МИНОБРНАУКИ РОССИИ
ФГБОУ ВО «СГУ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

АЛГОРИТМЫ АЛГЕБРЫ И ТЕОРИИ ЧИСЕЛ

ЛАБОРАТОРНАЯ РАБОТА №8

студента 4 курса 431 группы
направления 10.05.01 — Компьютерная безопасность
факультета КНиИТ
Никитина Арсения Владимировича

Проверил
доцент ____________ А. С. Гераськин

Саратов 2022

СОДЕРЖАНИЕ

1 Задание лабораторной работы

Осуществить проверку чисел на простоту с помощью полиномиального теста распознавания простоты.

2 Теоретическая часть

Формулировка теста Агравала-Каяла-Саксены

Если существует $r \in \mathbb{Z}$ такое, что $o_r(n) > \log^2 n$ и $\forall a$ от 1 до $\left\lfloor \sqrt{\varphi(r)} \log(n) \right\rfloor$ выполняется сравнение $(x+a)^n \equiv (x^n + a) \pmod{x^r - 1,\ n}$, то n — либо простое число, либо степень простого числа.

$o_r(n)$ обозначает показатель числа n по модулю r, $\log$ — двоичный логарифм и $\varphi(\cdot)$ — функция Эйлера.

Сравнение по двум модулям вида $a(x) \equiv b(x) \pmod{h(x),\ n}$ для многочленов $a(x),\ b(x) \in \mathbb{Z}[x]$ означает, что существует $g(x) \in \mathbb{Z}[x]$ такой, что все коэффициенты многочлена $a(x) - b(x) - g(x)h(x)$ кратны n, где $\mathbb{Z}[x]$ — кольцо многочленов от x над целыми числами.

Оновная идея алгоритма

Основной идеей алгоритма является обобщение малой теоремы Ферма на многочлены, утверждающее, что для всех $a \in \mathbb{Z}_n^*$ (где кольцо $\mathbb{Z}_n$ взято без обратных элементов по умножению и нулевого элемента) $n \in \mathbb{N}$, n — простое тогда и только тогда, когда $(x+a)^n \equiv (x^n + a) \pmod{n}$.

На проверку этого выражения требуется время, оцениваемое в $\Omega(n)$, поскольку в худшем случае следует оценить n коэффициентов в левой части. Для сокращения числа коэффициентов и сложности вычислений было выбрано такое r, чтобы использовать в качестве теста на простоту выражение: $(x+a)^n \equiv (x^n + a) \pmod{x^r - 1,\ n}$, которое получается делением обеих частей исходного выражения на $x^r - 1$.

Здесь количество подлежащих проверке значений a и значение r уже ограничены многочленом от $\log n$.

В этом случае вместо факторкольца $\mathbb{F}_p[x]/\langle x^r - 1\rangle$ рассматривается поле $F = \mathbb{F}_p[x]/\langle h\rangle$, где $h = h(x)$ — неприводимый делитель $x^r - 1$ над конечным полем $\mathbb{F}_p$, отличный от $x - 1$. Оценивается число многочленов этого поля, для которых выполняется сравнение: $(x+a)^n \equiv (x^n + a) \pmod{x^r - 1,\ n}$.

Алгоритм теста

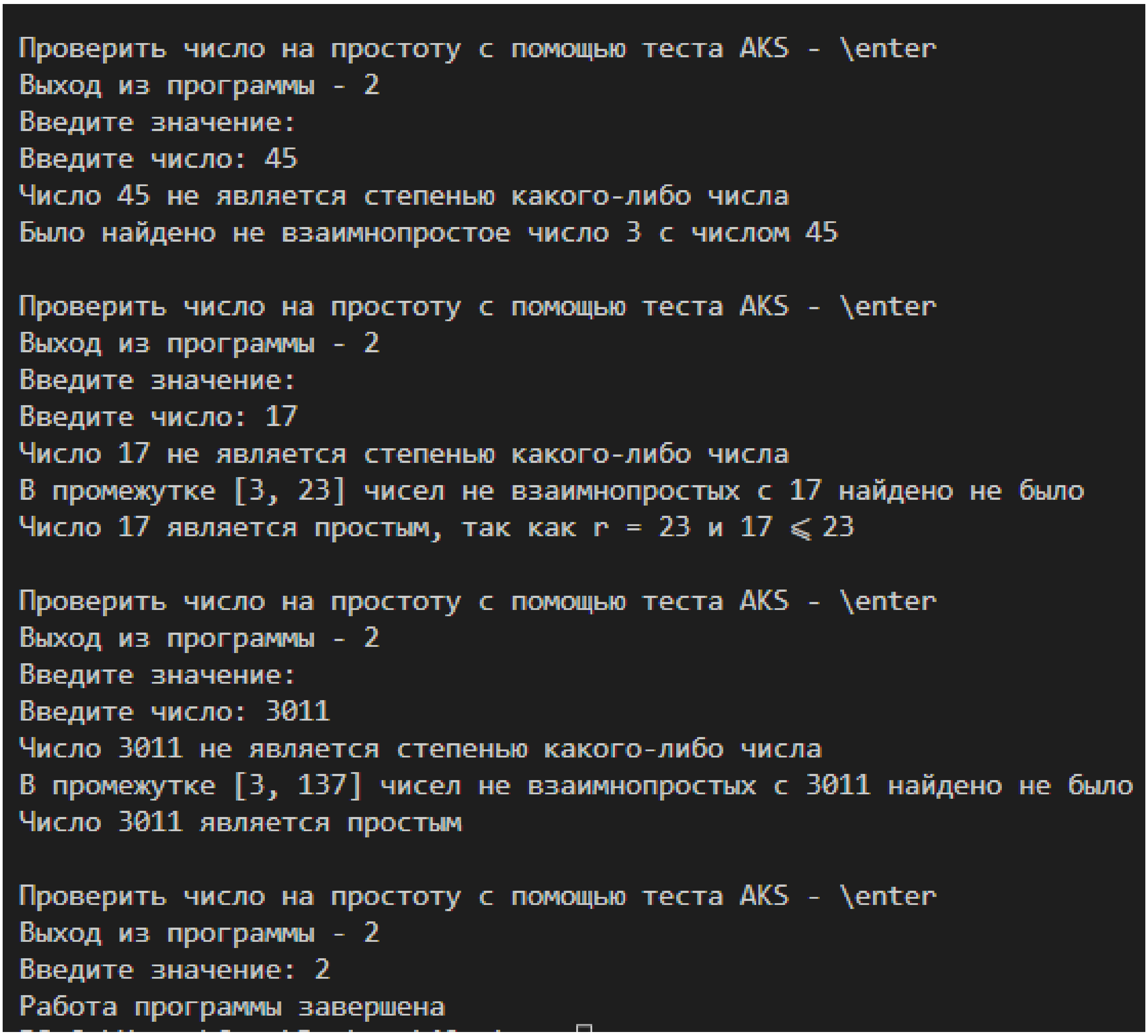
Ввод: целое число $n > 1$.

1. Если $n = a^b$ для целых чисел $a > 1$ и $b > 1$, вернуть «составное».

2. Найдем наименьшее r, такое что $o_r(n) > (\log_2(n))^2$.
3. Если $1 <$НОД$(a,\ n) < n$ для некоторого $a \leqslant r$, вернуть «составное».
4. Если $n \leqslant r$, вернуть «простое».
5. Если для всех a от 1 до $\left\lfloor\sqrt{\varphi(r)}\log(n)\right\rfloor$ верно, что $(x+a)^n \equiv x^n + a \pmod{x^r - 1,\ n}$, вернуть «простое».
6. Иначе вернуть «составное».

3 Практическая часть

3.1 Пример работы алгоритма


```
Проверить число на простоту с помощью теста AKS - \enter
Выход из программы - 2
Введите значение:
Введите число: 45
Число 45 не является степенью какого-либо числа
Было найдено не взаимнопростое число 3 с числом 45

Проверить число на простоту с помощью теста AKS - \enter
Выход из программы - 2
Введите значение:
Введите число: 17
Число 17 не является степенью какого-либо числа
В промежутке [3, 23] чисел не взаимнопростых с 17 найдено не было
Число 17 является простым, так как r = 23 и 17 ⩽ 23

Проверить число на простоту с помощью теста AKS - \enter
Выход из программы - 2
Введите значение:
Введите число: 3011
Число 3011 не является степенью какого-либо числа
В промежутке [3, 137] чисел не взаимнопростых с 3011 найдено не было
Число 3011 является простым

Проверить число на простоту с помощью теста AKS - \enter
Выход из программы - 2
Введите значение: 2
Работа программы завершена
```

Рисунок 1

3.2 Код программы, реализующей рассмотренный алгоритм

```
import math
import numpy


def factor(p):

    d, factors, unique_factors = 2, [], set()

    while d*d <= p:
```

```

        while (p % d) == 0:
            factors.append(d)
            unique_factors.add(d)
            p //= d

        d += 1

    if p > 1:
       factors.append(p)
       unique_factors.add(p)

    return list(unique_factors), [factors.count(i) for i in unique_factors]


def pascal_triangle(n):

    a = [1]

    for _ in range(n):
        b = [1]
        b += [a[k] + a[k + 1] for k in range(len(a) - 1)] + [1]
        a = b

    return [[elem, i] for i, elem in enumerate(a)]


def get_phi(p):

    factors, powers = factor(p)

    if len(factors) == 1:
        return p - 1
    else:
        res = [factors[i] ** powers[i] - factors[i] ** (powers[i] - 1) for i
        in range(len(powers))]

        return numpy.prod(res)


def gcd(a, b):
```

```
    if a == 0:
        return b
    else:
        return gcd(b % a, a)


def is_power(number):

    if not number % 2:
        return True, 2, 2

    for i in range(3, math.ceil(math.sqrt(number)), 2):
        i_new = i
        counter = 1
        while i_new < number:
            i_new *= i
            counter += 1
            if i_new == number:
                return True, i, counter
    return False, None, None


def fast_power(a, n):
    return (1 if n == 0
            else fast_power(a * a, n // 2) if n % 2 == 0
            else a * fast_power(a, n - 1))


def find_r(n, n_log):

    right = n_log ** 2

    first_value = 2

    while True:

        if (l := gcd(first_value, n)) != 1 and first_value % n:
            return False, l

        elems = [n % first_value]
        elem_save = elem = n % first_value
```

```
        order = 1

        while True:

            elem = (elem * elem_save) % first_value

            if elem not in elems:
                elems.append(elem)
                order += 1
            else:
                break

        if order > right:

            return True, first_value

        else:
            first_value += 1


def aks(n):

    result, number, power = is_power(n)

    if result:
        if number == 2:
            print(f'Число {n} четное, а, значит, не простое')
        else:
            print(f'Число {n} является степенью другого числа '\
                f'({n} = {number} ^ {power})')
        return


    print(f'Число {n} не является степенью какого-либо числа')

    n_log = math.log2(n)

    res, r = find_r(n, n_log)

    if not res :
        print(f'Было найдено не взаимнопростое число {r} с числом {n}')
```

```
            return
        else:
            print(f'В промежутке [3, {r}] чисел не взаимнопростых с {n} '\
                'найдено не было')

        if n <= r:
            print(f'Число {n} является простым, так как r = {r} и {n} \u2A7D {r}')
            return

        pascal = pascal_triangle(n)

        for a in range(1, math.floor(math.sqrt(get_phi(r)) * n_log) + 1):

            current_pascal = [(coef * fast_power(a, pow_a)) % n == 0 for \
                [coef, pow_a] in pascal][1:-1]

            if not all(current_pascal):
                print(f'Число {n} не является простым')
                return

        print(f'Число {n} является простым')



def main():

    while True:

        print('\nПроверить число на простоту с помощью теста AKS - \enter')
        print('Выход из программы - 2')

        try:
            value = int(input('Введите значение: '))

        except ValueError:
            value = 1

        if value == 1:

            n = int(input('Введите число: '))
```

```
            aks(n)

        elif value == 2:
            print('Работа программы завершена')
            return


if __name__ == "__main__":
    main()
```