МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ДАГЕСТАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

# Методическое пособие
# по курсу «Теоретико числовые методы криптографии»

Махачкала
Издательство ДГУ
2022

Методическое пособие предназначено для студентов очного отделения информационной безопасности ФИ и ИТ Дагестанского государственного университета по курсу «Теоретико числовые методы криптографии».

В лабораторный практикум включены описания лабораторных работ по ТЧМК. К каждой работе дается краткое описание, приводятся требования к выполнению работы, даются упражнения и варианты заданий, которые необходимо выполнить.

Составители: Муртузалиева *А.А*. – ст. преподаватель

Рецензенты:

Лабораторная работа №1

Алгоритмы Евклида

**Алгоритм 1. Алгоритм Евклида.**

Вход. Целые числа a, b; $0 < b < a$.

Выход. d = НОД (a,b).

1. r0 := a, r1 := b, i := 1.
2. Найти остаток r[i+1] от деления r[i–1] на r[i].
3. Если r[i+1] = 0, то d := r[i]. В противном случае положить i := i + 1 и вернуться на шаг 2.
4. Результат: d.

Сложность алгоритма Евклида равна $O(log^2 \boldsymbol{a})$

**Алгоритм 2. Расширенный алгоритм Евклида.**

*Вход.* Целые числа ***a, b;*** $0 < \boldsymbol{b} \leq \boldsymbol{a}$.

*Выход.* $\boldsymbol{d}$=НОД($\boldsymbol{a}$, $\boldsymbol{b}$)**;** такие целые числа ***x,y,*** что $\boldsymbol{ax + by = d}$.

1. $r_0 := a$, $r_1 := b$, $x_0 := 1$, $x_1 := 0$, $y_0 := 0$, $y_1 := 1$, $i := 1$.
2. Разделить с остатком $r_{i-1}$ на $r_i$ : $r_{i-1} = q_i r_i + r_{i+1}$.
3. Если $r_{i+1} = 0$, то $d := r_i$, $x := x_i$ , $y := y_i$. В противном случае положить $x_{i+1} := x_{i-1} - q_i x_i$, $y_{i+1} := y_{i-1} - q_i y_i$ , $i := i+1$ и вернуться на шаг 2.
4. Результат: ***d, x, y.***

Сложность этого алгоритма равна $O(\log^2 a)$.

**Алгоритм 3. Бинарный алгоритм Евклида [3].**

Вход. Целые числа a, b; $0 < b \leq a$.

Выход. d = НОД(a,b).

1. g :=1.
2. Пока оба числа a и b четные, выполнять a:= a/2 , b := b/2, g := 2g до получения хотя бы одного нечетного значения a или b.
3. u := a, v := b.
4. Пока $u \neq 0$, выполнять следующие действия.

   4.1. Пока u четное, u := u/2.

   4.2. Пока v четное, v := v/2.

4.3. Если $u \geq v$, то $u := u - v$. В противном случае $v := v - u$.

5. $d := gv$.

6. Результат: d.

Сложность этого алгоритма равна $O(\log^2 a)$.

**Алгоритм 4. Расширенный бинарный алгоритм Евклида**

*Вход**,*** Целые числа ***a, b;*** $0 < b \leq a$.

*Выход*, ***d*** =НОД(***a***, ***b***); такие целые числа ***x***, ***y*** (множители Безу), что $ax + by = d$.

1. ***g*** :=1.
2. Пока оба числа ***a и b*** четные, выполнять $a := a/2$, $b := b/2$, $g := 2g$ до получения хотя бы одного нечетного значения ***a*** или ***b.***
3. $u := a$, $v := b$, $A := 1$, $B := 0$, $C := 0$, $D := 1$.
4. Пока $u \neq 0$, выполнять следующие действия.

4.1. Пока ***u*** четное:

1. $u \leftarrow u/2$
2. Если оба числа ***A*** и ***B*** четные, то $A := A/2$, $B := B/2$

В противном случае положить $A := (A+b)/2$ , $B := (B-a)/2$

4.2. Пока ***v*** четное:

4.2.1. $v := v/2$

4.2.2. Если оба числа C и ***D*** четные, то $C := C/2$, $D := D/2$, В противном случае $C := (C+b)/2$ , $D := (D-a)/2$

4.3. Если $u \geq v$, то $u := u - v$, $A := A - C$, $B := B - D$.

В противном случае $v := v - u$, $C := C - A$, $D := D - B$.

5. $d := gv$, $x := C$, $y := D$.
6. Результат: ***d, x, y.***

Сложность этого алгоритма равна $O(\log^2 a)$.

Функция в **python** - объект, принимающий аргументы и возвращающий значение.

Любая объявленная функция может быть передана в другую функцию в качестве аргумента. Поскольку каждая функция является объектом, то передается ссылка на эту функцию. Функция, которая получает ссылку может по этой ссылке вызывать другую функцию соблюдая правильное задание количества и типа параметров.

```
from math import exp, sin, pi, log
def metod_tr(fun, a,b,n):
    s=(fun(a)+fun(b))/2
    h=(b-a)/n
    for k in range(1,n):
        s+=fun(a+k*h)
    return s*h/3
def f1(x):
    return (x**3)/(3+x)
def f2(x):
    return (exp(x)-1)**0.5
def f3(x):
    return (sin(x))**2
print(metod_tr(f1,1,3,72))
print(metod_tr(f3,0,pi/2,22))
print(metod_tr(f2,0,log(2),104))
```

**Задание**

Составить программу, в которой будут реализованы все четыре алгоритма Евклида. Пользователь вводит любое количество целых чисел в одной строке и в следующей строке вводит название или номер алгоритма, по которому надо вычислить наибольший общий делитель.

Программа должна содержать пять функций: в четырех функциях реализованы алгоритмы Евклида, в пятой функции реализован алгоритм нахождения НОД(a1, a2, …, an), где n >=2, по алгоритму, который задает пользователь.

Реализованные функции должны корректно работать на таких тестах:

НОД(1, 10) = 1

НОД (5, 10) = 5

НОД d(24, 24) = 24

**Лабораторная работа№2**

**Возведение в степень по модулю**

**Алгоритмы быстрого возведения в степень по модулю** широко используются в различных криптосистемах, для ускорения вычислительных операций с большими числами.

Данный алгоритм основывается на том факте, что для заданных $a$ и $b$ следующие 2 уравнения эквивалентны:

C = (a*b) (mod m)

C = a (b (mod m))(mod m)

Алгоритм следующий:

1. Пусть $c = 1$, $n' = 0$.
2. Увеличим $n'$ на 1.
3. Установим c=(a*c) (mod p).
4. Если $n' <$ n, возвращаемся к шагу 2. В противном случае, $c$ содержит правильный ответ .

**Пример**

$5^4$ mod 13

5 mod 13≡5

5*5 mod 13≡12

12*5 mod 13≡8

8*5 mod 13≡1

**Алгоритмы быстрого возведения в степень**

Алгоритмы быстрого возведения в степень предназначены для возведения числа в натуральную степень за меньшее число умножений, чем это требуется в определении степени.

***Наивный алгоритм***

$$a^n = \begin{cases} 1 \text{ если } n = 0 \\ a * a^{n-1} \text{ если } n > 1 \end{cases}$$

***Быстрый рекурсивный алгоритм***

$$a^n = \begin{cases} (a^2)^{n/2} \text{ если } n \bmod 2 = 0 \\ a * (a^2)^{n/2} \text{ если } n \bmod 2 \neq 0 \end{cases}$$

***Бинарный алгоритм***

***Схема «слева направо»***

Основным алгоритмом быстрого возведения в степень является схема «слева направо». Она получила своё название вследствие того, что биты показателя степени просматриваются слева направо, то есть от старшего к младшему.

Пусть количество цифр в двоичном представлении числа n есть len(n)=t: $(n_{t-1} \dots n_0)_2$

Для каждого k∈[0…t−1] обозначим $m_k = (n_{t-1} \dots n_k)_2$

Если k=0, то $m_k$=n и поэтому $a^{m_k} = a^n$

Если 0 < k ≤ t−1, то

$$m_{k-1} = (n_{t-1} \dots n_k\, n_{k-1})_2 = 2 * (n_{t-1} \dots n_k)_2 + n_{k-1} = 2 * m_k + n_{k-1}$$

и поэтому

$$a^{m_{k-1}} = (a^{m_k})^2 * a^{n_{k-1}} = \begin{cases} (a^{m_k})^2 \quad \text{ если } n_{k-1} = 0 \\ a * (a^{m_k})^2 \text{ если } n_{k-1} = 1 \end{cases}$$

***Алгоритм***

1. Представить показатель степени n в двоичном виде

2. Если $\mathbf{m_i = 1}$, то текущий результат возводится в квадрат и затем умножается на x. Если $\mathbf{m_i = 0}$, то текущий результат просто возводится в квадрат. Индекс **i** изменяется от **k-1** до **0**.

Таким образом, алгоритм быстрого возведения в степень сводится к мультипликативному аналогу схемы Горнера:

$$\left\{ \begin{array}{c} s_1 = x \\ s_{i+1} = s_i^2 \cdot x^{m_{k-i}} \\ i = 1, 2, \dots, k \end{array} \right\}.$$

***Схема «справа налево»***

В данной схеме, в отличие от схемы «слева направо», биты показателя степени просматриваются от младшего к старшему.

Пусть количество цифр в двоичном представлении числа n есть len(n)=t: $(n_{t-1} \dots n_0)_2$

Для каждого k∈[0…t−1] обозначим $m_k = (n_{t-1} \dots n_k)_2$

Если k=t-1, то $m_k$=n и поэтому $a^{m_k} = a^n$

Если 0 ≤ k < t−1, то

$$m_{k+1} = (n_{k+1}n_k \dots n_0)_2 = 2^{k+1}n_{k+1} + (n_k \dots n_0)_2 = 2^{k+1}n_{k-1} + m_k$$

и поэтому

$$a^{m_{k+1}} = \left(a^{2^{k+1}}\right)^{n_{k+1}} * a^{m_k} = \begin{cases} a^{m_k} & \text{если } n_{k+1} = 0 \\ a^{2^{k+1}} a^{m_k} & \text{если } n_{k+1} = 1 \end{cases}$$

Последовательность действий при реализации данного алгоритма.

1. Представить показатель степени ***n*** в двоичном виде.
2. Положить вспомогательную переменную ***z*** равной числу ***x***.
1. Если $\mathbf{m_i}$**,** = **1** то текущий результат умножается на **z,** а само число **z** возводится в квадрат. Если $\mathbf{m_i}$ = **0**, то требуется только возвести **z** в квадрат. При этом индекс ***i,*** в отличие от схемы слева направо, изменяется от **0** до ***k*-1** включительно.

**Задание**

Составить программу, в которой будут реализованы два алгоритма быстрого возведения в степень (схема «слева направо» и схема «справа налево») и алгоритм быстрого возведения в степень по модулю. Вычисление возведение в степень оформить в виде функции.

## Лабораторная работа№3

## Символы Лежандра и Якоби

Для того чтобы определить является ли число квадратным вычетом или квадратным невычетом в поле простого числа используют символ Лежандра.

Значение символа Лежандра для целого числа $a$ и простого числа $p$ отвечает на вопрос, является число $a$ квадратичным вычетом или квадратичным невычетом по модулю $p$.

***Алгоритм вычисления символа Лежандра.***

1. Если a = 1, то L(a, p) = 1.
2. Если число a четное, то

$$\mathrm{L}(\mathrm{a}, \mathrm{p}) = \mathrm{L}\left(\frac{a}{2}, p\right) * (-1)^{\frac{(p^2-1)}{8}}$$

3. Если число a — нечетное и a != 1, то

$$\mathrm{L}(\mathrm{a}, \mathrm{p}) = \mathrm{L}(p \bmod a, a) * (-1)^{\frac{(a-1)*(p-1)}{4}}$$

*Символ Якоби*, который обозначается как J(a, n) — это обобщение символа Лежандра на составные модули. Это функция, определенная для всех целых чисел a и нечетных целых чисел n. Символ Якоби может принимать значения 0, 1 и –1.

**Символ Якоби** можно задать следующим образом.

1. Символ Якоби определен только для нечетных чисел n.
2. J(0, n) = 0.
3. Если n – простое число, то J(0, n) = 0, если a делится на n.
4. Если n – простое число, то J(0, n) = 1, если a — квадратичный вычет по модулю n.
5. Если n – простое число, то J(0, n) = –1, если a — квадратичный невычет по модулю n.
6. Если n – составное число, то J(a, n) = J(a, $p_1$)*...*J(a, $p_m$), где $p_1$,..., $p_m$ — разложение n на простые множители.

Алгоритм 2.1. Вычисление символа Якоби.

*Вход*. Нечетное целое число $n \geq 3$, целое число $a$, $0 \leq a < n$.

*Выход*. Символ Якоби $\left(\frac{a}{n}\right)$.

1. Положить $g \leftarrow 1$.
2. При $a = 0$ результат: 0.
3. При $a = 1$ результат: $g$.
4. Представить $a$ в виде $a = 2^k a_1$, где число $a_1$ нечетное.
5. При четном $k$ положить $s \leftarrow 1$. При нечетном $k$ положить $s \leftarrow 1$, если $n \equiv \pm 1 \pmod 8$; положить $s \leftarrow -1$, если $n \equiv \pm 3 \pmod 8$.
6. При $a_1 = 1$ результат: $g \cdot s$.

7. Если $n \equiv 3 \pmod 4$ и $a_1 \equiv 3 \pmod 4$, то $s \leftarrow -s$.
8. Положить $a \leftarrow n \pmod{a_1}$, $n \leftarrow a_1$, $g \leftarrow g \cdot s$ и вернуться на шаг 2.

Сложность алгоритма равна $O(\log^2 n)$.

***Алгоритм вычисления символа Якоби.***

1. J(1, n) = 1.

2. J(a*b, n) = J(a, n)*J(b, n).

3. J(2, n) = 1, если (n^2 – 1)/8 является четным, и –1 в противном случае.

4. J(a, n) = J((a mod n), n).

5. J(a, b1*b2) = J(a, b1)J(a, b2).

6. Если gcd(a, b) = 1 и, кроме того, числа a и b являются нечетными, то

6.1. J(a, b) = J(b, a), если (a – 1)*(b – 1)/4 является четным числом.

6.2. J(a, b) = –J(b, a), если (a – 1)*(b – 1)/4 является нечетным числом.

Если n — простое число, то символ Якоби эквивалентен символу Лежандра.

Символ Якоби нельзя использовать для проверки, является ли число a квадратичным вычетом по модулю n (кроме случая, когда число n — простое). Если J(a, n) = 1 и n — составное число, то число a не всегда является квадратичным вычетом:

J(7, 143) = J(7, 11) * J(7, 13) = (–1)*(–1) = 1,

хотя не существует целых чисел x таких, что x2 ≡ 7 (mod 143).

***Задание***

Составить программу с функциями, в которых будут реализованы алгоритмы вычисления символов Лежандра и Якоби.

## Лабораторная работа№4

## Вероятностные тесты простоты

***Тест Ферма***

1) Ввод числа N
2) число a– выбирается случайно из диапазона [1 , N-1].
3) Вычислить НОД (a, N)
4) Если НОД (a, N)>1

, то N — составное.

5) При НОД (a, N)=1

проверить

$$a^{N-1} \equiv 1 \mod N.$$

Если не выполняется, то — составное. Иначе – неизвестно, то есть или простое или составное.

Если алгоритм выдал ответ «неизвестно», то можно повторять тест для следующего числа a.

***Тест Соловея- Штрассена***

***Алгоритм Соловея–Штрассена***

1. Введите число p.
2. Выберите случайное число a, меньшее p.
3. Если gcd(a, p) != 1, то число p – составное и тест можно не продолжать.

4. Вычислите j = a^((p–1)/2) mod p.

5. Вычислите символ Якоби J(a, p).

6. Если j != J(a, p), то число p точно не является простым.

7. Если j = J(a, p), то вероятность того, что число p не является простым, не превышает 50%.

Число a, которое не указывает явно, что число p не простое, называется свидетелем. Если число p — составное, то вероятность того, что случайное число является свидетелем, составляет не менее 50%. Вероятность того, что составное число пройдет t испытаний, равняется 1/(2^t).

***Тест Миллера-Рабина***

Пусть есть нечетное число p, которое мы хотим проверить на простоту. Тогда p−1 делится на 2 и можно представить $p-1=2^n\cdot m$, где m- нечетное. Далее можно взять случайное число a от 1 до p. С помощью быстрого возведения в степень можно посчитать $a^m$ по модулю p. Если число сравнимо с -1 или 1, то a — свидетель простоты и выводим сообщение, что p –простое. Если же нет, то n раз возводим число в квадрат и проверяем, что не сравнимо ли оно с −1 по модулю p.

**Задание:**

Составить программу с функциями, в которых будут реализованы тесты: Ферма, Соловэя-Штрассена и Миллера-Рабина.

## Лабораторная работа№5

### Решение сравнения второй степени по простому модулю.

Существуют различные способы решения сравнения $x^2\equiv a(\mathrm{mod}\ p)$ в зависимости от вида модуля.

При $p\equiv 3\ (\mathrm{mod}\ 4)$ решение имеет вид $x\equiv\pm a^{m+1}(\mathrm{mod}\ p)$.

При $p\equiv 5\ (\mathrm{mod}\ 8)$ решение имеет вид $x\equiv\pm a^{m+1}2^{2m+1}(\mathrm{mod}\ p)$.

Если модуль $P\equiv 1(\mathrm{mod}\ 8)$, используйте следующий алгоритм

*Вход.* Простое число $p \neq 2$; такие целые числа $a$ и $N$, что $\left(\frac{a}{p}\right) = -\left(\frac{N}{p}\right) = 1$.

*Выход.* Решение сравнения $x^2 \equiv a \pmod p$.

1. Представить число $p$ в виде $p = 2^k \cdot h + 1$, где число $h$ нечетное.
2. Положить $a_1 \leftarrow a^{\frac{h+1}{2}} \pmod p$, $a_2 \leftarrow a^{-1} \pmod p$, $N_1 \leftarrow N^h \pmod p$, $N_2 \leftarrow 1, j \leftarrow 0$.
3. Для $i = 0, 1, \ldots, k-2$ выполнять следующие действия.
   - 3.1. Положить $b \leftarrow a_1 N_2 \pmod p$.
   - 3.2. Вычислить $c \leftarrow a_2 b^2 \pmod p$.
   - 3.3. Вычислить абсолютно наименьший вычет $d \leftarrow c^{2^{k-2-i}} \pmod p$. При $d = 1$ положить $j_i \leftarrow 0$, при $d = -1$ положить $j_i \leftarrow 1$.
   - 3.4. Положить $N_2 \leftarrow N_2 N_1^{2^i j_i} \pmod p$.
4. Результат: $\pm a_1 N_2 \pmod p$.

Сложность этого алгоритма равна $O(\log^4 p)$.

**Задание:**

Составить программу для решения сравнения второй степени по простому модулю.

## Лабораторная работа№6

## Каноническое разложение числа

### *ρ-алгоритм Полларда*

Основная статья: **Р-алгоритм Полларда**

*Сложность* $O(n^{1/4})$.

Алгоритм Полларда является вероятностным алгоритмом, позволяющим находить делитель составного числа $n$, работающим со сложностью, зависящей лишь от величины делителя, но не величины факторизуемого числа $n$. Это обуславливает удобство применимости данного алгоритма в тех случаях, когда другие алгоритмы, сложность которых зависит от $n$, становятся неэффективны[11]. Примечателен так же тем, что существует вариант реализации такого алгоритма, при котором достаточно в памяти хранить всего 3 целых числа[12].

Пример алгоритма[13]

**Шаг 1.** Выбираем небольшое число $x_0$ и строим последовательность чисел $x_n, n = 0, 1, 2, ...$, определяя каждое следующее $x_{n+1}$ по формуле: $x_{n+1} = x_n^2 - 1 \mod n$

**Шаг 2.** Одновременно на каждом шаге $i$ вычисляем наибольший общий делитель $d$ числа $n$ и всевозможных разностей $|x_i - x_j|$, где $j < i$.

**Шаг 3.** Когда будет найден $d = \gcd(n, |x_i - x_j|)$, отличный от $1$, вычисление заканчивается. Найденное $d$ является делителем $n$. Если $n/d$ не является простым числом, то процедуру можно продолжить, взяв вместо $n$ число $n/d$.