МОСКОВСКИЙ ИНСТИТУТ ЭЛЕКТРОННОЙ ТЕХНИКИ
Институт системной и программной инженерии
и информационных технологий (Институт СПИНТех)

Практикум по курсу "Эффективные методологии разработки программного обеспечения "

Работа 2. ВРЕМЕННАЯ СЛОЖНОСТЬ АЛГОРИТМА

Алгоритм — это точное предписание, определяющее вычислительный процесс, идущий от варьируемых исходных данных к искомому результату.

Алгоритм – инструмент для решения *корректно поставленной* вычислительной задачи. В постановке задачи задаются отношения между *входом и выходом* и описывается конкретная вычислительная процедура.

Алгоритм корректен, если для любых входных данных результатом его работы являются корректные выходные данные. Если алгоритм *некорректный*, то для некоторых входных данных он может вообще не завершить свою работу или выдать неправильный ответ.

Например, рассмотрим задачу сортировки последовательности чисел в неубывающем порядке. Эта задача часто решается на практике и достаточно проста для ознакомления на ее примере со многими стандартными методами разработки и анализа алгоритмов. Задача сортировки формально определяется следующим образом.

Вход: последовательность из *n* чисел $(a_1, a_2, \dots, a_n)$.

Выход: переупорядоченная входная последовательность $(a'_1, a'_2, \dots, a'_n)$ *такая, что* $a'_1 <= a'_2 <= \dots <= a'_n$.

Если на вход подается последовательность (31, 41, 59, 26, 41, 58), то вывод алгоритма сортировки должен быть таким: (26, 31, 41, 41, 58, 59). Подобная входная последовательность называется *экземпляром задачи сортировки*. *Экземпляр задачи* состоит из входных данных (удовлетворяющих всем ограничениям, наложенным при постановке задачи), необходимых для решения задачи.

Для доказательства правильности результата, полученного циклическим алгоритмом в теории верификации программ используются инварианты цикла, котрые позволяют понять, корректно ли работает алгоритм.

Инвариант цикла — в программировании — *логическое выражение* (обычно — равенство), истинное после каждого прохода тела цикла (после выполнения фиксированного оператора) и перед началом выполнения цикла, зависящее от переменных, изменяющихся в теле цикла.

1.Инварианты используются в теории верификации программ для доказательства правильности результата, полученного циклическим алгоритмом.

2.Инварианты цикла позволяют понять, корректно ли работает алгоритм.

Инвариант строится таким образом, чтобы быть истинным непосредственно перед началом выполнения цикла (перед входом в первую итерацию) и после каждой его итерации.

Причём если в инвариант входят переменные, определённые только внутри цикла, то для входа в цикл они учитываются с теми значениями, которые получат в момент инициализации.

Доказательство корректности цикла с применением инварианта

Порядок доказательства работоспособности цикла с помощью инварианта сводится к следующему:

*Инициализация.*Доказывается, что выражение инварианта истинно перед началом цикла.

Сохранение. Доказывается, что выражение инварианта сохраняет свою истинность после выполнения тела цикла; таким образом, *по индукции*, доказывается, что по завершении всего цикла инвариант будет выполняться.

Завершение. Доказывается, что при истинности инварианта после завершения цикла переменные примут именно те значения, которые требуется получить (это элементарно определяется из выражения инварианта и известных конечных значениях переменных, на которых основывается условие завершения цикла).

Доказывается (возможно — без применения инварианта), что цикл завершится, то есть условие завершения рано или поздно будет выполнено.

Истинность утверждений, доказанных на предыдущих этапах, однозначно свидетельствует о том, что цикл выполнится за конечное время и даст желаемый результат.

Инварианты используют при проектировании и оптимизации циклических алгоритмов. Например, чтобы убедиться, что оптимизированный цикл остался корректным, достаточно доказать, что инвариант цикла не нарушен и условие завершения цикла достижимо.

Если выполняются первые два свойства, инварианты цикла остаются истинными перед каждой очередной итерацией цикла. (Для доказательства того, что инвариант остается истинным перед каждой итерацией, можно использовать любые другие установленные факты, помимо самого инварианта.)

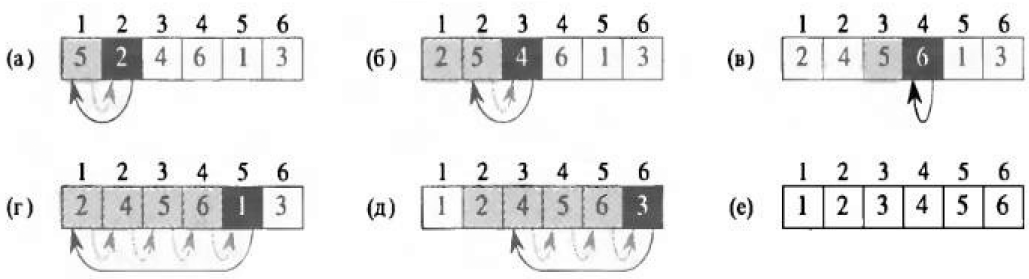
Обратите внимание на сходство с математической индукцией, когда для доказательства определенного свойства для всех элементов упорядоченной последовательности нужно доказать его справедливость для начального элемента этой последовательности, а затем обосновать шаг индукции. В данном случае первой части доказательства соответствует обоснование того, что инвариант цикла выполняется перед первой итерацией, а второй части – доказательство того, что инвариант цикла выполняется после очередной итерации (шаг индукции).

Для наших целей третье свойство, пожалуй, самое важное, так как нам нужно с помощью инварианта цикла продемонстрировать корректность алгоритма. Обычно

инвариант цикла используется вместе с условием, заставляющим цикл завершиться. Свойство завершения отличает рассматриваемый нами метод от обычной математической индукции, в которой шаг индукции используется в бесконечных последовательностях. В данном случае по окончании цикла "индукция" останавливается.

Псевдокод сортировки вставкой представлен ниже в виде процедуры под названием INSERTION-SORT, которая получает в качестве параметра массив A[l .. *n*], содержащий последовательность длиной *n,* которая должна быть отсортирована.

INSERTION-SORT(A)	*Стоимость*	*Повторы*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Вставка $A[j]$ в отсортированную последовательность $A[1..j-1]$.	0	$n-1$
4 $i = j - 1$	c_4	$n-1$
5 **while** $i > 0$ и $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n}(t_j - 1)$
7 $i = i - 1$	c_7	$\sum_{j=2}^{n}(t_j - 1)$
8 $A[i+1] = key$	c_8	$n-1$



Анализ алгоритма заключается в том, чтобы предсказать требуемые для его выполнения ресурсы. Чаще всего определяется время вычисления. Путем анализа нескольких алгоритмов, предназначенных для решения одной и той же задачи, можно выбрать наиболее эффективный из них. В процессе такого анализа может также оказаться, что несколько алгоритмов примерно равноценны, а многие алгоритмы в процессе анализа часто приходится отбрасывать.

Время работы алгоритма для тех или иных входных данных измеряется в количестве элементарных операций, или "шагов', которые необходимо выполнить.

Введем понятие шага. будем исходить из точки зрения, согласно которой для выполнения каждой строки псевдокода требуется фиксированное время.

Время выполнения различных строк может различаться, но мы предположим, что одна и та же *i*-я строка выполняется за время c_i, где c_i, - константа.

Время работы алгоритма - это сумма промежутков времени, необходимых для выполнения каждой входящей в его состав выполняемой инструкции. Если выполнение инструкции длится в течение времени c_i, и она повторяется в алгоритме n раз, то ее вклад в полное время работы алгоритма равен $c_i n$. Чтобы вычислить время работы алгоритма INSERTION-SORT (обозначим его через $T(n)$), нужно просуммировать произведения значений, стоящих в столбцах стоимость и повторы.

В наихудшем случае время работы INSERTION-SORT составляет:

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right) n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right) n \\ &\quad - (c_2 + c_4 + c_5 + c_8) \,. \end{aligned}$$

Это время работы в наихудшем случае можно записать как $an2 + bn + c$, где a, b и c - константы, зависящие от стоимостей c_i; таким образом, мы имеем *квадратичную функцию* от n, сложность которой оценивается как $\Theta(n^2)$, ибо при больших значениях n константы и значения полинома низших порядков существенного значения не имеют.

МЕТОД ДЕКОМПОЗИЦИИ

Многие алгоритмы имеют *рекурсивную* структуру: для решения поставленной задачи они рекурсивно вызывают сами себя один или несколько раз, решая вспомогательные подзадачи, тесно связанные с основной задачей. Такие алгоритмы зачастую разрабатываются с помощью метода *декомпозиции,* или метода *''разделяй и властвуй":* сложная задача разбивается на несколько простых, которые подобны исходной задаче, но имеют меньший объем; далее эти вспомогательные задачи решаются рекурсивным методом, после чего полученные решения комбинируются для получения решения исходной задачи.

Парадигма, лежащая в основе метода декомпозиции "разделяй и властвуй': на каждом уровне рекурсии включает в себя три шага.

Разделение задачи на несколько подзадач, которые представляют собой меньшие экземпляры той же задачи.

Властвование над подзадачами путем их рекурсивного решения. Если размеры подзадач достаточно малы, такие подзадачи могут решаться непосредственно.

Комбинирование решений подзадач в решение исходной задачи.

Существуют три метода решения рекуррентных соотношений:

- *метод подстановки*;

- *метод деревьев рекурсии;*
- *основной метод.*

1. Метод подстановки для решения рекуррентных соотношений состоит из двух шагов:

1. делается предположение о виде решения;
2. с помощью метода математической индукции определяются константы и доказывается, что решение правильное.

Название "метод подстановки" связано с тем, что мы подставляем предполагаемое решение вместо функции при применении гипотезы индукции для меньших значений.

Это хороший метод, но для его применения нужно суметь сделать предположение о виде решения и его можно применять для определения либо верхней, либо нижней границы рекуррентного соотношения.

В качестве примера определим верхнюю границу рекуррентного соотношения

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

Мы предполагаем, что решение имеет вид $T(n) = O(n \lg n)$. Из определения асимптотической оценки в О обозначениях следует: что при подходящем выборе константы $c > 0$ должно выполняться перавеяство $T(n) <= cn \lg n$.

Начнем с того, что предположим справедливость этого перавеяства для всех положительных

$$m < n, \text{ в частности для } m = \lfloor n/2 \rfloor$$

что дает

$$T(\lfloor n/2 \rfloor) \leq c \lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)$$

Подстановка в рекуррентное соотношение приводит к

$$
\begin{aligned}
T(n) &\le 2(c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n \\
&\le cn \lg(n/2) + n \\
&= cn \lg n - cn \lg 2 + n \\
&= cn \lg n - cn + n \\
&\le cn \lg n \ ,
\end{aligned}
$$

и выполняется при *c*>=1.

Теперь, согласно методу математической индукции, необходимо доказать, что наше решение справедливо для граничных условий.

Обычно для этого достаточно показать, что граничные условия являются подходящей базой для доказательства по индукции.

Однако брать здесь базу *n*=1 нельзя (почему?). Поэтому *делается различие между базой рекуррентного соотношения (n = 1) и базой индукции (n = 2 и n = 3).*

Из рекуррентного соотношения следует, что *T*(2) = 4, а *T(3)* = 5. Теперь доказательство по методу математической индукции соотношения *T(n)* <= *cn* lg *n* для пекоторой константы *c*>=1 можно завершить, выбрав ее достаточно большой для того, чтобы были справедливы перавеяства T(2) <= c2lg 2 и *T(3)* <= c3 lg 3. Для этого достаточно выбрать *c*<=2. Итак, гипотеза *T(n)* <= *cn* lg *n верна при* n_0*=2 и c* <= 2.

Метод деревьев рекурсии

Метод подстановок способен обеспечить краткий путь к доказательству того факта, что предполагаемое решение рекуррентного соотношения является правильным, однако сделать хорошую догадку зачастую довольно трудно.

Построение дерева рекурсии, подобного тому, с которым мы имели дело в разделе 2.3.2 в ходе анализа рекуррентного соотношения, описывающего время сортировки слиянием, - прямой путь к хорошей догадке.

В *дереве рекурсии* каждый узел представляет стоимость выполнения отдельно взятой подзадачи, которая решается при одном из многочисленных рекурсивных вызовов функций. Далее стоимости отдельных этапов суммируются в пределах каждого уровня, а затем - по всем уровням дерева, в результате чего получаем полную стоимость всех уровней рекурсии.

Деревья рекурсии лучше всего подходят для того, чтобы помочь сделать догадку о виде решения, которая затем проверяется методом подстановок. При этом в догадке часто допускается наличие небольших неточностей, поскольку впоследствии она все равно проверяется.

Если же построение дерева рекурсии и суммирование времени работы по всем его составляющим производится достаточно тщательно, то само дерево рекурсии может стать средством доказательства корректности решения

Например, посмотрим, как с помощью дерева рекурсии можно догадаться о виде решения рекуррентного соотношения *T (n)* = ЗТ(l*n*/4J) +8(*n*2).

Начнем с поиска верхней границы решения.

При решении рекуррентных соотношений тот факт, что от аргументов функций берется целая часть, при решении рекуррентных соотношений обычно является несущественным (это пример отклонений, которые можно допустить), поэтому построим дерево рекурсии для рекуррентного соотношения *T(n)* = ЗТ(п/4) + cn2 , записанного с использованием константы с > О.

2. Метод деревьев рекурсии

В *дереве рекурсии* каждый узел представляет стоимость выполнения отдельно взятой подзадачи, которая решается при одном из многочисленных рекурсивных вызовов функций. Далее стоимости отдельных этапов суммируются в пределах каждого уровня, а затем - по всем уровням дерева, в результате чего получаем полную стоимость всех уровней рекурсии.

Деревья рекурсии лучше всего подходят для того, чтобы помочь сделать догадку о виде решения, которая затем проверяется методом подстановок. При этом в догадке часто допускается наличие небольших неточностей, поскольку впоследствии она все равно проверяется.

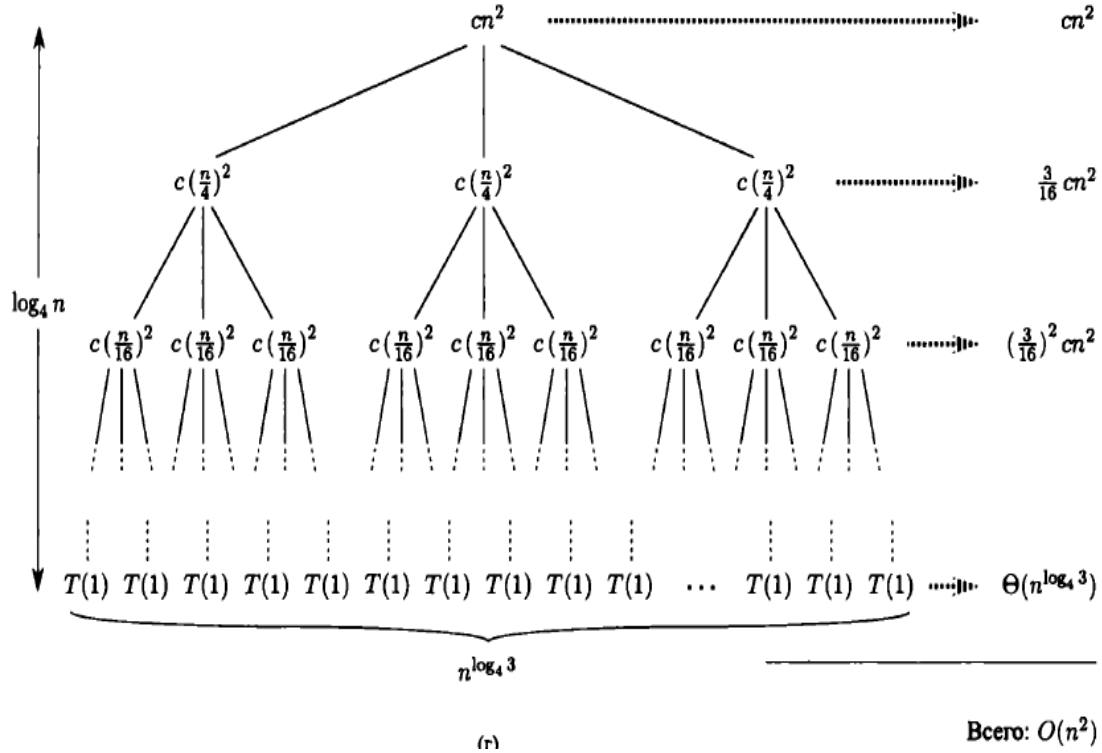
Если же построение дерева рекурсии и суммирование времени работы по всем его составляющим производится достаточно тщательно, то само дерево рекурсии может стать средством доказательства корректности решения.

Например, посмотрим, как с помощью дерева рекурсии можно догадаться о виде решения рекуррентного соотношения

$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2)$$

$$T(n) = 3T(n/4) + cn^2, \quad c > 0.$$

Дерево рекурсии будет иметь вид:



Просуммируем стоимости всех уровней, чтобы найти стоимость всего дерева:

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&< \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{1}{1-(3/16)}\, cn^2 + \Theta(n^{\log_4 3}) \\
&= \frac{16}{13}\, cn^2 + \Theta(n^{\log_4 3}) \\
&= O(n^2)\ .
\end{aligned}
$$

Таким образом, для исходного рекуррентного соотношения получен предполагаемый вид решения $T(n) = O(n^2)$. В рассматриваемом примере коэффициенты при cn^2 образуют убывающую геометрическую прогрес сию, а их сумма ограничена сверху константой 16/13.

Поскольку вклад корня дерева в полное время работы равен cn^2, время работы корня представляет собой некоторую постоянную часть от общего времени работы всего дерева в целом. Другими словами, полное время работы всего дерева в основном определяется временем работы его корня.

3. В *основном методе* граничные оценки рекуррентных соотношений представляются в виде

$$T(n) = aT(n/b) + f(n).$$

где $a >= 1$, $b > 1$, а функция $f(n)$ — это заданная функция.

Рекуррентное соотношение такого вида создает a подзадач, каждая из которых имеет размер, равный n/b от размера исходной задачи, и шаги разделения и комбинирования которого в сумме занимают время $f(n)$.

Для применения этого метода необходимо учитывать три различных случая в соответствии с основной теоремой:

Теорема 4.1 (основная теорема)

Пусть $a \geq 1$ и $b > 1$ — константы, $f(n)$ — функция, а $T(n)$ определена на множестве неотрицательных целых чисел с помощью рекуррентного соотношения

$$T(n) = aT(n/b) + f(n) ,$$

где n/b интерпретируется либо как $\lfloor n/b \rfloor$, либо как $\lceil n/b \rceil$. Тогда $T(n)$ имеет следующие асимптотические границы.

1. Если $f(n) = O(n^{\log_b a - \epsilon})$ для некоторой константы $\epsilon > 0$, то $T(n) = \Theta(n^{\log_b a})$.
2. Если $f(n) = \Theta(n^{\log_b a})$, то $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. Если $f(n) = \Omega(n^{\log_b a + \epsilon})$ для некоторой константы $\epsilon > 0$ и если $af(n/b) \leq cf(n)$ для некоторой константы $c < 1$ и всех достаточно больших n, то $T(n) = \Theta(f(n))$. ■

Использование основного метода

При использовании основного метода мы просто определяем, какой из случаев основной теоремы (если таковой имеет место) применим в данной ситуации, и записываем ответ.

Случай 1

В качестве примера рассмотрим

T(n) = 9T(n/3) + n.

В этом рекуррентном соотношении мы имеем

$$a = 9,\ b = 3,\ f(n) = n,$$

$$n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$$

$$\text{Поскольку } f(n) = O(n^{\log_3 9 - \epsilon}), \text{ где } \epsilon = 1,$$

$$T(n) = \Theta(n^2)$$

Случай 2

$$T(n) = T(2n/3) + 1,$$
$$a = 1,\ b = 3/2,\ f(n) = 1$$
$$n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$$
$$f(n) = \Theta(n^{\log_b a}) = \Theta(1)$$
$$T(n) = \Theta(\lg n)$$

Случай 3

$$T(n) = 3T(n/4) + n \lg n$$

$$a = 3,\ b = 4,\ f(n) = n \lg n$$

$$n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$$

$$f(n) = \Omega(n^{\log_4 3 + \epsilon}), \text{ где } \epsilon \approx 0.2,$$

$$af(n/b) = 3(n/4) \lg(n/4) \leq (3/4) n \lg n = cf(n)$$

$$c = 3/4. \quad T(n) = \Theta(n \lg n)$$

Задание к лабораторной работе

1. *Написать программу (в виде псевдокода или на произвольном языке программирования) для реккурентного алгоритма.*
2. *Доказать инвариант цикла.*
3. *Доказать асимптотическую сложность реализованного алгоритма следующими способами:*
   - *Методом анализа алгоритма*
   - *методом подстановки;*
   - *методом деревьев рекурсии;*
   - *основным методом.*

Варианты алгоритмов:

1. Метод пузырьковой сортировки.
2. Правило Горнера для вычисления многочлена

$$P(x) = \sum_{k=0}^{n} a_k x^k$$
$$= a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + xa_n)\cdots))$$

3. Умножение двух матриц