DOI: 10.34185/1991-7848.itmm.2023.01.072

КОМБІНОВАНІ АЛГОРИТМИ СОРТУВАННЯ

Шинкаренко В.І., Макаров О.В.
Український державний університет науки та технологій, Україна

Вступ. Існує велика кількість алгоритмів сортування з різними показниками часової ефективності, стабільності і вимог до виділення додаткової пам'яті. Найчастіше алгоритми оцінюють за показником обчислювальної складності (O). Найбільш популярні алгоритми сортування, такі як швидке сортування (Quick sort) чи сортування злиттям (Merge sort), мають обчислювальну складність у середньому $O(n*\log(n))$. Однак на невеликих наборах даних сортування вставками (Insertion sort), з обчислювальною складністю $O(n^2)$, працює швидше.

Ефективні реалізації зазвичай використовують гібридний алгоритм, що поєднує асимптотично ефективний алгоритм для загального сортування з сортуванням вставками для невеликих масивів у нижній частині рекурсії [1]. Одним із комбінованих алгоритмів є Flash sort [2]. Він поєднує у собі принципи сортування комірками (Bucket sort) та сортування вставками (Insertion sort). Алгоритм надзвичайно ефективний $O(n)$ для рівномірно розподілених елементів, але в гіршому випадку часова ефективність знижується до $O(n^2)$.

Використовуються і більш складні варіанти, наприклад Introsort [3] поєднує три базових алгоритми. Спершу використовується швидке сортування (Quick sort). Коли розмір підмасиву на черговому кроці рекурсії стає менше заданої константи, викликається сортування вставками. Після досягнення певної глибини рекурсії, викликається пірамідальне сортування (Heap sort). Таким чином створений комбінований алгоритм має у собі сильніші сторони трьох різних алгоритмів сортування.

Основний матеріал. Запропоновано декілька видів попередньої обробки масивів даних, метою яких є покращення часової ефективності сортування. Це може бути досягнуто кількома способами. Можна спробувати попередньо змінити порядок елементів у масиві і тим самим зменшити час роботи основного алгоритму сортування. Також можливо попередньою перестановкою елементів запобігти ситуації з найгіршою комбінацією елементів в масиві, при якій сортування займатиме максимально великий час.

Перший спосіб полягає в тому, щоб спробувати «передбачити» позицію елемента у масиві і виконати перестановку. Повторивши цю операцію для

кожного елемента, або для якогось відсотка від довжини масиву, можна значно знизити рівень невпорядкованості масиву і тим самим зменшити час роботи основного алгоритму сортування. Передбачуваний індекс залежатиме від діапазону значень елементів і від значення поточного елементу. Якщо масив матиме багато однакових значень, то буде виконано перестановку в одне місце. Таким чином усі перестановки після першої не матимуть впливу на порядок елементів масиву, а тільки марнуватимуть час. Для вирішення даної проблеми можна запам'ятати усі передбачені індекси і не робити перестановку, або знайти найближчий індекс що не був передбачений раніше.

Наступний спосіб попередньої обробки полягає у тому щоб розвернути усі послідовності елементів відсортованих у зворотньому порядку. Такиж чином зменшується невпорядкованість масиву. Також у випадку коли масив відсортовано у зворотньому порядку це дасть змогу відсортувати його в один прохід, а для швидкого сортування (Quick sort) – уникнути найгіршого випадку у якому сортування займе максимально великий час.

Для великих масивів запропоновано ще кілька видів попередньої обробки. Ідея локальної передобробки полягає в умовному розбитті масиву на блоки певної довжини і проведенні передобробок, описаних раніше, на кожному підмасиві. Таким чином початковий масив буде мати N частково (або повністю) відсортованих підмасивів. Масив після локальної передобробки матиме «форму пилки». При правильному виборі довжини блока, менше ніш довжина кешу процесора поділена на розмір елементів у сортованому масиві, можна значно зменшити кількість або взагалі уникнути промахів кешу (cache miss). У разі якщо розмір блока буде дуже великими, отримаємо багато промахів кешу і зменшення часової ефективності передобробки.

Глобальна передобробка також використовує умовне розбиттся масиву на блоки певної довжини. Але для кожного елемента масиву передбачається глобальний індекс, тобто приблизне місце на якому стояв би даний елемент у відсортованому масиві. Після передбачення елемент може бути переставлений тільки якщо передбачений індекс не виходить за межі поточного блока. Такий підхід також дозволяє уникнути промахів кешу (cache miss). Але приблизна вірогідність того що елемент буде переставлений дорівнює $1/K$ (де K – кількість блоків), для масиву рівномірно розподілених випадкових чисел.

Проведено експеримент у якому массиви випадкових цілих чисел сортувались за допомогою різних алгоритмів сортування з попедньою обробкою і без. Використовувались масиви довжиною від 1000 до 100000 елементів з кроком 1000. Кожний масив заповнювався випадковими цілими

позитивними числами у діапазоні від 0 до n, де n – довжина масиву. Було використано шейкерне сортування, сортування вставками та швидке сортування.

Шейкерне та сортування вставками мають середню складність $O(n^2)$, але у кращому випадку можуть виконуватись за $O(n)$. Ефективна передобробка може зробити масив майже відсортованим, що мінімізує кількість перестановок виконуваних сортуваннями і знизить загальний час роботи комбінованого алгоритму. У порявнянні, швидке сортування у середньому має набагато кращу складність $O(n*\log(n))$, але знизивши кількість перестановок, також можна покращити його часову ефективність.

Для оцінки часової ефективності передобробки необхідно порівняти час виконання сортування і сортування з передобробкою. Ефективність визначимо як відношення різниці у часі виконання комбінованого і чистого до часу чистого алгоритму сортування [2]. Ефективність може бути як позитивною так і негативною. Негативне значення означатиме що сортування з передобробкою виконується довше ніж окреме сортування.

Висновки. Розроблено кілька видів попередньої обробки масивів даних для покращення часової ефективності сортування. Проведений експеримент показав що передобробки загалом позитивно впливають на роботу алгоритмів сортування. Найбільше прискорив усі типи сортування метод передобробки що запам’ятовує передбачені індекси. Для шейкерного сортування на великих обсягах даних прискорення перевищило 1000%. Найменше прискорення показала передобробка із розворотом послідовностей відсортованих у зворотньому порядку. Для всіх видів сортувань відсоток покращення часової ефективності був близьким до 0.

Порівнявши час сортування із комбінованим алгоритмом передобробки та сортування маємо значний виграш для шейкерного та сортування вставками. Для швидкого сортування відсоток покращення часової ефективності коливається в діапазоні +-5% у комбінації зі швидкою передобробкою. Передобробка із запам’ятовуванням індексів займає надто багато часу що нівелює прискорення часу роботи основного алгоритму. Необхідно прискорити роботу попередньої обробки. А також провести експерименти з іншими алгоритмами сортування.

Література

1. Двохкомпонентні алгоритми сортування / В.І. Шинкаренко, А.Ю. Дорошенко, О.А. Яценко, В.В. Разносілін, К.К. Галанін // Проблеми програмування. — 2022. — № 3-4. — С. 32-41
2. The Flashsort1 Algorithm / Karl-Dietrich Neubert.
URL: https://www.neubert.net/Flapaper/9802n.htm [дата звернення 03.02.2023]
3. Introsort – C++'s Sorting Weapon /URL: https://www.geeksforgeeks.org/introsort-cs-sorting-weapon/ [дата звернення 10.02.2023]

COMBINED SORTING ALGORITHMS

Shynkarenko Viktor, Makarov Oleksii

Abstract. This study represents the usage of data preprocessing for performance improvement of sorting algorithms. Combining of different basic sorting algorithms was already successfully used to outperform any sole algorithm. The purpose of this research is to discover and estimate different ways to reorganize unsorted data just before executing the main sorting algorithm. Five preprocessing technics were developed and tested in combination with cocktail sort, insertion sort and quick sort. Time efficiency was estimated by comparing time of sole sorting algorithm and combined algorithm which includes preprocessing. Further studying of preprocessing methods and their impact on different sorting algorithms is required.

Keywords: algorithm, sorting, time efficiency, combined algorithm, preprocessing.

References

1. Bicomponent sorting algorithms / V I Shynkarenko, A. Y. Doroshenko, O. A. Yatsenko, V. V. Raznosilin, K. K. Halanin // The problems of programming — 2022. — № 3-4. — P. 32-41
2. The Flashsort1 Algorithm / Karl-Dietrich Neubert.
URL: https://www.neubert.net/Flapaper /9802n.htm [Accessed: 3rd February 2023]
3. Introsort – C++'s Sorting Weapon /URL: https://www.geeksforgeeks.org/introsort-cs-sorting-weapon/ [Accessed: 10th February 2023]