Отчёт по лабораторной работе №4

Вычисление наибольшего общего делителя

Каймакджыоглу Мерич Дорук

Содержание

1 Цель работы **5**

2 Задание **6**

3 Теоретическое введение **7**

4 Выполнение лабораторной работы **9**

5 Выводы **14**

Список литературы **15**

Список иллюстраций

Список таблиц

1 Цель работы

Целью данной лабораторной работы является изучение и практическая реализация фундаментальных алгоритмов вычисления наибольшего общего делителя (НОД).

Особое внимание уделяется не только классическому алгоритму Евклида, но и его бинарной (более быстрой) версии, а также расширенным вариантам обоих алгоритмов.

Понимание и реализация расширенного алгоритма Евклида имеет критическое значение, поскольку он составляет математическую основу для вычисления модульных обратных элементов, что является ключевой операцией в асимметричных криптосистемах, таких как RSA.

2 Задание

Реализовать программно все рассмотренные алгоритмы: - Классический алгоритм Евклида - Бинарный алгоритм Евклида - Расширенный алгоритм Евклида для нахождения d, x, y таких, что $ax + by = d$ - Расширенный бинарный алгоритм Евклида Обеспечить корректную работу алгоритмов для целых положительных чисел a, b при $0 < b \leq a$ Продемонстрировать работу алгоритмов на примерах.

3 Теоретическое введение

Наибольший общий делитель (НОД) двух целых чисел a и b — это наибольшее целое число d, на которое делятся и a, и b без остатка. Все алгоритмы, рассмотренные в работе, так или иначе, базируются на основной лемме Евклида, которая следует из операции деления с остатком: $a = qb+r$. Из этого равенства следует, что НОД$(a, b) =$ НОД(b, r).

1. **Алгоритм Евклида** рекурсивно применяет эту лемму, заменяя пару (a, b) парой $(b, a \mod b)$ до тех пор, пока остаток не станет равен нулю. Последний ненулевой остаток и является НОД.
2. **Бинарный алгоритм Евклида** оптимизирует этот процесс для вычислительных машин, заменяя дорогостоящую операцию деления на быстрые битовые сдвиги (деление на 2) и вычитание. Он использует следующие свойства:
   - НОД$(a, b) = 2 \cdot$ НОД$(a/2, b/2)$, если a, b четные.
   - НОД$(a, b) =$ НОД$(a, b/2)$, если a нечетное, b четное.
   - НОД$(a, b) =$ НОД$(a - b, b)$, если a, b нечетные и $a > b$.
3. **Расширенный алгоритм Евклида** не только находит $d =$ НОД(a, b), но и находит пару целых чисел (x, y), известных как коэффициенты Безу, которые удовлетворяют тождеству:

   $$ax + by = d$$

   Это тождество является представлением НОД в виде линейной комбинации исходных чисел. Если НОД$(a, b) = 1$, то $ax + by = 1$. Взяв это уравнение по модулю b, получаем $ax \equiv 1 \pmod{b}$, что означает, что x является модульным мультипликативным обратным для a по модулю b.

4 Выполнение лабораторной работы

Для выполнения задания был разработан единый скрипт на языке Python.

1. Классический алгоритм Евклида

```python
def euclidean_gcd(a, b):
    r_prev, r_curr = a, b
    # prev = a, curr = b

    while r_curr != 0:
        r_next = r_prev % r_curr
        r_prev = r_curr
        r_curr = r_next

    d = r_prev

    return d

vala = 12345
valb = [24690, 54321, 12541]
for val in valb:
    print(f"GCD({vala}, {val}) = {euclidean_gcd(vala, val)}")
```

2. Бинарный алгоритм Евклида

```python
def binaru_gcd(a, b):
    g = 1

    while a % 2 == 0 and b % 2 == 0:
        a //= 2
        b //= 2
        g *= 2

    u, v = a, b

    while u != 0:
        while u % 2 == 0:
            u //= 2
        while v % 2 == 0:
            v //= 2
        if u >= v:
            u = u - v
        else:
            v = v - u

    d = g * v

    return d

vala = 12345
valb = [24690, 54321, 12541]
for val in valb:
    print(f"GCD({vala}, {val}) = {binaru_gcd(vala, val)}")
```

3. Расширенный алгоритм Евклида

```python
# test = 91, 105, 154 == 7

def ext_euc_gcd(a, b):
    r_prev, r_curr = a, b
    x_prev, x_curr = 1, 0
    y_prev, y_curr = 0, 1

    while r_curr != 0:
        q = r_prev // r_curr
        r_next = r_prev % r_curr

        x_next = x_prev - q * x_curr
        y_next = y_prev - q * y_curr

        r_prev, r_curr = r_curr, r_next
        x_prev, x_curr = x_curr, x_next
        y_prev, y_curr = y_curr, y_next

    d, x, y = r_prev, x_prev, y_prev

    return d, x, y

d1, x1, y1 = ext_euc_gcd(105, 91)
d2, x2, y2 = ext_euc_gcd(154, d1)
print(f"GDC(105, 91) = {d1, x1, y1}")
print(f"final GDC(154, 7) = {d2, x2, y2}")
```

4. Расширенный бинарный алгоритм Евклида

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

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

    while u != 0:
        while u % 2 == 0:
            u //= 2
            if A % 2 == 0 and B % 2 == 0:
                A //= 2
                B //= 2
            else:
                A = (A + b) // 2
                B = (B - a) // 2
            print("v: " + str(v))

        while v % 2 == 0:
            v //= 2
            if C % 2 == 0 and D % 2 == 0:
                C //= 2
                D //= 2
            else:
```

```
                C = (C + b) // 2
                D = (D - a) // 2
            print("u: " + str(u))

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

dbin, xbin, ybin = ext_bin_gcd(105, 91)
print(f"GCD(105, 91) = {dbin, xbin, ybin}")
```

5 Выводы

В ходе выполнения данной лабораторной работы были успешно реализованы все четыре алгоритма вычисления НОД, указанные в задании. Разработанные функции корректно вычисляют как сам НОД, так и коэффициенты Безу (x,y) для линейного представления ax+by=d.

Программная реализация бинарных алгоритмов продемонстрировала, как можно заменить дорогостоящие операции деления и умножения на более эффективные с точки зрения вычислений операции сдвига, сложения и вычитания. Наиболее важным результатом является практическая реализация расширенного алгоритма Евклида, который является краеугольным камнем для многих криптографических протоколов. Эта работа позволила углубить понимание математических основ, на которых строится информационная безопасность, в частности, механизм получения мультипликативного обратного элемента в конечном поле, что напрямую используется в алгоритме RSA.

Список литературы

::: {#Методические указания к лабораторной работе №4} :::