Выводы (к практикам 1-4)

Любая программа должна эффективно использовать ресурсы компьютера и выполняться как можно быстрее.

Однако, измерение *времени выполнения программ* – вещь не однозначная, так как на время выполнения влияют, например, и компьютер, на котором вы работаете, и компилятор, который вы используете, а они могут по разному влиять на время исполнения вашей программы. Поэтому вычислять время выполнения программы в единицах времени (сек) не имеет смысла.

Поэтому под *трудоемкостью алгоритма* для данной конкретной проблемы, заданной множеством D, понимают количество элементарных операций, задаваемых алгоритмом в принятой модели вычислений. В качестве модели вычислений рассматривают абстрактную машину, имеющую процессор с фон-Неймановской архитектурой, адресную память и набор элементарных операций, соответствующий языкам высокого уровня. Такая модель вычислений называется машиной с произвольным доступом к памяти (RAM).

Функция трудоемкости f_A (D) – это отношение, которое связывает входные данные с количеством элементарных операций. Значением этой функции является целое положительное число.

Опыт показывает, что количество элементарных операций, задаваемых алгоритмом, то есть значение f_A (D) на входе длины n, где n=|D| , не всегда совпадает с количеством операций на другом входе такой же длины. (потому что существует много множеств D длины n – то есть может быть много различных входных наборов данных)

Поэтому при анализе алгоритмов различают худший случай, лучший случай, и средний случай.

Худший случай – это наибольшее количество операций, совершаемых алгоритмом А для решения конкретных проблем размерности n.
Лучший случай – это наименьшее количество операций, совершаемых алгоритмом А для решения конкретных проблем размерности n.
Средний случай – среднее количество операций, совершаемых алгоритмом А для решения конкретных проблем размерности n.

Для получения значений функции трудоемкости f_A (D) определили элементарные операции, используемые в языках высокого уровня, которые необходимо учитывать при расчетах.
Такими операциями являются :

Операия присваивание: a = b;

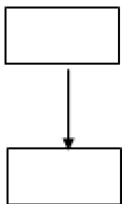
Одномерная индексация a[i] : (адрес (а)+i*длинна элемента);

Арифметические операции: (* , / , - , +);

Операции сравнения: a < b;

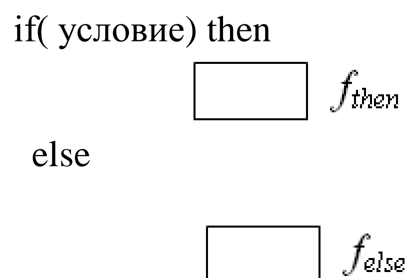
Логические операции {or, and, not} ;

Конструкция «Следование» Трудоемкость конструкции есть сумма трудоемкостей блоков, следующих друг за другом, где k – количество блоков.



$$F_{«следование»} = f_1 + \ldots + f_k$$

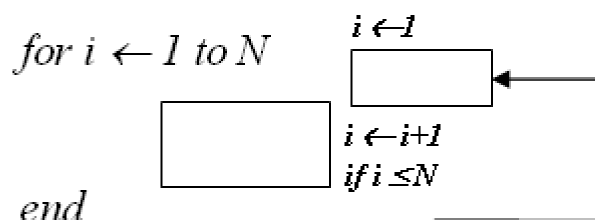
Конструкция «Ветвление»



Общая трудоемкость конструкции «Ветвление» требует анализа вероятности выполнения переходов на блоки «Then» и «Else» и определяется как:

$$F_{«ветвление»} = f_{then} * p + f_{else} * (1-p).$$

Конструкция “Цикл”



Общая трудоемкость определяется как

$$F_{«цикл»} = 1+3*N+N*f_{«тела\ цикла»}$$

Пример 1 Задача суммирования элементов квадратной матрицы

```
SumM (A, n; Sum)
Sum =0                    //  1 операция
For i = 1 to n            //  3 операции на один проход цикла
For j = 1 to n            //  3 операции на один проход цикла
Sum =Sum + A[i,j]         //  4 операции
end for
Return (Sum)
End
```

Алгоритм выполняет одинаковое количество операций при фиксированном значении n, и, следовательно, является количественно-зависимым. Его функция трудоемкости зависит только от размерности конкретного входа. Применение методики анализа конструкции «Цикл » дает:

$f_A(n)=1+1+ n *(3+1+ n *(3+4))=7*n^2+4* n +2 =\theta\ (n^2)$,

где внутренний цикл : $f_1(n) = 1+3*n+4*n$

внешний цикл : $f_2(n)= 1+3*n+n*f_1(n)$

замечание по задаче:

под n понимается линейная размерность матрицы, в то время как на вход алгоритма подается n^2 значений.

Общее замечание:

Количественно-зависимые по трудоемкости алгоритмы - это алгоритмы, функция трудоемкости которых зависит только от размерности конкретного входа, и не зависит от конкретных значений:

Fa(n), n=f(N)

К ним можно отнести алгоритмы для стандартных операций с массивами и матрицами – умножение матриц, умножение матрицы на вектор и т .д.

Пример 2 Задача поиска максимума в массиве

```
MaxS (S,n; Max)
Max = S[1]              //  2 операции
For i = 2 to n          //  3 операции на один проход цикла
if Max < S[i]           //  2 операции
then Max =S[i]          //  2 операции
end for
return Max
```

End

Данный алгоритм является количественно-параметрическим.

Трудоемкость таких алгоритмов определяется конкретными значениями обрабатываемых слов памяти:

$$Fa(n),\ n=f(p1\ldots,pi)$$

У таких алгоритмов на входе два числовых значения – аргумент функции и точность.

К ним можно отнести алгоритмы вычисления стандартных функций с заданной точностью . Они работают с помощью вычисления соответствующих степенных рядов.

В данном конкретном случае трудоемкость зависит от размера входа и от порядка расположения однородных элементов. Зависимость от значений не может быть полностью исключена, но она не является существенной, поэтому для фиксированной размерности исходных данных необходимо проводить анализ для худшего, лучшего и среднего случая. Примерами такого класса могут служить алгоритмы сортировки сравнениями, алгоритмы поиска минимума и максимума в массиве.

А). Худший случай

Максимальное количество переприсваиваний максимума (на каждом проходе цикла) будет в том случае, если элементы массива отсортированы по возрастанию. Трудоемкость алгоритма в этом случае равна:

$$f_A(n)=1+1+1+(n-1)(3+2+2)=7n-4=\theta(n).$$

Б) Лучший случай

Минимальное количество переприсваивания максимума (ни одного на каждом проходе цикла) будет в том случае, если максимальный элемент расположен на первом месте в массиве. Трудоемкость алгоритма в этом случае равна:

$$f_A(n)=1+1+1+(n-1)(3+2)=5n-2=\theta(n).$$

В) Средний случай

Алгоритм поиска максимума последовательно перебирает элементы массива, сравнивая текущий элемент массива с текущим значением максимума. На очередном шаге, когда просматривается к-ый элемент массива, переприсваивание максимума произойдет, если в подмассиве из первых к элементов максимальным элементом является последний. Очевидно, что в случае равномерного распределения исходных данных, вероятность того, что максимальный из к элементов расположен в определенной (последней) позиции

равна 1/к. Тогда в массиве из n элементов общее количество операций переприсваивания максимума определяется как:

$$\sum_{i=1}^{N} \frac{1}{i} = Hn \approx \ln(N) + \gamma, \gamma \approx 0{,}57$$

Величина Hn называется n-ым гармоническим числом. Таким образом, точное значение (математическое ожидание) среднего количества операций присваивания в алгоритме поиска максимума в массиве из n элементов определяется величиной Hn (на бесконечности количества испытаний), тогда:

$$f_A(n)=1 + (n-1)\ (3+2) + 2\ (\ln(n) +)=5\ n +2 \ln(n) - 4 +2\ = \theta\ (n).$$

Анализ сложности рекурсивных алгоритмов был разобран на нашей практике для задачи вычисления факториала для значения n=5. (n – параметр алгоритма, а не количество слов на входе)

В асимптотическом анализе используются обозначения, позволяющие показать скорость роста функции : (классификация скоростей роста сложности алгоритмов)

1. Оценка θ (тетта)
2. Оценка О (О большое)
3. Оценка Ω (Омега)

Эти классы описаны в практике 1-4.

Оценка θ

1. $f(n)=4\ n^2 +n \ln(n)+174 \qquad f(n)= \theta\ (n^2)$

2. $f(n)= \theta(1)$- запись означает, что f(n) или равна константе, не равной 0, или f(n) ограничена константой на бесконечности:

$$f(n)=7+1/n = \theta(1).$$

Пример

$$c_1 n^2 \le \frac{1}{2} n^2 - 3n \le c_2 n^2$$

$$c_1 \le \frac{1}{2} - \frac{3}{n} \le c_2$$

Неравенства выполняются, если выбрать $n \ge 7 c \le \frac{1}{14}$ и $n \to \propto c_2 \ge \frac{1}{2}$

Оценка O

$f(n)=1/n$; $f(n)=12$; $f(n)=3n+17$; $f(n)=n \ln(n)$; $f(n)=6n^2 +24n +77$
для всех этих функций справедлива оценка $O(n^2)$

Оценка Ω

Теорема Для любых двух функций f(n) и g(n) соотношение $f(n) =\theta(g(n))$ выполняется тогда и только тогда, когда $f(n)=O(g(n))$ и $f(n)= \Omega(g(n))$.

Например, $\Omega(n \ln(n))$ обозначает класс функций, которые растут не медленнее, чем $g(n)=n \ln(n))$. В этот класс попадают все полиномы со степенью n , больше двух и все степенные функции с основанием, большим единицы.

Таким образом, в асимптотическом анализе алгоритмов разработаны специальные методы получения асимптотических оценок.

Классификация алгоритмов по трудоёмкости.

O (1)
Большинство инструкций программы запускается один или несколько раз, независимо от n. Т.е. время выполнения программы постоянно
O (n)
Время выполнения программы линейно зависит от n. Каждый входной элемент подвергается небольшой обработке.
(поиск min/max в массиве, значения в связанном списке) (for…; i<n ;...)

$O (n^2)$
Время выполнения программы является квадратичным. Такие алгоритмы можно использовать для относительно небольших n. Такое время характерно, когда обрабатываются все пары элементов (в цикле двойного уровня вложенности).
(сортировки выбором, вставками)

$O (n^3)$
Алгоритм обрабатывает тройки элементов (возможно, в цикле тройного уровня вложенности) и имеет кубическое время выполнения. Такой алгоритм практически применим только для малых задач. (умножение матриц)

O (log n)
Программа работает медленнее с ростом n. Такое время характерно для программ, которые сводят большую задачу к набору меньших подзадач, уменьшая на каждом шаге размер подзадачи на некоторый постоянный коэффициент. Общее решение находится в одной из подзадач. Логарифмическая зависимость. (бинарный поиск)

O (n*log n)
Время выполнения программы пропорционально n*log n возникает тогда, когда алгоритм решает задачу, разбивая ее на меньшие подзадачи, решая их независимо и затем объединяя решения подзадач (комбинация).
(сортировки быстрая, слиянием)

O (2^n)
Лишь несколько алгоритмов с экспоненциальным временем имеет практическое применение, хотя такие алгоритмы возникают естественным образом при попытке прямого решения задач. (перебор и сравнение различных решений) Для комбинаторных задач нереализуемы. Сведение к приближенному алгоритму с приближенным значением.

Обычно время выполнения в результате анализа выглядит как многочлен, где главным членом является один из вышеприведенных. Коэффициент при главном члене зависит от количества инструкций во внутреннем цикле. Поэтому так важно максимально упрощать количество инструкций во внутреннем цикле. При росте N начинает доминировать главный член, поэтому остальными слагаемыми в многочлене можно пренебречь.