# РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

### Факультет физико-математических и естественных наук

### Кафедра прикладной информатики и теории вероятностей

# ОТЧЕТ

# ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4

### дисциплина: Математические основы защиты информации и информационной безопасности

### Студент: Пиняева Анна Андреевна

### Группа: НПИмд-01-24

### МОСКВА

### 2025

# Теоретическое введение

**1. Алгоритм Евклида**

Основан на свойстве: НОД(a,b) = НОД(b, a mod b) Процесс: Последовательное деление с остатком до получения нулевого остатка Сложность: O(log(min(a,b))) делений Преимущества: Простота реализации, минимальная память

**2. Бинарный алгоритм Евклида**

Использует свойства:

НОД(2a,2b) = 2·НОД(a,b)

НОД(2a,b) = НОД(a,b) для нечетного b

НОД(a,b) = НОД(a-b,b) для нечетных a,b

Процесс: Замена делений на сдвиги и вычитания Преимущества: Быстрее на компьютерах (операции с битами)

**3. Расширенный алгоритм Евклида**  
Дополнительно находит коэффициенты Безу: x,y такие что ax + by = НОД(a,b) Процесс: Сохраняет коэффициенты при итерациях алгоритма Евклида Применение: Решение линейных диофантовых уравнений, модульная арифметика

**3. Расширенный бинарный алгоритм Евклида**  
Комбинация: Бинарного алгоритма + расширенного Особенность: Работает с коэффициентами Безу при операциях сдвига Эффективность: Максимальная скорость для компьютерных вычислений

В задании лабораторной предлагается рассмотреть все алгоритмы. Исходный код написан на языке Julia [@doc-julia]. Код функции, осуществляющей шифрование гаммированием с конечной гаммой, представлен ниже.

# Цель работы

Изучение и реализация вычисления НОД с помощью алгоритма Еклида на языке Julia.

# Ход работы

1. Алгоритм Евклида

function euclidean\_gcd(a::Int, b::Int)  
 a, b = abs(a), abs(b)  
 while b != 0  
 a, b = b, a % b  
 end  
 return a  
end

Что происходит в функции: - Нормализация входных чисел - Цикл пока остаток не станет нулевым - Ключевой шаг алгоритма Евклида - Возвращает НОД (последний ненулевой остаток)

1. Бинарный алгоритм Евклида

function binary\_gcd(a::Int, b::Int)  
 a, b = abs(a), abs(b)  
   
 a == 0 && return b  
 b == 0 && return a  
   
 shift = 0  
 while iseven(a) && iseven(b)  
 a ÷= 2  
 b ÷= 2  
 shift += 1  
 end  
   
 while a != 0  
 while iseven(a)  
 a ÷= 2  
 end  
 while iseven(b)  
 b ÷= 2  
 end  
   
 if a >= b  
 a = a - b  
 else  
 b = b - a  
 end  
 end  
   
 return b << shift # b \* 2^shift  
end

Что происходит в функции: - Счетчик общей степени двойки (shift=0) - Удаление общих множителей 2 - Делает числа нечеными - Вычитание по свойству НОД(a,b)=НОД(a-b,b) - Восстанавливает общую степень двойки

1. Расширенный алгоритм Евклида

function extended\_euclidean(a::Int, b::Int)  
 a, b = abs(a), abs(b)  
   
 if b == 0  
 return (a, 1, 0)  
 end  
   
 x0, x1 = 1, 0  
 y0, y1 = 0, 1  
   
 while b != 0  
 q = a ÷ b  
 a, b = b, a % b  
 x0, x1 = x1, x0 - q \* x1  
 y0, y1 = y1, y0 - q \* y1  
 end  
   
 return (a, x0, y0)  
end

Что происходит в функции: - Инициализация матрицы коэффициентов - Вычисление частного - Обновление х-коэффициентов - Обновление у-коэффициентов - Возвращает тройку удовлетворяющую уравнению Безу

1. Расширенный бинарный алгоритм Евклида

function extended\_binary\_gcd(a::Int, b::Int)  
 a, b = abs(a), abs(b)  
   
 if a == 0  
 return (b, 0, 1)  
 elseif b == 0  
 return (a, 1, 0)  
 end  
   
 g = 1  
 while iseven(a) && iseven(b)  
 a ÷= 2  
 b ÷= 2  
 g \*= 2  
 end  
   
 u, v = a, b  
 A, B, C, D = 1, 0, 0, 1  
   
 while u != 0  
 while iseven(u)  
 u ÷= 2  
 if iseven(A) && iseven(B)  
 A ÷= 2  
 B ÷= 2  
 else  
 A = (A + b) ÷ 2  
 B = (B - a) ÷ 2  
 end  
 end  
 while iseven(v)  
 v ÷= 2  
 if iseven(C) && iseven(D)  
 C ÷= 2  
 D ÷= 2  
 else  
 C = (C + b) ÷ 2  
 D = (D - a) ÷ 2  
 end  
 end  
   
 if u >= v  
 u -= v  
 A -= C  
 B -= D  
 else  
 v -= u  
 C -= A  
 D -= B  
 end  
 end  
   
 d = g \* v  
 return (d, C, D)  
end

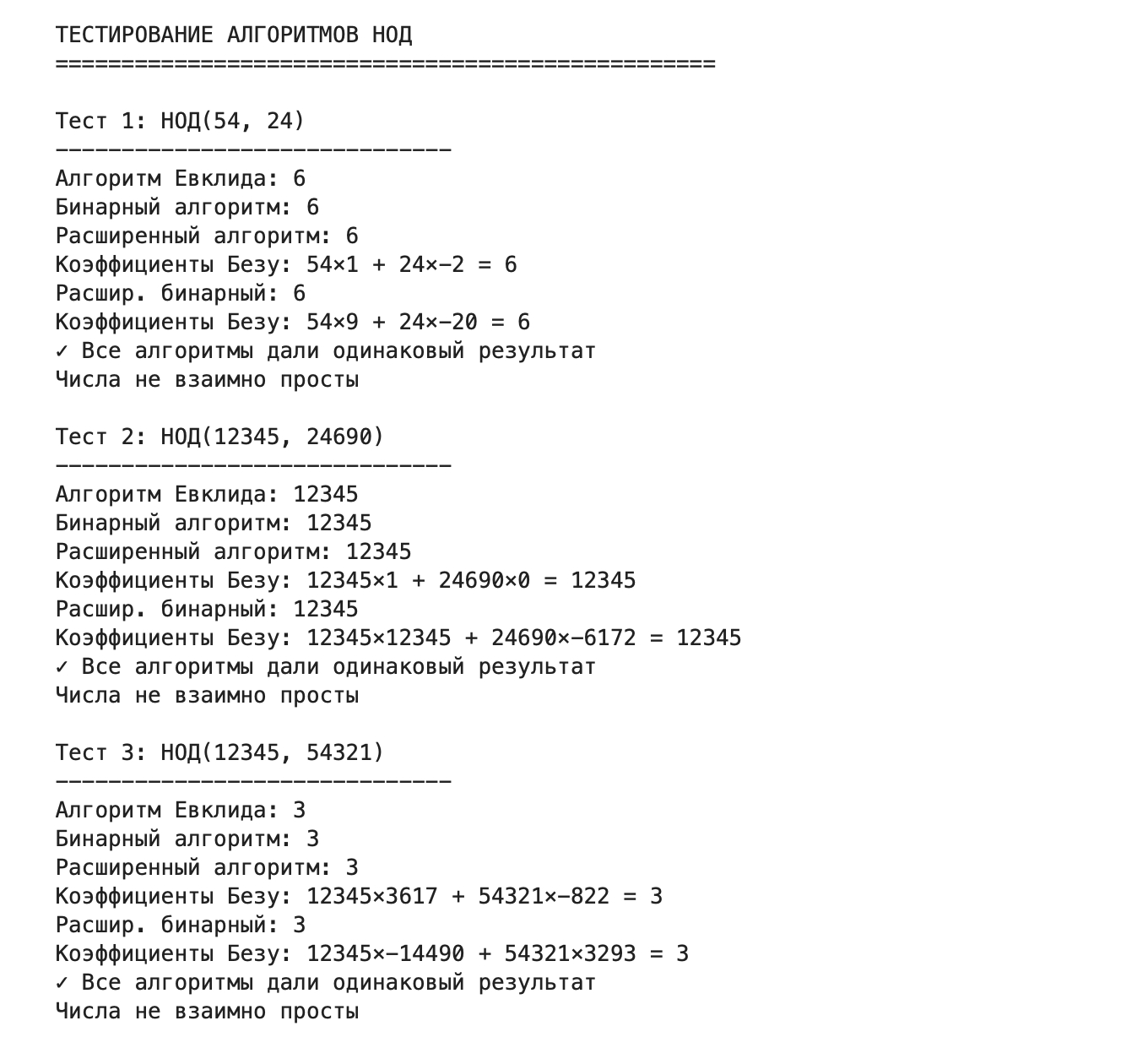
Что происходит в функции: - Накопитель общего множителя 2 - Матрица для отслеживания коэффициентов Безу - Обработка четности с коррекцией коэффициентов - Обновление матрицы при вычитании - Восстановление НОД с общим множителем

1. Вывод результатов

function test\_algorithms()  
 test\_cases = [  
 (54, 24),  
 (12345, 24690),  
 (12345, 54321),  
 (12345, 12541),  
 (91, 105),  
 (105, 154),  
 (17, 13), # Простые числа  
 (100, 25), # Одно делит другое  
 (0, 15), # Ноль  
 (15, 0), # Ноль  
 (-54, 24), # Отрицательные числа  
 (54, -24) # Отрицательные числа  
 ]  
   
 println("ТЕСТИРОВАНИЕ АЛГОРИТМОВ НОД")  
 println("="^50)  
   
 for (i, (a, b)) in enumerate(test\_cases)  
 println("\nТест $i: НОД($a, $b)")  
 println("-"^30)  
   
 # Алгоритм Евклида  
 gcd1 = euclidean\_gcd(a, b)  
 println("Алгоритм Евклида: $gcd1")  
 # Бинарный алгоритм  
 gcd2 = binary\_gcd(a, b)  
 println("Бинарный алгоритм: $gcd2")  
   
 # Расширенный алгоритм  
 gcd3, x3, y3 = extended\_euclidean(a, b)  
 println("Расширенный алгоритм: $gcd3")  
 println("Коэффициенты Безу: $a×$x3 + $b×$y3 = $(a\*x3 + b\*y3)")  
   
 # Расширенный бинарный алгоритм  
 gcd4, x4, y4 = extended\_binary\_gcd(a, b)  
 println("Расшир. бинарный: $gcd4")  
 println("Коэффициенты Безу: $a×$x4 + $b×$y4 = $(a\*x4 + b\*y4)")  
   
 # Проверка согласованности  
 if gcd1 == gcd2 == gcd3 == gcd4  
 println("✓ Все алгоритмы дали одинаковый результат")  
 else  
 println("✗ Ошибка: результаты различаются!")  
 end  
   
 # Проверка взаимной простоты  
 if gcd1 == 1  
 println("✓ Числа взаимно просты")  
 else  
 println("Числа не взаимно просты")  
 end  
 end  
end  
  
function main()  
   
 println()  
   
 test\_algorithms()   
end  
main()

Результат тестирования представлен на рис.1

*Рис. 1 Тестирование:*



Вывод: В ходе данной работы мной были изучены и реализованы различные вариации метода Евклида для вычисления НОД. Написан программный код на языке Julia и протестирован.