Лабораторна робота №6

Швидкі методи сортування

Мета роботи: реалізація швидких алгоритмів сортування та дослідження їх характеристик (швидкодія, необхідний обсяг пам'яті, застосування тощо).

Методичні вказівки

Під **сортуванням** звичайно розуміють процес перестановки об'єктів заданої множини у певному порядку.

Мета сортування – полегшення подальшого пошуку елементів у відсортованій множині. У цьому сенсі елементи сортування присутні майже у всіх завданнях.

1. Пірамідальне сортування (сортування кучою)

Пірамідальне сортування (англ. Heapsort) — алгоритм сортування, який використовує бінарну кучу.

Куча (англ. Heap) – структура даних типу дерево, яка задовольняє наступній властивості: для будь-якого заданого вузла B, який є нащадком вузла A виконується умова: ключ (A) ≥ ключ (B). Таким чином, кореневий вузол кучи буде зберігати найбільше значення, тому іноді таку кучу називають max-кучою. Якщо змінити порівняння на протилежне, то кореневий вузол буде зберігати найменше значення і таку кучу називають min-кучою.

Поширеною реалізацією кучи є бінарна куча (англ. Binary heap), в якій дерево є двійковим деревом (рисунок 6.1 та рисунок 6.2). Двійкова куча задовольняє трьом умовам:

- значення в будь-якій вершині не менш, ніж значення її нащадків (max-куча);
- глибина всього листя відрізняється не більше ніж на 1 шар;
- останній шар заповнюється зліва направо.

Висота кореневого вузла (дерева): int(log2 N).

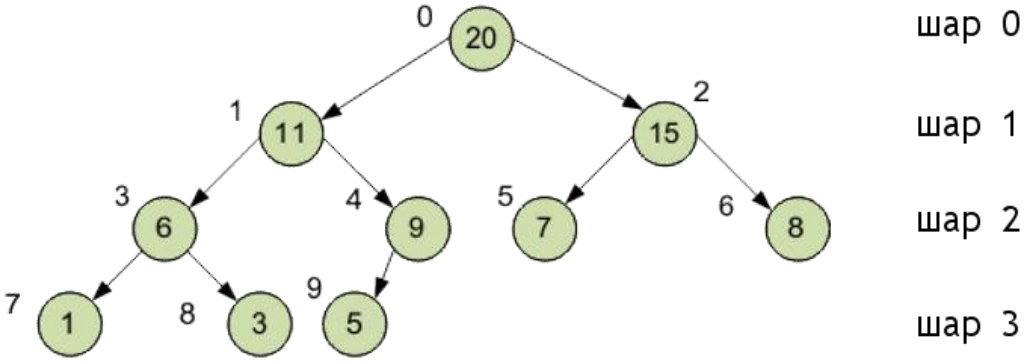


Рисунок 6.1 – Приклад max-кучи

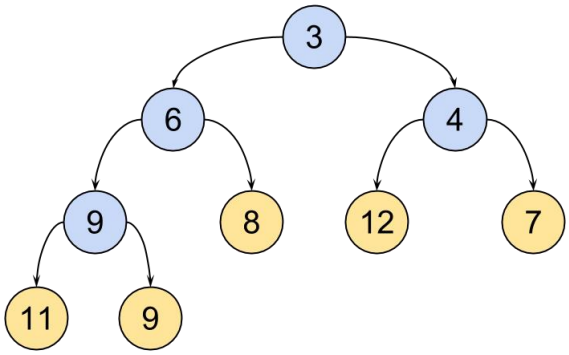


Рисунок 6.2 – Приклад min-кучи

Основні операції з кучею:

- Вилучити вузол з максимальним (мінімальним для min-кучи) значенням з кучи;
- Додати вузол у кучу;
- Змінити значення будь-якого вузла кучи.

Бінарну кучу можна зберігати у вигляді динамічного масиву:

- a[0] – корінь дерева;
- a[i] – вузел дерева;
- a[2i + 1] и a[2i + 2] – нащадки.

Приклад зберігання кучи (рисунок 6.2) у масиві (рисунок 6.3):

0	1	2	3	4	5	6	7	8
3	6	4	9	8	12	7	11	9

Рисунок 6.3 – Приклад зберігання min-кучи у масиві

Для реалізації основних операцій використовуються додаткові дві операції: просіювання вниз (sift_down) і просіювання вверх (sift_up). Алгоритми основних операцій представлені у вигляді псевдокоду:

```
// Метод просіювання вниз
def a.sift_down( i ): // O(log(n))
  for ; left = 2 * i + 1 < a.len(); i = j:
    right = left + 1
    j = left
    if right < a.len() && a[right] < a[left]:
      j = right
    if a[i] <= a[j]:
      break
    a[i], a[j] = a[j], a[i]
```

```
// Метод просіювання вверх
def a.sift_up( i ): // O(log(n))
  for j = (i - 1) / 2; a[i] < a[j]; j = (i - 1) / 2:
    a[i], a[j] = a[j], a[i]
    i = (i - 1) / 2


// Метод зміни значення вузла кучи
def a.node_change( i, value ): // O(log(n))
  if a[i] > value:
    a[i] = value
    a.sift_down( i )
  elif a[i] < value:
    a[i] = value
    a.sift_up( i )


// Метод вилучення мінімального елементу
def a.pop(): // O(log(n))
  min = a[0]
  a[0] = a[a.len() - 1]
  a.resize(a.len() - 1)
  sift_down(0)
  return min


// Метод додавання елементу
def a.push(value): // O(log(n))
  a[a.len()] = value
  sift_up(a.len() - 1)
```

Тепер, коли розглянуто роботу з кучою, розглянемо пірамідальне сортування. Суть алгоритму:

1. Вибудовуємо елементи масиву у вигляді бінарної кучи;

2. Видаляємо вузол з кореня дерева по одному за раз та зберегаємо;

3. Перебудовуваємо кучу;

4. Повторюємо пункти 2-4.

```
// Функція формування бінарної кучи з масиву
def buld_heap( a ):
  for i = a.len() / 2; i > 0; i--:
    a.sift_down(i)


// Функція сортування бінарної кучи
def heap_sort( a ): // O(nlog(n))
  buld_heap( a )
  for i = 0; i < a.len(); i++:
     b[i] = a.pop()
  return b
```

Показаний алгоритм потребує додаткової пам'яті O(n). Алгоритм можна змінити так, що буде потрібно пом'яти O(1), якщо врахувати, що після вилучення з кучи мінімального елементу розмір кучи зменшується на 1 комірку пам'яті, тому її можна використати для зберігання мінімального елементу. Однак в цьому випадку дані будуть відсортовані від більшого значення до меншого (за реалізацію даного варіанту алгоритму додатково буде нараховано 1 бал).

2. Сортування Шелла

Сортування Шелла є модифікацією алгоритму сортування вставками та класифікується як сортування вставками з убиваючим кроком.

Ефективність алгоритму полягає в тому, що на кожному з проміжних кроків сортується або невелике число елементів, або вже досить добре впорядковані набори елементів. Впорядкованість масиву зростає після кожного проходу.

В ході вивчення алгоритму досліджувалася залежність середнього числа перестановок від розміру масивів при N від 100 до 60000 для декількох типів послідовностей кроків, для яких були отримані відповідні залежності часу роботи від розміру масиву:

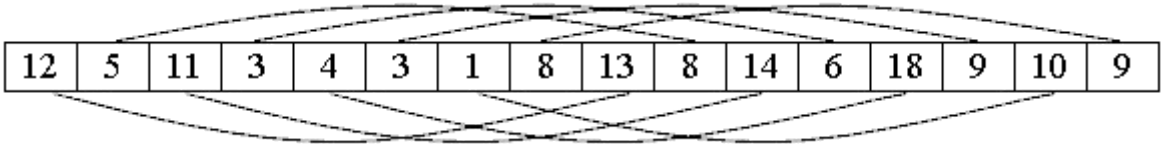
$2^k+1, \ldots, 9, 5, 3, 1 \Rightarrow 1{,}09N^{1{,}27}$
$2^k-1, \ldots, 15, 7, 3, 1 \Rightarrow 1{,}22N^{1{,}26}$
$(2^k-(-1)^k)/3, \ldots, 11, 5, 3, 1 \Rightarrow 1{,}12N^{1{,}28}$
$(3^k-1)/2, \ldots, 40, 13, 4, 1 \Rightarrow 1{,}66N^{1{,}25}$

Розглянемо алгоритм сортування масиву a [0] .. a [15] з кроками 8, 4, 2, 1.

Припустимо маємо початковий масив:

12	8	14	6	4	9	1	8	13	5	11	3	18	3	10	9

1. Спочатку сортуємо простими вставками кожні 8 груп з 2-х елементів (a[0], a[8[), (a[1], a[9]), ... , (a[7], a[15]).

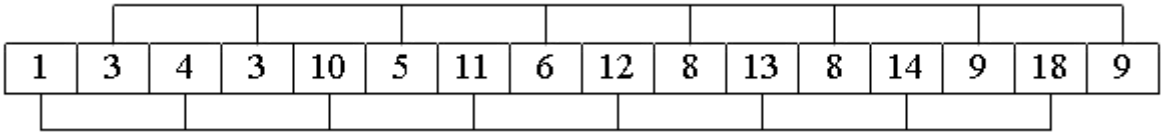


2. Потім сортуємо кожну з чотирьох груп по 4 елементи: (a[0], a[4], a[8], a[12]), ..., (a[3], a[7], a[11], a[15]).

номер группы:															
0	*1*	*2*	*3*	*0*	*1*	*2*	*3*	*0*	*1*	*2*	*3*	*0*	*1*	*2*	*3*
4	3	1	3	12	5	10	6	13	8	11	8	18	9	14	9

В нульової групи будуть елементи 4, 12, 13, 18, а в першій - 3, 5, 8, 9 і т.д.

3. Далі сортуємо 2 групи по 8 елементів, починаючи з (a[0], a[2], a[4], a[6], a[8], a[10], a[12], a[14]), …



4. В кінці сортуємо вставками всі 16 елементів (одну єдину групу).

1	3	3	4	5	6	8	8	9	9	10	11	12	13	14	18

Очевидно, що останнє сортування необхідне, щоб розташувати всі елементи по своїх місцях. Перші перестановки просувають елементи максимально близько до відповідних позицій, так що в останній стадії число переміщень буде досить невелике (послідовність буде майже відсортована). Прискорення підтверджено численними дослідженнями і на практиці виявляється досить суттєвим.

Єдиною характеристикою сортування Шелла є приріст – відстань між сортованими елементами, в залежності від проходу. В кінці приріст завжди дорівнює одиниці (метод завершується звичайним сортуванням вставками, але саме послідовність приросту визначає зростання ефективності).

Використаний в прикладі набір 8, 4, 2, 1 – непоганий вибір, особливо, коли кількість елементів – ступінь двійки. Однак набагато кращий варіант запропонував Р.Седжвік. Його послідовність має вигляд:

$$inc[s] = \begin{cases} 9*2^s - 9*2^{s/2} + 1, \text{ если } s \text{ четно} \\ 8*2^s - 6*2^{(s+1)/2} + 1, \text{ если } s \text{ нечетно} \end{cases}$$

При використанні таких приростів середня кількість операцій: O ($n^{7/6}$), в гіршому випадку – порядку O ($n^{4/3}$).

Звернемо увагу на те, що послідовність обчислюється в порядку протилежному використовуваному: inc[0] = 1, inc[1] = 5, ... Формула дає спочатку менші числа, а потім все більші і більші, в той час як відстань між сортованими елементами, навпаки, має зменшуватися. Тому масив приростів inc обчислюється перед запуском сортування (обчислюють до максимального приросту між елементами, який буде першим кроком в сортуванні Шелла).

При використанні формули Седжвика слід зупинитися на значенні inc[s-1], якщо 3*inc[s] > N.

3. Сортування підрахунком

Сортування підрахунком (англ. Counting sort) – алгоритм впорядкування, що застосовується при малій кількості різних елементів у масиві даних.

Застосування даного сортування доцільно використовувати лише тоді, коли кількість елементів масиву набагато більша за діапазон можливих значень. Наприклад, мільйон натуральних чисел, менших 100.

Суть алгоритму. Спочатку порахувати, скільки разів кожен елемент зустрічається у вихідному масиві. Спираючись на ці дані, можна вирахувати, на якому місці має стояти кожен елемент, і за один прохід поставити всі елементи на свої місця. Оцінка складності алгоритму O(n).

Розглянемо алгоритм сортування масиву A[0] .. A[7] із натуральних чисел, що не перевищують **10**.

Припустимо маємо початковий масив:

A	1	5	3	9	3	5	8	5

Оскільки елементи масиву – натуральні числа, що не перевищують 10, то значення належать проміжку [0, 10].

Заведемо масив B, в якому комірка масиву з номером **k** міститиме кількість значень **k** в масиві A.

Оскільки нас цікавлять комірки з номерами від 0 до 10, то потрібно виділити пам’ять під 11 клітинок.

Підрахуємо скільки разів зустрічаються числа [0, 10] та запишемо в масив B.

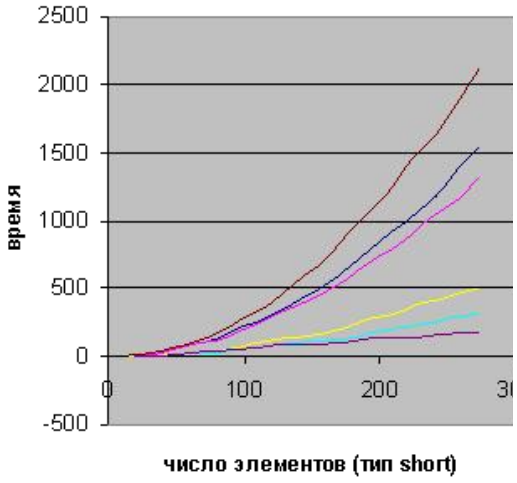
	0	1	2	3	4	5	6	7	8	9	10
B	0	1	0	2	0	3	0	0	1	1	0

Тоді можна записати результат сортування у масив A наступним чином: проходимо по масиву B и записуємо у масив A нуль разів “0”, один раз “1”, нуль разів “2”, два рази “3”, нуль разів “4”, три рази “5”, …

Відсортований масив A буде виглядати наступним чином:

A	1	3	3	5	5	5	8	9

Зображений нижче графік (рисунок 6.4) ілюструє різницю в ефективності розглянутих алгоритмів.



- коричнева лінія – сортування бульбашкою;
- синя лінія – шейкер-сортування;
- рожева лінія – сортування вибором;
- жовта лінія – сортування вставками;
- блакитна лінія – сортування Шелла;
- фіолетова лінія – сортування підрахунком.

Рисунок 6.4 – Ефективність алгоритмів

Порядок виконання роботи

1. Реалізувати алгоритми сортування (структура даних – масив, тип даних – int, (діапазон – [0, 1000])):
   1) пірамідальне сортування ;
   2) сортування Шелла (формула приросту – $inc(s)=(3^s-1)/2$);
   3) сортування підрахунком;
2. Виміряти час сортування даних різної розмірності: 10, 100, 500, 1000, 2000, 5000, 10000. Дані сформувати з використанням генератора випадкових чисел, де в якості початкового значення використати номер Вашого варіанту (для вимірювання

інтервалів часу можна використовувати клас Stopwatch з простору імен System.Diagnostics, якщо для C # або використати бібліотеку C++ <chrono>).

3. За отриманими даними побудувати графіки залежностей часу сортування від кількості вхідних даних (з використанням Excel).

Формат вхідних даних:

Єдиний рядок вхідних даних містить 3 цілих числа, розділених пробілами:

N V K

де:

N – номер алгоритму сортування (від 1 до 3);

V – номер варіанту (вхідний параметр для функції srand());

K – приймає кількість значень для сортування

Формат вихідних даних:

Програма повинна вивести розраховані перші 5 значень через пробіл

Приклад:

Вказуємо, наприклад, такі значення:

1 12 100

Результат виведення перших 5 значень у консоль:

-12 -3 2 3 9

Зміст звіту

1. Опис алгоритму (словесна форма або блок-схема алгоритму).
2. Текст функцій сортування з коментарями.
3. Таблиця результатів вимірів часу.
4. Графіки результатів вимірів часу.
5. Висновки по роботі (опис досліджених характеристик кожного алгоритму, порівняння алгоритмів, відзначити переваги та недоліки).