Лекція 4. Прості алгоритми внутрішнього сортування

Алгоритм сортування – це алгоритм для впорядкування елементів у деякій структурі даних по зростанню чи спаданню. У випадку наявності елементів з однаковими значеннями, в упорядкованій послідовності вони розташовуються поруч один за одним у будь-якому порядку. Однак іноді буває корисно дотримуватися початкового порядку елементів з однаковими значеннями.

В залежності від того, над якою структурою даних здійснюється сортування, буває сортування масиву, зв'язаного списку, дерева, графа, таблиці.

У випадку, коли елемент структури даних має декілька полів, вводиться поняття ключ сортування.

Ключ сортування – це поле структури даних, за значенням якого визначається порядок елементів (тобто за яким відбувається сортування). На практиці як ключ часто виступає число, а в інших полях зберігаються які-небудь дані, що ніяк не впливають на роботу алгоритму.

Простий алгоритм сортування можна розбити на 3 частини:

- порівняння, що визначає впорядкованість пари елементів;
- перестановка, що міняє місцями цю пару елементів;
- повтор перших двох дій доти, поки всі елементи структури даних не будуть впорядковані.

Алгоритми сортування мають велике практичне значення. Їх можна зустріти там, де відбувається обробка й зберігання великих обсягів інформації. Вони використовуються також у криптографії, кодуванні, вирішенні математичних задач тощо.

Жодна інша проблема не породила такої кількості різних рішень, як задача сортування. Універсального, найкращого алгоритму сортування не існує. Однак, маючи приблизні характеристики вхідних даних, можна підібрати метод, що працює оптимальним чином. Для цього необхідно знати параметри, за якими буде здійснюватися оцінка алгоритмів.

Параметри алгоритмів сортування:

- *Час сортування* – основний параметр, що характеризує швидкодію алгоритму.
- *Пам'ять* – один з параметрів, що характеризується тим, що ряд алгоритмів сортування вимагають виділення додаткової пам'яті під тимчасове зберігання даних. При оцінці використовуваної пам'яті не буде враховуватися місце, що займає вихідний масив даних і незалежні від вхідної послідовності витрати, наприклад, на зберігання коду програми.
- *Стійкість* – це параметр, що відповідає за те, що сортування не міняє взаємного розташування рівних елементів. Наприклад, якщо алфавітний список групи сортується по оцінках, те стабільний метод створює список у якому прізвища студентів з однаковими оцінками будуть впорядковані за алфавітом, а нестабільний метод створить список у якому, можливо, вихідний порядок буде порушений.

– *Природність поводження* – параметр, який вказує на ефективність методу при обробці вже відсортованих, або частково відсортованих даних. Алгоритм поводиться природно, якщо враховує цю характеристику вхідної послідовності й працює краще.

– *Використання операції порівняння.* Алгоритми, що використовують для сортування порівняння елементів між собою, називаються заснованими на порівняннях. Мінімальна трудомісткість гіршого випадку для цих алгоритмів становить $O(n \log n)$, але вони відрізняються гнучкістю застосування. Для спеціальних випадків (типів даних) існують більш ефективні алгоритми.

Класифікація алгоритмів сортування

Всі алгоритми сортування можна класифікувати за різними ознаками, наприклад, за стійкістю, за особливостями функціонування, за використанням операцій порівняння, за потребою в додатковій пам'яті, за структурою даних, за шириною області застосування, за потребою в знаннях про структуру даних, що виходять за рамки операцій порівняння ключів тощо.

Розглянемо більш детально класифікацію алгоритмів сортування за використанням пам'яті. У цьому випадку основні типи сортування діляться на:

– *Внутрішнє сортування* – це алгоритм сортування, що у процесі впорядкування даних використовує тільки оперативну пам'ять (ОЗП) комп'ютера. Тобто оперативної пам'яті досить для розміщення в ній сортуємого масиву даних з довільним доступом до будь-якої комірки й власне для виконання алгоритму. Внутрішнє сортування застосовується у всіх випадках, за винятком однопрохідного зчитування даних і однопрохідного запису відсортованих даних. Залежно від конкретного алгоритму і його реалізації дані можуть сортуватися в тій же області пам'яті, або використовувати додаткову оперативну пам'ять.

– *Зовнішнє сортування* – це алгоритм сортування, що при проведенні впорядкування даних використовує зовнішню пам'ять, як правило, жорсткі диски. Зовнішнє сортування розроблене для обробки великих структур даних, які не поміщаються в оперативну пам'ять. Звертання до різних носіїв накладає деякі додаткові обмеження на даний алгоритм: доступ до носія здійснюється послідовним чином, тобто в кожний момент часу можна зчитати або записати тільки елемент наступний за поточним; обсяг даних не дозволяє їм розміститися в ОЗП.

Внутрішнє сортування є базовим для будь-якого алгоритму зовнішнього сортування - окремі частини масиву даних сортуються в оперативній пам'яті й за допомогою спеціального алгоритму з’єднуються в один масив, упорядкований за ключем.

Слід зазначити, що внутрішнє сортування значно ефективніше зовнішнього, тому що на звертання до оперативної пам'яті затрачується набагато менше часу, ніж до носіїв.

Одна з найбільш розповсюджених класифікацій методів сортування

наведена на рисунку 1.1.

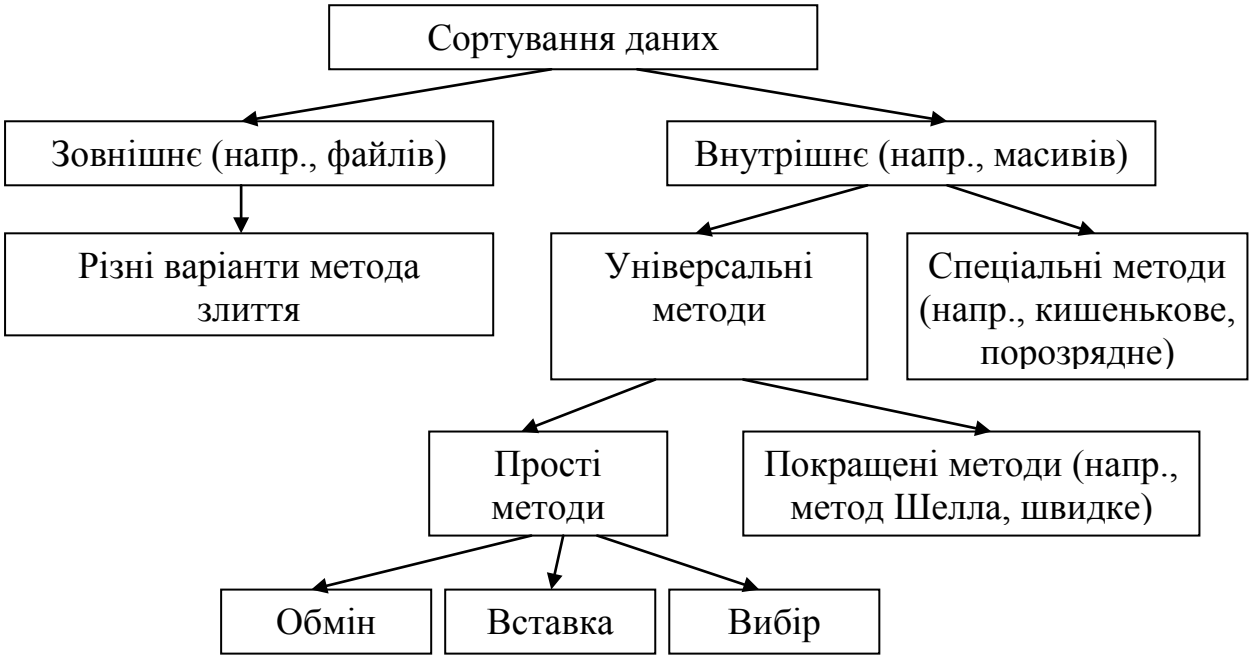

Рисунок 1.1 – Загальна класифікація методів сортування

Класифікація за часом виконання:

За час $O(n^2)$

- сортування вибором
- сортування вставкою
- сортування обміном

За час $O(n \log\ n)$

- пірамідальне сортування
- швидке сортування
- сортування злиттям

За час $O(n)$ з використанням додаткової інформації про елементи

- сортування підрахунком
- сортування за розрядами
- сортування комірками

За час $O(n \log^2\ n)$

- сортування злиттям модифіковане
- сортування Шелла

Алгоритми стабільного впорядкування:

- сортування вибором
- сортування вставкою
- сортування обміном
- сортування злиттям
- сортування підрахунком
- сортування за розрядами

- сортування комірками
- сортування злиттям модифіковане

Прості алгоритми внутрішнього сортування

Щоб сконцентруватися на алгоритмічних питаннях, ми будемо працювати з алгоритмами, які сортують масиви цілих чисел. Ці алгоритми легко адаптувати для сортування записів.

В основному програми сортування працюють із записами двома способами: або вони порівнюють і сортують тільки ключі, або пересувають записи цілком. Більшість алгоритмів які ми вивчимо можна застосовувати, за допомогою їхнього переформулювання в термінах цих двох операцій, для довільних записів.

Якщо записи досить великі, то звичайно намагаються уникнути їхнього пересування за допомогою "непрямого сортування": при цьому самі записи не сортуються, а замість цього сортується масив покажчиків (індексів), так, що перший покажчик вказує на самий маленький елемент і так далі. Ключі можуть зберігатися або із записами (якщо вони великі), або з покажчиками (якщо вони маленькі).

Сортування Вибором

Один з найпростіших методів сортування працює в такий спосіб:

1) знаходимо найменший елемент у масиві;

2) міняємо його місцями з елементом, що знаходиться на першому місці;

3) повторюємо процес із другої позиції у файлі й знайдений найменший елемент обмінюємо із другим елементом і так далі поки весь масив не буде відсортований.

У міру просування покажчика зліва направо через масив, елементи ліворуч від покажчика перебувають уже в своїй кінцевій позиції (і їх не вже не будуть більше пересувати), тому масив стає повністю відсортованим до того моменту, коли покажчик досягає правого краю.

Цей метод називається сортуванням вибором оскільки він працює циклічно вибираючи найменший з елементів, що залишилися.

Псевдокод процедури сортування вибором:

```
Процедура Сортування вибором (a: масив, N: довжина масиву)
    змінні i, j, min, t : цілі
Початок
   для і від 1 до N-1 // N-розмір масиву
   початок
      min := i
      для j від i+1 до N  // цикл знаходження мінімального елемента
         якщо a[j]<a[min] тоді
         min := j
    t := a[min] //заміна елементів
    a[min] :=a[i]
    a[i] := t
  кінець
Кінець
```

Приклад сортування масиву з п'яти елементів за даним алгоритмом:

7 5 2 3 1
1 5 2 3 **7**
1 **2** **5** 3 7
1 2 **3** **5** 7

Цей метод - один з найпростіших, і він працює дуже добре для невеликих структур даних. Його "внутрішній цикл" складається з порівняння a[i]<a[min] (плюс код необхідний для збільшення j та перевірки того, що він не перевищив N), що навряд чи можна ще спростити.

Більш того, незважаючи на те, що цей метод очевидно є методом "грубої сили", він має дуже важливе застосування: оскільки кожний елемент пересувається не більше ніж раз, то він дуже гарний для великих записів з маленькими ключами.

Характеристика алгоритму:

Структура даних: Масив

Швидкодія: $O(n^2)$

Простір: O(n), O(1)

Оптимальність: Не практичний

Сортування Вставкою

Сортування вставкою - це метод який майже настільки ж простий, що й сортування вибором, але набагато більш гнучкий. Більшість людей при сортуванні колоди гральних карт, використовують метод, схожий на алгоритм сортування включенням. Суть алгоритму: беремо один елемент і вставляємо його в потрібне місце серед тих, що ми вже обробили (тим самим залишаючи їх відсортованими). Алгоритм:

1) зліва направо проходимо масив, порівнюючи сусідні елементи, поки не знайдемо елемент, що розташований не в порядку сортування;

2) обмінюємо цей елемент з елементами розташованими ліворуч від нього, поки він не займе потрібну позицію;

3) повторюємо перші дві дії поки масив не буде відсортовано.

Псевдокод процедури сортування вставкою:

```
Процедура Сортування вставкою (a: масив, N: довжина масиву)
Змінні i, j, t: цілі
Початок
    для i від 2 до N
    початок
        t:=a[i]
        j:=i-1
        поки j>0 та t<a[j]
        початок
            a[j+1]:=a[j]
            j:=j-1
            кінець
    a[j+1]:=t //для прискорення операція пересунута в верхній
цикл
    кінець
Кінець
```

Приклад сортування масиву з семи елементів за даним алгоритмом:

3	8	9	10	6	7	11	j=5, i=4	6<10
3	8	9	6	10	7	11	i=3	6<9
3	8	6	9	10	7	11	i=2	6<8
3	6	8	9	10	7	11	i=1	6>3
3	6	8	9	10	7	11	j=6, i=5	7<10
3	6	8	9	7	10	11	i=4	7<9
3	6	8	7	9	10	11	i=3	7<8
3	6	7	8	9	10	11	i=2	7>6

Також як і в сортуванні вибором, у процесі сортування елементи ліворуч від покажчика і перебувають уже в відсортованому порядку, але вони не обов'язково перебувають у своїй останній позиції, оскільки їх ще можуть пересунути праворуч щоб вставити більш маленькі елементи зустрінуті пізніше. Однак масив стає повністю відсортованим, коли покажчик досягає правого краю.

Даний алгоритм також простий в реалізації та ефективний для невеликих масивів. Є стабільним алгоритмом.

Ефективний при сортуванні масивів, дані в яких вже непогано відсортовані: продуктивність рівна O(n + d), де d – кількість інверсій.

Характеристика алгоритму:

Структура даних: Масив

Швидкодія: $O(n^2)$, для найкращого випадку O(n + d)

Простір: O(n), O(1)

Оптимальний: Переважно ні

Бульбашкове сортування (сортування простими обмінами)

Алгоритм працює таким чином – у масиві порівнюються два сусідні елементи. Якщо один з елементів, не відповідає критерію сортування (є більшим, або ж, навпаки, меншим за свого сусіда), то ці два елементи міняються місцями. Прохід по списку продовжується до тих пір, доки дані не будуть відсортованими. Алгоритм отримав свою назву від того, що процес сортування за ним нагадує поведінку бульбашок повітря у резервуарі з водою. Оскільки для роботи з елементами масиву він використовує лише порівняння, це сортування на основі порівнянь.

```
Процедура Бульбашкове сортування (a: масив, N: довжина масиву)
   Змінні i, j, t: цілі
   Початок
         для і від 1 до N
            для j від 1  до N-1
              якщо a[j]>a[j+1] тоді
                   початок
            t:=a[j]
            a[j]:=a[j+1]
            a[j+1]:=t
            кінець
   Кінець
```

Приклад реалізації крок за кроком

Візьмемо масив чисел "5 1 4 2 8", і за допомогою даного алгоритму, відсортуємо його від найменшого до найбільшого елементу. На кожному кроці, елементи, виділені **жирним** шрифтом будуть порівнюватись.

Перший прохід:

(**5** **1** 4 2 8) → (**1** **5** 4 2 8)
(1 **5** **4** 2 8) → (1 **4** **5** 2 8)
(1 4 **5** **2** 8) → (1 4 **2** **5** 8)
(1 4 2 **5** **8**) → (1 4 2 **5** **8**)

Другий прохід:

(**1** **4** 2 5 8) → (**1** **4** 2 5 8)
(1 **4** **2** 5 8) → (1 **2** **4** 5 8)
(1 2 **4** **5** 8) → (1 2 **4** **5** 8)
(1 2 4 **5** **8**) → (1 2 4 **5** **8**)

Тепер наш масив повністю відсортований, однак, алгоритм цього ще не знає. Йому потрібен ще один "пустий" прохід, під час якого від не поміняє місцями жодного елементу (якщо реалізувати відповідний прапор, інакше проходів буде стільки скільки елементів у масиві – за класичною реалізацією).

Третій прохід:

(**1** **2** 4 5 8) → (**1** **2** 4 5 8)
(1 **2** **4** 5 8) → (1 **2** **4** 5 8)
(1 2 **4** **5** 8) → (1 2 **4** **5** 8)
(1 2 4 **5** **8**) → (1 2 4 **5** **8**)

Нарешті, масив відсортовано, і алгоритм може припинити свою роботу.