Лабораторна робота № 2

Теоретичні відомості. Прості методи сортування

Сортування (упорядкування) є однією з найбільш поширених операцій обробки даних. Розрізняють методи внутрішнього та зовнішнього сортування. Методи внутрішнього сортування застосовують до невеликих об’ємів даних і вони не вимагають додаткової пам’яті. Методи зовнішнього сортування застосовують до великих масивів даних, що зберігаються на зовнішніх носіях.

Серед методів внутрішнього сортування розрізняють прості та удосконалені методи.

Найбільш відомими простими методами сортування є:

- сортування обміном (бульбашкове сортування);
- сортування вставкою;
- сортування вибором.

Серед удосконалених методів найчастіше вживаються [3, 4]:

- швидке сортування (метод Хоара);
- сортування злиттям;
- сортування включенням зі спадним приростом (метод Шелла);
- сортування за допомогою дерева (пірамідальне сортування).

Зауваження. Надалі, розглядаючи різні методи сортування, не втрачаючи загальності, будемо розглядати сортування *за зростанням*. При сортуванні *за спаданням* алгоритми є аналогічними з точністю до знаку порівняння елементів.

Зауваження. Нижченаведений опис алгоритмів буде здійснюватись на прикладі сортування *одномірного масиву* (вектора), кожен елемент якого характеризується *індексом* (порядковим номером) та *ключем* (даними певного типу для яких можна застосувати операції порівняння). Звичайно, ці ж алгоритми сортування можна застосовувати і для інших

структур даних, зокрема, *лінійних зв'язаних списків* (див. лабораторну роботу № 4).

Сортування обміном (бульбашкове сортування)

Бульбашкове сортування (англ. *bubble sort*) полягає у порівнянні двох сусідніх елементів і їх подальшій перестановці місцями, якщо їхній порядок не є вірний. Застосування цієї процедури до всіх пар елементів масиву, від останньої пари до першої, дозволяє виявити елемент з найменшим ключем і перемістити його в першу позицію. Цей процес нагадує спливання бульбашки (звідки і походить назва методу). За кожну ітерацію черговий елемент з найменшим ключем потрапляє на потрібну позицію у лівій частині масиву, збільшуючи таким чином розмір відсортованої ділянки. З огляду на це, при кожній новій ітерації не потрібно розглядати вже відсортовані елементи у лівій частині масиву, зменшуючи розмір правої частини на один елемент.

У вдосконаленому алгоритмі вводиться індикатор (прапорець) який відслідковує факт перестановки елементів. Перед кожною ітерацією цей індикатор встановлюється у стан `false`, а якщо перестановка відбулася – у стан `true`. Таким чином, якщо після завершення ітерації прапорець залишився у стані `false`, то всі пари елементів розміщенні у вірному порядку і сортування можна припинити. У випадку частково впорядкованих масивів такий підхід дозволяє скоротити кількість операцій.

Зауваження. У сортуванні обміном (а також і в багатьох інших алгоритмах) використовується операція перестановки місцями двох елементів. Для цієї мети служить стандартна (`std::swap(…)`) чи користувацька функція `swap(a,b)` яка реалізується наступним кодом:

```
temp = a;
a = b;
b = temp;
```

Параметрами цієї функції повинні служити вказівники чи посилання на елементи, які потрібно переставити [5, 6]

Алгоритм сортування обміном (функція `SortBubble(…)`*).*

1. Встановити лічильник ітерацій рівний одиниці. Поки лічильник не досяг значення індексу останнього елементу масиву (цикл за зростанням індексу):
   1.1. Для елементів масиву: від останнього елементу до елементу з індексом, рівному значенню лічильника ітерацій (цикл за спаданням індексу):
      1.1.1. Якщо поточний елемент є менший за наступний, то поміняти ці елементи місцями.

Сортування вставкою

В сортуванні методом вставки (англ. *insertion sort*) масив даних розділяється на дві частини: ліву – відсортовану і праву – невідсортовану. Перший елемент невідсортованої частини вставляється у відсортовану частину таким чином, щоб вона залишалася відсортованою. У результаті відсортована частина збільшується на один елемент, а невідсортована – зменшується. Таким чином, на кожному кроці потрібно знайти позицію у відсортованій частині для вставки нового елемента та здійснити саму вставку. Для забезпечення місця під вставку, елементи відсортованого підмасиву, ключі яких більші за ключ елемента, що вставляється, повинні бути зсунуті на одну позицію вправо. Оскільки такий зсув зітре перший елемент невідсортованої частини (власне той, що вставляється), перед вставкою його потрібно запам’ятати в допоміжній змінній.

На першій ітерації відсортована частина містить тільки перший елемент масиву, решту елементів масиву відносимо до невідсортованої частини.

Зауваження. Існує інша реалізація алгоритму, коли поточний елемент з невідсортованої частини переставляється місцями з попереднім елементом до тих пір, поки ключ цього елементу є більшим за його ключ. Така версія не вимагає введення допоміжної змінної, але потребує більшу кількість операцій (для здійснення перестановок).

Алгоритм сортування вставкою (функція `SortInsertion(…)`*).*

1. Встановити лічильник ітерацій рівний одиниці. Поки лічильник не досяг значення індексу останнього елементу масиву (цикл за зростанням індексу):
   1.1. Занести елемент, що вставляється (перший в невідсортованій ділянці – його індекс рівний лічильнику ітерацій) в допоміжну змінну.
   1.2. Для елементів масиву: від останнього елементу відсортованої частини допоки не знайдено місце для вставки, або не досягнуто початку масиву (цикл за спаданням індексу):
      1.2.1. Якщо ключ поточного елементу є більший за ключ елементу, що вставляється, то зсунути поточний елемент на одну позицію вправо (пошук місця для вставки).
   1.3. Вставити на місце останнього поточного елементу значення елементу, що вставляється з додаткової змінної.

Сортування вибором

При сортуванні вибором (англ. *selection sort*), як і в попередньому випадку, масив даних складається з відсортованої і невідсортованої частини. На кожному кроці у невідсортованій частині шукається елемент з мінімальним ключем і вставляється на її початок. Таким чином відсортована частина збільшується на один елемент, а невідсортована – зменшується. Процедура вставки здійснюється, як правило, шляхом обміну

елементу з мінімальним ключем з першим елементом у невідсортованій частині.

На першій ітерації невідсортована частина складає весь масив.

Зауваження. Пошук мінімального (максимального) елементу у послідовності (масиві) є самостійною задачею і може бути реалізований у вигляді окремої функції(напр. `MinItem(…)` або `MaxItem(…)`). Ця функція повинна повертати індекс елемента з мінімальним (максимальним) ключем та містити як параметри сам масив, у якому здійснюється пошук, та індекси елементів, які визначають ділянку пошуку (оскілки пошук відбувається не в усьому масиві, то потрібно задати перший та останній елементи підмасиву пошуку).

Алгоритм пошуку мінімального елементу в масиві (функція `MinItem(…)`*).*

1. Присвоїти змінній `min` значення індексу першого елементу підмасиву пошуку. Для елементів масиву: від другого до останнього елементів підмасиву (цикл за зростанням індексу):
   1.1. Якщо значення ключа поточного елементу є меншим за значення ключа елементу з індексом `min`, то присвоїти змінній `min` значення індексу поточного елементу.
2. Повернути значення `min` як результат функції.

Зауваження. Алгоритм пошуку максимального елементу в масиві є аналогічним попередньому.

Алгоритм сортування вибором (функція `SortSelection(…)`*).*

1. Для елементів масиву: від першого до останнього елементів (цикл за зростанням індексу):
   1.1. Знайти елемент з мінімальним ключем (функція `MinItem(…)`) серед елементів від поточного до останнього.

1.2. Переставити місцями знайдений елемент з мінімальним ключем з поточним.

Обчислювальна складність простих алгоритмів сортування становить $O(n^2)$, де n – кількість елементів масиву, що обумовлено наявністю двох вкладених циклів. Окрім того, у зв'язку з великою кількістю перестановок, метод обміну вважається найгіршим і на практиці не використовується. Серед вищеописаних алгоритмів найкращим вважається метод вставки, однак для великих наборів даних доцільніше використовувати удосконалені методи [3].

Хід роботи:

1. Створити нову бібліотеку `Sort` (файли `Sort.h, Sort.cpp`).
2. У файлі `Sort.h` за допомогою команди `typedef` зв'язати тип ключа елементу масиву `datatype` з типом даних, заданим викладачем.
3. У бібліотеці `Sort`, згідно описаних вище алгоритмів, реалізувати функції сортування методом обміну (бульбашки), вставки та вибору (функції `SortBubble(…), SortInsertion(…), SortSelection(…)`). Ці функції повинні приймати як параметри вказівник (ім'я) на масив з даними, та кількість елементів цього масиву.
4. В цій же бібліотеці запрограмувати допоміжні функції для перестановки елементів масиву (`swap(…)`), відображення вмісту масиву (`show(…)`) та пошуку елементу з мінімальним та максимальним ключем у частині масиву (`MinItem(…), MaxItem(…)`).
5. Створити новий проект `Lab_2` до якого підключити бібліотеку `Sort`. У функції `main()` проекту реалізувати меню для вибору методу сортування масиву з даними.
6. Модифікувати алгоритми сортування таким чином, щоб вони дозволяли відсортовувати тільки задану частину масиву. Для цього потрібно, як параметри відповідних функцій передавати індекси

елементів, які визначають ділянку (підмасив) сортування та змінити межі відповідних циклів у коді цих функцій відповідно до цих меж (аналогічно, як в алгоритмі пошуку мінімального значення в масиві). Зауваження. Процедура сортування частини масиву використовується в лабораторній роботі № 3.

7. Отримати від викладача завдання: масив даних відповідного типу. Продемонструвати викладачеві результат сортування цього масиву різними методами та для різних ділянок масиву.

Додаткові завдання

Класи, шаблони