Анализ производительности алгоритмов

Литература

1. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. — М.: Изд. Дом Вильямс, 2000. — 960 с.
2. Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы. – М.: Изд. Дом Вильямс, 2000. – 384с.
3. Кнут Д. Искусство программирования, т.1 Основные алгоритмы, Изд. Дом Вильямс, 2000. – 384с.

Литература

1. Левитин, А. Алгоритмы: введение в разработку и анализ. : Пер. с англ. — М. : Издательский дом "Вильяме", 2006. — 576 с.
2. Дж. Макконнелл Основы современных алгоритмов. 2-е издание. М.: Техносфера, 2004. - 368с.

Понятие сложности алгоритмов

- Анализом задач с точки зрения вычислительной сложности занимается раздел теории алгоритмов – *теория сложности вычислений*
- Для программиста **теория сложности** – это набор общих методов, позволяющих:
  - существенно минимизировать прямолинейный перебор вариантов,
  - или показать, что задача в рассматриваемой постановке неразрешима.

Как оценить эффективность алгоритма?

- Используют *порядок роста необходимого времени или емкости памяти при увеличении входных данных*.
- **Время (память),** затраченное алгоритмом, как **функция размера задачи**, называется **временной (емкостной) сложностью алгоритма**.
- Поведение этой сложности в пределе при увеличении размера задачи называется асимптотической временной (емкостной) сложностью.

Пример

- Алгоритм вычисления значения многочлена степени *n* в заданной точке *x*.

$$P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_i x^i + \ldots + a_1 x^1 + a_0$$

Алгоритм 1

- Для каждого слагаемого, кроме a_0 возвести *x* в заданную степень последовательным умножением и затем домножить на коэффициент. Затем слагаемые сложить.
- Вычисление i-го слагаемого *(i=1..n)* требует *i* умножений.
- всего $1 + 2 + 3 + \ldots + n = n(n+1)/2$ умножений.
- кроме того, требуется $n+1$ сложение.
- Всего $n(n+1)/2 + n + 1 = n^2/2 + 3n/2 + 1$ операций.

Пример

Алгоритм 2

- Вынесем *x* за скобки и перепишем многочлен в виде

$$P_n(x) = a_0 + x(a_1 + x(a_2 + \ldots (a_i + .. x(a_{n-1} + a_n x)))$$

- Самая внутренняя скобка требует одно умножение и одно сложение. Ее значение используется для следующей скобки.
- И так, одно умножение и одно сложение на каждую скобку, которых *n-1* штука.
- И еще после вычисления самой внешней скобки умножить на *x* и прибавить a_0.
- Таким образом, всего *n* умножений и *n* сложений, всего *2n* операций.

Обозначения сложности

- широкое распространение для оценивания алгоритмов в отношении размера входных данных получили обозначения с использованием символа *O(*).*
- Типичный результат:
  - сложность алгоритма сортировки – *O(nlogn).* Читается как «сложность алгоритма порядка *nlogn*»
  - это следует понимать так: существует константа C > 0, такая, что время работы алгоритма в худшем случае не превышает $C \cdot n \cdot log_2 n$.
  - Существует и другой подход, который заключается в рассмотрении сложности в среднем.

- Выражение *O(*)* показывает, насколько быстро увеличивается время работы алгоритма с увеличением размерности, т.е. алгоритм, работающий за время *O(nlogn)* лучше алгоритма с временем работы $O(n^2)$.
- Таким образом, сложность алгоритма определяется как порядок функции, выражающее время его работы или затрачиваемую память.

Примеры

Алгоритм	Сложность
a[1]:=10	$O(1)$
i:=1…n a[i]:=i	$O(n)$
i:=1…n j:=1…m a[i]:=a[i]+a[j]	$O(n^2)$
Умножение матриц	$O(n^3)$

Сравнение среднего, худшего и лучшего случаев

Количество элементов (n)	***Средний случай:*** $nlogn$	***Худший случай:*** n^2
8	24	64
64	384	4096
2048	22528	4194304