Функция сложности алгоритма

Анализ алгоритмов обусловлен рядом важных причин. Одной из них является необходимость получения оценок или границ для объема памяти или времени работы, которое потребуется алгоритму для успешной обработки конкретных данных.

Для оценки качества алгоритма вводится понятие *сложность алгоритма*, или обратное понятие — *эффективность алгоритма*. Чем большее время и объем памяти требуются для реализации алгоритма, тем больше его сложность и соответственно ниже эффективность. Сложность алгоритма делится на временную и емкостную. Временная сложность — это критерий, характеризующий временные затраты на реализацию алгоритма. Емкостная сложность — критерий, характеризующий затраты памяти на те же цели. В зависимости от конкретной формы этих критериев сложность алгоритма в свою очередь подразделяется на *практическую* и *теоретическую.* Практическая временная сложность обычно оценивается во временных единицах (секунды, милисекунды, количество временных тактов процессора, количество выполнения циклов и т.п.). Практическая емкостная сложность выражается, как правило, в битах, байтах, словах и т.п. Способы представления теоретической сложности алгоритма будут рассмотрены далее.

Перечислим основные факторы, от которых может зависеть сложность алгоритма:

- быстродействие компьютера и его емкостные ресурсы (в первую очередь — объем оперативной памяти). В самом деле, чем ниже тактовая частота процессора и меньше объем оперативного запоминающего устройства, тем медленнее выполняются арифметические и логические операции, тем чаще (для больших задач) приходится обращаться к медленно действующей внешней памяти, и, следовательно, больше времени уходит на реализацию алгоритма;
- выбранный язык программирования. Задача, запрограммированная, например, на языке Ассемблера, в общем случае решится быстрее, чем по тому же самому алгоритму, но запрограммированному на языке более высокого уровня, например на Паскале;
- выбранный математический метод формулирования задачи;
- искусство и опыт программиста. В общем случае по одному и тому же алгоритму опытный программист напишет более эффективно работающую программу, чем его начинающий коллега.

Для решения многих задач можно применить несколько разных алгоритмов. Какой же из них выбрать? В программировании этот вопрос тщательно прорабатывается. Важной характеристикой программы является ее эффективность, которая имеет две составляющие: память и время. Пространственная эффективность программы определяется объемом памяти, требуемой для ее выполнения. Временная эффективность измеряется временем, необходимым для выполнения программы.

Раньше память была доминирующим фактором в оценке эффективности программы, т.к. компьютеры имеют ограниченный объем памяти. При реализации идентичных функций несколькими программами, та программа, которая использует меньшее количество памяти, обладает большей пространственной эффективностью. Но в последнее время в связи с удешевлением памяти эта составляющая эффективности стала терять свое значение.

Сравнение эффективностей алгоритмов состоит в сопоставлении их порядков временной или пространственной сложности. Порядок сложности алгоритма может определяться количеством обрабатываемых данных. Например, некоторые алгоритмы существенно зависят от размера обрабатываемого массива. В этом случае, если с удвоением размера массива время обработки удваивается, то порядок временной сложности алгоритма определяется размером массива. Порядок алгоритма является функцией, доминирующей над точным выражением временной сложности.Если имеется константа k и счетчик n_0, такие, что $f(n) \le kg(n)$ для $n > n_0$ то функция $f(n)$ имеет порядок $O(g(n))$.Действительное время — это функция, которая имеет завимость от длины массива.

Например, если точное время обработки массива определяется из такого уравнения:

Действительное время (Длина массива) =

Длина массива^2 + 5 • Длина массива +100

Грубое значение определяется вспомогательной функцией:

Оценка времени (Длина массива) = 1,1 •Длина массива^2

Функция сложности ***O*** выражает относительную скорость алгоритма в зависимости от некоторых переменных (или переменной). Для определения функции сложности существуют три правила:

1. ***O(kf) = O(f)*** —порядок сложности не зависит от постоянных множителей, например:

O(1,5N) = O(N)

2. ***O(fg) = O(f) O(g)*** или ***O(f/g) = O(f)/O(g)*** — порядок сложности произведения функций равен произведению их сложностей, например:

O(17N x N) = O(17N) x O(N) = O(N) x O(N)= O(N x N) = O(N^2)

3. ***O(f + g)*** равна доминанте ***O(f)*** **и** ***O(g)*** — порядок сложности суммы функций определяется порядком доминанты слагаемых (выбирается наибольший порядок), например:

O(N^5 + N^2) = O(N^5)

Здесь ***k*** обозначает константу, а ***f*** и ***g*** — функции.

Виды функции сложности алгоритмов

Функция сложности O(1). Любой алгоритм, выполняющийся всегда за одно и то же время независимо от размера данных, имеет константную сложность. В алгоритмах константной сложности операции в программе выполняется один или несколько раз.

Функция сложности O(N). Время работы программы линейно, когда каждый элемент входных данных обрабатывается линейное число раз.

Функция сложности O(N^2), O(N^3), O(N^a) — полиномиальная функция сложности.

Функция сложности O($\log_2 N$), *O*($N \log_2 N$). Такое время работы имеют алгоритмы, которые делят одну большую проблему на несколько небольших, а решив их, объединяют решения.

Функция сложности O(2^N). Экспоненциальная сложность. Чаще всего такие алгоритмы возникают в результате подхода, называемого «метод грубой силы».

Временная функция сложности

Программисту надо уметь анализировать алгоритмы и определять их сложность. Временная сложность алгоритма может быть определена после анализа его управляющих структур.

Алгоритмы без рекурсивных вызовов и циклов имеют константную сложность. Определение сложности алгоритма обычно сводится к анализу рекурсивных вызовов и циклов.

Например, рассмотрим алгоритм обработки элементов массива:

```
for(i=1; i<N; i++) {  …  }
```

Сложность этого алгоритма ***O(N)***, так как тело цикла выполняется ***N*** раз, и сложность тела цикла равна ***O*(1)**. Если один цикл вложен в другой и оба цикла зависят от размера одной и той же переменной, то вся конструкция характеризуется квадратичной сложностью:

```
for(i=1; i<N; i++)
for(j=1; j<N; j++) {    …   }
```

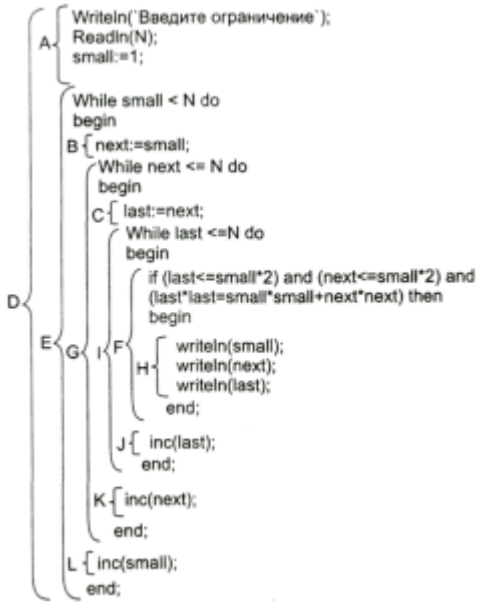
Сложность этой программы ***O(N ^{2})***.

Анализ функции сложности по программе

Существуют два способа анализа сложности алгоритма: восходящий (от внутренних управляющих структур к внешним) и нисходящий (от внешних структур к внутренним).

Как правило, около 90% времени работы программы требует выполнение повторений и только 10% составляют непосредственно вычисления. Анализ сложности программ показывает, на какие фрагменты выпадают эти 90% — это циклы наибольшей глубины вложенности. Повторения могут быть организованы в виде вложенных циклов или вложенной рекурсии. Эта информация может использоваться программистом для построения более эффективной программы следующим образом. Прежде всего можно попытаться сократить глубину вложенности повторений. Затем следует рассмотреть возможность сокращения количества операторов в циклах с наибольшей глубиной вложенности. Если 90% времени выполнения составляет выполнение внутренних циклов, то тридцатипроцентное сокращение этих небольших секций приводит к 27-процентному снижению времени выполнения всей программы.

Оценим сложность программы «Тройки Пифагора»:



O(H) = O(1) + O(1) + O(1) = O(1);

O(I) = O(N) x *(O(F) + O(J)) = O(N)* x *O*(доминанты условия) = *O(N);*

O(G) = O(N) x *(O(C) + O(I) + O(K)) = O(N)* x *(O(1) + O(N) + O(1)) = O(N²);*

O(E) = O(N) x *(O(B) + O(G) + O(L)) = O(N)* x *O(N²) = O(N³);*

O(D) = O(A) + O(E) = O(1) + O(N²) = O(N³).

Сложность этой программы ***O(N ³)***.

Теоретическая и практическая функции сложности

Для оценки эффективности алгоритмов используется функция сложности алгоритма, которая обозначается заглавной буквой «***O***», в круглых скобках записывается аргумент. Например, функция сложности ***O(n²)*** читается как функция сложности порядка ***n²***. Функция сложности алгоритма — это функция, которая определяет количество сравнений, перестановок, а также временные и ресурсные затраты на реализацию алгоритма (рис.).

Ряд значений функции сложности

Функция ***f(п)*** в ряде случаев может иметь достаточно сложную аналитическую форму. Поскольку для временной теоретической сложности большее значение имеет не столько вид функции, сколько порядок ее роста, то во многих математических дисциплинах, в том числе и в теории алгоритмов, функцию ***f(n)***определяют как ***O(g(п))*** и говорят, что она порядка ***g(п)*** для больших ***n***, если

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \text{const} \neq 0,$$

где ***f(n)*** и ***g(n)*** — экспериментальная и теоретическая функции сложности. При этом если ***f(n)*** = ***O(g(n))***, то предел отношения равен константе и тогда функция ***f(n)*** имеет порядок ***O(g(n))***.

Пример Определить функцию сложности алгоритма по результатам эксперимента:

N	Количество сравнений
6	54

Решение. Вначале найдем экспериментальную функцию сложности *Оэ.* Экспериментальная функция сложности алгоритма принимает следующий ряд значений:

*an, an*log₂*n, an², an³ , aeⁿ, an!*

Для правильно подобранной экспериментальной функции *a* = 1, однако на практике допускается 1 <= *a* <= 2;

а) допустим, $O_э$ = *an*, тогда *an* = 54, 6*a* = 54, a = 54/6 (не удовлетворяет условию 1 <= *a* <= 2).

При *a* = 1 значение экспериментальной функции совпадает со значением теоретической функции сложности;

б) допустим, $O_э$ = *an*log₂*n*, тогда *an*log₂*n* = 54, *a*61og₂6 = 54,

a = 54/6$\log_2$6 =9/$\log_2$6 (a > 2, не удовлетворяет условию);

в) допустим, $O_э = an^2$, тогда $an^2 = 54$, *a* x 36 = 54, *a* = 54/36 = 1,5 (a < 2 — удовлетворяет условию).

Таким образом, экспериментальная функция сложности имеет вид $O_э(1{,}5n^2)$.

Найдем теоретическую функцию сложности:

$$\lim_{n\to\infty}\frac{O_э(n)}{O_т(n)} = \text{const} \neq 0, \qquad \lim_{n\to\infty}\frac{1{,}5n^2}{X} = \text{const} \neq 0,$$

$$\lim_{n\to\infty}\frac{1{,}5n^2}{n^2} = \text{const} \neq 0.$$

Отсюда теоретическая функция сложности — ***O(n²)***.

Анализ алгоритмов всегда тщательно прорабатывается в программировании. Эффективных Вам алгоритмов!