

Лабораторная работа №4. Вычисление наибольшего общего делителя

Выполнила: Лебедева Ольга Андреевна

РУДН, Москва, Россия

2025

Изучить и реализовать на языке Julia[1] классический[2], бинарный, расширенный[3] и бинарный расширенный алгоритмы Евклида для нахождения наибольшего общего делителя.

- 1 Реализовать четыре варианта алгоритма Евклида на языке Julia.

Объект исследования: алгоритмы вычисления НОД.

Предмет исследования: классический, бинарный, расширенный и бинарный расширенный алгоритмы Евклида.

НОД (наибольший общий делитель) — наибольшее целое число, на которое делятся оба числа без остатка.

Коэффициенты линейной комбинации (x, y) — числа, удовлетворяющие уравнению $a \cdot x + b \cdot y = \text{НОД}(a, b)$.

Бинарный алгоритм — вариант метода Евклида, использующий деление на 2 вместо деления с остатком.

Расширенный алгоритм — модификация метода Евклида, позволяющая вычислить не только НОД, но и коэффициенты x и y .

Программное обеспечение:

- Язык программирования Julia.
- Среда разработки JupyterLab / VS Code.

Методы:

- Использование циклов и целочисленного деления.
- Работа с арифметическими операциями по модулю.
- Побитовые операции для бинарного алгоритма.
- Реализация шагов расширенного алгоритма с сохранением коэффициентов x и y .

Метод Евклида является классическим способом нахождения наибольшего общего делителя двух целых чисел. Он основан на свойстве:

если $a = b \cdot q + r$, то $\text{НОД}(a, b) = \text{НОД}(b, r)$

Алгоритм повторяется, пока остаток не станет равен нулю.

Бинарный алгоритм Евклида оптимизирует вычисления, заменяя деление на операции вычитания и деления на 2 (битовые сдвиги), что ускоряет выполнение на компьютере.

Расширенный алгоритм Евклида помимо НОД находит коэффициенты x и y , при которых выполняется равенство

$$a \cdot x + b \cdot y = \text{НОД}(a, b)$$

Эти коэффициенты широко применяются в криптографии и теории чисел.

Расширенный бинарный алгоритм Евклида объединяет оба подхода, повышая эффективность вычислений при сохранении возможности нахождения коэффициентов.

- 1 Реализовать четыре алгоритма Евклида на языке Julia.
- 2 Проверить работу программ.
- 3 Вывести результаты (НОД и коэффициенты).

Напишем код 1 с помощью языка Julia: См. рис. 1

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

function gcd_binary(a::Int, b::Int)
    if a == 0 return b end; if b == 0 return a end
    shift = 0
    while iseven(a) && iseven(b); a >>= 1; b >>= 1; shift += 1; end
    while iseven(a); a >>= 1; end
    while b != 0
        while iseven(b); b >>= 1; end
        if a > b; a, b = b, a; end
        b -= a
    end
    return a << shift
end
```

Рис. 1: Классический Евклид

Классический алгоритм Евклида основан на свойстве: если $a = b \cdot q + r$, где r — остаток от деления, то $\text{НОД}(a, b) = \text{НОД}(b, r)$. Это означает, что общий делитель двух чисел не изменится, если большее число заменить на остаток от деления его на меньшее.

В программе используется цикл `while`, который выполняется до тех пор, пока второе число не станет равным нулю.

На каждом шаге происходит операция:

$$a, b = b, a \% b$$

то есть старшее число заменяется младшим, а младшее — остатком от деления. Когда остаток становится равным нулю, переменная a содержит значение наибольшего общего делителя. Алгоритм является базовым, надёжным и демонстрирует принцип постепенного уменьшения пары чисел до их общего делителя.

Напишем код 2 с помощью языка Julia: См. рис. 2

```
function gcd_binary(a::Int, b::Int)
    if a == 0 return b end; if b == 0 return a end
    shift = 0
    while iseven(a) && iseven(b); a >>= 1; b >>= 1; shift += 1; end
    while iseven(a); a >>= 1; end
    while b != 0
        while iseven(b); b >>= 1; end
        if a > b; a, b = b, a; end
        b -= a
    end
    return a << shift
end
```

Рис. 2: Бинарный Евклид

Бинарный (или двоичный) алгоритм Евклида является оптимизированной версией классического метода. Он основан на том, что деление на 2 можно заменить побитовым сдвигом, что существенно ускоряет вычисления на уровне машинных операций.

Программа проверяет чётность чисел с помощью функции `iseven()`. Если оба числа чётные — они делятся на 2, а общий множитель 2 запоминается. Если одно из чисел чётное, оно делится на 2, пока не станет нечётным. Если оба числа нечётные, из большего вычитается меньшее — при этом НОД не меняется. Цикл продолжается до тех пор, пока одно из чисел не обнулится. В конце результат восстанавливается умножением на сохранённую степень двойки. Таким образом, алгоритм избегает операций деления с остатком и использует только побитовые сдвиги и вычитания.

Напишем код 3 с помощью языка Julia: См. рис. 3

```
function gcd_extended(a::Int, b::Int)
    old_r, r = a, b; old_s, s = 1, 0; old_t, t = 0, 1
    while r != 0
        q = div(old_r, r)
        old_r, r = r, old_r - q * r; old_s, s = s, old_s - q * s; old_t, t = t, old_t - q * t
    end
    return (abs(old_r), old_s, old_t)
end
```

Рис. 3: Расширенный Евклид

Расширенный алгоритм Евклида не только находит НОД, но и вычисляет такие коэффициенты x и y , для которых выполняется равенство:

$$a \cdot x + b \cdot y = \text{НОД}(a, b)$$

Алгоритм использует те же шаги, что и классический, но дополнительно сохраняет линейные комбинации исходных чисел. На каждом шаге, кроме вычисления остатка, пересчитываются коэффициенты s и t , которые отражают, как текущий остаток выражается через a и b .

Когда один из остатков становится равным нулю, предыдущие значения s и t содержат коэффициенты, удовлетворяющие приведённому уравнению. Эти значения часто применяются в криптографии (например, при поиске обратных элементов по модулю). В итоге алгоритм возвращает три значения: сам НОД и коэффициенты x , y .

Напишем код 4 с помощью языка Julia: См. рис. 4

```
function gcd_binary_extended(a::Int, b::Int)
    if a == 0 return (b, 0, 1) end; if b == 0 return (a, 1, 0) end
    g = 1
    while iseven(a) && iseven(b); a >>= 1; b >>= 1; g <= 1; end
    u, v = a, b; A, B, C, D = 1, 0, 0, 1
    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
    d = g * v; return (abs(d), C, D)
end
```

Рис. 4: Бинарный расширенный Евклид

Расширенный бинарный алгоритм Евклида объединяет идеи двух предыдущих методов: он выполняет операции над числами с использованием только сдвигов и вычитаний, но при этом вычисляет коэффициенты x и y в выражении:

$$a \cdot x + b \cdot y = \text{НОД}(a, b)$$

Алгоритм начинается с удаления общих множителей 2, которые накапливаются в переменной g .

Далее формируются две пары коэффициентов (A, B) и (C, D) , которые позволяют отслеживать, как каждое текущее число u и v выражается через исходные a и b . На каждом шаге алгоритм делает одно из чисел нечётным, затем вычитает меньшее из большего, корректируя коэффициенты, чтобы сохранить равенство.

Когда одно из чисел становится нулём, второе содержит значение НОД, а соответствующие коэффициенты (x, y) — решение линейной комбинации. Благодаря бинарной оптимизации этот метод работает быстрее, чем классический расширенный вариант, особенно для больших чисел.

Запустим код и проверим результаты работы алгоритмов: См. рис. 5

```
println("1) Евклид: ", gcd_euclid(91,154))  
println("2) Бинарный: ", gcd_binary(91,154))  
println("3) Расширенный: ", gcd_extended(91,154))  
println("4) Расширенный бинарный: ", gcd_binary_extended(91,154))
```

- 1) Евклид: 7
- 2) Бинарный: 7
- 3) Расширенный: (7, -5, 3)
- 4) Расширенный бинарный: (7, -5, 3)

Рис. 5: Вывод результатов

В ходе работы были изучены и реализованы четыре варианта алгоритма Евклида. Все методы дают одинаковый результат по величине НОД, что подтверждает их корректность. Бинарные алгоритмы демонстрируют более высокую эффективность при работе с большими числами, так как используют операции побитового сдвига. Расширенные версии позволяют получить дополнительные параметры (коэффициенты линейной комбинации), применяемые в криптографии.

[1] Julia: <https://ru.wikipedia.org/wiki/Julia>

[2] Алгоритм Евклида — https://ru.wikipedia.org/wiki/Алгоритм_Евклида

[3] Расширенный алгоритм Евклида —
https://ru.wikipedia.org/wiki/Расширенный_алгоритм_Евклида