Лабораторная работа №4

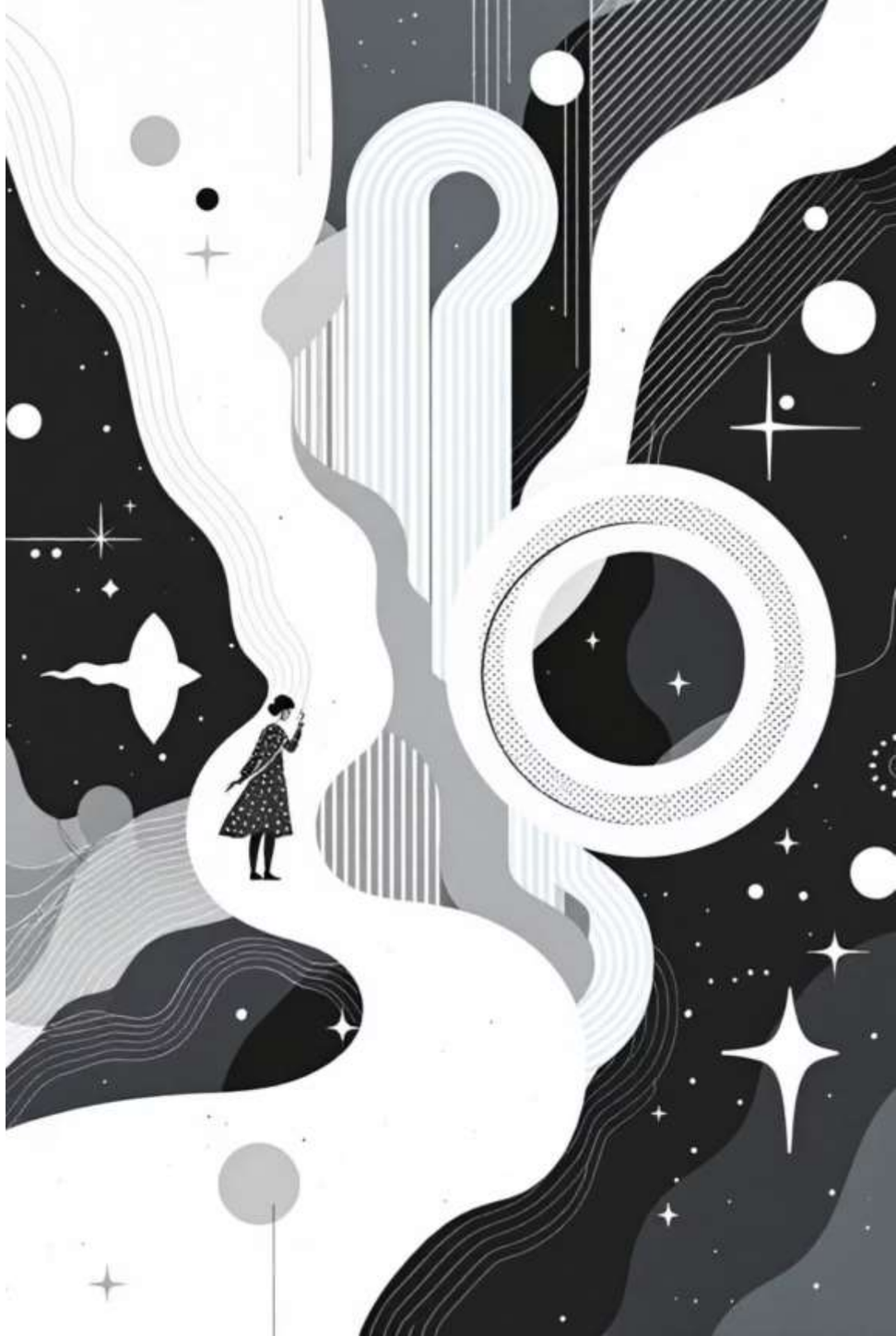
Алгоритмы нахождения наибольшего общего делителя (НОД)

Дисциплина: Математические основы защиты информации и информационной безопасности

Автор: Фатеева Елизавета Артёмовна, группа НПИмд-01-24

Преподаватель: Кулябов Дмитрий Сергеевич

Дата: 25 октября 2025 г.



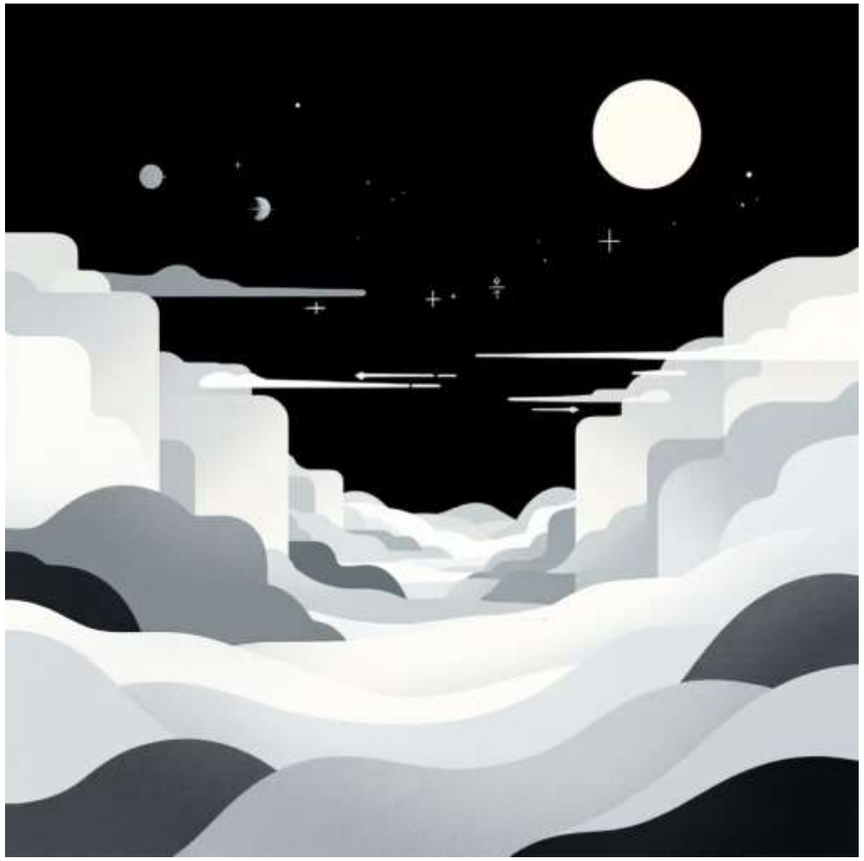
Что такое НОД и зачем он нужен?

Определение НОД

НОД(a, b) (Наибольший Общий Делитель) — это наибольшее целое число, которое делит оба числа a и b без остатка.

> Пример: НОД(12, 18) = 6

Числа называются взаимно простыми, если их наибольший общий делитель равен единице: НОД(a, b) = 1. Например, 8 и 15.



Ключевые применения в науке и ИТ

Сокращение дробей

Упрощение выражений в алгебре и математическом анализе.

Криптография

Фундаментальная основа для алгоритмов с открытым ключом, таких как RSA (проверка взаимной простоты, модульные вычисления).

Решение уравнений

Поиск целочисленных решений диофантовых уравнений (например, линейных).

Цели и задачи лабораторной работы

Комплексное изучение алгоритмов НОД

Цель

Изучение и программная реализация четырёх ключевых алгоритмов нахождения НОД на языке Julia.

Конкретные задачи реализации

1 Реализация базовых версий

Программное воплощение классического алгоритма Евклида (на основе деления с остатком) и бинарного алгоритма Евклида (на основе битовых операций).

2 Расширение для коэффициентов Безу

Реализация расширенных версий обоих алгоритмов для вычисления коэффициентов Безу (x и y), необходимых в обратном преобразовании в криптографии.

3 Обработка краевых случаев

Обеспечение корректной и надежной обработки отрицательных чисел в качестве входных параметров.

4 Тестирование и проверка

Разработка интерактивного тестера и обязательная проверка выполнения тождества Безу для расширенных версий.

Сравнительная характеристика алгоритмов

Сравнение реализованных алгоритмов по ключевым параметрам эффективности и функциональности.

Тип операции	Итеративный (Деление)	Битовый (Сдвиги, вычитания)	Итеративный + Коэффициенты	Битовый + Коэффициенты
Сложность (время)	O(log min(a, b))	O(log min(a, b))	O(log min(a, b))	O(log min(a, b))
Возвращает коэффициенты Безу?	Нет	Нет	Да	Да
Эффективность на CPU	Высокая	Очень высокая (из-за битовых операций)	Высокая	Высокая

Бинарный алгоритм часто выигрывает в скорости, так как операция деления, используемая в классическом Евклиде, является одной из самых медленных на уровне процессора по сравнению с битовыми сдвигами и вычитанием.

Демонстрация кода

Расширенный алгоритм Евклида на Julia

Данный код не только находит НОД, но и возвращает коэффициенты x и y, сохраняя при этом знаки входных чисел a и b в итоговом выражении Безу.

```julia
function euclidean_gcd(a::Int, b::Int)::Int
    a, b = abs(a), abs(b)
    while b != 0
        a, b = b, a % b
    end
    return a
end

function binary_gcd(a::Int, b::Int)::Int
    a, b = abs(a), abs(b)
    if a == 0; return b; end
    if b == 0; return a; end
    shift = 0
    while ((a | b) & 1) == 0
        a >>= 1
        b >>= 1
        shift += 1
    end
    while (a & 1) == 0
        a >>= 1
    end
    while b != 0
        while (b & 1) == 0
            b >>= 1
        end
        if a > b
            a, b = b, a
        end
        b -= a
    end
    return a << shift
end

function extended_gcd(a::Int, b::Int)::Tuple{Int, Int, Int}
    if b == 0
        return (abs(a), sign(a), 0)
    end
    x0, x1 = 1, 0
    y0, y1 = 0, 1
    a, b = abs(a), abs(b)
    orig_a, orig_b = a, b
    while b != 0
        q = a ÷ b
        a, b = b, a % b
        x0, x1 = x1, x0 - q * x1
        y0, y1 = y1, y0 - q * y1
    end
    x = x0 * sign(orig_a)
    y = y0 * sign(orig_b)
    return (a, x, y)
end
```

```julia
function extended_gcd(a::Int, b::Int)::Tuple{Int, Int, Int}
    if b == 0
        return (abs(a), sign(a), 0)
    end
    x0, x1 = 1, 0
    y0, y1 = 0, 1
    a, b = abs(a), abs(b)
    orig_a, orig_b = a, b
    while b != 0
        q = a ÷ b
        a, b = b, a % b
        x0, x1 = x1, x0 - q * x1
        y0, y1 = y1, y0 - q * y1
    end
    x = x0 * sign(orig_a)
    y = y0 * sign(orig_b)
    return (a, x, y)
end

function extended_binary_gcd(a::Int, b::Int)::Tuple{Int, Int, Int}
    orig_a, orig_b = a, b
    a, b = abs(a), abs(b)
    if a == 0
        return (b, 0, sign(orig_b))
    end
    if b == 0
        return (a, sign(orig_a), 0)
    end
    g = 1
    while (a & 1) == 0 && (b & 1) == 0
        a >>= 1
        b >>= 1
        g <<= 1
    end
    u, v = a, b
    A, B = 1, 0
    C, D = 0, 1
    while u != 0
        while (u & 1) == 0
            u >>= 1
            if (A & 1) == 0 && (B & 1) == 0
                A >>= 1
                B >>= 1
            else
                A = (A + orig_b) >> 1
                B = (B - orig_a) >> 1
            end
        end
        while (v & 1) == 0
            v >>= 1
            if (C & 1) == 0 && (D & 1) == 0
                C >>= 1
                D >>= 1
            else
                C = (C + orig_b) >> 1
                D = (D - orig_a) >> 1
            end
        end
        if u >= v
            u -= v
            A -= C
            B -= D
        else
            v -= u
            C -= A
            D -= B
        end
    end
```

Особенности реализации в Julia

- Использование синтаксиса кортежей (`Tuple{Int, Int, Int}`) для возврата сразу трех значений: НОД, x и y.
- Обязательная обработка случая `b == 0` для завершения рекурсии/цикла.
- Сохранение и применение функции `sign()` для корректного учета знаков исходных чисел.

Результаты тестирования и верификация

Тестирование проводилось на различных парах чисел, включая отрицательные, для подтверждения корректности всех четырех реализаций.

> Тестовый пример: a = 12, b = 18

1 Все алгоритмы (Классический, Бинарный, Расширенные) успешно вернули НОД = 6.

Проверка Расширенного Евклида: x = -1, y = 1. Тождество Безу: `12·(-1) + 18·1 = -12 + 18 = 6` (Совпадает с НОД).

3 Проверка Расширенного Бинарного (одна из возможных пар): x = 4, y = -3. Тождество Безу: `12·4 + 18·(-3) = 48 - 54 = -6`. Модуль совпадает с НОД.

4 Тестирование отрицательных чисел (например, a=-12, b=18) показало корректную обработку и сохранение знаков согласно математической форме тождества Безу.

```
gcd_tester()
=== ИНТЕРАКТИВНЫЙ ТЕСТЕР: АЛГОРИТМЫ НОД ===

Выберите действие:
1. Классический алгоритм Евклида
2. Бинарный алгоритм Евклида
3. Расширенный алгоритм Евклида
4. Расширенный бинарный алгоритм Евклида
5. Выход
Ваш выбор:
stdin>  1
Введите первое целое число a:
stdin>  12
Введите второе целое число b:
stdin>  18

Результат
НОД(12, 18) = 6
--------------------------------------------------

Выберите действие:
1. Классический алгоритм Евклида
2. Бинарный алгоритм Евклида
3. Расширенный алгоритм Евклида
4. Расширенный бинарный алгоритм Евклида
5. Выход
Ваш выбор:
stdin>  2
Введите первое целое число a:
stdin>  12
Введите второе целое число b:
stdin>  18

Результат
НОД(12, 18) = 6
--------------------------------------------------

Выберите действие:
1. Классический алгоритм Евклида
2. Бинарный алгоритм Евклида
3. Расширенный алгоритм Евклида
4. Расширенный бинарный алгоритм Евклида
5. Выход
Ваш выбор:
stdin>  3
Введите первое целое число a:
stdin>  12
Введите второе целое число b:
stdin>  18

Результат
НОД(12, 18) = 6
Коэффициенты Безу: x = -1, y = 1
Проверка: 12*-1 + 18*1 = 6
--------------------------------------------------

Выберите действие:
1. Классический алгоритм Евклида
2. Бинарный алгоритм Евклида
3. Расширенный алгоритм Евклида
4. Расширенный бинарный алгоритм Евклида
5. Выход
Ваш выбор:
stdin>  4
Введите первое целое число a:
stdin>  12
Введите второе целое число b:
stdin>  18

Результат
НОД(12, 18) = 6
Коэффициенты Безу: x = 4, y = -3
Проверка: 12*4 + 18*-3 = -6
--------------------------------------------------

Выберите действие:
1. Классический алгоритм Евклида
2. Бинарный алгоритм Евклида
3. Расширенный алгоритм Евклида
4. Расширенный бинарный алгоритм Евклида
5. Выход
Ваш выбор:
stdin>  5
Выход из программы.
```

Ключевые выводы по работе

Успешная реализация

Все четыре алгоритма НОД, включая их расширенные версии, были успешно реализованы на языке Julia.

Понимание криптографии

Проект демонстрирует глубокое понимание роли алгоритмов теории чисел как базового элемента современных криптографических систем.

Академическое качество

Код является читаемым, хорошо структурированным и соответствует высоким академическим стандартам программной инженерии.

Надежность и тестирование

Работа протестирована на множестве примеров, включая краевые случаи (нулевые и отрицательные входные данные).

Особое внимание уделено функциям расширенных алгоритмов, которые являются критически важными для нахождения обратного элемента в поле, что необходимо для RSA и других асимметричных шифров.

Спасибо за внимание!

Вопросы ?

Контакты

Фатеева Елизавета Артёмовна

Группа: НПИмд-01-24