СТАТЬИ

УДК 004.051

ПРИМЕНЕНИЕ ПРИБЛИЖЕНИЯ С ПОМОЩЬЮ КРИВЫХ ДЛЯ ОПРЕДЕЛЕНИЯ ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ РЕШЕНИЙ ЗАДАЧ ПО ПРОГРАММИРОВАНИЮ

Буянова И.В., Замулин И.С.

ФГБОУ ВО «Хакасский государственный университет им. Н.Ф. Катанова», Абакан, e-mail: irinazvzd@gmail.com

Существующие электронные образовательные ресурсы, предлагающие механизм автоматической проверки решений задач по программированию, зачастую ограничиваются оценками «решено» или «не решено». Обычно этого достаточно для проведения олимпиад по спортивному программированию, однако наличие более подробного анализа решений, включающего рекомендации по улучшению написанного программного кода, может стать полезным в ходе проведения практикумов по программированию для школьников и студентов. Одним из пунктов отчёта о проверке решения должна быть оценка вычислительной сложности алгоритма на основе нотации «О» большое. В статье рассматривается способ автоматического определения категории сложности алгоритма, реализованного в качестве решения задачи. Предлагаемый способ основывается на трассировке программного кода. Полученная статистика обрабатывается методом приближения с помощью кривых, задаваемых одной из нескольких заранее определённых функций. Каждая из функций должна соответствовать одной из категорий сложности. При помощи метода RMSE прогнозы функций сравниваются с полученными результатами трассировки. Категория сложности алгоритма определяется функцией, дающей наименьшую ошибку аппроксимации. Полученные результаты говорят о практической применимости предложенного подхода и достаточно высокой точности рассмотренного метода.

Ключевые слова: «О» большое, анализ алгоритмов, приближение с помощью кривых, вычислительная сложность, практикум по программированию

APPLICATION OF CURVE-FITTING APPROXIMATION TO DETERMINE THE COMPUTATIONAL COMPLEXITY OF SOLUTIONS TO PROGRAMMING PROBLEMS

Buyanova I.V., Zamulin I.S.

Khakassian State University named after N. F. Katanov, Abakan, e-mail: irinazvzd@gmail.com

Existing electronic educational resources that offer a mechanism for automatically checking solutions to programming problems are often limited to «solved» or «not solved» grades. Usually this is enough to hold competitions in sports programming, however, the presence of a more detailed analysis of decisions, including recommendations for improving the program code, can be useful in the course of programming workshops for schoolchildren and students. One of the points of the solution verification report should be an estimate of the computational complexity of the algorithm based on the big O notation. The article discusses a method for automatically determining the category of complexity of an algorithm implemented as a solution to a problem. The proposed method is based on tracing the program code. The resulting statistics is processed by the curve-fitting approximation method using one of several predefined functions. Each of the functions must correspond to one of the categories of complexity. Using the RMSE method, function predictions are compared with the received trace results. The complexity category of the algorithm is determined by the function that gives the smallest approximation error. The results obtained indicate the practical applicability of the proposed approach and the rather high accuracy of the considered method.

Keywords: Big O notation, analysis of algorithms, curve-fitting, computational complexity, programming practice

К настоящему времени разработано множество электронных ресурсов [1], предназначенных для автоматизации проверки решений задач по программированию. В большинстве случаев они ориентированы на спортивное программирование, но также находят применение при обучении программированию в школах, колледжах и высших учебных заведениях. Зачастую такие ресурсы ограничиваются проверкой корректности решения и его способности выдать результат за заранее определённое время. Этого более чем достаточно для соревновательного программирования, но для облегчения работы преподавателя в ходе проведения практикумов по программированию для школьников и студентов мог бы оказаться полезным более подробный отчёт. Помимо других пунктов, характеризующих решение, такой отчёт должен включать информацию об эффективности алгоритма, предложенного обучающимся.

Чтобы судить об эффективности алгоритма, необходимо оценить его временную и пространственную сложность. Временная сложность определяется количеством элементарных операций, совершаемых алгоритмом для решения поставленной задачи. Пространственная сложность характеризуется объемом затраченной памяти. Для оценки обеих характеристик применятся нотация «О» большое [2, с. 12], ставшая популярной после того, как в 1976 году Дональд Кнут предложил её использование для анализа алгоритмов.

Таблица 1

Наиболее распространённые категории сложности алгоритмов по времени

№	Название	Оценка	Пример
1	Константное время	$O(1)$	Умножение числа на два
2	Линейное время	$O(n)$	Нахождение суммы элементов массива
3	Логарифмическое время	$O(\log_2 n)$	Бинарный поиск
4	Линейно-логарифмическое время	$O(n \cdot \log_2 n)$	Сортировка слиянием
5	Квадратичное время	$O(n^2)$	Сортировка пузырьком
6	Кубическое время	$O(n^3)$	Подбор корней уравнения с четырьмя неизвестными
7	Экспоненциальное время	$O(2^n)$	Нахождения множества всех подмножеств
8	Факториальное время	$O(n!)$	Решение задачи коммивояжёра полным перебором

Нотация «О» большое описывает сложность алгоритма в виде функции от объёма входных данных. Поскольку эта функция должна давать примерную оценку наихудшего возможного времени выполнения алгоритма, для неё берут только самый высокий порядок переменной и не берут константные коэффициенты. Например, если требуется выполнить поиск элемента в отсортированном массиве, то наихудшее время выполнения алгоритма бинарного поиска будет оцениваться как $O(\log_2 n)$, где n – количество элементов в массиве.

Существует несколько наиболее распространённых категорий сложности алгоритмов по времени, каждая из которых характеризуется собственной О-нотацией. В учебных задачах чаще всего встречаются категории, приведённые в таблице 1.

В данном исследовании предпринимается попытка предложить способ автоматического определения категории сложности алгоритма, применённого студентом для решения задачи по программированию на языке Python. С этой целью рассматривается механизм трассировки, как источник данных для дальнейшего анализа методом приближения с помощью кривых.

Материалы и методы исследования

Программный код решений задач зачастую содержит блоки, ответственные за чтение входных данных и вывод результатов, что может исказить результаты анализа решения. Так, если от студента требуется реализовать алгоритм бинарного поиска, сложность которого определяется логарифмической функцией, то основные временные затраты придутся на чтение входных данных – процесс, описываемый линейной функцией. Чтобы корректно оценить решение задачи как имеющее временную сложность $O(\log_2 n)$, необходимо оценивать только фрагмент кода, осуществляющего основную обработку данных.

Поиск основного алгоритма в программном коде решения задачи может представлять собой проблему, которая серьёзно упростится, если сформулировать задание особым образом. Например, студенту можно предложить закончить частично написанную программу, в которой уже реализованы загрузка входных данных и вывод результатов.

Одним из возможных способов определения категории сложности алгоритма является его трассировка, в ходе которой в реальном времени собирается статистика о выполняемых им действиях. В отличие от других методов, таких как статический анализ программного кода, трассировка легко осуществима, особенно в интерпретируемых языках. Например, язык программирования Python предоставляет функцию *sys.settrace*, позволяющую задать обработчик, который будет вызываться при переходе к каждой следующей строке программы. Анализ этих вызовов позволяет оценить количество выполненных в программе итераций циклов, а следовательно, и собрать статистику о временной сложности выполняющегося алгоритма.

Результаты исследования и их обсуждение

График на рисунке 1 показывает пример расчёта сложности сортировки массива методом «пузырька» и статистику, полученную путём подсчёта количества итераций во время трассировки. Как видно из графика, прогнозируемая и полученная кривые имеют схожую форму, отличие объясняется тем, что сортировка выполнялась на массиве, заполненном случайными данными,

в то время как оценочная формула $O(n^2)$ даёт прогноз по наихудшему случаю. Для метода «пузырька» наихудшим случаем является массив, отсортированный в порядке, обратном требуемому. При тестировании на таком массиве различие между кривыми было бы менее заметным. Однако, для цели оценки, полученного результата достаточно, поскольку требуется определить категорию алгоритма, а не получить точную кривую.

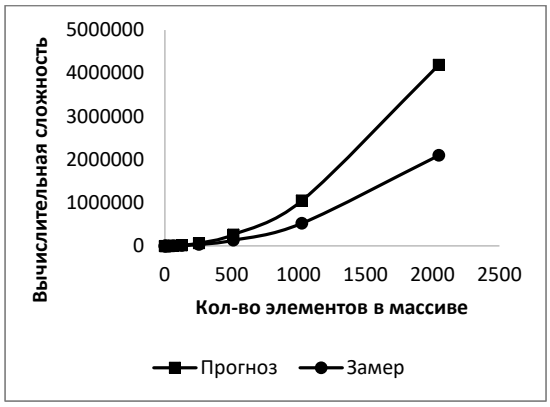

Рис. 1. Прогноз и результаты замера вычислительной сложности алгоритма сортировки массива методом «пузырька»

Аналогично рассмотренному алгоритму были выполнены замеры для примеров алгоритмов №№ 3-8 из таблицы 1. Полученные результаты вместе с использованными исходными кодами разработанного программного обеспечения доступны в виде дополнительных материалов на GitHub (https://github.com/irinaby/bigO).

Следующим этапом после тестирования алгоритмов и сбора статистики о количестве итераций является выбор наиболее подходящей категории для каждого из них. Поскольку категории задаются функциями от количества обрабатываемых элементов, задачу подбора категории можно решить, выбрав кривую, наиболее точно аппроксимирующую полученные результаты измерений. Для этого была выбрана функция *scipy.optimize.curve_fit* из пакета SciPy [3], использующая алгоритм TRF (Trust Region Reflective) [4]. Такой подход даёт оптимальные результаты аппроксимации при малом количестве выполненных измерений, что позволит минимизировать необходимое количество циклов тестирования решения и повысить производительность тестирующей системы образовательного ресурса. Функции, для которых осуществлялся подбор коэффициентов, приведены в таблице 2. После подстановки коэффициентов в функции было выполнено вычисление выдаваемого ими прогноза для каждого из тестов. Данные, полученные для всех рассмотренных алгоритмов, доступны в дополнительных материалах к данной статье.

Таблица 2

Функции, применённые для аппроксимации результатов подсчёта количества итераций

Категория	Использованная функция	Реализация на Python
Константное время	$f(x) = a$	`def constFn(x, a):` `  return [a] * len(x)`
Линейное время	$f(x) = k \cdot x + b$	`def linearFn(x, a, b):` `  return a * x + b`

Логарифмическое время	$f(x) = \begin{cases} 0, b \le 0 \\ a \cdot \log_2 b \cdot x, b > 0 \end{cases}$	`def logarithmicFn(x, a, b, c):`
		`  if (b <= 0): return [0] * len(x)`
		`  return a * log2(b * x) + c`

Линейно-логарифмическое время	$f(x) = \begin{cases} 0, b \le 0 \\ a \cdot x \cdot \log_2 b \cdot x + c, b > 0 \end{cases}$	`def linearithmicFn(x, a, b, c):`
		`  if (b <= 0): return [0] * len(x)`
		`  return a * x * log2(b * x) + c`

Квадратичное время	$f(x) = a \cdot x^2 + b \cdot x + c$	`def quadraticFn(x, a, b, c):` `  return a * x**2 + b * x + c`
Кубическое время	$f(x) = a \cdot x^3 + b \cdot x^2 + c \cdot x + d$	`def cubicFn(x, a, b, c, d):` `  return a * x**3 + b * x ** 2 + c * x + d`
Экспоненциальное время	$f(x) = a \cdot 2^{b \cdot x} + c$	`def exponentialFn(x, a, b, c):` `  return a * power(2, b * x) + c`
Факториальное время	$f(x) = a \cdot \left(\prod_{k=1}^{x} k \right) + b$	`def factorialFn(x, a, b):` `  return list(a * prod(range(1, int(k)+1)) + b for k in x)`

Таблица 3

Результаты трассировки решения задачи коммивояжёра и прогноз функций, дающих наиболее точную аппроксимацию

n – кол-во городов	Кол-во итераций		
	трассировка	$a \cdot \left(\prod_{k=1}^{x} x\right) + b$ a = 1,000001517 b = 3,887447327	$a \cdot 2^{b \cdot x} + c$ a = 0,00036394 b = 3,3214786 c = 657,832586
1	1	4,88	657,83
2	3	5,88	657,86
3	8	9,88	658,19
4	27	27,88	661,46
5	124	123,88	694,17
6	725	723,88	1021,09
7	5046	5043,89	4289,34
8	40327	40323,94	36961,60
9	362888	362884,43	363582,48
10	3628809	3628809,39	3628773,92

Для примера рассмотрим решение задачи коммивояжёра [5, с. 176] методом полного перебора. Имеются результаты десяти тестов, начиная с одного города и далее до десяти, с шагом один. Количество итераций, определённое при трассировке, а также прогнозы двух функций, давших наилучшую аппроксимацию, приведены в таблице 3.

На рисунке 2 приведена визуализация данных из таблицы 3.

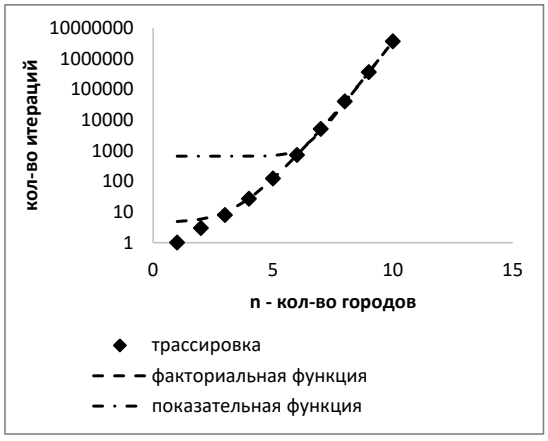

Рис. 2. Визуализация результатов трассировки решения задачи коммивояжёра и прогнозы функций, дающих наиболее точное приближение

Ромбами обозначены результаты замера количества выполненных итераций. Штрихами и штрих-пунктиром показаны графики двух функций, давших наилучшие результаты приближения. Заметно, что кривая, полученная для функции факториала (штрихи), дала наилучший результат, в то время как показательная функция (штрих-пунктир) заметно отклоняется при малых значениях n.

Для определения наиболее подходящих функций был применён метод RMSE (1), в силу квадратичной природы которого даже небольшие ошибки в предсказаниях аппроксимирующих функций оказывают существенное влияние на получаемые оценки. Это особенно важно при работе с такими функциями, как показательная или факториал, поскольку они дают значительный прирост значения при небольшом увеличении аргумента.

$$E = \sqrt{\frac{1}{N}\sum_{i=0}^{N}\left(p_i - t_i\right)^2}, \qquad (1)$$

где E – среднеквадратичная ошибка, N – количество проведённых тестов, p_i – предсказанные значения, t_i – результаты трассировки.

После выполнения серии тестов для алгоритмов №№ 3-8 из таблицы 1 была предпринята попытка подбора коэффициентов для каждой из функций в таблице 3. Результаты вычисления оценки RMSE для всех пар алгоритм-функция приведены в таблице 4. Ячейки пар, давших наилучшие совпадения, выделены тёмным фоном. Выполнить подбор коэффициентов для показательной функции удалось только для алгоритмов, работающих с относительно небольшим количеством входных данных (n <= 32).

Таблица 4

Оценка методом RMSE прогнозов аппроксимирующих функций

Алгоритм / O(n)	Бинарный поиск	Сортировка слиянием	Сортировка пузырьком	Подбор корней уравнения с четырьмя неизвестными	Нахождение множества всех подмножеств	Решение задачи коммивояжёра полным перебором
Константное время	146.90	205697358.8	$1.19 \cdot 10^{12}$	$6.15 \cdot 10^{11}$	$2.78 \cdot 10^{11}$	$3.69 \cdot 10^{12}$
Линейное время	92.16	469462.27	91524960974	84025107392	$1.71 \cdot 10^{11}$	$2.48 \cdot 10^{12}$
Логарифмическое время	6.58	90342184.8	$7.32 \cdot 10^{11}$	$2.352 \cdot 10^{11}$	$2.25 \cdot 10^{11}$	$3.04 \cdot 10^{12}$
Линейно-логарифмическое время	470.02	57.20	6499875441	8518378383	$1.03 \cdot 10^{11}$	$1.59 \cdot 10^{12}$
Квадратичное время	289.90	35251.08	$1.30 \cdot 10^{-21}$	1736469909	77628512124	$1.28 \cdot 10^{12}$
Кубическое время	3592.00	14156682.39	$1.74 \cdot 10^{-20}$	$2.16 \cdot 10^{-21}$	26071065573	$4.99 \cdot 10^{11}$
Экспоненциальное время	переполн.	переполн.	переполн.	переполн.	$5.46 \cdot 10^{-21}$	4579076.11
Факториальное время	146.20	205693163.9	$1.19 \cdot 10^{12}$	$5.56 \cdot 10^{11}$	$1.90 \cdot 10^{11}$	17.59

При больших значениях n вычисление показательной функции приводит к ошибке переполнения. Это не является проблемой, поскольку сам факт возникновения переполнения говорит о неприменимости функции к алгоритму.

Заключение

По результатам проделанной работы видно, что трассировка алгоритмов помогает получить необходимые данные для подбора коэффициентов аппроксимирующих функций методом приближения с помощью кривых. Также удалось показать, что применение метода RMSE для сравнения выдаваемых функциями оценок с результатами трассировки позволяет с высокой точностью определить категорию сложности алгоритма. Однако следует помнить, что многие языки программирования содержат функции, предназначенные для обработки большого количества данных за одну операцию. Оценить категорию сложности решений, использующих такие функции, одним лишь предложенным методом невозможно. Одним из путей преодоления этого препятствия является применение статического анализа кода, что может стать темой дальнейших исследований в данном направлении.

Преимуществами предложенного метода определения категории сложности являются простота его реализации и независимость от производительности системы, на которой выполняется тестирование, поскольку подсчитывается количество итераций циклов, а не время выполнения программного кода, получаемые результаты оказываются значительно более точными. Также следует отметить, что рассмотренные в работе алгоритмы и категории сложности являются наиболее часто встречающимися в учебных материалах по программированию. Вместе с тем существуют и другие, которые также имеет смысл рассмотреть на применимость предлагаемого метода оценки, особенно случаи, когда решение задачи требует последовательно применить несколько различных алгоритмов.

Список литературы

1. Буянова И.В. Требования к системе онлайн-обучения студентов программированию и обзор существующих решений // Инженерные технологии: традиции, инновации, векторы развития: сборник материалов VII Всероссийской научно-практической конференции с международным участием (Абакан, 10-12 ноября 2021 г.) / отв. ред. Д.Ю. Карандеев. Абакан: Издательство ФГБОУ ВО «Хакасский государственный университет им. Н.Ф. Катанова», 2021. С. 109-111.

2. Селиванова И.А., Блинов В.А. Построение и анализ алгоритмов обработки данных: учеб.-метод. пособие. Екатеринбург: Изд-во Урал. ун-та, 2015. 108 с.

3. Virtanen P., Gommers R., Oliphant T.E. et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. Nat. Methods. 2020. V. 17. P. 261–272. DOI: 10.1038/s41592-019-0686-2.

4. Branch M.A., Coleman T.F., Li Y. A Subspace, Interior, and Conjugate Gradient Method for Large-Scale Bound-Constrained Minimization Problems. SIAM Journal on Scientific Computing. 1999. V. 21(1). P. 1–23. DOI: 10.1137/S1064827595289108.

5. Кормен Т.Х. Алгоритмы: вводный курс / Пер. с англ. М.: ООО «И.Д. Вильямс», 2014. 208 с.