Алгоритмы и структуры данных

Сергей Григорян

11 сентября 2024 г.

Содержание

1 Инфа

Лектор: Степанов Илья Данилович.

`telegram: @irkstepanov`

2 Основные понятия

Фабула решения задачи

- Условие
- Алгоритм (реализация)
- Корректность
- Асимптотика/Время работы

Элементарные действия

- Сложение, умножения, сравнение чисел;
- Условные конструкции;
- Обращение по индексу (! Большое кол-во = иногда вредно);

Модель вычислений: RAM модель (Random Access Memory)

Замечание. *Бывают и др. модели:*

- *Параллельные вычисления*
- *Внешняя память*

3 Асимптотика, время работы

Пример. *Найти минимум в массиве.*

```
int n;
read(n);
int a[n];
read(a);
int x = +inf;
for i = 0..n-1:
    if (x < a[i]):
        x = a[i]
print(x)
```

Листинг 1: Нахождение минимума

Асимптотика: $O(n)$

Определение 3.1. Пусть $f, g : \mathbb{N} \to \mathbb{N}$. Тогда:

$$f = O(g) \iff \exists N, C \colon \forall n \geq N \colon f(n) \leq C * g(n)$$

Пример.

$$6n + 4 = O(n), 6n + 4 \leq 7n, \text{ при } n \geq 4.$$

Утверждение 3.1. $f = O(g) \iff \exists D \colon \forall n \colon f(n) \leq D * g(n)$

Доказательство.

$\Leftarrow$) $N = 1 \colon n \geq N, C = D, f(n) \leq c * g(n)$

$\Rightarrow$) Надо обеспечить:

$$f(1) \leq Dg(1)$$
$$f(2) \leq Dg(2)$$
$$\vdots$$
$$f(N) \leq Dg(N).$$

$$\Rightarrow D = max(C, \frac{f(1)}{g(1)}, \frac{f(2)}{g(2)}, \cdots, \frac{f(N)}{g(N)})$$

$\square$

Определение 3.2. Пусть $f, g : \mathbb{N} \to \mathbb{N}$. Тогда $f = \Omega(g)$, если $\exists C > 0, N \colon \forall n \geq N$:

$$f(n) \geq C * g(n)$$

Определение 3.3. $f, g : \mathbb{N} \to \mathbb{N}$. Тогда $f = \Theta(g) \iff \exists c_1, c_2 > 0, N \colon \forall n \geq N$:

$$c_1 * g(n) \leq f(n) \leq c_2 * g(n).$$

Пример. *1.* $n^a, n^b,\ a, b = const$

$$n^a = O(n^b), a \leq b.$$
$$n^a = \Omega(n^b), a \geq b.$$
$$n^a = \Theta(n^b), a = b.$$

2. $\log_a n = \Theta(\log_b n); a, b = const$

3. $n^n = O(2^{2^n}), 2^{2^n} = \omega(n^n)$

Утверждение 3.2.

$$\log n^a < n^b < c^n, \forall a > 0, b > 0, c > 1.$$

Утверждение 3.3. *Пусть $T(n)$ - время работы влгоритма на входных данных. Пусть:*

$$T(n) = T(\left\lfloor \frac{n}{2} \right\rfloor) + T(\left\lceil \frac{n}{2} \right\rceil) + O(n).$$

Тогда $T(n) = O(n \log n)$

Доказательство. $T(n) \leq T(\left\lfloor \frac{n}{2} \right\rfloor) + T(\left\lceil \frac{n}{2} \right\rceil) + Dn$

Докажем по индукции, что:

$$T(n) \leq C * n \log n,\ \text{при } n \geq 2$$

База индукции: $T(2), T(3), \cdots, T(10)$ - Взяли C, чтоб было верно.
Переход: Пусть $T(k) \leq Ck \log_2 k, k \leq n - 1$
Докажем для $k = n$:

$$T(n) \leq 2 * T(\left\lceil \frac{n}{2} \right\rceil) + Dn$$

$$\left\lceil \frac{n}{2} \right\rceil \leq \frac{n+1}{2} \Rightarrow$$

$$\Rightarrow T(n) \leq 2 * T(\left\lceil \frac{n}{2} \right\rceil) + Dn \leq$$

$$\leq 2 * C * \frac{n+1}{2} \log_2 \frac{n+1}{2} + Dn = C(n+1)(\log_2 (n+1) - 1) + Dn ==$$

$$n + 1 \leq n\sqrt{2}$$

$$\Rightarrow \log_2 n + 1 \leq \log_2 n\sqrt{2} = \log_2 n + \frac{1}{2}$$

$$== C(n+1)(\log_2 n - \frac{1}{2}) + Dn = Cn \log_2 n - \frac{1}{2}Cn + C \log_2 n - \frac{1}{2}C + Dn \leq Cn \log_2 n.$$

Достаточно д-ть, что $Dn + C \log_2 n \leq \frac{1}{2}Cn$

Для этого дост. положить $C \geq 6D$:

$$C = 6D.$$

$$Dn + 6D \log_2 n \leq 3Dn.$$

$$6 \log_2 n \leq 2n.$$

$\square$

4 Бинарный поиск

Задача 4.1. $a_0 \leq a_1 \leq a_2 \leq \cdots \leq a_{n-1}$

Узнать, есть ли x в a.

Наивное решение: q запросов $\Rightarrow O(nq)$

Решение. *Используем бинпоиск:*

```
int left = 0, right = n;
while (right - left > 1) {
    mid = (left + right) / 2
    if (a[mid] > x) right = mid
    else left = mid
}
```

```
if (a[left] == x) print("Yes");
else print("No");
```

Листинг 2: Binary Search

Асимптотика: $O(\log_2 n)$

5 Сортировки

Задача 5.1. $a_1, a_2, \ldots, a_n$ - дано Найти $b =$sort(a)

Задача 5.2. $a_1, a_2, \ldots, a_n$ - дано. Найти перестановку $G : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$: $a_{G(1)} \leq a_{G(2)} \leq \ldots \leq a_{G(n)}$

Утверждение 5.1. *Пусть $T(n) \geq n$. Тогда, если одна задача решается за $O(T(n))$, то и вторая тоже.*

Доказательство. $2 \Rightarrow 1$: $b_1 = a_{G(1)}, \ldots, b_n = a_{G(n)}$

$1 \Rightarrow 2$: Отсортируем массив пар: $(a_1, 1), (a_2, 2), \ldots, (a_n, n)$

$$(x_1, y_1) \leq (x_2, y_2) \iff \left[\begin{aligned} &x_1 < x_2 \\ &x_1 = x_2, y_1 < y_2 \end{aligned}\right.$$

□

Теорема 5.1 (Оценка кол-во сравнений в сортировке сравнениями)**.** *Если алгоритм может только сравнивать эл-ты, то время сортировки массива* $\in \Omega(n \log n)$

Доказательство. $a_1, a_2, \ldots, a_n$ - все эл-ты попарно различны.

Вопрос алгоса: $a_i?a_j$ (Дерево сравнений)

Рез-т работы алгоритма зависит от n и п-ти получаемых ответов $(<, >)$

Пусть алгоритм всегда совершает $\leq q$ запросов $(a_i?a_j)$. Тогда есть не более $2^0 + 2^1 + \ldots + 2^q = 2^{q+1} - 1 \leq 2^{q+1}$ возм. протоколов работы алгоритма. Должно быть хотя бы $n!$ разл. протоколов. $\Rightarrow$

$$n! \leq 2^{q+1} \Rightarrow q = \Omega(n \log n)?$$

□

<u>Лемма</u> 5.2.

$$\log(n!) = \theta(n \log n)$$

Доказательство. 1)

$$n! = 1 * 2 * \ldots * n \leq n^n$$

$$\log_2(n!) \leq n \log_2(n) \Rightarrow \log(n!) = O(n \log n)$$

2)

$$n = 2k \Rightarrow n! \geq (k+1) * \ldots * (2k) \geq k^{k+1}$$

$$\log_2(n!) \geq \log_2(k^{k+1}) = (k+1)\log_2 k \geq \frac{n}{2}\log_2(\frac{n}{2}) = \frac{1}{2}n(\log n - 1) = \frac{1}{2}n\log n - \frac{1}{2} \geq \frac{1}{2}n\log n$$

$$\Rightarrow \log_2(n!) = \Omega(n \log_2 n)$$

Аналогично, при $n = 2k + 1$

□

5.1 Merge Sort (Сортировка слиянием)

Алгоритм:

1) Если массив длины 1, то выходим, иначе делим его на 2 половины;

2) Рекурсивно сортируем половины

3) "Мёрджим"две половины.

Операция merge:

```
merge(a[0..n - 1], b[0..m - 1], to[0..n + m - 1]):
    i = 0, j = 0
    while (i < n || j < m):
        if (j == m || (i < n && a[i] <= b[j])):
            to[i + j] = a[i]
            ++i
        else
            to[i + j] = b[j]
            ++j
```

Листинг 3: Merge

```
MergeSort(a[0..n - 1]):
    if (n <= 1) return
    k = n / 2
    l = a[0..k]
    r = a[k + 1..n - 1]
    MergeSort(l)
    MergeSort(r)
    Merge(l, r, a)
```

Листинг 4: MergeSort

Асимптотика: $T(n) = 2T(\frac{n}{2}) + O(n) \Rightarrow T(n) = O(n \log n)$

Задача 5.3. 1) Сделать потребление памяти $O(n)$

2) Сделать нерекурсивный MergeSort

Задача 5.4 (О числе инверсий в массиве)**.**

$a_1, a_2, \ldots, a_n$ - дано
Инверсия $(i, j) := i < j \wedge a_i > a_j$

Решение. *Для решения просто модифицируем merge:*

```
// ans - global variable
merge(a[0..n - 1], b[0..m - 1], to[0..n + m - 1]):
    i = 0, j = 0
    while (i < n || j < m):
        if (j == m || (i < n && a[i] <= b[j])):
            to[i + j] = a[i]
            ++i
            ans += j
        else
            to[i + j] = b[j]
            ++j
            ans += i
```

Листинг 5: Merge for inversions

5.2 Quick Sort (Быстрая сортировка; Сортировка Хоара)

```
Partition(a[0..n - 1], x):
    B =>  < x
    C =>  = x
    D =>  > x
    return (|B|, |C|, |D|)

QuickSort(a[0..n - 1]):
    if (n <= 1) return;
    x = a[random(0, n - 1)]
    l, m, r = Partition(a, x)
    QuickSort(a[1..l - 1])
    QuickSort(a[l + m..n - 1])
```

Листинг 6: Quick Sort

Теорема 5.3. *В среднем асимпт.* $= O(n \log n)$

5.3 Quick Select

Определение 5.1. $a_1, a_2, \ldots, a_n$ - массив **k-ая порядковая статистика:** - эл-т на k-ом эл-те после сортировки.

Задача 5.5. Найти k-ую порядковую статистику в массиве a.

Решение.

```
QuickSelect(a[1..n], k):
    if (n == 1) return a[1];
    l, m, r = Partition(a, a[random(1, n)])
    if (k <= l) return QuickSelect(a[1..l], k)
    if (k <= l + m) return x
    return QuickSelect(a[l + m..n], k - l - m)
```

Листинг 7: QuickSelect