Основы программирования

Анализ трудоемкости алгоритмов

Трудоемкость алгоритма

Элементарный шаг – это действие, время выполнения которого не зависит от числа входных переменных и их значений.

Трудоемкость – это функция зависимости количества элементарных действий от входного параметра n при $n \to \infty$ (асимптотическая трудоемкость).

На практике важно не точное значение, а порядок роста $T(n)$ при $n \to \infty$.

Используется обозначение $T(n) = O\big(f(n)\big)$, если существуют такие константы $c > 0, n_0 > 0$,

что $\forall\, n > n_0 \;\; T(n) \le cf(n)$.

Типичные случаи для трудоемкости

- $T(n) = O(\log n)$ - логарифмическая,
  $\log_a n = \log_a 2 \cdot \log_2 n$ (отличаются в const раз)

$T(n) = O(n)$ - линейная

$T(n) = O(n \log n)$ - линейно-логарифмическая

$T(n) = O(n^k), k > 1$ – полиномиальная

$T(n) = O(2^{cn}), c = const$ – экспоненциальная,

для случая $a^{cn}, a > 1, k > 0$: $a^{cn} = 2^{\log a \cdot cn}$, т.е.
разница в $2^{(c \log a - 1)n} \neq const$ раз.

$T(n) = O(n!)$

Соотношения для оценки и сравнения трудоемкостей

$\forall\ a > 0, b > 0$ выполняется:

$$\lim_{n\to\infty} \frac{\log^a n}{n^b} = 0 \qquad \lim_{n\to\infty} \frac{n^b}{2^n} = 0$$

Формула Стирлинга для больших значений n:

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n = O(n^n)$$

$$\log(n!) \approx n\log n - n\log e + \frac{1}{2}(\log n + \log 2\pi) = O(n\log n)$$

Типы трудоемкостей

- Трудоемкость в наихудшем $T_{worst}(n)$

Трудоемкость в наилучшем $T_{best}(n)$

Трудоемкость в среднем:

для генеральной совокупности всех случаев выполнения алгоритма на разных входах i с вероятностями p_i и трудоемкостями $T_i(n)$ вычисляется $T_{mid}(n) = \sum_i p_i T_i(n)$

Эффективный алгоритм:

- имеет наилучшее соотношение трудоемкости и емкостной сложности или
- имеет трудоемкость на уровне известной нижней границы

Алгоритмы, основанные на сравнениях

- Пусть имеется множество решений V.

Первое сравнение приводит к разделению всего множества решений на два подмножества и выбору одного из них. После 2 сравнений мы потенциально можем проверить $2^2 = 4$ различных подмножеств, после m сравнений - 2^m.

Наша цель – выйти на одно из $|V|$ конечных решений. Гарантированно сделать это за m сравнений для любого входа можно только при $2^m \geq |V|$, т.е. при $m \geq \log(|V|)$ – это минимальная гарантированная трудоемкость в наихудшем для алгоритмов основанных на сравнениях.

Поиск в массиве

- Мощность множества решений $|V| = n$.

В неупорядоченном массиве $T_{worst}(n) = O(n)$.

Для дихотомического поиска:

в общем случае $2^{k-1} < n \leq 2^k$, и при первом разделении массив делится на части длины $\lfloor n/2 \rfloor$ и $\lceil n/2 \rceil$, причем $2^{k-2} \leq \lfloor n/2 \rfloor \leq \lceil n/2 \rceil \leq 2^{k-1}$.

Поэтому требуется не более $k = \lceil \log n \rceil$ разделений массива и $T_{worst}(n) = \lceil \log n \rceil + 1 = O(\log n)$, т.е. совпадает с минимальной гарантированной трудоемкостью в наихудшем. Следовательно, алгоритм дихотомического поиска является эффективным.

1-я теорема о временной сложности

- Пусть трудоемкость можно представить соотношением

$$T(n)=\begin{cases} b, n=1 \\ aT(n-1)+bn^p, n>1, p\geq 0 \end{cases}$$

Тогда:

1. $T(n)=O(n^{p+1})$ при $a=1$
2. $T(n)=O(2^{cn}), c=const$, при $a>1$.

Доказательство основано на последовательном разложении рекуррентного соотношения.

1-я теорема о временной сложности

Доказательство:

$$T(n) = aT(n-1) + bn^p = a^2T(n-2) + ab(n-1)^p + bn^p =$$

$$= a^3T(n-3) + a^2b(n-2)^p + ab(n-1)^p + bn^p =$$

$$= b(a^{n-1}1^p + a^{n-2}2^p + \ldots + a(n-1)^p + 1 \cdot n^p)$$

1. При $a = 1$ и больших n: $T(n) \approx b\dfrac{n^{p+1}}{p+1} = O(n^{p+1})$

2. Если $a > 1$, то $T(n)$ минимальна при $p = 0$:

$$T_{\min}(n) = \frac{b(a^n - 1)}{a-1} = \frac{b(2^{\log a \cdot n} - 1)}{a-1} = O(2^{cn})$$

В случае $T(n) = aT(n-c) + bn^p$ порядок сохраняется

Примеры использования 1-й теоремы

- Все простые алгоритмы сортировки массива длины n:

$$T(n) = cn + T(n-1) = O(n^2)$$

Рекурсивный алгоритм для чисел Фибоначчи:

$$T(n) = T(n-1) + T(n-2) + b = O(2^n)$$

Задача «Ханойские башни»:

$$T(n) = 2T(n-1) + b = O(2^n)$$

2-я теорема о временной сложности

- Пусть трудоемкость можно представить соотношением

$$T(n)=\begin{cases} b, n=1 \\ aT\left(\frac{n}{c}\right)+bn^p, n>1, c>1, p\geq 0 \end{cases}$$

Тогда:

1. $T(n)=O(n^p)$ при $a<c^p$
2. $T(n)=O(n^p \log n)$ при $a=c^p$
3. $T(n)=O(n^{\log_c a})$ при $a>c^p$

Доказательство основано на последовательном разложении рекуррентного соотношения.

2-я теорема о временной сложности

Доказательство: положим $n = c^k \, (n, k \to \infty)$, тогда

$$T(n) = aT\left(\frac{n}{c}\right) + bn^p = a^2 T\left(\frac{n}{c^2}\right) + ab\left(\frac{n}{c}\right)^p + bn^p =$$

$$= a^3 T\left(\frac{n}{c^3}\right) + a^2 b\left(\frac{n}{c^2}\right)^p + ab\left(\frac{n}{c}\right)^p + bn^p =$$

$$= b\left(a^k \left(\frac{n}{c^k}\right)^p + a^{k-1}\left(\frac{n}{c^{k-1}}\right)^p + \ldots + a^1\left(\frac{n}{c^1}\right)^p + a^0\left(\frac{n}{c^0}\right)^p \right) =$$

$$= bn^p \left(\left(\frac{a}{c^p}\right)^k + \left(\frac{a}{c^p}\right)^{k-1} + \ldots + \left(\frac{a}{c^p}\right)^1 + \left(\frac{a}{c^p}\right)^0 \right)$$

где $k = \log_c n = \log_c 2 \cdot \log n.$

2-я теорема о временной сложности

- 1. Если $a < c^p$, то $T(n) = bn^p \dfrac{1 - \left(\dfrac{a}{c^p}\right)^{k+1}}{1 - \dfrac{a}{c^p}} = O(n^p)$

2. Если $a = c^p$, то $T(n) = bn^p(\log_c n + 1) = O(n^p \log n)$

3. Если $a > c^p$, то $T(n) = bn^p \dfrac{\left(\dfrac{a}{c^p}\right)^{k+1} - 1}{\dfrac{a}{c^p} - 1} = \dfrac{bn^p}{\dfrac{a}{c^p} - 1}\left(\dfrac{a^{k+1}}{n^p c^p} - 1\right)$

где $\log_c a > p$, $a^k = a^{\log_c n} = n^{\log_c a}$ и $T(n) = O(n^{\log_c a})$.

Порядок $T(n)$ сохраняется и при $c^{k-1} < n \le c^k$.

Примеры использования 2-й теоремы

- Дихотомический поиск в массиве длины n:

$$T(n) = b + T(n/2) = O(\log n)$$

Рекурсивная сортировка слиянием:

$$T(n) = 2T(n/2) + bn = O(n \log n)$$