УДК 004.05

О ПРАКТИЧЕСКОЙ НЕОБХОДИМОСТИ ОПРЕДЕЛЕНИЯ ТЕОРЕТИЧЕСКОЙ СЛОЖНОСТИ АЛГОРИТМОВ

Самуйлов Сергей Владимирович

канд. тех. наук
Финансовый университет при Правительстве
Российской Федерации (филиал), Пенза

author@apriori-journal.ru

Аннотация. В статье рассматривается понятие сложности алгоритмов и необходимость ее определения и анализа на начальных этапах решения поставленной задачи. На примерах показана необходимость сравнения временной и пространственной сложности алгоритмов, а также эффективности их реализации.

Ключевые слова: сложность алгоритма; критерии сравнения алгоритмов; полиномиальные и экспоненциальные алгоритмы.

ABOUT PRACTICAL NEED OF DETERMINATION OF THEORETICAL COMPLEXITY OF ALGORITHMS

Samuylov Sergey Vladimirovich

candidate of technical sciences
Financial University under the Government
of the Russian Federation (branch), Penza

Abstract. The article discusses the concept of algorithm complexity and the need of its determination and analysis at the initial stages of the task. The examples show the necessity of comparison of temporal and spatial complexity of algorithms and the effectiveness of their implementation.

Key words: complexity of algorithm; criteria of comparison of algorithms; polynomial and exponential algorithms.

Для решения большинства задач, как правило, существует много различных алгоритмов. При решении конкретной задачи вопрос выбора наиболее эффективного алгоритма является одним из основных [1].

Под сложностью алгоритма понимается, как правило, время его выполнения и количество требуемой памяти.

Обычно указывают на три причины, обосновывающие необходимость анализа сложности алгоритмов [2].

1. Необходимость получения оценок объема памяти и времени работы, которые потребуются алгоритму для успешной обработки конкретных данных.
2. Хотелось бы иметь некий количественный критерий для сравнения двух алгоритмов, претендующих на решения одной и той же задачи. Это позволит среди множества алгоритмов выбирать для реализации наиболее эффективный.
3. Поиск абсолютного критерия. Когда можно считать решение задачи оптимальным, когда алгоритм настолько хорош, что его значительное улучшение невозможно?

Проиллюстрируем причины необходимости предварительного анализа сложности алгоритмов с помощью простых, но наглядных примеров. Две основные характеристики любого компьютера – быстродействие и доступная память. С каждым годом быстродействие компьютера растет, память увеличивается. Однако и в настоящее время, и через много лет эти параметры, судя по всему, все равно останутся конечными. Следовательно, еще долго будут существовать задачи, в принципе невыполнимые с помощью современного компьютера из-за недостаточной памяти и/или низкого быстродействия.

Как известно, все множество алгоритмов можно разбить на две группы: алгоритмы полиномиальной сложности и алгоритмы экспоненциальной сложности. Следует помнить, что в настоящее время только полиномиальные алгоритмы считаются практически полезными, т.е. ре-

ализуемыми с помощью современных компьютеров, причем полином должен быть небольшой степени. Если же для решения поставленной задачи существуют только экспоненциальные алгоритмы, то даже для небольшого количества исходных данных время реализации алгоритма может быть неприемлемо.

В качестве примера алгоритма экспоненциальной сложности рассмотрим алгоритм решения задачи коммивояжера. Это один из известных примеров таких задач, которые очень легко поставить и промоделировать, но очень сложно решить [3].

Постановка задачи. Есть агент по продаже компьютеров (коммивояжер). На его территории n городов. Известна стоимость переезда из одного города в другой для каждой пары городов. Эта стоимость может быть задана матрицей стоимости C размерности $n \times n$, где c_{ij} – стоимость проезда между i-м и j-м городами.

Необходимо найти маршрут минимальной стоимости, начинающийся и оканчивающийся в базовом городе и проходящий по одному разу через все остальные города его территории.

Разработка алгоритма. Перенумеруем n городов целыми числами от 1 до n. Базовому городу приписывается номер n. Обход всех городов назовем **туром**. Заметим, что каждый тур однозначно соответствует единственной перестановке целых чисел $1, 2, ..., n–1$, и каждой перестановке соответствует единственный тур.

Таким образом, для любой заданной перестановки мы можем легко построить тур по городам и вычислить стоимость этого тура.

Следовательно, можно решить поставленную задачу, образуя все перестановки первых $n–1$ целых положительных чисел. Для каждой перестановки необходимо строить соответствующий тур и вычислять его стоимость. Обрабатывая все перестановки, запоминаем тур с наименьшей стоимостью.

Анализ сложности. В задаче требуется найти все перестановки первых *n–1* положительных целых чисел. Количество таких перестановок *(n–1)!*. Следовательно, эта часть алгоритма потребует минимум *O[(n–1)!]* шагов.

Для каждой перестановки нужно найти соответствующий тур и его стоимость за *O(n)* шагов. Поэтому верхняя граница для времени работы всего алгоритма равна *O(n!).*

Расчет времени выполнения. Предположим, что на территории коммивояжера 20 городов, то есть *n = 20*. Пусть каждая перестановка генерируется за один шаг, что является очень большой недооценкой. Пусть быстродействие нашей машины 1 миллиард операций в секунду, т.е. 10^9 оп./сек.

Наш алгоритм содержит 20! ≈ $2 \cdot 10^{18}$ операций, которые могут быть выполнены за $2 \cdot 10^{18} / 10^9 = 2 \cdot 10^9$ секунд. В году 31.536.000 секунд. Тогда наша задача будет решаться чуть более 63 лет.

Пусть быстродействие компьютера выросло с одного миллиарда операций в секунду до одного триллиона. Тогда этот же алгоритм будет работать $2 \cdot 10^{18} / 10^{12} = 2 \cdot 10^6$ секунд. Один день состоит из 86.400 секунд. В этом случае наша задача будет решаться 23 дня. И первое, и второе время выполнения алгоритма, несмотря на небольшое число городов (*n = 20*), не может быть удовлетворительным.

Рассмотренный выше пример демонстрирует необходимость анализа сложности алгоритма еще до его программирования и выполнения. Это избавит пользователя от попытки выполнения на ЭВМ алгоритмов, которые не могут быть решены за приемлемое время.

Вторая из приведенных выше причин, требующая предварительного анализа сложности алгоритмов, свидетельствует о необходимости иметь количественный критерий для сравнения двух или более алгоритмов, претендующих на решение одной и той же задачи [4].

Как уже упоминалось, для решения задачи можно использовать, как правило, несколько различных алгоритмов. Зачастую программисты, не утруждая себя теоретической оценкой сложности решения поставленной задачи, не выполнив анализ и сравнение приемлемых для решения задачи алгоритмов, реализуют либо наиболее очевидный с их точки зрения алгоритм, либо наиболее простой в реализации, либо наиболее известный.

Результатом такого подхода к программированию может быть как неэффективно работающая программа, так и, в худшем случае, напрасно потраченное время и средства на попытку реализации алгоритма, который изначально не должен был быть выбран для реализации.

Проиллюстрируем вышеизложенное на следующем примере. Пусть необходимо решить следующую задачу – определить для любого целого *N (N > 1)* его наибольший делитель, отличный от *N*. Один из наиболее очевидных алгоритмов решения этой задачи основывается на том факте, что для любого числа *N* его наибольший делитель находится в интервале от (*N*-1) до 1. Вследствие этого алгоритм заключается в следующем. Пусть некоторая переменная *K* последовательно принимает значения от (*N*-1) до 1. На каждой итерации цикла проверяем, если *K* делит без остатка переменную *N,* то наибольший делитель найден, алгоритм заканчивается. В случае если *N* простое число, цикл закончится при *K = 1.* Назовем описанный выше алгоритм – алгоритм *A*. Этот алгоритм будет состоять из следующих шагов.

Алгоритм А.

1. K= N – 1
2. Пока (N mod K) ≠ 0 повторять K = K – 1
3. Печать «Наибольший делитель = », K.

В алгоритме операция *mod* возвращает остаток, полученный при выполнении целочисленного деления.

Казалось бы, мы рассмотрели наиболее логичный и простой вариант решения поставленной задачи. Однако существуют и другие подхо-

ды к ее решению. Один из альтернативных вариантов основан на том, что разыскиваемый результат есть *N / R*, где *R* – наименьший делитель *N (R ≥ 1).* Если *N* простое число, то *R = 1*, но если *N* непростое, то его наименьший делитель находится в интервале от 2 до $\sqrt{n}$.

Тогда алгоритм решения задачи может быть следующим. Некоторая переменная *R* последовательно принимает значения от 2 до $\sqrt{n}$. На каждой итерации цикла проверяем, если *R* делит без остатка переменную *N,* то наименьший делитель найден. Вычисляем и выводим наибольший делитель, и алгоритм заканчивается. Назовем описанный выше алгоритм – алгоритм В. Этот алгоритм будет состоять из следующих шагов

Алгоритм В.

1. R = 2
2. Пока $\left(R < \sqrt{N}\right) u \left((N \bmod R) \neq 0\right)$ повторять R = R + 1
3. Если (N mod R) = 0, то Печать «НД=», N / R, иначе Печать «НД=», 1.

Чтобы сравнить время выполнения алгоритма *A* и алгоритма *B*, отметим, что оба они состоят из трех шагов и имеют вид:

1. O_1
2. Пока <условие> повторять O_2
3. O_3

где O_1, O_2 и O_3 – некоторые операторы.

Пусть t_1, t_2 и t_3 время выполнения операторов O_1, O_2 и O_3 соответственно, причем t_2 – время выполнения одной итерации цикла. Тогда время выполнения и одного, и другого алгоритма определяется как

$$t = t_1 + t_3 + m \cdot t_2,$$

где *m* – число повторений цикла. Так как t_1, t_2, t_3 – константы, то выражение можно записать как

$$t = P + Q \cdot m,$$

где *P* и *Q* – константы, различные для алгоритма *A* и алгоритма *B*.

Кроме того, эти алгоритмы отличаются и числом повторений цикла (значением переменной *m*). В алгоритме *A* максимальное число итераций цикла m_A вычисляется по формуле

$$max(m_A) = n - 2.$$

В алгоритме *B* –

$$max(m_B) = \lfloor \sqrt{n} \rfloor - 2 .$$

Тогда максимальная временная сложность алгоритма *A* определяется как

$$t_A^{\max} = P_A + Q_A \cdot n ,$$

где P_A и Q_A – константы, а для алгоритма *B* эта величина равна

$$t_B^{\max} = P_B + Q_B \cdot \sqrt{n} ,$$

где P_B и Q_B – другие константы.

Практическая сложность этих алгоритмов определяется константами P_A, Q_A, P_B и Q_B и зависит от конкретной ЭВМ, в частности, от времени выполнения ее команд.

Гораздо более полезна **теоретическая сложность** этих алгоритмов. Когда *n* велико, этот параметр становится определяющим для сложности как в алгоритме *A*, так и в алгоритме *B*. Из вышеизложенного можно следует, что сложность алгоритма A определяется как $O(N)$, сложность алгоритма *B* – $O(\sqrt{N})$. Отсюда можно сделать вывод, что алгоритм B значительно эффективней алгоритма *A*.

Определение теоретической сложности алгоритмов в некоторых случаях позволяет сделать однозначный выбор одного алгоритма из нескольких альтернативных. Для понимания того, насколько же один алгоритм эффективнее другого при заданных исходных данных, рассмотрим практическую сложность этих алгоритмов. Для сравнения практической сложности алгоритмов *A* и *B* была выполнена их реализация в среде программирования Delphi на языке программирования Pascal.

Для алгоритмов *A* и *B* были сняты характеристики их работы. Для простого числа 914 748 389 алгоритм *A* работал 13120 тиков (≈ 13 секунд), а алгоритм *B* – 1,25 тика.

Для числа 1914748439, наибольший делитель, равный 107347, алгоритм *A* искал 28455 тиков (≈ 28 секунд), а алгоритм *B* – 0,78 тика.

Проанализируем рассмотренный выше пример. Как известно, алгоритмы, претендующие на решения одной и той же задачи, можно сравнить по требуемым вычислительным ресурсам, т.е. их временной и пространственной эффективности, а также по простоте реализации. Временная эффективность программы определяется временем, необходимым для ее выполнения, пространственная – количеством необходимой памяти.

Алгоритмы *A* и *B* имеют одинаковую структуру, т.е. состоят из оператора присваивания, оператора цикла и оператора вывода, следовательно, по простоте реализации эти алгоритмы не отличаются друг от друга. Кроме того, они используют по две вспомогательные переменные. Из этого можно сделать вывод, что пространственная эффективность этих алгоритмов также одинакова. Очень часто, основываясь на этом, автоматически делается вывод об отсутствии различий между анализируемыми алгоритмами. Между тем, как показал рассмотренный выше пример, время выполнения внешне очень похожих алгоритмов может отличаться на несколько порядков.

Рассмотренные в статье примеры демонстрирует необходимость поиска для решения поставленной задачи именно эффективного алгоритма, а не наиболее очевидного или простого в программировании. Причем анализ эффективности обязательно должен учитывать и временную, и пространственную эффективность, и эффективность реализации.

Список использованных источников

1. Самуйлов С.В. Методика сравнительного анализа алгоритмов на примере алгоритмов последовательного поиска // Концепт. 2014. № 9. С. 46-50.
2. 2 Казакова И.А., Самуйлов С.В. Структуры данных. Учебное пособие. Пенза, 2011.
3. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ. М.: «Вильямс», 2005.
4. Самуйлов С.В., Казакова И.А. О критерии естественности поведения в оценке алгоритмов внутренней сортировки // Научное обозрение. 2014. № 10-1. С. 98-104.