

# 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

# 2 Идея

# 3 Обозначения

# 4 Алгоритм

```
Data: n: integer
Result: True, если n простое, False - иначе
if  $n = a^b$ , где  $a \in N$ ,  $b > 1$  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
end
```

Algorithm 1: AKS алгоритм

# 5 Доказательство корректности

# 6 Анализ временной сложности алгоритма

**Теорема 1.** Алгоритм определяет простоту числа за время  $O^{\sim}(\log^{\frac{21}{2}} n)$ .

**Лемма 1.** Первый шаг алгоритма работает за время  $O^{\sim}(\log^3 n)$ .

**Доказательство леммы 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 2.** Второй шаг алгоритма работает за время  $O^{\sim}(\log^7 n)$ .

**Доказательство леммы 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 3.** Третий шаг алгоритма работает за время  $O(\log^6 n)$ .

**Доказательство леммы 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 4.** Пятый шаг алгоритма работает за время  $O^\sim(\log^{\frac{21}{2}} n)$ .

**Доказательство леммы 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

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

$\square$

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

## 7 Сравнение с другими алгоритмами

## 8 Литература