Лекція 8. Метод грубої сили.
План.

1. Визначення методу розробки алгоритмів.
2. Квадратичні алгоритми сортування.

1.

Метод проектування алгоритму (algorithm design technique) (або "стратегія", або "принцип") — це універсальний підхід, застосовуваний для алгоритмічного рішення широкого кола задач, що ставляться до різних областей обчислювальної техніки.
Метод грубої сили являє собою прямий підхід до розв'язання задачі, звичайно заснований безпосередньо на формулюванні задачі й визначеннях використовуваних нею концепцій.
Перефразувати визначення даної стратегії можна простіше: "Нема чого думати, треба діяти!". Найчастіше стратегія грубої сили виявляється найбільш простою у застосуванні.

Як приклад розглянемо задачу піднесення в ступінь: обчислення a^n для деякого числа a й ненегативного цілого n. Хоча ця задача може здатися тривіальної, вона дозволяє проілюструвати кілька методів розробки алгоритмів, у тому числі й підхід грубої сили. По визначенню піднесення в ступінь

$$a^n = \underbrace{a \cdot a \cdot \ldots \cdot a}_{n \text{ раз}}.$$

Звідси відразу треба найпростіший алгоритм обчислення a^n — шляхом множення на a початкового значення, рівного 1, n раз.
Ми вже зустрічалися на попередніх лекціях як мінімум із двома алгоритмами з використанням грубої сили: послідовна перевірка всіх цілих чисел при пошуку *gcd* (*m*,*n*) і алгоритм множення матриць, заснований на визначенні даної операції. Багато інших прикладів ви знайдете на цій лекції. (чи можете ви вказати трохи відомих вам алгоритмів, заснованих на використанні грубої сили?)
Хоча метод грубої сили рідко дає красиві або ефективні алгоритми, його розгляд не можна опустити, оскільки даний метод являє собою важливу стратегію розробки алгоритмів. По-перше, на відміну від інших стратегій, метод грубої сили застосуємо до дуже широкого діапазону задач. (Схоже, це єдиний підхід, для якого істотно складніше вказати задачу, для рішення якої він незастосовний.) Зокрема, метод грубої сили використається для багатьох елементарних, але важливих алгоритмічних задач, таких як обчислення суми n чисел, пошук найбільшого елемента в списку й тому подібних. По-друге, для деяких важливих задач (наприклад, сортування, пошуку, множення матриць, пошуку підстрок) метод грубої сили дає цілком раціональні алгоритми. По-третє, вартість розробки більше ефективного алгоритму може виявитися неприйнятної, якщо потрібно вирішити тільки кілька екземплярів задачі, а алгоритм, заснований на грубій силі, дозволяє вирішити їх за прийнятний час. По-четверте, навіть будучи неефективним у загальному випадку, метод грубої

сили може виявитися корисний для рішення невеликих по розмірі екземплярів задачі.
Нарешті, алгоритм, заснований на грубій силі, може служити для важливих теоретичних або дидактичних цілей, наприклад мірилом для визначення ефективності інших алгоритмів для рішення даної задачі.

2.

Розглянемо застосування методу грубої сили до задачі сортування, що полягає в наступному: даний список з *n* елементів, що впорядковуються, (наприклад, чисел, символів деякого алфавіту, символьних рядків) , які треба розмістити в неубутному порядку. Є десятки алгоритмів, розроблених для рішення цієї дуже важливої задачі. Деякі з них вам уже, можливо, зустрічалися. У такому випадку спробуйте на час забути про їх.

Бульбашкове сортування

Застосування методу грубої сили до задачі сортування складається в порівнянні сусідніх елементів і їхньому обміні, якщо вони перебувають не в належному порядку. Неодноразово виконуючи цю дію, ми змушуємо найбільший елемент "спливати" до кінця списку. Наступний прохід приведе до спливання другого найбільшого елемента, і так доти, поки після *n-1* ітерації список не буде повністю відсортований, *i-ий* прохід *0<=i<=n-2*) бульбашкового сортування можна представити наступною діаграмою:

$$A_0, \ldots, A_j \overset{?}{\leftrightarrow} A_{j+1}, \ldots, A_{n-i-1} \mid A_{n-i} \leq \cdots \leq A_{n-1}$$

Элементы в окончательных позициях

Нижче наведений псевдокод даного алгоритму.

```
Алгоритм BubbleSort (A[0..n − 1])
    // Сортировка массива пузырьковой сортировкой
    // Входные данные:  Массив A[0..n − 1] упорядочиваемых
    //                  элементов
    // Выходные данные: Массив A[0..n − 1], отсортированный
    //                  в неубывающем порядке
    for i ← 0 to n − 2 do
        for j ← 0 to n − 2 − i do
            if A[j + 1] < A[j]
                Обмен A[j] и A[j + 1]
```

Кількість порівнянь ключів у даній версії бульбашкового сортування однаково для всіх масивів розміром *n*. Воно представляється сумою, практично ідентичній сумі для сортування вибором:

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=0}^{n-2-i} 1 = \sum_{i=0}^{n-2} [(n-2-i) - 0 + 1] =$$

$$= \sum_{i=0}^{n-2} (n-1-i) = \frac{(n-1)\,n}{2} \in \Theta\left(n^2\right).$$

Кількість же обмінів елементів залежить від вхідних даних. У найгіршому випадку зменшуваного масиву воно дорівнює кількості порівнянь:

$$S_{worst}(n) = C(n) = \frac{(n-1)\,n}{2} \in \Theta\left(n^2\right).$$

Найчастіше при використанні методу грубої сили перша версія алгоритму може бути вдосконалена ціною досить скромних зусиль. Зокрема, наведену сиру версію бульбашкового сортування можна поліпшити, скориставшись наступним спостереженням: якщо при проході за списком не зроблено жодного обміну, виходить, список відсортований, і виконання алгоритму можна припинити.

Сортування вибором

Ми починаємо сортування вибором з пошуку найменшого елемента в списку й обміну його з першим елементом (таким чином, найменший елемент міститься в остаточну позицію у відсортованому списку). Потім ми скануємо список, починаючи із другого елемента, у пошуках найменшого серед що залишилися *n - 1* елементів і обмінюємо знайдений найменший елемент із другим, тобто поміщаємо другий найменший елемент в остаточну позицію у відсортованому списку. У загальному випадку, при *i-ом* проході за списком (0 =< *i* =< *n-2*) алгоритм шукає найменший елемент серед останніх *n-i* елементів і обмінює його з A_i.

$$A_0 \le A_1 \le \cdots \le A_{i-1} \mid \overbrace{A_i, \ldots, A_{min}, \ldots, A_{n-1}}$$

Элементы в окончательных позициях Последние $n - i$ элементов

Після виконання *n - 1* проходів список виявляється відсортований.

От псевдокод даного алгоритму, у якому для простоти передбачається, що список реалізований у вигляді масиву.

```
Алгоритм SelectionSort (A[0..n − 1])
  // Сортировка массива методом выбора
  // Входные данные:  Массив A[0..n − 1] упорядочиваемых
  //                  элементов
  // Выходные данные: Масєив A[0..n − 1], отсортированный
  //                  в неубывающем порядке
  for i ← 0 to n − 2 do
    min ← i
    for j ← i + 1 to n − 1 do
      if A[j] < A[min]
        min ← j
    Обмен A[i] и A[min]
```

Аналіз сортування вибором виконується досить просто. Розмір вхідних даних визначається кількістю *n* елементів у списку. Базовою операцією алгоритму є порівняння ключів *A [j] < A [min]*. Загальна кількість порівнянь залежить тільки від розміру масиву й визначається наступною сумою:

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i).$$

Обчислите зазначену суму.

$$C(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} (n-i-1) = \frac{(n-1)\,n}{2}.$$

Таким чином, для будь-яких вхідних даних алгоритм сортування вибором належить $\Theta(n^2)$. Помітимо, однак, що кількість обмінів елементів масиву дорівнює $\Theta(n)$, точніше, рівно n-1 — по одному для кожної ітерації циклу *i*. Ця властивість відрізняє сортування вибором від багатьох інших алгоритмів сортування.