Лабораторная работа №4

Дисциплина: Математические основы защиты информации и информационной безопасности

Студент: Леон Фернандо Хосе Фернандо

Цель работы

Вычислить максимальный общий делитель, используя алгоритмы, представленные в лабораторном рабочем материале 4.

Задание

1. Реализовать рассмотренные алгоритмы программно (4 коды)

---

2. Выполнение лабораторной работы

Алгоритм Евклида (1/2)

В этой отчете описывается программная реализация евклида алгоритма для нахождения наибольшего общего делителя (НОД) двух чисел. Этот алгоритм, написанный на языке программирования Julia, эффективен при вычислении НОД путем итеравтиного применения операций по модулю.

Алгоритм Евклида (2/2)

```julia
function gcd(a:: Int, b:: Int)
    r_0 = a
    r_1 = b
    i = 1

    while true
        r_next = r_0 % r_1

        if r_next == 0
            return r_1
        else
            r_0 = r_1
            r_1 = r_next
            i += 1
        end
    end
end
```

```
function input(prompt:: AbstractString)
    print(prompt)
    return chomp(readline())
end

a = input("a = ")
num1 = parse(Int, a)

b = input("b = ")
num2 = parse(Int, b)

d = gcd(num1, num2)
println("НОД = $d")
```

Бинарный алгоритм Евклида

В этом отчете описывается реализация двоичного евклидова алгоритма (также известного как алгоритм Штейна) в коде Julia для вычисления наибольшего общего делителя (НОД) двух целых чисел. Этот метод представляет собой вариацию традиционного евклидова алгоритма, использующего побитовые операции для более эффективной обработки четных и нечетных чисел.

```
function binary_gdc(a::Int, b::Int)
    g = 1

    while iseven(a) && iseven(b)
        a = a/2
        b = b/2
        g *= 2
    end

    u, v = a, b
    while u ≠ 0
        while iseven(u)
            u =u/2
        end
```

```
        while iseven(v)
            v =v/2
        end

        if u >= v
            u -= v
        else
            v -= u
```

```
        end
    end
    d = g * v
    return d
end

function input(prompt::AbstractString)
    print(prompt)
    return chomp(readline())
end

a = input("a = ")
num1 = parse(Int, a)

b = input("b = ")
num2 = parse(Int, b)

d = binary_gdc(num1, num2)
println("НОД = $d")
```

Расширенный алгоритм Евклида (1/2)

```
function extended_euclidean(a::Int, b::Int)
    r0, r1 = a, b
    x0, x1 = 1, 0
    y0, y1 = 0, 1
    i = 1

    while true
        q = r0 / r1
        r_next = r0 - q * r1

        if r_next == 0
            d, x, y = r1, x1, y1
            return d, x, y
        end

        x_next = x0 - q * x1
        y_next = y0 - q * y1

        r0, r1 = r1, r_next
        x0, x1 = x1, x_next
        y0, y1 = y1, y_next
        i += 1
    end
end
```

Расширенный алгоритм Евклида (2/2)

```
function input(prompt::AbstractString)
    print(prompt)
    return chomp(readline())
end

a = input("a = ")
num1 = parse(Int, a)

b = input("b = ")
num2 = parse(Int, b)

d, x, y = extended_euclidean(num1, num2)
println("НОД d = $d, x = $x, y = $y")
```

Расширенный бинанрный алгоритм Евклида(1/2)

```
function binary_extended(a::Int, b::Int)
   g = 1
   while iseven(a) && iseven(b)
        a = a/2
        b = b/2
        g *= 2
   end

   u, v = a, b
   A, B = 1, 0
   C, D = 0, 1

   while u ≠ 0
        while iseven(u)
            u = u/2
            if iseven(A) && iseven(B)
                A = A/2
                B = B/2
            else
                A = (A+b)/2
                B = (B-a)/2
            end
        end
```

Расширенный бинанрный алгоритм Евклида(2/2)

```
        while iseven(v)
            v = v/2
            if iseven(C) && iseven(D)
                C = C/2
                D = D/2
            else
                C = (C+b)/2
                D = (D-a)/2
            end
        end

        if u >= 0
            u = u - v
            A = A - C
            B = B - D
        else
            v = v - u
            C = C - A
            D = D - B
        end
    end
    d = g * v
    x, y = C, D
    return d, x, y
end
```

Вывод

Этот проект успешно демонстрирует реализацию и применение алгоритма Евклида и его расширенных версий, включая двоичный алгоритм Евклида и расширенный двоичный алгоритм Евклида. С помощью этих реализаций мы изучили различные эффективные методы вычисления наибольшего общего делителя (GOD) двух целых чисел, а также их коэффициентов Безу. Эти коэффициенты необходимы для выражения GCD в виде линейной комбинации исходных целых чисел, которая является фундаментальной при решении линейных диофантовых уравнений и имеет практическое применение в таких областях, как криптография и модульная арифметика.