Лабораторная Работа No 4

Математические Основы Защиты Информации и Информационной Безопасности

Хосе Фернандо Леон Атупанья | НФИмд-01-24

Содержание

1. Цель работы
2. Выполнение лабораторной работы
3. Выводы

1. Цель работы

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

2. Выполнение лабораторной работы

Алгоритм Евклида

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.

Функция: Реализация алгоритма Евклида Функция gcd вычисляет GCD из двух целых чисел, используя алгоритм Евклида. Вот краткое описание функции:

```
1   #Алгоритм Евклида
2   function gcd(a:: Int, b:: Int)
3       r_0 = a
4       r_1 = b
5       i = 1
```

Механизм цикла: Этот цикл while true выполняется бесконечно долго, пока не завершится при обнаружении GCD. Вычисление остатка: r_next присваивается значение r_0 % r_1, которое вычисляет остаток при делении r_0 на r_1.

Условие завершения: Цикл останавливается, когда значение r_next становится равным нулю, что означает, что значение r_1 является GCD для исходных входных данных. Этот GCD возвращается в качестве результата.

```
 7      while true
 8          r_next = r_0 % r_1
 9
10          if r_next == 0
11              return r_1
```

Если значение r_next не равно нулю, значение r_0 изменяется на значение r_1, а значение r_1 изменяется на значение r_next. В этой последовательности числа уменьшаются до тех пор, пока остаток не станет равным нулю. Результат: когда r_next равно нулю, r_1 является GCD и возвращается.

```
        else
            r_0 = r_1
            r_1 = r_next
            i += 1
        end
    end
end
```

Функция ввода предлагает пользователю ввести значения, которые затем преобразуются в целые числа.

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

Пользователю предлагается ввести значение, которое хранится в a. Затем оно преобразуется в целое число num1 с помощью parse(Int, a). Аналогичным образом запрашивается другое значение, которое сохраняется в b и преобразуется в num2.

GCD чисел num1 и num2 вычисляется с использованием функции gcd.

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

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

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

Результат выводится на печать с сообщением, отображающим НОД, что на русском языке расшифровывается как GCD.

```
julia> 56
56
b = 15
НОД = 1
```

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.

Функция binary_gdc вычисляет GCD из двух целых чисел, используя двоичный алгоритм Евклида.

```
#Бинарный алгоритм Евклида
function binary_gdc(a::Int, b::Int)
    g = 1
```

Хотя a и b четные, они делятся на 2, а g умножается на 2, чтобы отслеживать этот коэффициент. u and v are initialized to a and b, respectively, after their even factors have been divided out.Внешний цикл продолжается до тех пор, пока u не станет равным нулю. Делаем v нечетным: этот внутренний цикл продолжает делить v на 2 до тех пор, пока v не станет нечетным.

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

После того, как u и v станут нечетными, алгоритм использует вычитание для постепенного уменьшения большего значения: Если u больше или равно v, алгоритм вычитает v из u; в противном случае он вычитает u из v.

```
    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
```

Как только u становится равным нулю, оставшееся значение v умножается на g (общий коэффициент 2s, удаленный ранее), чтобы получить окончательный GCD:

```
27      d = g * v
28      return d
29  end
```

Программа запускается с запроса у пользователя двух целочисленных входных данных, Вызов binary_gdc: Значение GCD для num1 и num2 вычисляется с помощью функции binary_gdc.

```
36  a = input("a = ")
37  num1 = parse(Int, a)
38
39  b = input("b = ")
40  num2 = parse(Int, b)
41
42  d = binary_gdc(num1, num2)
43  println("НОД = $d")
```

Результат, обозначенный как NOD (что в переводе на русский означает НОД), отображается пользователю.

```
julia> 40
40
b = 20
НОД = 20.0
```

Расширенный алгоритм Евклида

Функция extended_euclidean является ядром этой программы. Она вычисляет как код двух целых чисел, так и коэффициенты для тождества Безу. Давайте рассмотрим каждую часть этой функции.

r0 и r1 хранят начальные значения a и b соответственно. x0 и x1 инициализируются как 1 и 0, соответственно, представляя начальные коэффициенты для a. y0 и y1 инициализируются как 0 и 1, представляя начальные коэффициенты для b. i - это счетчик итераций, который не является строго необходимым для работы алгоритма, но может быть полезен для отладки или отслеживания итераций.

```
1   #Расширенный алгоритм Евклида
2   function extended_euclidean(a::Int, b::Int)
3       r0, r1 = a, b
4       x0, x1 = 1, 0
5       y0, y1 = 0, 1
6       i = 1
```

На каждой итерации q вычисляется как целочисленное деление r0 на r1, представляющее, сколько раз r1 делится на r0. Следующий остаток r_next вычисляется как r0 - q * r1, что является ключевым шагом в евклидовом алгоритме для итеративного уменьшения значений до тех пор, пока остаток не достигнет нуля.

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

Когда r_next равно нулю, это означает, что r1 содержит GCD из a и b. d присваивается значение r1, которое является GCD. x и y присваиваются значения x1 и y1, соответственно, представляющие собой конечные коэффициенты Безу.

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

Следующие значения коэффициентов Безу вычисляются как x_next = x0 - q * x1 и y_next = y0 - q * y1. Эти значения выводятся из текущих коэффициентов и частного q, сохраняя связь x и y с исходными входными данными.

r0 и r1 обновляются значениями r1 и r_next, постепенно приближаясь к нулевому остатку. Аналогично, x0 и x1 устанавливаются в значения x1 и x_next, а y0 и y1 обновляются до y1 и y_next. Счетчик итераций i увеличивается.

```
        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
```

Программа предлагает пользователю ввести два значения, a и b, которые затем преобразуются в целые числа и присваиваются числам1 и num2 соответственно. Вызов extended_euclidean: Функция вызывается с числами num1 и num2 в качестве аргументов, возвращая GCD d и коэффициенты x и y.

```
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")
```

Инструкция println выводит результаты на русском языке с NOD d = (БОГ) и значениями x и y, показанными в качестве коэффициентов Безу.

```
julia> 56
56
b = 15
НОД d = 15, x = 0, y = 1
```

Расширенный бинанрный алгоритм Евклида

Функция binary_extended вычисляет коэффициенты GCD и Безу, используя расширенную версию алгоритма Штейна. Ниже приведена расшифровка функции:

```
#Расширенный бинанрный алгоритм Евклида
function binary_extended(a::Int, b::Int)
    g = 1
```

Удаление общих множителей, равных 2: Хотя a и b равны, они делятся на 2, а g умножается на 2, чтобы записать эти общие множители. Условие окончания цикла: Этот цикл завершается, как только значение a или b становится нечетным, удаляя все общие степени 2.

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

u и v инициализируются значениями a и b. A, B, C и D - коэффициенты для определения Безу. Первоначально, A=1, B=0, C=0, и D=1, представляющий единичную матрицу.

Внешний цикл продолжается до тех пор, пока u не станет равным нулю, что указывает на то, что НОД найден. Если A и B четные, они делятся на 2. В противном случае A и B корректируются для сохранения целочисленных значений путем добавления b к A и вычитания a из B, а затем деления обоих на 2. Аналогично, если C и D четные, они делятся на 2. В противном случае C и D корректируются таким образом, чтобы сохранить целочисленные значения, добавляя b к C и вычитая a из D.

```
10      u, v = a, b
11      A, B = 1, 0
12      C, D = 0, 1
13
14      while u ≠ 0
15          while iseven(u)
16              u = u/2
17              if iseven(A) && iseven(B)
18                  A = A/2
19                  B = B/2
20              else
21                  A = (A+b)/2
22                  B = (B-a)/2
23              end
24          end
```

Затем алгоритм сравнивает u и v, чтобы уменьшить их путем вычитания, а также обновляет коэффициенты. Если u больше или равно v, v вычитается из u, а коэффициенты A и B корректируются путем вычитания C и D. И наоборот, если v больше u, u вычитается из v, а C и D обновляются путем вычитания A и B.

```
25          while iseven(v)
26              v = v/2
27              if iseven(C) && iseven(D)
28                  C = C/2
29                  D = D/2
30              else
31                  C = (C+b)/2
32                  D = (D-a)/2
33              end
34          end
```

Вычисление конечного коэффициента полезного действия: Коэффициент полезного действия рассчитывается путем умножения v на g, которое включает все коэффициенты из 2, разделенные ранее.

```
        if u >= 0
            u = u - v
            A = A - C
            B = B - D
        else
            v = v - u
            C = C - A
            D = D - B
        end
    end
end
```

Запрашивает у пользователя ввод данных: a и b извлекаются пользователем, сохраняются в виде строк и преобразуются в целые числа с помощью parse(Int, a) и parse(Int, b). Вызов функции binary_extended: Коэффициенты БОГА d и Безу x и y вычисляются с помощью функции binary_extended.

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

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

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

3. Выводы

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.