Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Кафедра інтелектуальних інформаційних систем

Алгоритми та складність

Завдання №2

“Болти та гайки”

Виконав студент 2-го курсу

Групи К-28

Самойлич Євгеній Костянтинович

2020

**Завдання**:

Нехай є n болтів різного розміру та n відповідних гайок. Можна порівнювати, чи підходять гайка і болт одне до одного, або гайка більша чи менша. Порівнювати між собою дві гайки чи два болти не можна. Розробіть і реалізуйте алгоритм розбивки всіх гайок і болтів на відповідні пари за час Θ(n log n).

**Теорія**

Так як час роботи має бути рівним Θ(n log n) використаємо алгоритм швидкого сортування.

На першому кроці треба взяти першу гайку і розділити всі болти на три групи: 1) ті болти, що більші за цю гайку 2) ті болти, що менші за цю гайку 3) болт, який відповідає цій гайці. Перенести першу гайку і відповідний болт в розв’язок задачі.

Наступну гайку треба порівняти з вибраним для першої гайки болтом і тим самим визначити, до якої групи входить ця друга гайка. Тим самим способом знайти відповідний болт для другої гайки.

Цю процедуру виконати для кожної наступної гайки.

Такий алгоритм працює за **n\*log(n)**, що задовольняє умові завдання.

**Алгоритм**

Алгоритм швидкого сортування складається з трьох кроків:

• Вибрати елемент з масиву. Назвемо його опорним.

• Розбиття: перерозподіл елементів в масиві таким чином, що елементи менші опорного елементу поміщаються перед ним, а більше або рівні після.

• Рекурсивно застосувати перші два кроки до двох підмасивів зліва і праворуч від опорного елемента. Рекурсія не застосовується до масиву, в якому тільки один елемент або відсутні елементи.

**Складність**

На кожному кроці алгоритму знаходиться одна пара (болт,гайка), і задача розбивається на дві підзадачі меншого розміру.

Найгірший випадок – коли масив гайок розташований у порядку спадання, тому на кожному кроці виникає не дві підзадачі, а всього одна, тільки на 1 менша за попередню. Тоді складність алгоритму *n* + (*n*-1) + (*n* – 2) + … + 1 = *n*(*n*-1)/2 – квадратична.

Кращий та середній випадки – коли на кожному кроці масив болтів ділиться приблизно пополам. Тут складність – *O(n∙*log*n).*

**Мова програмування**

С++

**Модулі програми**

* void quickSort(Bolt\* B, Nut\* N, int left, int right)

Рекурсивна функція, яка сортує масив *arr2*. На вхід приймає масив *arr1* з якого береться опорне значення для допоміжної функції *partition* під індексом *index*, масив *arr2*, який треба відсортувати та значення лівого *left* та правого right індексу масиву *arr2*

* int partition(vector<int> arr, int left, int right, int pivot)

У функції partition відбувається перерозподіл елементів в масиві *arr* таким чином, що елементи менші опорного елементу *pivot* поміщаються перед ним, а більше або рівні після повертає позицію елемента у відсортованому масиві

**Інтерфейс користувача**

Вхідні дані вводяться з консолі і виводяться також в консоль.

**Тестові приклади**

|  |  |
| --- | --- |
| **input** | **output** |
| Number of bolts: 5  Bolts array: 39 9 32 6 3  Nuts array: 6 39 3 9 32 | Sorted bolts array: 2 3 6 9 29 32 34 39 41 46  Sorted nuts array: 2 3 6 9 29 32 34 39 41 46 |

Нехай масив болтів Б ={6, 9, 32, 3, 39}, масив гайок Г = {6, 39, 3, 9, 32}.

Спочатку елементи в масиві гайок розташовуються таким чином, що всі елементі лівіші за індекс p будуть менші за 6, а ті, що правіші – більше або рівні. Тобто масив гайок буде мати вигляд {3, 6, 9, 32, 39}. Тут p = 1 (нумерація іде з нуля). Далі рекурсивно застосовуємо функцію для масивів {3, 6} та {9, 32, 39} та бовтом розміром 9. Перший масив не зміниться, а другий розіб’ється на два масиви: {9} та {32, 39}. Далі рекурсивно застосовуємо функцію для бовта з розміром 3. Масив {3, 6} розіб’ється на два {3} та {6}, масиви {9} та {32, 39} не зміняться. Після останнього входження рекурсії можна бачити, що отримаємо відсортований масив {3, 6, 9, 32, 39}. Так само відсортуємо масив болтів. Отримаємо 2 масиви {3, 6, 9, 32, 39} та {3, 6, 9, 32, 39}, де, вочевидь, нульовому бовту відповідає нульова гайка, 1-ому бовту відповідає 1-ша гайка і так далі.

**Висновки**

В практичних задачах вхідні дані формуються випадковим чином. Малоймовірно, щоб вони розташувались у строгому порядку. Тому при застосуванні алгоритму складність, як правило, буде *O(n∙*log*n).*

**Література**

* Лекція №4
* http://kytok.org.ua/?p=347