ВикипедиЯ

Сортировка вставками

Материал из Википедии — свободной энциклопедии

Сортировка вставками (англ. *Insertion sort*) — алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов[1]. Вычислительная сложность — $O(n^2)$.

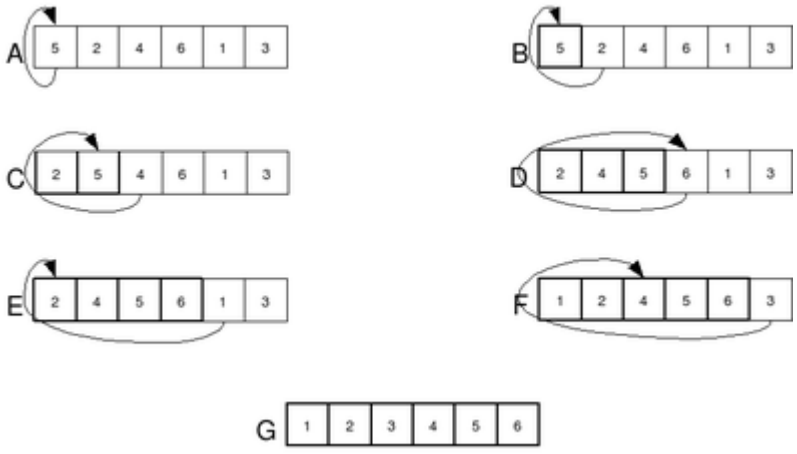
Сортировка вставками

Пример сортировки вставками

Предназначение	Алгоритм сортировки
Структура данных	Массив
Худшее время	O(n^2) сравнений, обменов
Лучшее время	O(n) сравнений, 0 обменов
Среднее время	O(n^2) сравнений, обменов
Затраты памяти	O(n) всего, O(*1*) вспомогательный

Содержание

Описание



Пример сортировки вставками

На вход алгоритма подаётся последовательность n чисел: $a_1, a_2, \ldots, a_n$. Сортируемые числа также называют *ключами*. Входная последовательность на практике представляется в виде массива с n элементами. На выходе алгоритм должен вернуть перестановку исходной последовательности $a'_1, a'_2, \ldots, a'_n$, чтобы выполнялось следующее соотношение $a'_1 \leqslant a'_2 \leqslant \ldots \leqslant a'_n$[2]

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма[3].

Данный алгоритм можно ускорить при помощи использования бинарного поиска для нахождения места текущему элементу в отсортированной части. Проблема с долгим сдвигом массива вправо решается при помощи смены указателей[4].

Псевдокод

На вход процедуре сортировки подаётся массив $A[1..n]$, состоящий из элементов последовательности $A[1], A[2], \ldots, A[n]$, которые требуется отсортировать. n соответствует $A.\mathrm{length}$ — размеру исходного массива. Для сортировки не требуется привлечения дополнительной памяти, кроме постоянной величины для одного элемента, так как выполняется перестановка в пределах массива. В результате работы процедуры во входном массиве оказывается требуемая выходная последовательность элементов[5].

Псевдокод алгоритма :

```
for j = 2 to A.length do
    key = A[j]
    i = j-1
    while (i >= 0 and A[i] > key)
do
        A[i + 1] = A[i]
        i = i - 1
    end while
    A[i+1] = key
end for[5]
```

```
for i = 2 to n do
    x = A[i]
    j = i
    while (j > 1 and A[j-1] > x)
do
        A[j] = A[j-1]
        j = j - 1
    end while
    A[j] = x
end for[6]
```

```
A[0] = -∞
for i = 2 to n do
    j = i
    while (j > 0 and A[j] < A[j - 1])
do
        swap (A[j], A[j - 1])
        j = j - 1
    end while
end for[7][8]
```

В последнем варианте обмен `x = A[j]; A[j] = A[j-1]; A[j-1] = x` представлен операцией swap из-за чего он немного медленнее. Значение введённого A[0] меньше любого значения остальных элементов.[8]

Анализ алгоритма

Время выполнения алгоритма зависит от входных данных: чем большее множество нужно отсортировать, тем большее время потребуется для выполнения сортировки. Также на время выполнения влияет исходная упорядоченность массива. Время работы алгоритма для различных входных данных одинакового размера зависит от элементарных операций, или шагов, которые потребуется выполнить[9].

Для каждой инструкции алгоритма введём временную стоимость и количество повторений, где t_j — количество проверок условия во внутреннем цикле *while*[10]:

Код	Стоимость	Повторы
for j = 2 to A.length	c_1	n
key = A[j]	c_2	$n-1$
i = j — 1	c_3	$n-1$
while i > 0 and A[i] > key	c_4	$\sum_{j=2}^{n} t_j$
A[i+1] = A[i]	c_5	$\sum_{j=2}^{n} (t_j - 1)$
i = i — 1	c_6	$\sum_{j=2}^{n} (t_j - 1)$
A[i+1] = key	c_7	$n-1$

Время работы алгоритма сортировки вставками — это сумма времён работы каждого шага[11]:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7(n-1).$$

Самым благоприятным случаем является отсортированный массив. При этом все внутренние циклы состоят всего из одной итерации, то есть $t_j = 1$ для всех j. Тогда время работы алгоритма составит $T(n) = (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7) = O(n)$. Время работы линейно зависит от размера входных данных[12].

Анализ наихудшего случая

Наихудшим случаем является массив, отсортированный в порядке, обратном нужному. При этом каждый новый элемент сравнивается со всеми в отсортированной последовательности. Это означает, что все внутренние циклы состоят из j итераций, то есть $t_j = j$ для всех j. Тогда время работы алгоритма составит:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} j + c_5 \sum_{j=2}^{n} (j-1) + c_6 \sum_{j=2}^{n} (j-1) + c_7(n-1)$$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \left(\frac{n(n+1)}{2} - 1 \right) + c_5 \frac{n(n-1)}{2} + c_6 \frac{n(n-1)}{2} + c_7(n-1) = O(n^2).$$

Время работы является квадратичной функцией от размера входных данных[13].

Анализ среднего случая

Для анализа среднего случая нужно посчитать среднее число сравнений, необходимых для определения положения очередного элемента. При добавлении нового элемента потребуется, как минимум, одно сравнение, даже если этот элемент оказался в правильной позиции. i-й добавляемый элемент может занимать одно из $i+1$ положений. Предполагая случайные входные данные, новый элемент равновероятно может оказаться в любой позиции[14]. Среднее число сравнений для вставки i-го элемента[15]:

$$T_i = \frac{1}{i+1} \left(\sum_{p=1}^{i} p + 1 \right) = \frac{1}{i+1} \left(\frac{i(i+1)}{2} + 1 \right) = \frac{i}{2} + 1 - \frac{1}{i+1}$$

Для оценки среднего времени работы для n элементов нужно просуммировать[16]:

$$T(n) = \sum_{i=1}^{n-1} T_i = \sum_{i=1}^{n-1} \left(\frac{i}{2} + 1 - \frac{1}{i+1} \right) = \sum_{i=1}^{n-1} \frac{i}{2} + \sum_{i=1}^{n-1} 1 - \sum_{i=1}^{n-1} \left(\frac{1}{i+1} \right)$$

$$T(n) \approx \frac{n^2 - n}{4} + (n-1) - (\ln(n) - 1) = O(n^2)$$

Временная сложность алгоритма — $O(n^2)$. Однако, из-за константных множителей и членов более низкого порядка алгоритм с более высоким порядком роста может выполняться для небольших входных данных быстрее, чем алгоритм с более низким порядком роста[17].

См. также

- Список алгоритмов сортировки
- Сортировка пузырьком
- Сортировка выбором
- Гномья сортировка

Примечания

1. *Кнут Д. Э.* 5.2 Внутренняя сортировка // Искусство программирования. Том 3. Сортировка и поиск (https://books.google.ru/books?id=92rW-nktlbgC&printsec=frontcover&dq=editions:spjjKVwoQ3QC&hl=ru&sa=X&ved=0ahUKEwiUjLbc88rQAhVCfiwKHUJlBQoQuwUIIjAB#v=onepage&q&f=false) = The Art of Computer Programming. Volume 3. Sorting and Searching / под ред. В. Т. Тертышного (гл. 5) и И. В. Красикова (гл. 6). — 2-е изд. — Москва: Вильямс, 2007. — Т. 3. — 832 с. — ISBN 5-8459-0082-1.
2. Кормен, 2013, p. 38.
3. Кормен, 2013, p. 39.
4. *Кнут Д. Э.* 5.2.1 Сортировка путём вставок // Искусство программирования. Том 3. Сортировка и поиск (https://books.google.ru/books?id=92rW-nktlbgC&printsec=frontcover&dq=editions:spjjKVwoQ3QC&hl=ru&sa=X&ved=0ahUKEwiUjLbc88rQAhVCfiwKHUJlBQoQuwUIIjAB#v=onepage&q&f=false) = The Art of Computer Programming. Volume 3. Sorting and Searching / под ред. В. Т. Тертышного (гл. 5) и И. В. Красикова (гл. 6). — 2-е изд. — Москва: Вильямс, 2007. — Т. 3. — 832 с. — ISBN 5-8459-0082-1.
5. Кормен, 2013, p. 39-40.
6. *Н. Вирт.* Алгоритмы и структуры данных. — М.: ДМК Пресс, 2010. — С. 74. — 272 с. — ISBN 987-5-94074-584-6.
7. *Скиена С.* Алгоритмы. Руководство по разработке. — 2-е. — СПб.: БХВ-Петербург, 2014. — С. 137. — 720 с. — ISBN 978-5-9775-0560-4.
8. Ахо, 2000, p. 237.
9. Кормен, 2013, p. 47.
10. Кормен, 2013, p. 48.
11. Кормен, 2013, p. 48-49.
12. Кормен, 2013, p. 49.
13. Кормен, 2013, p. 49-50.
14. Макконнелл, 2004, p. 74.
15. Макконнелл, 2004, p. 75.
16. Макконнелл, 2004, p. 75-76.
17. Кормен, 2013, p. 51.

Литература

- *Ахо А. В., Хопкрофт Д. Э., Ульман Д. Д.* Структуры данных и алгоритмы = Data structures and algorithms / Под ред. А. А. Минько. — М.: Вильямс, 2000. — С. 231. — ISBN 5-8459-0122-7.
- *Кнут Д. Э.* 5.2.1 Сортировка путём вставок // Искусство программирования. Том 3. Сортировка и поиск (https://books.google.ru/books?id=92rW-nktlbgC&printsec=frontcover&dq=editions:spjjKVwoQ3QC&hl=ru&sa=X&ved=0ahUKEwiUjLbc88rQAhVCfiwKHUJlBQoQuwUIIjAB#v=onepage&q&f=false) = The Art of Computer Programming. Volume 3. Sorting and Searching / под ред. В. Т. Тертышного (гл. 5) и И. В. Красикова (гл. 6). — 2-е изд. — Москва: Вильямс, 2007. — Т. 3. — 832 с. — ISBN 5-8459-0082-1.
- *Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К.* 2.1. Сортировка вставкой // Алгоритмы: построение и анализ = Introduction to Algorithms / Под ред. И. В. Красикова. — 3-е изд. — М.: Вильямс, 2013. — С. 38-45. — ISBN 5-8459-1794-8.
- *Макконнелл Дж.* Основы современных алгоритмов = Analysis of Algorithms: An Active Learning Approach / Под ред. С. К. Ландо. — М.: Техносфера, 2004. — С. 72-76. — ISBN 5-94836-005-9.

Ссылки

- Сортировка вставками (http://algolist.manual.ru/sort/insert_sort.php)
- Реализация сортировки вставками на Pascal и C++ (http://kvodo.ru/sortirovka-vstavkami-2.html)
- Анимированное представление алгоритма сортировки вставками (http://www.sorting-algorithms.com/insertion-sort)
