Отчет по лабораторной работе №4

дисциплина: Математические основы защиты информации и информационной безопасности

Морозова Ульяна Константиновна

Содержание

1 Цель работы

Целью работы является изучение алгоритмов вычисления НОД и реализация их на языке Julia.

2 Выполнение лабораторной работы

2.1 Алгоритм Евклида

Алгоритм Евклида — метод для нахождения наибольшего общего делителя (НОД) двух чисел. Алгоритм основан на принципе, что НОД двух чисел остаётся неизменным, когда большее число заменяется его остатком при делении на меньшее число.

Далее приведена реализация шифра на языке Julia.

```
function euclidean(a::T, b::T) where T<:Integer
    a=abs(a)
    b=abs(b)

    while b != 0
        a,b = b, a%b
    end
    return a
end
```

2.2 Бинарный алгоритм Евклида

Бинарный алгоритм вычисления НОД (бинарный алгоритм Евклида) — метод нахождения наибольшего общего делителя (НОД) двух целых чисел, который

использует операции сдвига и вычитания вместо деления. Разработан в 1967 году Джозефом Стайном.

Преимущества алгоритма: - скорость работы за счёт использования бинарных операций, которые выполняются быстрее, чем деление и умножение; - отсутствие дорогостоящих операций деления и остатка.

```
function binary_euc(a::T, b::T) where T<:Integer
    a=abs(a)
    b=abs(b)

    if a == 0
        return b
    end
    if b == 0
        return a
    end

    s = 0
    while ((a|b)&1) == 0
        a>>=1
        b>>=1
        s += 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<<s
end
```

2.3 Расширенный алгоритм Евклида

Расширенный алгоритм Евклида — модификация алгоритма Евклида, которая, кроме вычисления наибольшего общего делителя (НОД) двух целых чисел, находит коэффициенты x и y для уравнения Безу: ax + by = НОД(a, b). Это означает, что алгоритм не только находит НОД, но и выражает его как линейную комбинацию a и b.

```
function ext_euc(a::T, b::T) where T<:Integer
    if b == 0
        return a, one(T), zero(T)
    end

    gcd_val, x1, y1 = ext_euc(b, a % b)
    x = y1
    y = x1 - (a ÷ b) * y1

    return gcd_val, x, y
```

```
end
```

2.4 Расширенный бинарный алгоритм Евклида

Бинарный расширенный алгоритм Евклида - это комбинация бинарного и расширенного алгоритмов, он использует битовые операции для повышения эффективности Все функции работают с целыми числами любого размера и корректно обрабатывают отрицательные числа и нули.

```
function binary_ext(a::T, b::T) where T <: Integer
    a = abs(a)
    b = abs(b)

    g = 1
    while iseven(a) && iseven(b)
        a >>= 1
        b >>= 1
        g <<= 1
    end

    u, v = a, b
    A, B, C, D = one(T), zero(T), zero(T), one(T)

    while u != 0
        while iseven(u)
            u >>= 1
            if iseven(A) && iseven(B)
                A >>= 1
                B >>= 1
            else
```

```
            A = (A + b) >> 1
            B = (B - a) >> 1
        end
    end

    while iseven(v)
        v >>= 1
        if iseven(C) && iseven(D)
            C >>= 1
            D >>= 1
        else
            C = (C + b) >> 1
            D = (D - a) >> 1
        end
    end

    if u >= v
        u -= v
        A -= C
        B -= D
    else
        v -= u
        C -= A
        D -= B
    end
end

val = v * g
return val, C, D
```

```
end
```

2.5 Результаты работы алгоритмов

Для проверки работы была использована следующая функция

```
function test_gcd_algorithms()
    test_cases = [
        (48, 18),
        (1071, 462),
        (17, 13),
        (100, 25),
        (0, 5),
        (169, 13)
    ]

    println("Тестирование алгоритмов НОД:")
    println("="^50)

    for (a, b) in test_cases
        gcd1 = euclidean(a, b)
        gcd2 = binary_euc(a, b)
        gcd3, x3, y3 = ext_euc(a, b)
        gcd4, x4, y4 = binary_ext(a, b)

        println("НОД($a, $b) = $gcd1")
        println("Классический: $gcd1")
        println("Бинарный: $gcd2")
        println("Расширенный: $gcd3 (коэффициенты: $x3, $y3)")
        println("Бинарный расширенный: $gcd4 (коэффициенты: $x4, $y4)")
```

```
        # Проверка тождества Безу
        bezout_check3 = a * x3 + b * y3 == gcd3
        bezout_check4 = a * x4 + b * y4 == gcd4

        println("Тождество Безу (расширенный): $bezout_check3")
        println("Тождество Безу (бинарный расширенный): $bezout_check4")
        println("-"^30)
    end
end
```

Результаты:

```
Тестирование алгоритмов НОД:
==================================================
НОД(48, 18) = 6
Классический: 6
Бинарный: 6
Расширенный: 6 (коэффициенты: -1, 3)
Бинарный расширенный: 6 (коэффициенты: -4, 11)
Тождество Безу (расширенный): true
Тождество Безу (бинарный расширенный): true
------------------------------
НОД(1071, 462) = 21
Классический: 21
Бинарный: 21
Расширенный: 21 (коэффициенты: -3, 7)
Бинарный расширенный: 21 (коэффициенты: -47, 109)
Тождество Безу (расширенный): true
Тождество Безу (бинарный расширенный): true
------------------------------
```

```
НОД(17, 13) = 1
Классический: 1
Бинарный: 1
Расширенный: 1 (коэффициенты: -3, 4)
Бинарный расширенный: 1 (коэффициенты: -3, 4)
Тождество Безу (расширенный): true
Тождество Безу (бинарный расширенный): true
------------------------------
НОД(100, 25) = 25
Классический: 25
Бинарный: 25
Расширенный: 25 (коэффициенты: 0, 1)
Бинарный расширенный: 25 (коэффициенты: 0, 1)
Тождество Безу (расширенный): true
Тождество Безу (бинарный расширенный): true
------------------------------
НОД(0, 5) = 5
Классический: 5
Бинарный: 5
Расширенный: 5 (коэффициенты: 0, 1)
Бинарный расширенный: 5 (коэффициенты: 0, 1)
Тождество Безу (расширенный): true
Тождество Безу (бинарный расширенный): true
------------------------------
НОД(169, 13) = 13
Классический: 13
Бинарный: 13
Расширенный: 13 (коэффициенты: 0, 1)
Бинарный расширенный: 13 (коэффициенты: 0, 1)
```

```
Тождество Безу (расширенный): true
Тождество Безу (бинарный расширенный): true
------------------------------
```

3 Выводы

Мы изучили работу алгоритмов вычисления НОД, а также реализовали их на языке Julia.