Математические основы защиты информации и информационной безопасности. Отчет по лабораторной работе №4

Алгоритмы нахождения НОД

Юдин Герман Станиславович 1132236901

Содержание

List of Figures

List of Tables

1 Цель работы

Освоить на практике алгоритмы вычисления НОД.

2 Выполнение лабораторной работы

Требуется реализовать:

1. Алгоритм Евклида
2. Бинарный алгоритм Евклида
3. Расширенный алгоритм Евклида
4. Расширенный бинарный алгоритм Евклида

2.1 Алгоритм Евклида

Основные шаги:

1. Берёт два числа a и b, где a > b
2. Повторяет деление a на b, заменяя a значением b и b остатком от деления, пока b не станет равным 0.
3. Последнее ненулевое значение a будет НОД.

Реализация на Python предствлена на рисунке 1 fig. 2.1.

```python
def euclidean_gcd(a, b):
    r0, r1, i = a, b, 1
    while True:
        ri_next = r0 % r1
        if ri_next == 0:
            d = r1
            return d
        r0, r1, i = r1, ri_next, i + 1
```

Figure 2.1: gdc_eucl

2.2 Бинарный алгоритм Евклида

Основные шаги:

1. Если оба числа четные, делит оба числа на 2 и удваивает итоговый НОД
2. Если только одно из чисел четное, делит только его на 2.
3. Из большего числа вычитает меньшее.
4. Повторяет процесс, пока числа не станут равными. Это число становится НОД, умноженным на полученный ранее множитель.

Реализация на Python предствлена на рисунке 2 fig. 2.2.

```python
def binary_gcd(a, b):
    g = 1
    while a % 2 == 0 and b % 2 == 0:
        a, b, g = a // 2, b // 2, 2 * g

    u, v = a, b
    while u != 0:
        while u % 2 == 0:
            u //= 2
        while v % 2 == 0:
            v //= 2
        if u >= v:
            u -= v
        else:
            v -= u
    return g * v
```

Figure 2.2: gdc_binary

2.3 Расширенный алгоритм Евклида

Основные шаги:

1. Кроме нахождения НОД, алгоритм находит такие числа x и y, что ax+by=НОД(a,b).

2. Начинается с базовых коэф.: x0 = 1, y0 = 0 (для a) и x1 = 0, y1 = 1 (для b).

3. При каждом шаге обновляются значения коэффициентов, используя остаток и частное от деления.

Реализация на Python предствлена на рисунке 3 fig. 2.3.

```python
def extended_gcd(a, b):
    r0, r1, x0, x1, y0, y1, i = a, b, 1, 0, 0, 1, 1
    while True:
        q, ri_next = divmod(r0, r1)
        if ri_next == 0:
            return r1, x1, y1
        x_next = x0 - q * x1
        y_next = y0 - q * y1
        r0, r1, x0, x1, y0, y1, i = r1, ri_next, x1, x_next, y1, y_next, i + 1
```

Figure 2.3: gdc_extended

2.4 Расширенный бинарный алгоритм Евклида

Основные шаги:

1. Как и обычный бинарный алгоритм, но также отслеживает коэффициенты x и y.

2. Когда числа делятся на 2, коэффициенты корректируются соответствующим образом.

3. Когда из одного числа вычитается другое, соответствующие коэффициенты также вычитаются.

Реализация на Python предствлена на рисунке 4 fig. 2.4.

```python
def extended_binary_gcd(a, b):
    g = 1
    while a % 2 == 0 and b % 2 == 0:
        a, b, g = a // 2, b // 2, 2 * g

    u, v, A, B, C, D = a, b, 1, 0, 0, 1
    while u != 0:
        while u % 2 == 0:
            u //= 2
            if A % 2 == 0 and B % 2 == 0:
                A, B = A // 2, B // 2
            else:
                A, B = (A + b) // 2, (B - a) // 2

        while v % 2 == 0:
            v //= 2
            if C % 2 == 0 and D % 2 == 0:
                C, D = C // 2, D // 2
            else:
                C, D = (C + b) // 2, (D - a) // 2

        if u >= v:
            u, A, B = u - v, A - C, B - D
        else:
            v, C, D = v - u, C - A, D - B

    return g * v, C, D
```

Figure 2.4: gdc_extended_binary

2.5 Результат работы программы

Выходные значения программы fig. 2.5.

```python
import math

a, b = 56, 98

print("Using Euclidean Algorithm:", euclidean_gcd(a, b))
print("Using Binary Euclidean Algorithm:", binary_gcd(a, b))
print("Using Extended Euclidean Algorithm:", extended_gcd(a, b)[0])  # Just the gcd, not x and y
print("Using Extended Binary Euclidean Algorithm:", extended_binary_gcd(a, b)[0])  # Just the gcd, not x and y
print("Using Python's built-in math.gcd:", math.gcd(a, b))
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS F:\учеба 5 курс\информационная безопасность> & C:/Python311/python.exe "f:/учеба 5 курс/информационная безопасность/Lab4_gcd/GCD.py"
Using Euclidean Algorithm: 14
Using Binary Euclidean Algorithm: 14
Using Extended Euclidean Algorithm: 14
Using Extended Binary Euclidean Algorithm: 14
Using Python's built-in math.gcd: 14
PS F:\учеба 5 курс\информационная безопасность>
```

Figure 2.5: output

3 Выводы

В результате выполнения работы я освоил на практике применение алгоритмов нахождения НОД.

4 Список литературы

1. Методические материалы курса