Отчёт по лабораторной работе №4

Вычисление наибольшего общего делителя

Каймакджыоглу Мерич Дорук

Содержание

# 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 таких, что - Расширенный бинарный алгоритм Евклида Обеспечить корректную работу алгоритмов для целых положительных чисел при Продемонстрировать работу алгоритмов на примерах.

# 3. Теоретическое введение

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

1. **Алгоритм Евклида** рекурсивно применяет эту лемму, заменяя пару парой 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:
   * , если четные.
   * , если нечетное, четное.
   * , если нечетные и .
3. **Расширенный алгоритм Евклида** не только находит , но и находит пару целых чисел , известных как коэффициенты Безу, которые удовлетворяют тождеству:

* Это тождество является представлением НОД в виде линейной комбинации исходных чисел. Если , то . Взяв это уравнение по модулю , получаем , что означает, что является модульным мультипликативным обратным для по модулю .

# 4. Выполнение лабораторной работы

Для выполнения задания был разработан единый скрипт на языке Python.

1. Классический алгоритм Евклида

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)}")

1. Бинарный алгоритм Евклида

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)}")

1. Расширенный алгоритм Евклида

# 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}")

1. Расширенный бинарный алгоритм Евклида

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} :::