if x > pivot]

Return quick\_sort(left) + middle + quick\_sort(right)

Для рівняння:

T(n)=aT(n/b)+f(n)

Де a=2, b=2, f(n)=O(n).

Розрахуємо logba:

Logba=log22=1

F(n) з n^logba:

* F(n)=O(n)
* N^logba=n1=n

Отже, (n^logba)=Θ(n). За другим випадком теореми рекурсії, коли f(n) та n^logba є однаковими, маємо:

T(n)=Θ(nlogn)

Відповідь на контрольні питання

1Що таке асимптотична складність алгоритму сортування і чому вона важлива для порівняння алгоритмів?

Асимптотична складність алгоритму сортування — це міра часу виконання або використання пам’яті алгоритму в залежності від розміру вхідних даних. Вона показує, як алгоритм масштабується з ростом обсягу даних. Асимптотична складність дозволяє описати продуктивність алгоритму для великих розмірів вхідних даних за допомогою математичних виразів

Які алгоритми сортування мають квадратичну складність у найгіршому випадку? Поясніть, чому це може бути проблемою для великих обсягів даних. Сортування вставками

Сортування вибором Бульбашкове сортування

При збільшенні розміру масиву час виконання алгоритмів із квадратичною складністю зростає набагато швидше, ніж для алгоритмів із лінійною або логарифмічною складністю.

Квадратичні алгоритми витрачають багато часу на порівняння і переміщення елементів, що може значно перевантажити процесор і пам’ять комп’ютера, особливо при обробці великих обсягів даних.

Для обробки дуже великих масивів даних (мільйони або більше елементів) квадратичні алгоритми стають практично непридатними через дуже тривалий час виконання.

. В чому полягає перевага сортування злиттям над сортуванням вставками для великих наборів даних?

Основна перевага сортування злиттям над сортуванням вставками для великих наборів даних полягає в його значно кращій часовій складності

4 Які алгоритми сортування використовуються для сортування списків у стандартних бібліотеках мов програмування, таких як Python, Java або C++?

Python: sorted() list.sort()

Java: Collections.sort() Arrays.sort()

C++: std::sort() std::stable\_sort()

1. Яка різниця між алгоритмами сортування злиттям і швидким сортуванням? У яких випадках краще використовувати кожен з цих алгоритмів?

Алгоритми сортування злиттям (Merge Sort) та швидким сортуванням (Quick Sort) є двома різними методами сортування, кожен з яких має свої унікальні характеристики та відмінності.

Ось деякі ключові відмінності між ними:

Метод роботи

Сортування злиттям Розділяє масив на дві половини, сортує кожну половину рекурсивно, а потім зливає їх разом відсортованим чином.

Швидким сортуванням Вибирає елемент з масиву та розміщує всі елементи менше опорного елемента ліворуч від нього, а всі елементи, більші або рівні йому, - праворуч. Потім цей процес повторюється рекурсивно для двох частин, розділених опорним елементом.

Швидкість

Сортування злиттям Зазвичай працює швидше за Merge Sort у середньому випадку та в кращому випадку.

Швидким сортуванням Може бути більш ефективним у випадках, коли необхідно гарантувати стабільність сортування, а також у випадках, коли робиться багато операцій з пам’яттю (оскільки Merge Sort не робить прямих операцій над вхідним масивом, а замість цього створює додатковий масив для зливання).

Робота в найгіршому випадку

Сортування злиттям Може мати найгіршу часову складність O(n^2), якщо опорний елемент вибирається неоптимально.

Швидким сортуванням Має часову складність O(n log n) у будь-якому випадку, навіть у найгіршому.

У випадку вибору між ними:

* Використовуйте сортування злиттям якщо вам потрібен швидкий алгоритм сортування та необхідно враховувати час виконання у середньому випадку.
* Використовуйте швидким сортуванням якщо важлива стабільність сортування або якщо пам’ять обмежена, і ви хочете уникнути найгіршого випадку з швидким сортуванням.

6 Які фактори слід враховувати при виборі алгоритму сортування для конкретної задачі?

1. Розмір вхідних даних: Деякі алгоритми сортування мають кращу часову складність для малих масивів, тоді як інші можуть бути ефективнішими для великих масивів.
2. Часові обмеження: Якщо час виконання є критичним фактором, важливо вибрати алгоритм з мінімальною часовою складністю, яка задовольняє потреби вашої задачі.
3. Стабільність сортування: Якщо вам важливо зберігати порядок елементів з однаковим значенням, то ви повинні вибрати стабільний алгоритм сортування
4. Пам’ять: Деякі алгоритми можуть вимагати більше пам’яті для роботи, особливо якщо вони використовують додатковий стек для рекурсії.
5. Властивості даних: Якість вибору алгоритму сортування також може залежати від властивостей самих даних.