Лекція 6. Швидке сортування. Спеціалізовані алгоритми внутрішнього сортування

Швидке сортування Хоара

Удосконалений метод сортування, що базується на обміні, К.Хоар запропонував алгоритм QuickSort сортування масивів, що дає на практиці відмінні результати і дуже просто програмується. Це сортування називають швидким, тому що на практиці воно виявляється найшвидшим методом сортування з тих, що оперують порівняннями.

Основна стратегія прискорення алгоритмів сортування - обмін між якомога більш віддаленими елементами вихідного файлу.

Ідея К. Хоара полягає в наступному: на кожному кроці методу ми спочатку вибираємо "середній" елемент, потім переставляємо елементи масиву так, що він поділяється на три частини: спочатку ідуть елементи, менші "середнього", потім рівні йому, а в третій частині - більші. Після такого розподілу масиву залишається тільки відсортувати першу і третю його частини, з якими ми зробимо аналогічно (розділимо на три частини). І так доти, доки ці частини не будуть складатися з одного елемента, а масив з одного елемента завжди відсортований.

Вибір "середнього" - задача непроста, тому що потрібно, не виконуючи сортування, знайти елемент зі значенням максимально близьким до середнього. Тут, звичайно, можна просто вибрати довільний елемент (звичайно вибирають елемент, що стоїть у середині підмасива, що сортується), але можемо вибирати з трьох елементів самого лівого, самого правого і того, що стоїть посередині.

Дано	17	35	48	52	27	9	15	13	89
1-й обмін	13	35	48	52	27	9	15	17	89
2-й обмін	13	17	18	52	27	9	15	35	89
3-й обмін	13	15	18	52	27	9	17	35	89
4-й обмін	13	15	17	52	27	9	18	35	89
5-й обмін	13	15	9	52	27	17	18	35	89
6-й обмін	13	15	9	17	27	52	18	35	89

Складність:

Аналіз складності алгоритму в середньому, що використовує гіпотезу про рівну імовірність усіх входів, показує, що

$C(n) = O(n \log_2 n)$, $M(n) = O(n \log_2 n)$.

У гіршому випадку, коли в якості бар'єрного вибирається, наприклад, максимальний елемент підмасива, складність алгоритму квадратична.

Швидке сортування є алгоритмом на основі порівнянь, і не є стабільним.

Класична реалізація

В класичному варіанті, запропонованому Хоаром, з масиву обирався один елемент, і весь масив розбивався на дві частини по принципу: в першій частині — ті що не більші даного елементу, в другій частині — ті що не менші даного елемента. Процедура *Quicksort*(*A*,*p*,*q*) здійснює часткове впорядкування масиву A з p-го по q-ий індекс:

$Quicksort(A,p,q)$
if $p \geq q$ **return**;
$r \leftarrow A[p]$
$i \leftarrow p-1$
$j \leftarrow q+1$
while $i < j$ **do**
    repeat
        $i \leftarrow i+1$
    until $A[i] \geq r$
    repeat
        $j \leftarrow j-1$
    until $A[j] \leq r$
    if $i < j$
        then Поміняти $A[i] \leftrightarrow A[j]$
$Quicksort(A,p,j)$
$Quicksort(A,j+1,q)$

Сучасна реалізація

На сьогодні в стандартних бібліотеках використовують таку реалізацію алгоритму:

$Partition(A,p,q)$
$x \leftarrow A[q]$
$i \leftarrow p-1$
for $j \leftarrow p$ **to** $q-1$
do **if** $A[j] \leq x$
    then $i \leftarrow i+1$
        Поміняти $A[i] \leftrightarrow A[j]$
$i \leftarrow i+1$
Поміняти $A[i] \leftrightarrow A[q]$
return i

$Quicksort(A,p,q)$
if $p \geq q$ **return**;
$i \leftarrow Partition(A,p,q)$
$Quicksort(A,p,i-1)$
$Quicksort(A,i+1,q)$

Аналіз

Час роботи алгоритму сортування залежить від збалансованості, що характеризує розбиття. Збалансованість, у свою чергу залежить від того, який елемент обрано як опорний (відносно якого елемента виконується розбиття). Якщо розбиття збалансоване, то асимптотично алгоритм працює так само швидко як і алгоритм сортування злиттям. У найгіршому випадку, асимптотична поведінка алгоритму настільки ж погана, як і в алгоритму сортування включенням.

Найгірше розбиття

Найгірша поведінка має місце у тому випадку, коли процедура, що виконує розбиття, породжує одну підзадачу з n-1 елементом, а другу — з 0 елементами. Нехай таке незбалансоване розбиття виникає при кожному рекурсивному виклику. Для самого розбиття потрібен час $\Theta(n)$. Тоді, рекурентне співвідношення для часу роботи, можна записати так:

$T(n) = T(n-1) + T(0) + \Theta(n) = T(n-1) + \Theta(n)$

Розв'язком такого співвідношення є $T(n) = \Theta(n^2)$.

Найкраще розбиття

В найкращому випадку процедура Partition ділить задачу на дві підзадачі, розмір кожної не перевищує n/2. Час роботи, описується нерівністю:

$T(n) \le 2T(n/2) + \Theta(n)$

Тоді:

$T(n) = O(n \log n)$ — асимптотично найкращий час.

Середній випадок

Математичне очікування часу роботи алгоритму на всіх можливих вхідних масивах є $O(n \log n)$, тобто середній випадок ближчий до найкращого.

Модифікації

В середньому алгоритм працює дуже швидко, але на практиці, не всі можливі вхідні масиви мають однакову імовірність. Тоді, шляхом додання рандомізації вдається отримати середній час роботи в будь-якому випадку.

Рандомізованний алгоритм

В рандомізованному алгоритмі, при кожному розбитті випадковий елемент обирається в якості опорного:

$Randomized\_Partition(A, p, q)$

1 $i \leftarrow Random(p, q)$

2 **Поміняти** $A[i] \leftrightarrow A[q]$

9 **return** $Partition(A, p, q)$

$Randomized\_Quicksort(A, p, q)$

1 **if** $p \ge q$ **return**;

2 $i \leftarrow Randomized\_Partition(A, p, q)$

3 $Randomized\_Quicksort(A, p, i-1)$

4 $Randomized\_Quicksort(A, i+1, q)$

Сортування підрахунком

Сортування підрахунком – алгоритм впорядкування, що застосовується при малій кількості різних елементів (ключів) у масиві даних. Час його роботи лінійно залежить як від загальної кількості елементів у масиві так і від кількості *різних* елементів.

Ідея алгоритму

Ідея алгоритму полягає в наступному: спочатку підрахувати скільки разів кожен елемент (ключ) зустрічається в вихідному масиві. Спираючись на ці дані можна одразу вирахувати на якому місці має стояти кожен елемент, а потім за один прохід поставити всі елементи на свої місця.

Псевдокод алгоритму

Для простоти будемо вважати, що всі елементи (ключі) є натуральними числами що лежать в діапазоні 1..K. Процедура $Counting_Sort(A)$виконує сортування масиву A:

$Counting\_Sort(A)$

1 C — масив з K елементів, заповнений нулями

2 **for** $i \leftarrow 1$ **to** $length[A]$

3 **do** $C[A[i]] \leftarrow C[A[i]] + 1$

4 **for** $i \leftarrow 2$ **to** K

5 **do** $C[i] \leftarrow C[i] + C[i-1]$

6 **for** $i \leftarrow length[A]$ **downto** 1

7 **do** $B[C[A[i]]] \leftarrow A[i]$

8 $C[A[i]] \leftarrow C[A[i]] - 1$

9 $A \leftarrow B$

Аналіз алгоритму

В алгоритмі присутні тільки прості цикли: в рядках 2, 6, 9 — цикл довжини N (довжина масиву), в рядку 4 — цикл довжини K (величина діапазону). Отже складність роботи алгоритму є $O(N+K)$.

В алгоритмі використовуються два додаткових масиви: Cі B. Тому алгоритм потребує $O(N+K)$додаткової пам'яті.

В такій реалізації алгоритм є стабільним. Саме ця його властивість дозволяє використовувати його як частину інших алгоритмів сортування (напр. сортування за розрядами).

Використання даного алгоритму є доцільним тільки у випадку малих K (порядку N).

Сортування за розрядами

Сортування за розрядами (англ. *Radix sort*) — швидкий стабільний алгоритм впорядкування даних. Застосовується для впорядкування елементів, що є ланцюжками над будь-яким скінченним алфавітом (напр. рядки, або цілі числа). В якості допоміжного використовує будь-який інший стабільний алгоритм сортування.

Алгоритм застосовувався для впорядкування перфокарт.

Ідея алгоритму

Ідея полягає в тому, щоб спочатку впорядкувати всі елементи за молодшим розрядом, потім стабільно впорядкувати за другим розрядом, потім за третім і так далі аж до найстаршого. Оскільки, припускається, що кожен разряд приймає значення з невеликого діапазону, то кожен цикл впорядкування можна виконувати швидко і з малими затратами пам'яті.

Приклад роботи

В прикладі показано, як впорядковувати таким алгоритмом масив трицифрових чисел:

```
572      572      523      266
266      523      349      349
783 -->  783 -->  266 -->  523
523      266      572      572
349      349      783      783
           ^        ^        ^
```

Аналіз

Час роботи кожного циклу сортування залежить від того алгоритму, що використовується в якості допоміжного. Найчастіше використовують сортування підрахунком, що пряцює за час $O(N + K)$(де N — кількість елементів в масиві; K — кількість символів у алфавіті, якщо впорядковуються десяткові числа, то K = 10) і використовує додатково $O(N + K)$пам'яті. Всього здійснюється стільки циклів впорядкування, скільки розрядів у максимальному єлементі.

Загальна складність роботи алгоритму з використанням сортування підрахунком є $O(D \cdot (N + K))$(D — кількість розрядів). Якщо впорядковувати цим алгоритмом цілі числа, то складність буде $O(N \log M)$, де M — найбільший елемент масиву.