

Отчёт по лабораторной работе №4

Математические основы защиты информации и информационной безопасности

Вычисление наибольшего общего делителя

Выполнил: Махорин Иван Сергеевич,
НФИмд-02-21, 1032259380

Содержание

1	Цель работы	4
2	Выполнение лабораторной работы	5
2.1	Реализация алгоритма Евклида	5
2.2	Реализация бинарного алгоритма Евклида	7
2.3	Реализация расширенного алгоритма Евклида	10
2.4	Реализация расширенного бинарного алгоритма Евклида	13
3	Список литературы. Библиография	18

Список иллюстраций

2.1	Реализация алгоритма Евклида	6
2.2	Реализация алгоритма Евклида	7
2.3	Проверка	7
2.4	Реализация бинарного алгоритма Евклида	8
2.5	Реализация бинарного алгоритма Евклида	9
2.6	Проверка	10
2.7	Реализация расширенного алгоритма Евклида	11
2.8	Реализация расширенного алгоритма Евклида	12
2.9	Проверка	13
2.10	Реализация расширенного бинарного алгоритма Евклида	14
2.11	Реализация расширенного бинарного алгоритма Евклида	15
2.12	Реализация расширенного бинарного алгоритма Евклида	16
2.13	Проверка	17

1 Цель работы

Изучить алгоритмы нахождения наибольшего общего делителя и научиться их реализовывать.

2 Выполнение лабораторной работы

2.1 Реализация алгоритма Евклида

Алгоритм Евклида — эффективный алгоритм для нахождения наибольшего общего делителя двух целых чисел (или общей меры двух отрезков). Алгоритм назван в честь греческого математика Евклида (III век до н. э.), который впервые описал его в VII и X книгах «Начал».

Выполним реализацию этого алгоритма на языке Julia (рис. 2.1) и (рис. 2.2):

```

# Алгоритм Евклида

function euclidean_algorithm(a::Int, b::Int)::Int
    # Проверяем, что b не равно 0
    b == 0 && throw(DomainError(b, "b must not be zero"))

    # Шаг 1: Инициализация
    r_prev, r_curr = a, b
    i = 1

    # Шаг 2-3: Повторное деление с остатком
    while true
        # Находим остаток от деления
        r_next = r_prev % r_curr

        # Если остаток равен 0, возвращаем текущий делитель
        if r_next == 0
            return abs(r_curr) # Возвращаем абсолютное значение
        end

        # Обновляем значения для следующей итерации
        r_prev, r_curr = r_curr, r_next
        i += 1
    end
end
end

```

Рис. 2.1: Реализация алгоритма Евклида

```

test_cases = [
    (12345, 24690),
    (12345, 54321),
    (12345, 12541),
    (91, 105),
    (105, 154),
    (91, 154)
]

for (a, b) in test_cases
    gcd_val = euclidean_algorithm(a, b)
    println("НОД($a, $b) = $gcd_val")
end
println()

```

Рис. 2.2: Реализация алгоритма Евклида

Проверим работу алгоритма (рис. 2.3):

```

НОД(12345, 24690) = 12345
НОД(12345, 54321) = 3
НОД(12345, 12541) = 1
НОД(91, 105) = 7
НОД(105, 154) = 7
НОД(91, 154) = 7

```

Рис. 2.3: Проверка

2.2 Реализация бинарного алгоритма Евклида

Бинарный алгоритм Евклида — метод нахождения наибольшего общего делителя двух целых чисел. Данный алгоритм «быстрее» обычного алгоритма Евклида,

так как вместо медленных операций деления и умножения используются сдвиги. Но это преимущество в скорости теряется с увеличением разницы между целыми числами более чем на несколько порядков, в результате чего число итераций вычитания может многократно превышать число итераций обычного алгоритма, использующего сравнение по модулю. То есть скорость бинарных сдвигов даёт эффект только для чисел, близких друг к другу.

Выполним реализацию этого алгоритма на языке Julia (рис. 2.4) и (рис. 2.5):

```
# Бинарный алгоритм Евклида

function binary_euclidean_algorithm(a::Int, b::Int)::Int
    # Проверяем, что b не равно 0
    b == 0 && throw(DomainError(b, "b must not be zero"))

    # Шаг 1: Инициализация
    g = 1

    # Шаг 2: Убираем общие множители 2
    while iseven(a) && iseven(b)
        a ÷= 2
        b ÷= 2
        g *= 2
    end

    # Шаг 3: Инициализация u и v
    u, v = a, b

    # Шаг 4: Основной цикл
    while u != 0
        # Убираем множители 2 из u
        while iseven(u)
            u ÷= 2
        end

        # Убираем множители 2 из v
        while iseven(v)
            v ÷= 2
        end
    end
end
```

Рис. 2.4: Реализация бинарного алгоритма Евклида


```

    # Вычитаем меньшее из большего
    if u >= v
        u = u - v
    else
        v = v - u
    end
end

# Шаг 5: Возвращаем результат
return g * v
end

test_cases = [
    (12345, 24690),
    (12345, 54321),
    (12345, 12541),
    (91, 105),
    (105, 154),
    (91, 154)
]

for (a, b) in test_cases
    gcd_val = binary_euclidean_algorithm(a, b)
    println("НОД($a, $b) = $gcd_val")
end
println()

```

Рис. 2.5: Реализация бинарного алгоритма Евклида

Проверим работу алгоритма (рис. 2.6):

$\text{НОД}(12345, 24690) = 12345$
 $\text{НОД}(12345, 54321) = 3$
 $\text{НОД}(12345, 12541) = 1$
 $\text{НОД}(91, 105) = 7$
 $\text{НОД}(105, 154) = 7$
 $\text{НОД}(91, 154) = 7$

Рис. 2.6: Проверка

2.3 Реализация расширенного алгоритма Евклида

Расширенный алгоритм Евклида — модификация алгоритма Евклида, вычисляющая, кроме наибольшего общего делителя (НОД) целых чисел a и b , ещё и коэффициенты соотношения Безу, то есть такие целые x и y , что $ax + by = \text{НОД}(a, b)$.

Выполним реализацию этого алгоритма на языке Julia (рис. 2.7) и (рис. 2.8):

```

# Расширенный алгоритм Евклида

function extended_euclidean_algorithm(a::Int, b::Int)::Tuple{Int, Int, Int}
    # Проверяем, что b не равно 0
    b == 0 && throw(DomainError(b, "b must not be zero"))

    # Шаг 1: Инициализация
    r_prev, r_curr = a, b
    x_prev, x_curr = 1, 0
    y_prev, y_curr = 0, 1
    i = 1

    # Шаг 2-3: Основной цикл
    while true
        # Находим частное и остаток
        q = r_prev ÷ r_curr
        r_next = r_prev % r_curr

        # Если остаток равен 0, возвращаем результат
        if r_next == 0
            return (r_curr, x_curr, y_curr)
        end

        # Вычисляем новые коэффициенты
        x_next = x_prev - q * x_curr
        y_next = y_prev - q * y_curr

        # Обновляем значения для следующей итерации
        r_prev, r_curr = r_curr, r_next
        x_prev, x_curr = x_curr, x_next
        y_prev, y_curr = y_curr, y_next
        i += 1
    end
end
end

```

Рис. 2.7: Реализация расширенного алгоритма Евклида

```

test_cases = [
    (12345, 24690),
    (12345, 54321),
    (12345, 12541),
    (91, 105),
    (105, 154),
    (91, 154)
]

for (a, b) in test_cases
    gcd_val, x, y = extended_euclidean_algorithm(a, b)
    println("НОД($a, $b) = $gcd_val")
    println("Коэффициенты Безу: x = $x, y = $y")
    println("Проверка: $a * $x + $b * $y = $(a*x + b*y)")
    println()
end

```

Рис. 2.8: Реализация расширенного алгоритма Евклида

Проверим работу алгоритма (рис. 2.9):

$$\text{НОД}(12345, 24690) = 12345$$

$$\text{Коэффициенты Безу: } x = 1, y = 0$$

$$\text{Проверка: } 12345 * 1 + 24690 * 0 = 12345$$

$$\text{НОД}(12345, 54321) = 3$$

$$\text{Коэффициенты Безу: } x = 3617, y = -822$$

$$\text{Проверка: } 12345 * 3617 + 54321 * -822 = 3$$

$$\text{НОД}(12345, 12541) = 1$$

$$\text{Коэффициенты Безу: } x = 4159, y = -4094$$

$$\text{Проверка: } 12345 * 4159 + 12541 * -4094 = 1$$

$$\text{НОД}(91, 105) = 7$$

$$\text{Коэффициенты Безу: } x = 7, y = -6$$

$$\text{Проверка: } 91 * 7 + 105 * -6 = 7$$

$$\text{НОД}(105, 154) = 7$$

$$\text{Коэффициенты Безу: } x = 3, y = -2$$

$$\text{Проверка: } 105 * 3 + 154 * -2 = 7$$

$$\text{НОД}(91, 154) = 7$$

$$\text{Коэффициенты Безу: } x = -5, y = 3$$

$$\text{Проверка: } 91 * -5 + 154 * 3 = 7$$

Рис. 2.9: Проверка

2.4 Реализация расширенного бинарного алгоритма

Евклида

Выполним реализацию этого алгоритма на языке Julia (рис. 2.10 - рис. 2.12):

```

# Расширенный бинарный алгоритм Евклида

function extended_binary_euclidean_algorithm(a::Int, b::Int)::Tuple{Int, Int, Int}
    # Проверяем, что b не равно 0
    b == 0 && throw(DomainError(b, "b must not be zero"))

    # Шаг 1: Инициализация
    g = 1

    # Шаг 2: Убираем общие множители 2
    while iseven(a) && iseven(b)
        a ÷= 2
        b ÷= 2
        g *= 2
    end

    # Шаг 3: Инициализация переменных
    u, v = a, b
    A, B, C, D = 1, 0, 0, 1

    # Шаг 4: Основной цикл
    while u != 0
        # Шаг 4.1: Обработка четного u
        while iseven(u)
            u ÷= 2
            if iseven(A) && iseven(B)
                A ÷= 2
                B ÷= 2
            else

```

Рис. 2.10: Реализация расширенного бинарного алгоритма Евклида

```

        else
            A = (A + b) ÷ 2
            B = (B - a) ÷ 2
        end
    end
end

# Шаг 4.2: Обработка четного v
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

# Шаг 4.3: Вычитание
if u >= v
    u = u - v
    A = A - C
    B = B - D
else
    v = v - u
    C = C - A
    D = D - B
end
end
end

```

Рис. 2.11: Реализация расширенного бинарного алгоритма Евклида

```

    # Шаг 5: Возвращаем результат
    return (g * v, C, D)
end

test_cases = [
    (12345, 24690),
    (12345, 54321),
    (12345, 12541),
    (91, 105),
    (105, 154),
    (91, 154)
]

for (a, b) in test_cases
    gcd_val, x, y = extended_binary_euclidean_algorithm(a, b)
    println("НОД($a, $b) = $gcd_val")
    println("Коэффициенты Безу: x = $x, y = $y")
    println("Проверка: $a * $x + $b * $y = $(a*x + b*y)")
    println()
end

```

Рис. 2.12: Реализация расширенного бинарного алгоритма Евклида

Проверим работу алгоритма (рис. 2.13):

$$\text{НОД}(12345, 24690) = 12345$$

$$\text{Коэффициенты Безу: } x = 12345, y = -6172$$

$$\text{Проверка: } 12345 * 12345 + 24690 * -6172 = 12345$$

$$\text{НОД}(12345, 54321) = 3$$

$$\text{Коэффициенты Безу: } x = -14490, y = 3293$$

$$\text{Проверка: } 12345 * -14490 + 54321 * 3293 = 3$$

$$\text{НОД}(12345, 12541) = 1$$

$$\text{Коэффициенты Безу: } x = 4159, y = -4094$$

$$\text{Проверка: } 12345 * 4159 + 12541 * -4094 = 1$$

$$\text{НОД}(91, 105) = 7$$

$$\text{Коэффициенты Безу: } x = 52, y = -45$$

$$\text{Проверка: } 91 * 52 + 105 * -45 = 7$$

$$\text{НОД}(105, 154) = 7$$

$$\text{Коэффициенты Безу: } x = 113, y = -77$$

$$\text{Проверка: } 105 * 113 + 154 * -77 = 7$$

$$\text{НОД}(91, 154) = 7$$

$$\text{Коэффициенты Безу: } x = -5, y = 3$$

$$\text{Проверка: } 91 * -5 + 154 * 3 = 7$$

Рис. 2.13: Проверка

3 Список литературы. Библиография

[1] Julia: <https://docs.julialang.org/en/v1/>