Лабораторная работа №4

Вычисление наибольшего общего делителя

Доборщук Владимир Владимирович, НФИмд-02-22

Содержание

# 1 Цель и задачи работы

Изучить алгоритмы вычисления наибольшего общего делителя

**Задачи:**

* Реализовать все представленные алгоритмы вычисления наибольшего общего делителя.

# 2 Теоретическая информация

Целое число называется *наибольшим общим делителем* целых чисел , если выполняются следующие условия:

1. каждое из чисел делится на ;
2. если - другой общий делитель чисел , то делится на .

Для любых целых чисел существует наибольший общий делитель и его можно представить в виде *линейной комбинации* этих чисел:

Существует ряд алгоритмов, позволяющих вычислить значение наибольшего общего делителя для пары целых чисел :

* Алгоритм Евклида;
* Бинарный алгоритм Евклида;
* Расширенный алгоритм Евклида;
* Расширенный бинарный алгоритм Евклида.

# 3 Выполнение лабораторной работы

Для реализации шифров мы будем использовать Python, так как его синтаксис позволяет быстро реализовать необходимые нам алгоритмы.

На вход каждой функции реализации алгоритмов у нас на вход подается пара ненулевых целых чисел и . Также мы учитываем, что если у нас выполняется неравенство , то мы меняем значения соответствующих переменных местами, для корректности выполнения алгоритмов.

Все реализации соответствуют алгоритмам, представленным в описании лабораторной работы.

## 3.1 Реализация алгоритма Евклида

*Алгоритм Евклида* реализуем в виде функции euclid следующего вида:

# --- Euclid Algorithm ---  
def euclid(a: int, b: int):  
 if a == 0 or b == 0:  
 print("Non zero numbers should be used")  
 return  
   
 r\_0 = a  
 r\_1 = b  
   
 if a < b:  
 r\_0, r\_1 = r\_1, r\_0  
   
 while True:  
 r = r\_0 % r\_1  
 if r == 0:  
 return r\_1  
   
 r\_0 = r\_1  
 r\_1 = r

## 3.2 Реализация бинарного алгоритма Евклида

*Бинарный алгоритм Евклида* реализуем в виде функции binary\_euclid следующего вида:

# --- Binary Euclid Algorithm ---  
def binary\_euclid(a: int, b: int):  
 if a == 0 or b == 0:  
 print("Non zero numbers should be used")  
 return  
   
 r\_0 = a  
 r\_1 = b  
   
 if a < b:  
 r\_0, r\_1 = r\_1, r\_0  
   
 g = 1  
   
 while r\_0 % 2 == 0 and r\_1 % 2 == 0:  
 r\_0 = int(r\_0 / 2)  
 r\_1 = int(r\_1 / 2)  
 g = 2\*g  
   
 u, v = r\_0, r\_1  
  
 while u != 0:  
 while u % 2 == 0:  
 u = int(u / 2)  
 while v % 2 == 0:  
 v = int(v / 2)  
 if u >= v:  
 u = u - v  
 else:  
 v = v - u  
   
 return g\*v

## 3.3 Реализация расширенного алгоритма Евклида

*Расширенный алгоритм Евклида* реализуем в виде функции extended\_euclid следующего вида:

# --- Extended Euclid Algorithm ---  
def extended\_euclid(a: int, b: int):  
 if a == 0 or b == 0:  
 print("Non zero numbers should be used")  
 return  
   
 r\_0, r\_1 = a, b  
   
 x = [1, 0]  
 y = [0, 1]  
   
 if a < b:  
 r\_0, r\_1 = r\_1, r\_0  
 x, y = y, x  
   
 while True:  
 r = r\_0 % r\_1  
 if r == 0:  
 return (r\_1, x[1], y[1])  
   
 q = int((r\_0 - r)/r\_1)  
 r\_0 = r\_1  
 r\_1 = r  
   
 x\_ = x[0] - q\*x[1]  
 x[0] = x[1]  
 x[1] = x\_  
   
 y\_ = y[0] - q\*y[1]  
 y[0] = y[1]  
 y[1] = y\_

## 3.4 Реализация расширенного бинарного алгоритма Евклида

*Расширенный бинарный алгоритм Евклида* реализуем в виде функции extended\_binary\_euclid следующего вида:

# --- Extended Binary Euclid Algorithm ---  
def extended\_binary\_euclid(a: int, b: int):  
 if a == 0 or b == 0:  
 print("Non zero numbers should be used")  
 return  
   
 r\_0 = a  
 r\_1 = b  
   
 A, B, C, D = 1, 0, 0, 1  
   
 if a < b:  
 r\_0, r\_1 = r\_1, r\_0  
   
 g = 1  
   
 while r\_0 % 2 == 0 and r\_1 % 2 == 0:  
 r\_0 = int(r\_0 / 2)  
 r\_1 = int(r\_1 / 2)  
 g = 2\*g  
   
 u, v = r\_0, r\_1  
  
 while u != 0:  
 while u % 2 == 0:  
 u = int(u / 2)  
 if A % 2 ==0 and B % 2 == 0:  
 A = int(A / 2)  
 B = int(B / 2)  
 else:  
 A = int((A + r\_1) / 2)  
 B = int((B - r\_0) / 2)  
 while v % 2 == 0:  
 v = int(v / 2)  
 if C % 2 ==0 and D % 2 == 0:  
 C = int(C / 2)  
 D = int(D / 2)  
 else:  
 C = int((C + r\_1) / 2)  
 D = int((D - r\_0) / 2)  
 if u >= v:  
 u = u - v  
 A = A - C  
 B = B - D  
 else:  
 v = v - u  
 C = C - A  
 D = D - B  
   
 if a < b:  
 C, D = D, C  
   
 return (g\*v, C, D)

## 3.5 Тестирование

Для тестирования мы создали следующие функции, которые вызываем в блоке *Main*:

# --- Tests ---  
  
def test\_euclid(a: list, b: list):  
 print("EUCLID ALGORITHM\n---")  
 result = list(map(lambda a, b: euclid(a, b), a, b))  
 for i in range(0, len(a)):  
 print(f'НОД({a[i]}, {b[i]}) = {result[i]}')  
 print("---\n")  
   
def test\_binary\_euclid(a: list, b: list):  
 print("BINARY EUCLID ALGORITHM\n---")  
 result = list(map(lambda a, b: binary\_euclid(a, b), a, b))  
 for i in range(0, len(a)):  
 print(f'НОД({a[i]}, {b[i]}) = {result[i]}')  
 print("---\n")  
  
def test\_extended\_euclid(a: list, b: list):  
 print("EXTENDED EUCLID ALGORITHM\n---")  
 result = list(map(lambda a, b: extended\_euclid(a, b), a, b))  
 for i in range(0, len(a)):  
 print(f'НОД({a[i]}, {b[i]}) = {a[i]} \* ({result[i][1]}) + {b[i]} \* ({result[i][2]}) = {result[i][0]}')  
 print("---\n")  
   
def test\_extended\_binary\_euclid(a: list, b: list):  
 print("EXTENDED BINARY EUCLID ALGORITHM\n---")  
 result = list(map(lambda a, b: extended\_binary\_euclid(a, b), a, b))  
 for i in range(0, len(a)):  
 print(f'НОД({a[i]}, {b[i]}) = {a[i]} \* ({result[i][1]}) + {b[i]} \* ({result[i][2]}) = {result[i][0]}')  
 print("---\n")

Данные тесты получают на вход списки чисел и и возврщают строки с результатом нахождения в качестве результата.

Для расширенных алгоритмов, мы также получаем линейные комбинации пар чисел для получения наибольшего общего делителя.

Для их вызова, реализуем функцию main следующим образом:

# --- Main ---  
  
def main():  
 a = [16, 3, 91]  
 b = [20, 21, 105]  
   
 test\_euclid(a,b)  
 test\_binary\_euclid(a,b)  
 test\_extended\_euclid(a,b)  
 test\_extended\_binary\_euclid(a,b)

## 3.6 Результаты тестирования

Запустив наш программный код, получим результат, изображенный на рисунке 1.

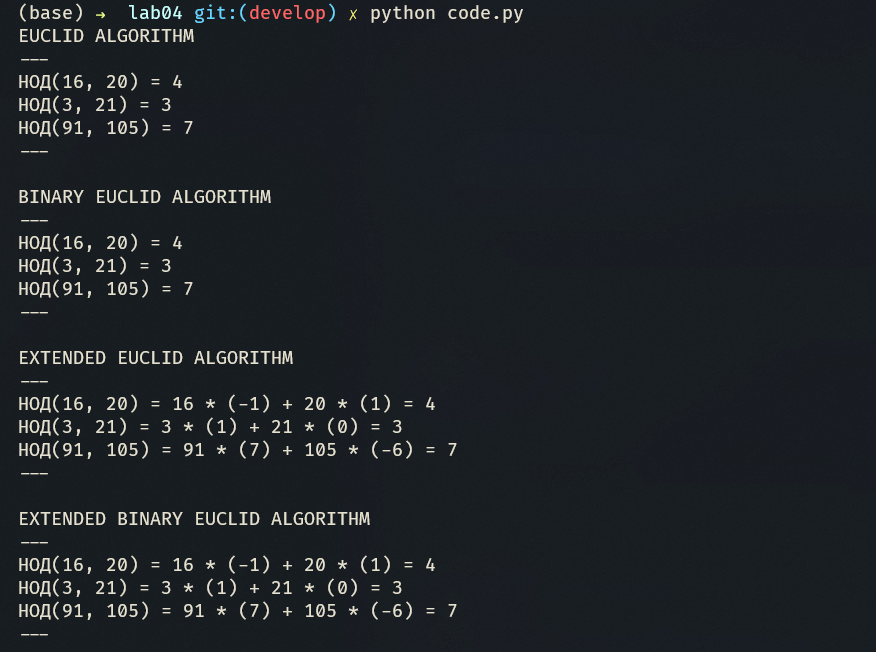


Рис. 1: Вывод результата тестирования алгоритмов нахождения

Явно получим вот такой результат:

EUCLID ALGORITHM  
---  
НОД(16, 20) = 4  
НОД(3, 21) = 3  
НОД(91, 105) = 7  
---  
  
BINARY EUCLID ALGORITHM  
---  
НОД(16, 20) = 4  
НОД(3, 21) = 3  
НОД(91, 105) = 7  
---  
  
EXTENDED EUCLID ALGORITHM  
---  
НОД(16, 20) = 16 \* (-1) + 20 \* (1) = 4  
НОД(3, 21) = 3 \* (1) + 21 \* (0) = 3  
НОД(91, 105) = 91 \* (7) + 105 \* (-6) = 7  
---  
  
EXTENDED BINARY EUCLID ALGORITHM  
---  
НОД(16, 20) = 16 \* (-1) + 20 \* (1) = 4  
НОД(3, 21) = 3 \* (1) + 21 \* (0) = 3  
НОД(91, 105) = 91 \* (7) + 105 \* (-6) = 7  
---

Ручная проверка дает идентичные результаты, из чего можем сделать вывод, что алгоритмы реализованы корректно.

# 4 Выводы

В рамках выполненной лабораторной работы мы изучили и реализовали алгоритмы вычисления наибольшего общего делителя.

# Список литературы