

9. Куликов А. С., Федин С. С. Автоматические доказательства верхних оценок на время работы алгоритмов расщепления // Теория сложности вычислений. IX. Зап. научн. сем. ПОМИ. СПб., 2004. Т. 316. С. 111–128.
10. Быкова В. В. Математические методы анализа рекурсивных алгоритмов // Журнал Сибирского федерального университета. Сер. Математика и физика. 2008. № 1(3). С. 236–246.

УДК 519.6

К РЕШЕНИЮ БОЛЬШИХ СИСТЕМ СРАВНЕНИЙ

К. Д. Жуков, А. С. Рыбаков

Пусть дано конечное множество S натуральных чисел, такое, что почти все его элементы попарно взаимно просты. Рассматривается алгоритм нахождения всех элементов $s \in S$, таких, что $(s, s') > 1$ для некоторого $s' \in S$, $s' \neq s$, который позволяет сводить произвольную систему полиномиальных сравнений к нескольким системам с взаимно простыми модулями.

Ключевые слова: взаимно простая база, наибольший общий делитель, дерево наибольших общих делителей, НОД слиянием.

Определение 1. Пусть S — конечное подмножество натуральных чисел. Взаимно простой базой для S называется конечное подмножество натуральных чисел B , такое, что любое число из S представляется в виде произведения элементов из B и любые два элемента множества B взаимно просты.

Взаимно простые базы используются в ряде приложений, например при решении систем сравнений в целых числах [1].

Пусть задана система полиномиальных сравнений вида $f_i(x) = 0 \pmod{a_i}$, $i = 0, 1, \dots, m-1$, где x — набор переменных. Построим для множества чисел $\{a_0, \dots, a_{m-1}\}$ взаимно простую базу $\{b_0, \dots, b_{r-1}\}$. Для каждого элемента b_j нужно последовательно составить системы сравнений по модулям b_j^l , где $l = 1, 2, \dots, t_j$, а t_j — максимальное, для которого найдётся a_i , делящееся на $b_j^{t_j}$. В эти системы войдут те уравнения $f_i(x) = 0$, для которых число a_i делится на $b_j^{t_j}$. Поднимая по лемме Гензеля решения от системы к системе, получим множество решений по модулю $b_j^{t_j}$. Найдя такие решения для каждого b_j , применим китайскую теорему об остатках, чтобы получить решения вида $x \pmod{\prod_j b_j^{t_j}}$. Каждое такое решение будет решением исходной системы.

Построение взаимно простой базы применяется и в других приложениях, например в задачах нахождения алгебраических зависимостей среди радикалов [2], вычисления нормальных базисов в расширениях полей [3] и др. [4].

В работе [4] Д. Бернштейн предложил алгоритм построения взаимно простой базы с некоторыми дополнительными свойствами, называемой естественной взаимно простой базой.

Ниже предлагается подход к построению взаимно простой базы для специального случая. Подход эффективен, когда заранее известно, что нетривиальная часть взаимно простой базы (элементы, отличные от 1 и не содержащиеся в S) достаточно мала. Метод состоит в том, чтобы быстро отбросить элементы $s \in S$, такие, что $(s, s') = 1$ для любого $s' \in S$, $s' \neq s$. Для остальных чисел можно применить известные алгоритмы построения взаимно простой базы.

С помощью известного алгоритма вычисления НОД слиянием можно проверить, являются ли все элементы множества S взаимно простыми. Данный алгоритм заключается в вычислении наибольших общих делителей в дереве произведений элементов множества S . С помощью небольшой модификации можно получить алгоритм 1 для выявления элементов $s \in S$, имеющих нетривиальный НОД с некоторым элементом из $S \setminus \{s\}$. Везде далее через $\lg$ обозначен логарифм по основанию 2.

Алгоритм 1

Вход: $S = \{a_0, \dots, a_{m-1}\}$ — множество различных n -битовых натуральных чисел (для упрощения m — степень двойки)

Выход: T — подмножество S , такое, что для любого $s \in T$ и для некоторого $s' \in S$, $s' \neq s$, выполняется условие $(s, s') > 1$.

1: Для всех $i = 0, \dots, m-1$

$a'_i \leftarrow a_i$.

2: Для всех $i = 1, 2, \dots, \lg m$

3: Для всех $j = 0, 1, \dots, 2^{\lg m - i} - 1$

4: $d \leftarrow (a'_{2j}, a'_{2j+1})$.

5: Если $d \neq 1$, то

6: Для всех $l = j2^i, j2^i + 1, \dots, (j+1)2^i - 1$

7: Если $(d, a_l) \neq 1$, то

$T \leftarrow T \cup \{a_l\}$;

8: $a'_j \leftarrow a'_{2j} \cdot a'_{2j+1}$.

9: Вывести T

Важное свойство алгоритма 1 заключается в том, что в памяти нужно хранить только текущий уровень дерева, а не всё дерево. Отсюда следует, что алгоритму 1 требуется $O(mn)$ битов памяти. Трудоёмкость алгоритма складывается из трудоёмкости вычисления дерева наибольших общих делителей ($O(mn \lg m \lg^2(n\sqrt{m}) \lg \lg(mn))$ битовых операций) и трудоёмкости всех вызовов цикла в строках 6 и 7 ($O(kmn \lg m \times \lg(kn) \lg \lg(kn))$ битовых операций, где $k = |T|$).

Несложно видеть, что при $k < \lg^2(mn)$ трудоёмкость описанного алгоритма не превосходит оценки трудоёмкости алгоритма Бернштейна в $O(mn \lg m \lg^3(nm) \lg \lg(mn))$ битовых операций.

Известен ещё один подход к выявлению множества T , вытекающий из работы [5].

Вычисляется произведение $U = \prod_{i=0}^{m-1} a_i$ с помощью дерева произведений. Затем вычисляются величины $b_i = U \bmod a_i^2$, $i = 0, \dots, m-1$, с помощью дерева остатков. На выход подаются те числа a_i , для которых $(b_i/a_i, a_i) \neq 1$. Трудоёмкость построения деревьев произведений и остатков оценивается в $O(mn \lg m \lg(nm) \lg \lg(mn))$ битовых операций. При этом построение дерева остатков требует $O(mn \lg m)$ битов памяти, что в $\lg m$ раз больше, чем требуемая память для алгоритма 1.

ЛИТЕРАТУРА

1. Bach E., Miller G. and Shallit J. Factor refinement // J. Algorithms. 1993. No. 15. P. 199–222.
2. Smedley T. J. Detecting algebraic dependencies between unnested radicals: extended abstract // Proc. Intern. Symp. on Symbolic and Algebraic Computation. Tokyo, Japan, 1990. P. 292–293.
3. Luneburg H. On a little but useful algorithm // AAECC'85. LNCS. 1986. V. 229. P. 296–301.

4. Bernstein D. J. Factoring into coprimes in essentially linear time // J. Algorithms. 2005. No. 54(1). P. 1–30.
5. Bernstein D. J. Scaled reminder trees // <http://cr.yp.to/papers.html#scaledmod>

УДК 519.688

ОПТИМИЗАЦИЯ $(p - 1)$ -АЛГОРИТМА ПОЛЛАРДА

А. С. Климина

Приведены критерии выбора параметров $(p - 1)$ -алгоритма Полларда и рассмотрен метод его оптимизации.

Ключевые слова: $(p - 1)$ -алгоритм Полларда, факторизация чисел.

Рассмотрим $(p - 1)$ -алгоритм Полларда факторизации числа N [1]. Алгоритм состоит из следующих шагов.

Шаг 1. Выбираем число k .

Шаг 2. Выбираем произвольное a , $1 < a < N$.

Шаг 3. Вычисляем $d = (a, N)$. Если $d > 1$, получили нетривиальный делитель N .

Шаг 4. Вычисляем $D = (a^k - 1, N)$. Если $1 < D < N$, то D — нетривиальный делитель N . Если $D = 1$, возвращаемся к шагу 1, а при $D = N$ — к шагу 2.

Вопрос оптимального выбора параметра k не исследован до настоящего времени с нужной полнотой. Автором реализованы следующие подходы к выбору числа k : k — произведение нескольких случайных чисел; k — факториал некоторого числа, k — произведение степеней простых чисел; k — наименьшее общее кратное нескольких чисел. После анализа работы программы выбран третий метод, поскольку он работает быстрее остальных и даёт больше удачных разложений [2].

Зафиксируем $a = 2$ и рассмотрим работу $(p - 1)$ -алгоритма Полларда при следующих допущениях:

- 1) k — произведение нескольких первых простых чисел в заданной степени;
- 2) исследуемое число N представляет собой произведение двух простых множителей, p и q .

Пусть $D = (a^k - 1, N)$; $O_p(a)$, $O_q(a)$ — показатели числа a по модулям p и q соответственно. Возможны три случая:

- 1) k кратно ровно одному из чисел $O_p(a)$, $O_q(a)$. В этом случае получим $1 < D < N$, и нетривиальный делитель D найден;
- 2) k не кратно ни одному из $O_p(a)$, $O_q(a)$. В этом случае $D = 1$;
- 3) k кратно и $O_p(a)$, и $O_q(a)$. В этом случае $D = N$.

Таким образом, для удачного нахождения делителя N нужно выбрать параметр k не слишком большим и не слишком маленьким. Заметим, что k кратно $O_p(a)$ для любого a , если k кратно $p - 1$. Это, в свою очередь, гарантированно выполняется, если k является произведением всех простых чисел $p \leq \sqrt{N}/2$ в степенях $\log_p(\sqrt{N}/2)$. Такой выбор k хорошо работает для сравнительно небольших чисел N , однако при увеличении числа N становится неприемлемым.

Предлагается следующий алгоритм выбора k в процессе работы программы.

Пусть $k = p_1^\alpha p_2^\alpha \dots p_l^\alpha$, где $p_1, \dots, p_l$ — первые l простых, α — некоторая константа. Будем рассматривать изменение k только за счёт изменения количества простых l . Идея выбора l состоит в том, чтобы выбрать его не слишком большим и не слишком малым, исходя из результатов работы программы (при слишком малом значе-