Сложность алгоритмов

Николай Вяххи

Санкт-Петербургский Государственный Университет

19 марта 2010 г.

Теория сложности

Проблема: необходимо сравнивать быстродействие алгоритмов.

Сложность алгоритма - функция от размера входных данных (например, количества байт).

Чем больше значение, тем алгоритм медленнее работает.

Обычно возрастает от размера входных данных
чем больше массив, тем дольше его сортировать.

Виды сложности

Бывают сложности:

- Временная - сколько времени (миллисекунд) понадобится алгоритму для работы.
- Пространственная - сколько памяти (байт) потребуется алгоритму для работы.

Бывают сложности:

- В худшем случае - для самого плохого ввода.
- В наилучшем случае - для самого хорошего ввода.
- В среднем - это математическое ожидание от всех возможных вводов.

Ассимптотическая сложность

Компьютеры у всех разные, языки программирования разные, пятна на солнце разные. Точное значение сложности искать бессмысленно.

Поэтому мы:

- Рассматриваем входные данные большого размера (можно мелочами пренебречь)
- Оцениваем порядок роста (то есть ищем значение с точностью до константы)

Куча формул (ассимптотические обозначения)

- $f(n) \in O(g(n))$
  f ограничена сверху функцией g (с точностью до постоянного множителя) асимптотически
  $\exists(C > 0), n_0 : \forall(n > n_0)\ |f(n)| \leq |Cg(n)|$
- $f(n) \in \Theta(g(n))$
  f ограничена снизу и сверху функцией g асимптотически
  $\exists(C, C' > 0), n_0 : \forall(n > n_0)\ |Cg(n)| < |f(n)| < |C'g(n)|$
- $f(n) \in o(g(n))$
  g доминирует над f асимптотически
  $\forall(C > 0), \exists n_0 : \forall(n > n_0)\ |f(n)| < |Cg(n)|$
- $f(n) \sim g(n)$
  f эквивалентна g асимптотически
  $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$

Ассимптотическая сложность

На самом деле, нам нужно только O-большое (big O notation):

$$f(n) \in O(g(n))$$

f ограничена сверху функцией g (с точностью до постоянного множителя) асимптотически

$$\exists(C > 0), n_0 : \forall(n > n_0)\ |f(n)| \leq |Cg(n)|$$

Стоит заметить

Иногда следует учитывать длину числа.

Например, сложение двух чисел занимает по времени:

- $O(1)$ если число атомарно (например, 64 битный int).
- $O(n)$ если длина числа n.

Стоит заметить

Иногда за $O()$ прячутся огромные константы, что для практического применения простой $O(n)$ лучше сложного мудреного $O(logn)$.

В численных алгоритмах точность и устойчивость алгоритмов не менее важны, чем их временная эффективность.

Примеры

$O(1)$ - перемножить два числа.
$O(logn)$ - найти число в отсортированном списке.
$O(n)$ - найти сумму чисел в списке.
$O(nlogn)$ - отсортировать список быстро.
$O(n^2)$ - отсортировать список пузырьком.
$O(n^3)$ - нахождение всех минимальных путей в графе.
$O(2^n)$ - перебрать все подмножества.
$O(n!)$ - перебрать все перестановки.