УДК 510.52:372.851

Рублев В.С., Юсуфов М.Т.

Ярославский государственный университет им. П.Г. Демидова, г. Ярославль, Россия

АВТОМАТИЗИРОВАННАЯ СИСТЕМА ДЛЯ ОБУЧЕНИЯ АНАЛИЗУ ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ АЛГОРИТМОВ

АННОТАЦИЯ

Исследуются вопросы построения автоматизированной обучающей системы «Анализ сложности алгоритмов», которая позволит учащемуся освоить сложный математический аппарат и развить логико-математическое мышление в этом направлении. Вводится технология символьной прокрутки алгоритма, позволяющая получать верхние и нижние оценки вычислительной сложности. Приводятся утверждения, облегчающие анализ в случае целочисленного округления параметров алгоритма, а также при оценке сложности сумм. Вводится нормальная система символьных преобразований, позволяющая, с одной стороны, делать учащемуся любые символьные преобразования, а с другой стороны – упростить автоматический контроль корректности таких преобразований.

КЛЮЧЕВЫЕ СЛОВА

Автоматизированное обучение; анализ сложности алгоритмов; оценка вычислительной сложности алгоритмов; таблица символьной прокрутки алгоритма; теоремы об анализе сложности алгоритма; нормальная система символьных преобразований.

Rublev V.S., Yusufov M.T.

Yaroslavl State University of Demidov, Yaroslavl, Russia

AUTOMATED SYSTEM FOR TEACHING COMPUTATIONAL COMPLEXITY OF ALGORITHMS COURSE

ABSTRACT

Article describes problems of designing automated teaching system for "Computational complexity of algorithms" course of study. This system should provide its students with means to familiarize themselves with complex mathematical apparatus and improve their mathematical thinking in respective area. Then article introduces the technique of algorithms symbol scroll table that allows estimating lower and upper bounds of computational complexity. Further, we introduce a set of theorems that facilitate analysis in cases when integer rounding of algorithm's parameters is involved and when analyzing complexity of a sum. At the end, article introduces a normal system of symbol transformations that both allows one to perform any symbol transformations and simplifies automated validation of such transformations.

KEYWORDS

Automated learning; algorithm complexity analysis; estimating algorithms' computational complexity; algorithm's symbol scroll table; theorems on analysis of computational complexity; normal system of symbol transformations.

Введение

Одной из важных сторон образования специалиста по компьютерным наукам является обучение анализу вычислительной сложности алгоритмов. Это позволяет в практической его деятельности выбирать лучший в тех или иных условиях алгоритм и прогнозировать время выполнения программы. Успех в этом направлении может быть достигнут только при индивидуальном обучении. Однако большое количество индивидуальных заданий требует

больших временных затрат от преподавателя. Возможным решением этой проблемы может являться некий программный комплекс, позволяющий управлять индивидуальной работой каждого из студентов – автоматизированная обучающая система. Обучение на расстоянии – это альтернативный способ обучения, не требующий непосредственного контакта с преподавателем. Рост популярности дистанционных методов обучения в настоящее время основан на том, что они предоставляют возможность заниматься в удобное время и практически в любом месте.

Исследование подходов к построению автоматизированной обучающей системы

Существующие автоматизированные системы обучения хорошо решают проблемы обучения в тех областях знаний, где определениям и неглубоким связям между ними можно обучить при помощи тестов. Но анализ алгоритмов требует развития логико-математического мышления, следовательно, существует необходимость в интеллектуальных системах обучения, направленных на развитие такого мышления и приобретения знаний и навыков в сложной области знаний [1].

Существует большое количество программных систем, называющихся обучающими (например, Moodle, Claroline, Dokeos, ATutor), но большинство из них не поддерживают полный цикл обучения (методики) – это всего лишь приложения, которые предоставляют доступ к текстам, выдают задания и проверяют их [2]. Более продвинутым решением является использование в процессе обучения различных методик, которые изменяют поведение системы по отношению к конкретному пользователю в зависимости от его индивидуальных особенностей. Для решения этой задачи предлагается использовать адаптивные обучающие системы (АОС), основная парадигма которых – адаптация к каждому студенту. Адаптивные технологии обучения появились сравнительно недавно, но завоевали популярность среди разработчиков обучающих систем. В основе адаптивного обучения лежат следующие методологии:

- построение последовательности курса обучения;
- интеллектуальный анализ решений задач;
- интерактивная поддержка в решении задач;
- поддержка в решении задач на примерах;
- адаптивная поддержка в навигации;
- адаптивное представление;
- адаптивная поддержка сотрудничества пользователей системы (обучающихся).

Применение этих методологий обеспечивает систему гибкостью в отношении к пользователю и в отношении к представлению материала для изучения.

В дополнение к концепции адаптивных систем *теоретической и методологической базой* становится утверждение о том, что обучение может быть сведено к совокупности следующих составляющих:

- информации, необходимой для изучения;
- контрольным мероприятиям, позволяющим проверять знания по данным материалам;
- способу оценки уровня полученных знаний;
- последующему управлению – важному и самому сложному механизму, делающему систему именно обучающей.

Следовательно, необходимо ответить на следующие вопросы: как провести разбиение, как вести контроль и насколько гибкой будет система по отношению к пользователю.

Методология адаптивного обучения хорошо ложится на следующие постулаты о процессе обучения:

- любой сложный для понимания материал может быть разбит на последовательность таких малых порций, что каждая порция после освоения предыдущих может быть понята;
- для любой порции изучаемого материала может быть разработано некоторое конечное количество контрольных тестов, упражнений, задач таких, что при выполнении их студентом можно с достаточной точностью гарантировать усвоение им материала этой порции.

Поэтому весь материал, подлежащий обучению, и контрольные тесты, упражнения, задачи разобьём на секции так, что каждая секция будет представлять совокупность материала (текста для обучения) и контроля (обучения этому материалу).

Весь материал для обучения можно разделить на две большие категории. В первую категорию войдут секции, дающие основные теоретические знания и понятия по определению алгоритма, вычислительной сложности и асимптотическим оценкам трудоёмкости. Основная цель этих секций – развитие и тренировка памяти обучаемого с использованием навыков запоминания. Контроль этих секций, чаще всего, производится с помощью тестирования.

Однако, в программу курса также входит обучение неформальному использованию математического аппарата, и в этом месте возникают дополнительные трудности. К ним относятся технические трудности, связанные с вводом формул и их преобразованием, и трудности, связанные с недостаточным уровнем логического мышления учащегося, который должен достигнуть цели, конструируя последовательность ведущих к ней преобразований. Таким образом, определяется вторая категория секций, связанных с математическими методами оценки сложности алгоритма и направленных на развитие логико-математического мышления.

Выделяются следующие черты, отличающие вторую категорию секций от первой. Во-первых, контроль материала секций второй категории не может быть основан только на тестировании, потому что необходимо проверять умение учащегося использовать различные математические приёмы. Во-вторых, при изучении материала требуется научить обучаемого связывать отдельные приёмы в целенаправленный процесс путём конструирования последовательности изученных приёмов. В-третьих, нужно проконтролировать умение связывать несколько процессов при решении итоговой задачи по оценке вычислительной сложности алгоритма. Одним из инструментов, используемых в контроле материала секций второй категории, является алгоритм проверки символьных преобразований.

Материал и контроль взаимодействуют с помощью третьего компонента системы, ответственного за определение объёма материала за один сеанс, набор заданий и их количество, а также другие параметры системы. Именно этот компонент придаёт системе гибкость и отличает АОС от приложения, которое умеет только выдавать текст и набор тестов по нему. Далее следует описание предлагаемого сценария взаимодействия системы и пользователя.

При входе в систему пользователь видит список секций курса, которые делятся на доступные и недоступные в соответствии с планом прохождения курса. Каждая секция должна быть пройдена одна за другой по порядку (принцип линейного обучения). При входе в секцию студенту предоставляется для изучения её материал. После изучения материала он переходит к контролю полученных знаний, для прохождения которого он должен выполнить определённое в секции начальное количество заданий (тестов, упражнений, задач). На каждый вопрос теста предлагается несколько вариантов ответов (обычно 6), случайно выбранных из заранее определённого для каждого задания множества ответов. Среди предлагаемых ответов может быть несколько правильных, или все правильные, или ни одного правильного. Студент должен отметить все правильные ответы. Но если, по его мнению, правильных ответов нет, то он должен выбрать именно такой ответ. Система считает задание выполненным, если студент отметил все правильные ответы и ни одного неправильного. В противном случае она выводит на экран или один из текстов: «Не все правильные ответы отмечены», «Некоторые из отмеченных ответов не верны», или их комбинацию и предоставляет фрагмент текста материала секции, связанный с совершенной ошибкой. После изучения этого фрагмента или повторного изучения всего материала секции студент имеет возможность повторно ответить на вопрос задания. Если же он опять не выполнит задание, то оно будет заменено на два дополнительных задания. Таким образом, число заданий для прохождения секции может расти. Сеанс работы со студентом будет прекращён при достижении некоторого предельного количества заданий секции, и он сможет вернуться к обучению только после определённого перерыва. Если студент добивается прерывания сеанса несколько раз подряд, то его учётная запись в системе временно блокируется, а сам он вызывается к преподавателю. В случае, когда сначала студент заработал много дополнительных заданий, а затем начал отвечать безошибочно, число контрольных заданий начинает снижаться по некоторой прогрессии. Такой подход заставляет студента внимательно и вдумчиво относиться к материалу секции. Таким образом, организуется не только контроль знаний секции, но и обучение. Только после выполнения всех заданий студент сможет перейти к изучению материала следующей секции.

Однако, существуют секции, материал которых опирается на предыдущие секции и не может быть освоен без безусловного владения материалом этих предыдущих секций. В таких секциях проводится дополнительный контроль знаний предыдущих секций. При этом количество начальных заданий для повторения секции снижается до одного.

Вычислительная сложность алгоритмов и методика получения оценок сложности

Перейдём к описанию предметной области, поскольку материал секций тесно связан с нею. Сначала напомним определение вычислительной сложности алгоритма. Под вычислительной сложностью алгоритма чаще всего понимают время (число шагов), требующееся для выполнения алгоритма в зависимости от некоторых входных параметров. И хотя в некоторых случаях интерес представляет не только рост времени с ростом параметров, но и рост памяти, используемой для работы алгоритма, мы будем использовать термин *сложность алгоритма* для обозначения его

временной сложности.

Для оценки быстроты роста числа шагов $T(n)$, где n – выделенный параметр данных, используют O-нотацию: $T(n) = O(f(n))$, означающую оценку сверху быстроты роста $T(n)$ скоростью изменения $f(n)$, т. е.

$$\exists C > 0, n_0, \forall n \geq n_0 : T(n) \leq C \cdot f(n),$$

а также используют Ω-нотацию: $T(n) = \Omega(g(n))$, означающую оценку снизу быстроты роста $T(n)$ скоростью изменения $g(n)$, т. е.

$$\exists C > 0, n_0, \forall n \geq n_0 : T(n) \geq C \cdot g(n).$$

Эти оценки могут быть как завышенными, так и заниженными соответственно. Например, для $T(n) = n^2 - n + 3$ нотации $T(n) = \Omega(n)$, $T(n) = O(n^3)$ дают соответственно заниженную и завышенную оценку быстроты роста $T(n)$.

Поэтому наиболее удачны двусторонние оценки, используемые Θ-нотацией [2]: $T(n) = \Theta(f(n))$, означающую оценки сверху и снизу быстроты роста одной и той же функцией $f(n)$, т. е.

$$\exists C_1 > 0, C_2 > 0, n_0, \forall n \geq n_0 : C_1 \cdot f(n) \leq T(n) \leq C_2 \cdot f(n).$$

Так, в примере выше: $T(n) \leq C \cdot n^2$.

Очень важен выбор видов функций $f(n)$, которые участвуют в оценке скорости роста сложности алгоритма. В теории сложности алгоритмов принят некоторый ряд из функций

$$\log n,\ n^{m/k},\ 2^n,$$

где n – параметр алгоритма (как правило, целочисленный), m, k – некоторые целочисленные константы, а также из функций, полученных суперпозицией функций этого ряда. Например,

$$\log\log n, \log^{1/2} n, n^{3/2}, n^{\log n}, 2^{2^n}.$$

Если выделено несколько параметров данных алгоритма, то Θ-нотация скорости роста сложности алгоритма определяется через суперпозицию функций от каждого параметра. Например, $T(n, m, k) = \Theta(2^n m \log k)$.

Перейдём к методике получения оценок вычислительной сложности алгоритмов. Нашей целью является получение оценок сложности алгоритмов в виде Θ-нотации скорости её роста, если это возможно, или в противном случае в виде O-нотации и Ω-нотации. Но так как эта оценка связана с оценкой сложности циклов, то в первую очередь мы рассмотрим, как получить эту оценку для одного цикла.

Поскольку сложность алгоритма, содержащего циклы, определяется количеством выполнения этих циклов, то в основе методики получения оценок лежит составление *символьной таблицы прокрутки алгоритма*. В этой таблице каждой переменной алгоритма соответствует свой столбец, а также в ней есть некоторые специальные столбцы:

- для каждого цикла столбец с номером выполнения цикла и символьным обозначением этого номера при последнем выполнении цикла;
- столбец *Условие цикла*, в котором записывается символьное условие выполнения цикла для последнего его выполнения с комментарием *посл. вып.* и условие выхода из цикла с комментарием *выход*.

В эти условия входит символьное значение параметра номера последнего выполнения цикла. Использование этих условий даёт 2 оценки количества выполнения цикла, определённого символом: снизу и сверху. Анализ этих оценок позволяет определить временную сложность цикла, если выражения условий не являются сложными. Чаще всего ([2]) оценки снизу и сверху разнятся только константным множителем, что приводит к Θ-нотации $T(n) = \Theta(f(n))$ скорости роста времени выполнения цикла по параметру алгоритма n. Рассмотрим пример 1 алгоритма с одним циклом:

```
void f1 (unsigned long n) {
  float x = n;
  while (x > 2)
    x = sqrt(x);
}
```

Составим таблицу символьной прокрутки, включив в неё столбец i с номером выполнения цикла, столбец со значением параметра цикла и столбец с условием выполнения цикла. При этом p в столбце i обозначает номер последнего выполнения цикла.

Анализ последнего выполнения цикла даёт неравенство $n^{(1/2)^{p-1}} > 2$, из которого следует $\log_2 n > 2^{p-1}$, откуда получаем неравенство $p < \log_2\log_2 n + 1$. Из условия выхода из цикла $n^{(1/2)p} \leq 2$ следует неравенство $p \geq \log_2\log_2 n$, и так как p определяет временную сложность алгоритма, то мы получаем

$$T_{f1}(n) = \Theta(\log\log n).$$

Таблица 1. Символьная прокрутка алгоритма 1

i	x	условие цикла	
	n		
1	$n^{1/2}$	$n > 2$	
2	$n^{(1/2)^2}$	$n^{1/2} > 2$	
3	$n^{(1/2)^3}$	$n^{(1/2)^2} > 2$	
…	…	…	
i	$n^{(1/2)^i}$	$n^{(1/2)^{i-1}} > 2$	
…	…	…	
p	$n^{(1/2)^p}$	$n^{(1/2)^{p-1}} > 2$	посл. вып.
$p+1$		$n^{(1/2)^p} \not> 2$	выход

Если циклы не являются вложенными и не являются зависимыми (параметр, определяемый первым циклом, не участвует во втором), то, определив временную сложность каждого из циклов, для временной сложности алгоритма берём наибольшую из них.

Если циклы вложены, но независимы (количество выполнения внутреннего цикла не зависит от номера выполнения внешнего цикла), то для определения общей сложности таких циклов их сложности перемножаются. Так, если внешний цикл A имеет сложность $T_A(n_A) = \Theta(f(n_A))$, а внутренний цикл B имеет сложность $T_B(n_B) = \Theta(g(n_B))$, то их общая сложность

$$T_{AB}(n_A, n_B) = \Theta(f(n_A) \cdot g(n_B)).$$

При невложенных, но зависимых циклах для первого из них надо определять значение параметра, участвующего во втором цикле. Рассмотрим пример 2 алгоритма для этого случая:

```
void f2 (unsigned long n) {
  float x = n, z = n;
  while (x > 2) {
    x = sqrt(x);
    z = z * z;
  }
  while (z /= 2 > 1);
}
```

Составим таблицу символьной прокрутки, включив в неё столбец $N_ц$ с номерами циклов 1 и 2, столбцы i_1, i_2 с номерами выполнения каждого цикла, столбцы x, z со значением этих переменных и столбец с условием выполнения цикла. При этом p_1 в столбце i_1 обозначает номер последнего выполнения цикла 1 (внешнего), а p_2 в столбце i_2 – номер последнего выполнения цикла 2 (внутреннего). В таблице мы будем записывать только изменение значений объектов каждого столбца.

Анализ цикла 1 такой же, как и в предыдущем примере 1, и приводит к оценкам снизу и сверху для количества выполнения этого цикла $\log_2 \log_2 n \le p_1 < \log_2 \log_2 n + 1$, из которого следует $p_1 = \log_2 \log_2 n$. Это дает следующую временную сложность цикла 1: $T_1(n) = \Theta(\log \log n)$.

Таблица 2. Символьная прокрутка алгоритма 2

$N_ц$	i_1	i_2	x	z	условие цикла	
			n	n		
1	1		$n^{1/2}$		$n > 2$	
	2		$n^{(1/2)^2}$		$n^{1/2} > 2$	
	…		…		…	
	i		$n^{(1/2)^i}$		$n^{(1/2)^{i-1}} > 2$	
	…		…		…	
	p_1		$n^{(1/2)^{p_1}}$		$n^{(1/2)^{p_1-1}} > 2$	посл. вып.
	p_1+1				$n^{(1/2)^{p_1}} \not> 2$	выход
2		1		$n^{2p_1}/2$	$n^{2p_1} > 1$	
		2		$n^{2p_1}/2^2$	$n^{2p_1}/2 > 1$	
		…		…	…	
		p_2		$n^{2p_1}/2^{p_2}$	$n^{2p_1}/2^{p_2-1} > 1$	посл. вып.
		p_2+1			$n^{2p_1}/2^{p_2} > 1$	выход

Анализ условия последнего выполнения цикла 2: $n^{2^{p_1}}/2^{p_2-1} > 1$, учитывая значение p_1

приводит к неравенству $2^{p_2-1} < n^{2^{\log_2 \log_2 n}} = n^{\log_2 n}$. Логарифмируя неравенство получаем $p_2 - 1 < \log_2^2 n$, что дает $p_2 < \log_2^2 n + 1$, а учитывая целочисленность p_2 получаем $p_2 \le \log_2^2 n$.

Анализ выхода из цикла 2: $n^{2^{p_1}}/2^{p_2} \le 1$ приводит, учитывая значение p_1, к неравенству $2^{p_2} \ge n^{2^{\log_2 \log_2 n}} = n^{\log_2 n}$. Логарифмируя его, получаем $p_2 \ge \log_2^2 n$, что вместе с предыдущим неравенством даёт $p_2 = \log_2^2 n$, и, следовательно, временная сложность цикла 2: $T_2(n) = \Theta(\log_2^2 n)$. Выбирая максимальную оценку временной сложности циклов окончательно получаем временную сложность алгоритма:

$$T_{f2}(n) = \Theta(\log^2 n).$$

Если циклы вложены и зависимы, то временная сложность алгоритма образуется из суммарного количества выполнения вложенного цикла по всем выполнениям внешнего. Рассмотрим пример 3 для этого случая.

```
void f3 (unsigned long n) {
  float x = n, y, z = n;
  while (x > 2) {
    x = sqrt(x);
    z = z * z;
    y = z;
    while (y /= 2 > 1);
  }
}
```

Составим таблицу символьной прокрутки (на следующей странице), включив в неё столбец $N_ц$ с номерами циклов 1 и 2, столбцы i_1, i_2 с номерами выполнения каждого цикла, столбцы x, z, y со значением этих переменных и столбец с условием выполнения цикла. При этом p_1 в столбце i_1 обозначает номер последнего выполнения цикла 1 (внешнего), а p_2 в столбце i_2 – номер последнего выполнения цикла 2 (внутреннего). В таблице мы будем записывать только изменение значений объектов каждого столбца.

Таблица 3. Символьная прокрутка алгоритма 3

$N_ц$	i_1	i_2	x	z	y	условие цикла	
			n	n			
1	1		$n^{1/2}$	n^2	n^2	$n > 2$	
2		1			$n^2/2$	$n^2/2 > 1$	
		2			$n^2/2^2$	$n^2/2^2 > 1$	
		…			…	…	
		$p_2(1)$			$n^2/2^{p_2(1)}$	$n^2/2^{p_2(1)} > 1$	посл. вып.
		$p_2(1)+1$			$n^2/2^{p_2(1)+1}$	$n^2/2^{p_2(1)+1} \not> 1$	выход
1	2		$n^{(1/2)^2}$	n^{2^2}	n^{2^2}	$n^{1/2} > 2$	
2		1			$n^{2^2}/2$	$n^{2^2}/2 > 1$	
		2			$n^{2^2}/2^2$	$n^{2^2}/2^2 > 1$	
		…			…	…	
		$p_2(2)$			$n^{2^2}/2^{p_2(2)}$	$n^{2^2}/2^{p_2(2)} > 1$	посл. вып.
		$p_2(2)+1$			$n^{2^2}/2^{p_2(2)+1}$	$n^{2^2}/2^{p_2(2)+1} \not> 1$	выход
…	…	…	…	…	…	…	
1	i		$n^{(1/2)^i}$	n^{2^i}	n^{2^i}	$n^{(1/2)^{i-1}} > 2$	
2		1			$n^{2^i}/2$	$n^{2^i}/2 > 1$	
		2			$n^{2^i}/2^2$	$n^{2^i}/2^2 > 1$	
		…			…	…	
		$p_2(i)$			$n^{2^i}/2^{p_2(i)}$	$n^{2^i}/2^{p_2(i)} > 1$	посл. вып.
		$p_2(i)+1$			$n^{2^i}/2^{p_2(i)+1}$	$n^{2^i}/2^{p_2(i)+1} \not> 1$	выход
…	…	…	…	…	…	…	
1	p_1		$n^{(1/2)^{p_1}}$	$n^{2^{p_1}}$	$n^{2^{p_1}}$	$n^{(1/2)^{p_1-1}} > 2$	посл. вып.
2		1			$n^{2^{p_1}}/2$	$n^{2^{p_1}}/2 > 1$	
		2			$n^{2^{p_1}}/2^2$	$n^{2^{p_1}}/2^2 > 1$	
		…			…	…	
		$p_2(p_1)$			$n^{2^{p_1}}/2^{p_2(p_1)}$	$n^{2^{p_1}}/2^{p_2(p_1)} > 1$	посл. вып.
		$p_2(p_1)+1$			$n^{2^{p_1}}/2^{p_2(p_1)+1}$	$n^{2^{p_1}}/2^{p_2(p_1)+1} \not> 1$	выход
1	p_1+1					$n^{(1/2)^{p_1}} \not> 2$	выход

Анализ количества выполнения цикла 1 такой же, как и в предыдущих примерах 1 и 2, и приводит к оценкам снизу и сверху для количества выполнения этого цикла $\log_2 \log_2 n \le p_1 < \log_2 \log_2 n + 1$, из которого следует $p_1 = \log_2 \log_2 n$.

Общее количество выполнения цикла 2 даётся формулой $T_2(n) = \sum_1^{p_1} p_2(i)$. Для его нахождения определим общий член этой суммы $p_2(i)$ как количество выполнений цикла 2 при i-м выполнении цикла 1. При последнем выполнении такого цикла 2 имеет место неравенство $n^{2^i}/2^{p_2(i)} > 1$, откуда получаем $2^{p_2(i)} < n^{2^i}$, что после логарифмирования даёт неравенство $p_2(i) < 2^i \log_2 n$. При выходе из такого цикла имеет место неравенство $n^{2^{p_1}}/2^{p_2(i)+1} \le 1$, откуда получаем $2^{p_2(i)+1} \ge n^{2^i}$, что после логарифмирования и переноса единицы даёт неравенство $p_2(i) \ge 2^i \log_2 n - 1$. Подставляя полученные неравенства в сумму, получим оценки снизу и сверху для общего количества выполнения цикла 2: $\sum_1^{p_1}(2^i \log_2 n - 1) \le \sum_1^{p_1} p_2(i) < \sum_1^{p_1} 2^i \log_2 n$.

Преобразуем сумму в верхней оценке: $\sum_1^{p_1} 2^i \log_2 n = \log_2 n \sum_1^{p_1} 2^i = 2(2^{p_1} - 1)\log_2 n = 2\left(2^{\log_2 \log_2 n} - 1\right)\log_2 n = 2(\log_2 n - 1)\log_2 n < 2\log_2^2 n$.

Теперь преобразуем сумму в нижней оценке: $\sum_1^{p_1}(2^i \log_2 n - 1) = \sum_1^{p_1} 2^i \log_2 n - p_1 = 2(\log_2 n - 1)\log_2 n - p_1 = 2\log_2^2 n - \log_2 n - \log_2 \log_2 n > 2\log_2^2 n - \log_2^2 n - \frac{1}{2}\log_2^2 n = \frac{1}{2}\log_2^2 n$.

Подставляя обе оценки, получим $\frac{1}{2}\log_2^2 n < \sum_1^{p_1} p_2(i) < 2\log_2^2 n$, откуда следует временная сложность алгоритма $T_{f3}(n) = \Theta(\log^2 n)$.

Заметим, что во всех примерах условия выхода из циклов задаётся неравенством, включающим параметр алгоритма и параметр номера последнего выполнения цикла. В общем случае условие выхода из цикла может быть сложной булевой функцией. Однако, любую булеву функцию можно преобразовать к дизъюнктивной нормальной форме (ДНФ), а затем исследовать для каждой элементарной конъюнкции ДНФ все входящие в неё неравенства и равенства, из которых оценить снизу выражение значения номера последнего выполнения цикла, при котором все входящие в элементарную конъюнкцию условия становятся истинными. Взяв минимальную из оценок по всем элементарным конъюнкциям ДНФ, получим оценку снизу на значение номера последнего выполнения цикла. Аналогичным образом можно получить оценку сверху этого параметра из ДНФ последнего выполнения цикла. Рассмотрим пример алгоритма 4:

```
void f4(unsigned long n) {
  float x, y;
  x = y = n;
  while (x > 1 || y > 2048) {
    x = x / 2;
    y = y - 128;
  }
}
```

Составим таблицу символьной прокрутки, включив в неё по одному столбцу для каждой элементарной конъюнкции из записи условия цикла в ДНФ.

Таблица 4. Символьная прокрутка алгоритма 4

i	x	y	*Условие 1*	Условие 2	
	n	n			
1	$\frac{n}{2}$	$n - 128$	$\frac{n}{2} > 1$	$n - 128 > 2048$	
2	$\frac{n}{2^2}$	$n - 128 \cdot 2$	$\frac{n}{2^2} > 1$	$n - 128 \cdot 2 > 2048$	
…	…	…	…	…	
i	$\frac{n}{2^i}$	$n - 128 \cdot i$	$\frac{n}{2^i} > 1$	$n - 128 \cdot i > 2048$	
…	…	…	…	…	
p	$\frac{n}{2^p}$	$n - 128 \cdot p$	$\frac{n}{2^p} > 1$	$n - 128 \cdot p > 2048$	посл. вып.
$p+1$	$\frac{n}{2^{p+1}}$	$n - 128 \cdot (p+1)$	$\frac{n}{2^{p+1}} \not> 1$	$n - 128 \cdot (p+1) \not> 2048$	выход

Анализ количества выполнений цикла с условием 1 приводит к оценкам сверху и снизу $\log_2 n - 1 \le p < \log_2 n$, что может дать оценку $\Theta(\log n)$. Анализ условия 2 дает иные оценки – $\frac{n}{128} - 17 \le p < \frac{n}{128} - 16$, что может дать оценку $\Theta(n)$. Из этого следует, что алгоритм может иметь разные

оценки в разных диапазонах параметра n. Чтобы уточнить эти диапазоны, приравняем верхние и нижние оценки для условий: $\frac{n}{128} - 16 = \log_2 n$, $\frac{n}{128} - 17 = \log_2 n - 1$; Уравнения получились эквивалентными, поэтому дальше рассматриваем только одно из них. Его корнями являются числа $n_1 = 3558$ и $n_2 \cong 0.00001$.

Если внимательно взглянуть на условие цикла, то можно заметить, что цикл не выполнится ни разу при значениях параметра от 0 до 2048 включительно, поэтому второй корень нас не интересует. При значениях параметра от 2048 до 3558 включительно оценка будет линейной, а при значениях параметра от 3559 и выше оценка будет логарифмической. Таким образом, сложность алгоритма $\Theta(\log n)$.

Анализ условий циклов и его упрощение в некоторых случаях

Общим для всех рассмотренных примеров условий (выполнения цикла или выхода из цикла) является отношение, которое может быть приведено к виду:

$f(p, n)$ <знак операции отношения> <выражение, не содержащее параметры p и n>,

где n – натуральный параметр алгоритма, а p – параметр номера выполнения цикла. В простых случаях функции $f(p, n)$ сравнительно просто получить Θ-нотацию роста количества выполнений цикла через рост n. Но в случае сложного вида этой функции достаточно получить скорость изменения функции $f(p, n)$ при росте ее параметров в виде Θ-нотации для более простой функции $g(p, n)$, позволяющей провести дальнейший анализ отношения:

$$f(p, n) = \Theta(g(p, n)),$$

означающую $\exists C_1 > 0, C_2 > 0, n_0, \forall p > 0, n \geq n_0 : C_1 \cdot g(p,n) \leq f(p,n) \leq C_2 \cdot g(p,n)$.

Так, если алгоритм целочисленный (параметры принимают только целые значения за счёт преобразования к целому), то анализ получаемых выражений может быть осложнён. Суммирование последовательностей, отличающихся от арифметической или геометрической последовательностей, также усложнит анализ. Следующие теоремы позволяют получить Θ-нотацию для многократных операций целых частей линейных выражений от параметра n и Θ-нотацию через интегралы для целочисленных сумм.

Пусть $\mu_1(a \cdot n + b) = a \cdot n + b$ линейная форма с натуральным параметром алгоритма n и положительным a. Обозначим через $\mu_p(a \cdot n + b)$ функцию $\mu_p(a \cdot n + b) = a \cdot (a \cdot \ldots (a \cdot n + b) + \cdots + b) + b$, полученную p-кратным применением формы μ_1, а через $\mu_p([a \cdot n + b]) = [a \cdot [a \cdot \ldots \cdot [a \cdot n + b] + \cdots + b] + b]$, где квадратные скобки означают целую часть выражения, функцию полученную p-кратным применением формы μ_1 к целой части значения линейной формы. В том случае, когда такая функция связана с количеством шагов цикла, нас интересует оценка скорости её изменения. Однако, при $a > 1$ указанная функция возрастает с ростом n и нас интересует Θ-нотация этого возрастания, а при $a < 1$ она убывает с ростом n и нас интересует Θ-нотация скорости её убывания. Следующая теорема позволяет при получении Θ-нотации снять операцию взятия целой части.

Теорема 1. *Пусть a, b вещественные коэффициенты линейной формы $a \cdot n + b$ с $a > 0, a \neq 1$ и натуральным n. Тогда $\mu_p([a \cdot n + b]) = \Theta(\mu_p(a \cdot n + b)) = \Theta(a^p n)$.*

В качестве примера использования теоремы 1 приведём анализ алгоритма следующей процедуры A:

```
procedure A (unsigned long N) {
  for (unsigned long k = N; k > 1; k = k/2);
}
```

Цикл будет выполняться p раз и при последнем выполнении цикла переменная k примет значение 1. Поэтому из условия $\left[\frac{1}{2} \cdot \left[\frac{1}{2} \cdot \ldots \cdot \left[\frac{1}{2} \cdot N\right]\right] \ldots\right] = 1$ получаем $1 \leq \left(\frac{1}{2}\right)^p \cdot N < 2$, что при логарифмировании дает оценку $\log_2 N - 1 < p \leq \log_2 N$, а, следовательно, временная сложность процедуры оценивается как $\Theta(\log_2 N)$.

Следующая теорема 2 устанавливает оценку вычислительной сложности конечной суммы через Θ-нотацию интеграла.

Теорема 2. *Пусть для неотрицательной монотонно возрастающей функции $f(x)$ $(x \geq 0)$ рост ее значений ограничен условием $f(x) \leq C \cdot f(x-1)$, где константа $C \geq 1$. Тогда справедлива следующая формула*

$$\sum_{x=m}^{n} f(x) = \Theta\left(\int_{m-1}^{n} f(x)\,dx\right) \quad (\forall m > 0).$$

В качестве примера использования теоремы 2 приведём анализ алгоритма следующей процедуры B:

```
procedure B (unsigned long n) {
  unsigned long m = 0;
  for (unsigned long i = 1, j = 2; i < n; i++, j <<= 1)
    m += i * j;
  while (m--);
}
```

Временная сложность первого цикла оценивается как Θ (n), а временную сложность второго цикла определяет суммарное время выполнения второго цикла $\Theta(\sum_{i=1}^{n} i \cdot 2^i) = \Theta\left(\int_{x=0}^{n} x \cdot 2^x \, dx\right) = \Theta(n2^n)$. Поэтому временная сложность процедуры B оценивается как $\Theta(n2^n)$.

Условие теоремы 2 является существенным, так как при его нарушении интеграл не является элементарной функцией. Однако и в этом случае удаётся получить оценку сложности суммы как Θ-нотацию верхнего предела суммирования, что утверждает следующая теорема 3.

Теорема 3. *Пусть для положительной монотонно возрастающей функции* $f(x), (x \geq 0)$, *функция* $g(x)$, *определённая отношением* $\frac{f(x)}{f(x-1)} = g(x), (x \geq 1)$, *является монотонно возрастающей неограниченной сверху функцией. Тогда справедлива следующая формула*

$$\sum_{x=m}^{n} f(x) = \Theta\big(f(n)\big), \qquad (\forall n > m > 0).$$

В качестве примера использования теоремы 3 приведём анализ алгоритма следующей процедуры C:

```
procedure C (unsigned long n) {
  unsigned long k = 0, m = 1;
  for (int i = 1; i <= n; i++) {
    m *= i;
    k += m;
    while (k--);
  }
}
```

Временная сложность первого цикла оценивается как $\Theta(n)$, а временную сложность второго цикла определяет суммарное количество его выполнений $k = \sum_{i=1}^{n} i!$. Так как $g(i) = \frac{i!}{(i-1)!} = i$ и $g(i) > 2$ при $i > n_0 = 2$, то по теореме 3 получаем $\Theta(\sum_{i=1}^{n} i!) = \Theta(n!)$. Используя формулу Стирлинга, получаем, что временная сложность процедуры C оценивается как $\Theta(n^n)$.

Нормальная система символьных преобразований

Приведённые теоремы позволяют ускорить анализ сложности алгоритмов определённого класса. Но, так как это возможно не во всех случаях, то для анализа приходится выполнять преобразования символьных выражений, а также равенств и неравенств с такими выражениями. Поэтому появляется проблема контроля правильности проведения преобразований студентом. Подход с использованием сложных систем преобразования символьных выражений, например, Mathcad, является неверным, так как они не помогают научить студента выполнять эти преобразования. Воспользуемся следующим подходом: сначала выделим те части анализа, которые требуют таких преобразований, а затем выделим некоторую ограниченную группу *допустимых* преобразований, при помощи которых может быть выполнено любое преобразование, необходимое для анализа сложности алгоритма. Назовём эту группу *нормальными* преобразованиями.

Случаями использования символьных преобразований являются следующие:

- символьная прокрутка алгоритма с преобразованием выражений, определяющих изменение данных алгоритма на этапе анализа значения переменных в таблице;
- конструирование неравенств для параметра отдельного цикла с помощью таблицы символьной прокрутки. Эти неравенства и определяют вычислительную сложность этого цикла;
- символьное преобразование равенств и неравенств как для переменных, так и для количества итераций циклов;
- проведение оценки итоговой сложности алгоритма по сложности отдельных циклов или по суммарным оценкам изменения параметров сложности алгоритма.

В качестве первого допустимого преобразования возьмём изменение порядка двух рядом стоящих аддитивных членов или множителей. Все остальные допустимые преобразования не будут изменять порядок преобразуемых членов. Контроль таких преобразований упрощается, но через них все равно можно выразить любое необходимое преобразование.

Перечислим допустимые шаги символьного преобразования формулы равенства – систему нормальных преобразований равенств:

- перестановка членов в формуле – замена местами двух членов только в одном месте формулы, где такая перестановка допустима;
- перестановка членов формулы из одной части равенства в другую со сменой знака;
- арифметические преобразования: сокращение только одного общего множителя;
- арифметические преобразования: вынесение только одного общего множителя;
- арифметические преобразования: разложение на множители, причём факторизация происходит атомарно (выносится только один из множителей);
- арифметические преобразования: использование основных тождеств для функций/операций (множество функций, заранее определённых в интерфейсе системы);
- символьное раскрытие скобок – только в одном месте формулы может быть раскрыта одна пара скобок;
- символьное группирование – однократное заключение в скобки некоторой части формулы (вся часть находится в одном месте формулы) с вынесением за скобки общего члена в скобках;
- символьное разложение на множители выражения, если оно может быть записано в виде произведения сомножителей;
- символьное разложение на множители степеней выражения, если она может быть записана в виде произведения сомножителей;
- символьное разложение на множители произведения сумм в выражении, если оно может быть записано в виде произведения сомножителей; порядок членов не должен изменяться
- однократное символьное перемножение степеней выражения, которое объединяет члены, содержащие одинаковые степени общего подвыражения;
- однократное символьное перемножение произведения сумм в выражении с соблюдением порядка записи каждого из полученных слагаемых (действие похоже на символьное раскрытие скобок).

Аналогично строится система нормальных преобразований для неравенств. Но при получении неравенств для преобразования к верхним и нижним оценкам сложности циклов вводится дополнительные нормальные преобразования:

- перемещение *ведущего аддитивного члена* (максимальная скорость роста) на первое место в одной из частей неравенства;
- усиление оценки сверху отбрасыванием аддитивных частей со знаком минус;
- усиление оценки снизу отбрасыванием аддитивных частей со знаком плюс;
- оценивание в оценке сверху неведущего аддитивного члена со знаком плюс через ведущий член;
- оценивание в оценке снизу неведущего аддитивного члена со знаком минус через ведущий член.

Указанная система нормальных преобразований требует введения дополнительных секций АОС для обучения этому материалу. Поэтому предлагается следующий порядок секций АОС:

1. Характеристики сложности алгоритма;
2. Определение временной сложности алгоритма;
3. Система нормальных преобразований равенств;
4. Таблица символьной прокрутки алгоритма;
5. Система нормальных преобразований неравенств;
6. Оценивание временной сложности алгоритма с простым циклом;
7. Оценивание временной сложности алгоритма с вложенным независимым циклом;
8. Оценивание временной сложности алгоритма с невложенными зависимыми циклами;
9. Оценивание временной сложности алгоритма с вложенными зависимыми циклами;
10. Оценивание временной сложности алгоритма с целочисленными преобразованиями выражений;
11. Оценивание временной сложности алгоритма с суммированием последовательностей количества выполнения цикла;
12. Итоговое задание.

При необходимости сложную по обучению секцию можно разбить на подсекции (например, секции с системой нормальных преобразований).

Заключение

Описанный подход обучения анализу сложности алгоритмов позволяет перейти

- к построению методического обеспечения, выделяющего для каждой секции и подсекции материал обучения и контрольные тесты, упражнения, задачи, необходимые для взаимодействия учащегося с материалом в процессе обучения и контроля усвоения материала;
- к построению программного обеспечения, позволяющего через интерфейс выполнять все этапы обучения.

Выразим надежду, что описанный подход к построению автоматизированной обучающей системы анализу вычислительной сложности алгоритмов будет реализован и покажет эффективность в обучении этому предмету и развитию логико-математического мышления студентов.

Литература

1. Ермилова А. В., Рублев В. С. Проблемы развития математического мышления учащихся на примере обучающей системы по курсу "Алгоритмы и анализ сложности" // Современные информационные технологии и ИТ-образование // Сборник избранных трудов IX Международной научно-практической конференции. Под ред. проф. В.А. Сухомлина. - М.: ИНТУИТ.РУ, 2014. – С.297-304
2. Кормен Т. и др. Алгоритмы: построение и анализ. — М.: «Вильямс», 2013.

References

1. Ermilova A. V., Rublev V. S. Problemy razvitiya matematicheskogo myshleniya uchashchikhsya na primere obuchayushchey sistemy po kursu "Algoritmy i analiz slozhnosti" // Sovremennye informatsionnye tekhnologii i IT-obrazovanie // Sbornik izbrannykh trudov IX Mezhdunarodnoy nauchno-prakticheskoy konferentsii. Pod red. prof. V.A. Sukhomlina. - M.: INTUIT.RU, 2014. – S.297-304
2. Kormen T. i dr. Algoritmy: postroenie i analiz. — M.: «Vil'yams», 2013.

Поступила: 10.09.2016

Об авторах:

Рублев Вадим Сергеевич, профессор кафедры теоретической информатики Ярославского государственного университета им. П.Г. Демидова, профессор, кандидат физико-математических наук, roublev@mail.ru**;**

Юсуфов Мурад Теймурович, аспирант кафедры теоретической информатики Ярославского государственного университета им. П.Г. Демидова, flood4life@gmail.com.