УВАЖАЕМЫЕ СТУДЕНТЫ!
ВАМ НЕОБХОДИМО ВЫПОЛНИТЬ СЛЕДУЮЩЕЕ:

1. Ознакомиться с теорией и законспектировать лекцию не меньше трех листов, составить и ответить на вопросы.

2. Предоставит отчет конспекта лекции прислать в виде скриншото в течении трех дней.

3. Отправить преподавателю на почту **v.vika2014@mail.ru** и указать свою Ф.И.О, группу, и название дисциплины тел 072-17-44-9-22

Тема: Сложность алгоритмов

Характеристики алгоритма, которые влияют на его применимость, принято называть характеристиками сложности алгоритма. Среди характеристик сложности наиболее важными являются две, характеризующие ресурсы исполнителя: время и память. Необходимо знать, как долго будет выполняться алгоритм и хватит ли ресурса памяти для этого. Время зависит от того, кто является исполнителем (человек, вычислительное устройство, компьютер), и от того, насколько быстро он делает операции (разные компьютеры обладают разной производительностью). Так как нужна объективная характеристика алгоритма, не зависящая от исполнителя, то вместо времени исполнения алгоритма будем рассматривать число шагов t выполнения алгоритма. Если $\bar{\tau}$ – среднее время одного шага исполнителя, то фактическое время работы алгоритма для этого исполнителя.

$$T_{\emptyset} = \bar{\tau} * t$$

Таким образом, t есть характеристика алгоритма, не зависящая от особенностей исполнителя, и потому математическая характеристика сложности алгоритма. Память S, используемая алгоритмом, также зависит от особенностей исполнителя. Если на каждом шаге алгоритма используется не более μ единиц памяти, то $S \leq \mu \cdot t$. Эта оценка очень грубая, так как t может значительно превосходить S. В большинстве случаев в качестве

характеристики сложности алгоритма применяется характеристика t – число шагов выполнения алгоритма.

Трудоемкость алгоритмов.

Итак, t зависит от исходных данных задачи. Зависимость эту не всегда возможно анализировать непосредственно. Вследствие этого целесообразно будет определить временные рамки выполнения алгоритма (максимальное и минимальное время), сколько в среднем будет выполняться алгоритм (среднее время). Но для любых вариантов задачи время (число шагов) ничем не ограничено. Так, при сортировке массива время, как правило, зависит от длины массива и растет с ростом числа элементов массива. Принято входные данные Vалгоритма характеризовать одним параметром или несколькими параметрами. Одной из таких характеристик является объем входных данных – число элементов входных данных. Эта характеристика объективно характеризует входные данные так же, как и число шагов объективно характеризует исполнение алгоритма. В свою очередь, устанавливают зависимость объема входных данных от одного или нескольких параметров, характеризующих задачу. Так, в задаче сортировки массива таким параметром является длина n массива.

Так как число шагов алгоритма зависит не только от выбранных в задаче параметров $P = (n, m, \dots)$, характеризующих объем входных данных, но и от других характеристик входных данных $X = (X_{1,}X_{2,\dots})$, то можно ввести оценку по всем этим характеристикам. Оценка наибольшего числа шагов, необходимых для выполнения алгоритма, в зависимости от параметров P:

$$t = (P, X) \leq \max_{X} T\,(P, X) \equiv T(P)$$

называется максимальной трудоемкостью алгоритма или просто трудоемкостью алгоритма. Максимальная трудоемкость дает возможность оценить максимальное время, необходимое для исполнения алгоритма. Эта оценка может быть очень завышенной в некоторых случаях. Поэтому важно иметь оценку наименьшего числа шагов, которую называют минимальной трудоемкостью:

$$t = (p, x) \geq \min_{x} t\,(P, X) \equiv Tmin(P)$$

и оценку среднего числа шагов, которую называют средней трудоемкостью:

$$t = t(P, X) \approx \frac{1}{K} \sum_{X} t(P, X) \equiv Tcp(P)$$

где k – число вариантов других характеристик входных данных.

Трудоемкость алгоритма позволяет оценить время выполнения алгоритма при решении той или иной задачи:

$$T_{\emptyset}\ (X) \leq \bar{\tau} * T(n)$$

При этом коэффициент $\bar{\tau}$ статистически определяется для исполнителя или оценивается некоторой константой. Однако точный вид зависимости *T(n)* от аргумента *n* часто очень трудно установить. Поэтому вместо установления вида функции для трудоемкости оценивается быстрота роста этой функции при помощи некоторой простой функции f(n).

Говорят, что T(n) = O(f(n)), если |T(n)| ≤ C|f(n)| для всех значений n > n_0. Такая оценка роста функции T(n) является односторонней, так как функция f(n) может расти быстрее. Лучше оценивать рост трудоемкости функцией f(n), имеющей тот же порядок роста, т. е. также |T(n)| ≥ C1|f(n)|. В этом случае пишут T(n) = Θ(f(n)) и говорят, что рост *T(n)* оценивается ростом f(n). Наиболее простыми функциями, оценивающими рост трудоемкости, являются полиномы

$$p(n) = n^k + c_1 * n^{k-1} + c_2 * n^{k-2} * t \ldots + c_k$$

В случае T(n) = Θ(p(n)), учитывая, что p(n) = Θ (n k), получаем T(n) = Θ (n k). Говорят, что в этом случае трудоемкость полиномиальная или алгоритм имеет полиномиальную трудоемкость. При k = 1 T(n) = Θ(n) и алгоритмы принято называть алгоритмами с линейной трудоемкостью.

Если есть два алгоритма A1 и A2 решения некоторой задачи и оба имеют полиномиальную трудоемкость, причем *k1* < *k2*, то говорят, что первый алгоритм имеет меньшую трудоемкость. Но меньшая трудоемкость не

означает, что время решения задачи первым алгоритмом будет меньше, чем вторым. Так, пусть

$$90n \le T_1(n) \le 100n\,, 9n^2 \le T_2(n) \le 10n^2$$

Тогда при n <10 оказывается, что время решения задачи для первого алгоритма больше, чем для второго. Однако, из определения ясно, что найдется такое n_0 (в примере n_0 = 10), что время решения задачи при n > n0 будет всегда меньше для первого алгоритма.

Трудоемкость алгоритма может иметь скорость роста меньшую, чем линейная. Например, $T(n) = \theta\left(\sqrt{n}\right)$ или $T(n) = \log_2 n$.

Но и в этом случае принято говорить о полиномиальной трудоемкости. Алгоритмы, трудоемкость которых растет быстрее любого полинома, принято называть алгоритмами экспоненциальной трудоемкости, даже если скорость роста трудоемкости оценивается более медленной функцией, чем экспонента. Например, экспоненциальными являются все алгоритмы со следующими трудоемкостями:

$$T(n) = \theta(2^n), T(n) = \theta\left(2^{\sqrt{n}}\right) T(n) = \theta\left(n^{n^2}\right)$$

Причина, по которой используются только эти два названия трудоемкости (полиномиальная и экспоненциальная), состоит в том, что алгоритмы полиномиальной трудоемкости, как правило, эффективны, если показатель степени у полинома не слишком большой. А алгоритмы экспоненциальной трудоемкости не являются эффективными, так как время вычисления по этим алгоритмам растет очень быстро. В таблице показана скорость нарастания времени работы алгоритмов различной трудоемкости в секундах на компьютере с быстродействием 10^6 оп/сек.

n	10	20	30	40	50
n	0.00001	0.00002	0.00003	0.00004	0.00005
n^2	0.0001	0.0004	0.0009	0.00016	0.00025
n^3	0.001	0.008	0.0027	0.0064	0.125
n^5	0.1	3.2	24.3	1.7 мин	5.3 мин
2^n	0.001	1.0	17.9 мин	12,7 дн	35,7 лет
3^n	0.059	58 мин	6.5 лет	385500 лет	200×10^8лет

При нескольких параметрах входных данных трудоемкость полиномиального алгоритма растет как полином от нескольких аргументов. Например,

$$T(n,m) = \theta(n \times m^2), T(n,m,k) = \theta(n^2 k \times \log_2 m), \sum_{i=1}^{n(A)} n_i\,(B)$$

Оценивание трудоемкости алгоритмов.

Процесс получения оценки трудоемкости называется оцениванием трудоемкости. Для этого следует анализировать алгоритм с точки зрения быстроты роста числа его шагов, при изменении параметров задачи (параметров входных данных). Прежде всего, в алгоритме следует выделить циклы. Если циклов нет, то число шагов линейной структуры алгоритма не зависит от параметров задачи и, следовательно, трудоемкость является константной, т. е. оценивается как Θ (1).

Циклическая структура алгоритма ведет к повторению выполнения его частей, что влияет на общее число шагов выполнения, т. е. на трудоемкость. Следует оценить для каждого цикла, от каких параметров задачи зависит число повторений цикла и как оно растет с ростом этих параметров.

Если цикл B с числом повторений $n(B)$ вложен в цикл A с числом повторений $n(A)$ и циклы независимы (число повторений цикла B не зависит от выполнения цикла A), то общее число повторений цикла B с учетом повторений цикла A составляет $n(A) \cdot n(B)$.

Отсюда правило: *для вложенных независимых циклов их трудоемкости перемножаются* $\Theta(AB) = \Theta(A) \cdot \Theta(B)$.

Если вложенные циклы не являются независимыми, т. е. число повторений внутреннего цикла $n_i(B)$ зависит от номера i повторения при выполнении внешнего цикла, то нужно проанализировать, как зависит общее число повторений внутреннего цикла от параметров задачи.

Если циклы не являются вложенными, то трудоемкость определяется наибольшей из трудоемкостей циклов

$$\Theta(A + B) = \Theta(A) + \Theta(B) = max\{\Theta(A), \Theta(B)\}.$$

При оценке максимальной трудоемкости следует подбирать такие примеры входных данных для тех или иных параметров задачи, на которых реализуется максимальное число шагов алгоритма. При оценке минимальной трудоемкости следует подбирать примеры, на которых реализуется минимальное число шагов алгоритма. Ввиду сложности некоторых алгоритмов такие примеры не всегда удается построить, но в таких случаях для оценки трудоемкости бывает достаточно примеров и близких по числу операций к максимальному или соответственно к минимальному числу.[11]

Рассмотрим примеры оценивания трудоемкости на примере алгоритма сортировки массива методом «пузырька». Блок – схема алгоритма сортировки методом «пузырька» см. рис. 15

Алгоритм содержит вложенные циклы. Внешний цикл, в случае массива входных данных, упорядоченного по убыванию, будет выполняться максимальное число раз: $n - 1$, а в случае входного массива, упорядоченного по возрастанию, будет выполняться только 1 раз. Внутренний цикл во втором случае выполняется $n - 1$ раз, а в первом случае циклы зависимы, но, внутренний цикл в среднем выполняется $n/2$ раз. Поэтому максимальная трудоемкость (входные данные первого случая) оценивается как

$$\Theta(n) \cdot \Theta(n) = \Theta(n^2),$$

а минимальная трудоемкость (входные данные второго случая) – как

$$\Theta(1) \cdot \Theta(n) = \Theta(n).$$

Во втором разделе рассмотрены методы сортировки элементов массива: метод простого выбора, метод «пузырька», сортировка слиянием и вставками. Разобран типовой пример нахождения максимального и минимального элементов в массив и принцип бинарного поиска в упорядоченном массиве. Для закрепления навыков создания алгоритмов сортировки можно рекомендовать задания для самостоятельной работы.