

AKS алгоритм проверки числа на простоту

Рубаненко Евгений

2017

Аннотация

В данной работе рассматривается тест Агравала - Каяла - Саксены проверки числа на простоту. Алгоритм работает за полиномиальное время. Приведено доказательство корректности и сравнение с другими алгоритмами проверки числа на простоту.

1 Введение

Долгое время считалось, что изучение простых чисел - пример "чистой" математики. Но в 70-ых годах XX века выяснилось, что простые числа могут быть использованы при создании криптографических алгоритмов. Это послужило толчком в развитии данной области. Для поиска простых чисел существует множество алгоритмов: простых и сложных. Но только в 2002 году был предложен алгоритм, который ответил на вопрос принадлежности задачи распознавания простоты классу P. Основное свойство теста AKS заключается в том, что он одновременно универсален (то есть может использоваться для проверки простоты любых чисел), полиномиален, детерминирован (что гарантирует получение уникального предопределенного результата) и безусловен (то есть корректность алгоритма не зависит от каких-либо недоказанных гипотез), предыдущие алгоритмы обладали лишь тремя из этих четырех свойств.

2 Идея

Идея алгоритма основана на следующей лемме.

Лемма 2.1. Пусть $a \in \mathbb{Z}$, $n \in \mathbb{N}$ и $(a, n) = 1$. Тогда n простое тогда и только тогда, когда

$$(X + a)^n = X^n + a \pmod{n} \quad (1)$$

Доказательство Леммы 2.1. Посмотрим на коэффициент перед X^i , $i \in \{0, \dots, n-1\}$ в многочлене $((X + a)^n - (X^n + a))$. Он равен $\binom{n}{i}a^{n-i}$. Тогда, если n простое, то $\binom{n}{i} = 0 \pmod{n}$ и сравнение (1) верно. Если n составное, то обозначим q - простой делитель n , входящий в его разложение на простые в степени k . Тогда $q^k \nmid \binom{n}{q}$ и $(q, a^{n-q}) = 1$, откуда получаем, что коэффициент при X^q не равен нулю. Но тогда многочлен $((X + a)^n - (X^n + a))$ не равен тождественно нулю, что завершает доказательство леммы. \square

Тогда можно придумать следующий тривиальный алгоритм: выбрать a и проверить (1). Проблема заключается в том, что он не эффективен - в худшем случае придется вычислить n коэффициентов в левой части (1).

Идея теста Агравала - Каяла - Саксены заключается в том, чтобы проверять следующее соотношение

$$(X + a)^n = X^n + a \pmod{X^r - 1, n}, \quad (2)$$

где r - специально выбранное число. Теперь проблема заключается в том, что соотношению (2) могут удовлетворять не только простые n . Далее будет показано, что можно проверить дополнительные условия, из которых будет следовать, что n простое.

3 Используемые обозначения

Большинство используемых обозначений являются общеизвестными. Дополнительную информацию можно найти в [1].

В работе используется символ $O^\sim(t(n))$, что есть $O(t(n) \cdot \text{poly}(\log t(n)))$.

4 Алгоритм

Data: n : integer

Result: True, если n простое, False - иначе

if $n = a^b$, где $a \in \mathbb{N}$, $b > 1$ **then**

 return False;

else

$r := \min\{r \mid o_r(n) > \log^2 n\}$;

if $1 < (a, n) < n$, для какого-то $a \leq r$ **then**

 return False;

else

if $n \leq r$ **then**

 return True;

else

for $a := 1$ to $\lfloor \sqrt{\phi(r)} \log n \rfloor$ **do**

if $((X + a)^n \neq X^n + a \pmod{X^r - 1, n})$ **then**

 return False;

end

end

 return True;

end

end

end

Algorithm 1: AKS алгоритм

5 Доказательство корректности

6 Анализ временной сложности алгоритма

Теорема 6.1. Алгоритм определяет простоту числа за время $O^{\sim}(\log^{\frac{21}{2}} n)$.

Лемма 6.1. Первый шаг алгоритма работает за время $O^{\sim}(\log^3 n)$.

Доказательство Леммы 6.1. На первом шаге проверяется, что $n \neq a^b$. Для этого надо перебрать $O(\log n)$ вариантов для a . Для конкретного a с помощью бинарного поиска проверяется, что не существует подходящего b . Перебор b требует $O(\log n)$ времени, а вычисление каждого числа вида a^b - $O^{\sim}(\log n)$. Тогда общая сложность первого шага составит $O^{\sim}(\log^3 n)$. \square

Лемма 6.2. Второй шаг алгоритма работает за время $O^{\sim}(\log^7 n)$.

Доказательство Леммы 6.2. На втором шаге алгоритма находится такое r , что $o_r(n) > \log^2 n$. Это можно сделать следующим образом: в цикле по r будем проверять, что $n^k \neq 1 \pmod r$ для всех $k \leq \log^2 n$. Для конкретного r потребуется не больше $O(\log^2 n)$ умножений по модулю r , откуда сложность одной итерации - $O^{\sim}(\log^2 n \log r)$. Согласно лемме *, необходимое r найдется, причем перебрать придется всего $O(\log^5 n)$ значений. Тогда общая сложность второго шага составит $O^{\sim}(\log^7 n)$. \square

Лемма 6.3. Третий шаг алгоритма работает за время $O(\log^6 n)$.

Доказательство Леммы 6.3. Третий шаг алгоритма - цикл из r итераций. На каждой итерации вычисляется НОД двух чисел, что требует $O(\log n)$ времени. Тогда общая сложность третьего шага составит $O(r \log n) = O(\log^6 n)$. \square

Лемма 6.4. Пятый шаг алгоритма работает за время $O^{\sim}(\log^{\frac{21}{2}} n)$.

Доказательство Леммы 6.4. Пятый шаг алгоритма - цикл из $\lfloor \sqrt{\phi(r)} \log n \rfloor$ итераций. На каждой итерации полином степени r возводится в степень n (что требует $O(\log n)$ времени); его коэффициенты можно оценить как $O(\log n)$. Таким образом, каждая итерация требует $O^{\sim}(r \log^2 n)$ времени. Тогда общая сложность пятого шага составит

$$O^{\sim}(r \sqrt{\phi(r)} \log^3 n) = O^{\sim}(r^{\frac{3}{2}} \log^3 n) = O^{\sim}(\log^{\frac{21}{2}} n)$$

□

Доказательство Теоремы 6.1. Так как четвертый шаг алгоритма выполняется за $O(\log n)$, то из Лемм 1 - 4 следует, что временная сложность алгоритма составляет $O^{\sim}(\log^{\frac{21}{2}} n)$. □

7 Сравнение с другими алгоритмами

8 Источники информации

[1] Э.Б. Винберг, Курс алгебры (2011)