

Лабораторная работа №4

Математические основы защиты информации и информационной безопасности

Дарижапов Тимур Андреевич

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	11
5	Список литературы	12

List of Tables

List of Figures

3.1	Программа реализации алгоритма Евклида	7
3.2	Программа реализации бинарного алгоритма Евклида	8
3.3	Программа расширенного алгоритма Евклида	9
3.4	Программа расширенного бинарного алгоритма Евклида	9
3.5	Результаты работы всех программ	10

1 Цель работы

Познакомиться с алгоритмами вычисления наибольшего общего делителя и практическая реализация алгоритмов.

2 Задание

Реализовать алгоритмы вычисления наибольшего общего делителя, рассмотренные в лабораторной работе.

3 Выполнение лабораторной работы

Данная работа была выполнена на языке Julia.

Для реализации алгоритма Евклида была написана следующая программа (рис. 3.1) :

```
In [4]: 1 number1 = 15
        2 number2 = 61
        3
        4 num1 = number1
        5 num2 = number2
        6 while num1 != 0 && num2 != 0
        7     if num1 >= num2
        8         num1 = num1 % num2
        9     else
       10         num2 = num2 % num1
       11     end
       12 end
       13
       14 nod1 = num1 + num2
       15 println(nod1)
```

1

Figure 3.1: Программа реализации алгоритма Евклида

В данной программе:

1-5 строки: задание чисел, НОД которого ищем.

6-12: реализация самого алгоритма Евклида: делим наибольшее число на наименьшее и записываем остаток до тех пор, пока одно из них не обнулится.

14: запись НОД в переменную.

15: вывод результата

Мы можем видеть результат на (рис. 3.1) . Программа работает верно.

Для реализации бинарного алгоритма Евклида была написана следующая программа (рис. 3.2)

```
In [5]: 1 num1 = number1
        2 num2 = number2
        3 shift = 0
        4
        5 while (num1 | num2) & 1 == 0
        6     shift += 1
        7     num1 >>= 1
        8     num2 >>= 1
        9 end
       10
       11 while num1 & 1 == 0
       12     num1 >>= 1
       13 end
       14
       15 while num2 != 0
       16     while num2 & 1 == 0
       17         num2 >>= 1
       18     end
       19     if num1 > num2
       20         num1, num2 = num2, num1
       21     end
       22     num2 -= num1
       23 end
       24
       25 println(num1 << shift)
```

1

Figure 3.2: Программа реализации бинарного алгоритма Евклида

В данной программе:

1-3 строки: задание чисел, НОД которого ищем, обнуление “сдвига”

5-23: реализация самого бинарного алгоритма Евклида: смотрим на четность получающихся чисел и записываем насколько нам необходимо “сдвинуть” число, чтобы получить НОД

25: сдвиг влево и вывод получившегося НОД.

Для реализации расширенного алгоритма Евклида была написана следующая программа (рис. 3.3)


```

In [21]: 1 function ext_evk(a, b)
          2     if b == 0
          3         return a, 1, 0
          4     else
          5         nod, x1, y1 = ext_evk_2(b, a % b)
          6         x = y1
          7         y = x1 - div(a, b) * y1
          8         return nod, x, y
          9     end
         10 end
         11 x, y, g = ext_evk(number1, number2)
         12 println(x, '*', number1, '+', '(', y, ')', '*', number2, '=', g)
1*15+(-4)*61=1

```

Figure 3.3: Программа расширенного алгоритма Евклида

1 строка: зададим рекурсивную функцию

2-3: если второе число равно нулю, возвращаем ответ из трёх чисел

5-7: в ином случае запускаем рекурсию, а затем выводим ответ согласно формуле на строке 7.

8: возвращаем вывод в качестве получившегося НОД, числа, что нужно домножить на первую цифру и на вторую, чтобы получить НОД.

11: вызов функции и сохранение данных в переменные

12: вывод на экран.

Для реализации расширенного бинарного алгоритма Евклида была написана следующая программа (рис. 3.4)

```

def ext_gcd(a, b):
    if a == 0: return 1, 1, b
    if b == 0: return 1, 1, a
    if not a & 1 | b & 1:
        x, y, g = ext_gcd(a >> 1, b >> 1)
        return x, y, g << 1
    elif not a & 1:
        x, y, g = ext_gcd(a >> 1, b)
        return (x - b >> 1, y + (a >> 1), g) if x & 1 else (x >> 1, y, g)
    elif not b & 1:
        x, y, g = ext_gcd(a, b >> 1)
        return (x + (b >> 1), y - a >> 1, g) if y & 1 else (x, y >> 1, g)
    elif b > a:
        x, y, g = ext_gcd(a, b - a)
        return x - y, y, g
    else:
        x, y, g = ext_gcd(a - b, b)
        return x, y - x, g

```

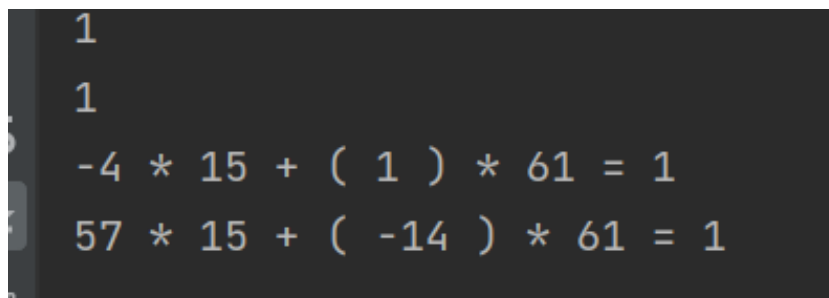
Figure 3.4: Программа расширенного бинарного алгоритма Евклида

Данная программа также работает рекурсивно, рассматривая 4 случая: 1) a четное 2) a нечетное и b четное 3) a нечетное и b нечетное, $b > a$ 4) a нечетное и b нечетное, $b < a$

Каждая рекурсия сдвигает биты в цифрах, формируя окончательный ответ.

В итоге выводит три значения: НОД, число, что нужно домножить на первую цифру и на вторую, чтобы получить НОД.

Выводы всех программ (рис. 3.5)



```
1
1
-4 * 15 + ( 1 ) * 61 = 1
57 * 15 + ( -14 ) * 61 = 1
```

Figure 3.5: Результаты работы всех программ

Необходимо обратить внимание, что расширенные алгоритмы выводят разные множители, однако оба ответа верны и дают верный НОД.

4 Выводы

Я познакомился с алгоритмами вычисления наибольшего общего делителя и практически реализовала алгоритмы.

5 Список литературы

Лабораторная работа №4

Вычисление наибольшего общего делителя [Электронный ресурс]. URL:
<https://esystem.rudn.ru/mod/folder/view.php?id=1150974>