РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Отчёт по лабораторной работе №4. Вычисление наибольшего общего делителя

Дисциплина: *Математические основы защиты информации и информационной безопасности*

Студент: Банникова Екатерина Алексеевна
Группа: НФИмд-02-23

Москва 2023

Содержание

List of Figures

List of Tables

1 Цель работы

Целью данной лабораторной работы является ознакомление с алгоритмами вычисления наибольшего общего делителя, – а так же реализация алгоритмов на произвольном языке программирования.

2 Задание

Реализовать все рассмотренные в инструкции к лабораторной работе алгоритмы нахождения наибольшего общего делителя программно.

3 Теоретическое введение

Рассмотрим, что такое наибольший общий делитель. Вспомним, что делитель – это число, на которое другое число делится без остатка. **Наибольшим общим делителем (НОД)** для двух целых чисел m и n называется наибольший из их общих делителей. Пример: для чисел 54 и 24 наибольший общий делитель равен 6.

Алгоритм Евклида — один из наиболее ранних численных алгоритмов в истории. Название было дано в честь греческого математика Евклида, который впервые дал ему описание в книгах «Начала». Изначально назывался «взаимным вычитанием», так как его принцип заключался в последовательном вычитании из большего числа меньшего, пока в результате не получится ноль. Сегодня чаще используется взятие остатка от деления вместо вычитания, но суть метода сохранилась.

Алгоритм представлен в следующей блок-схеме:

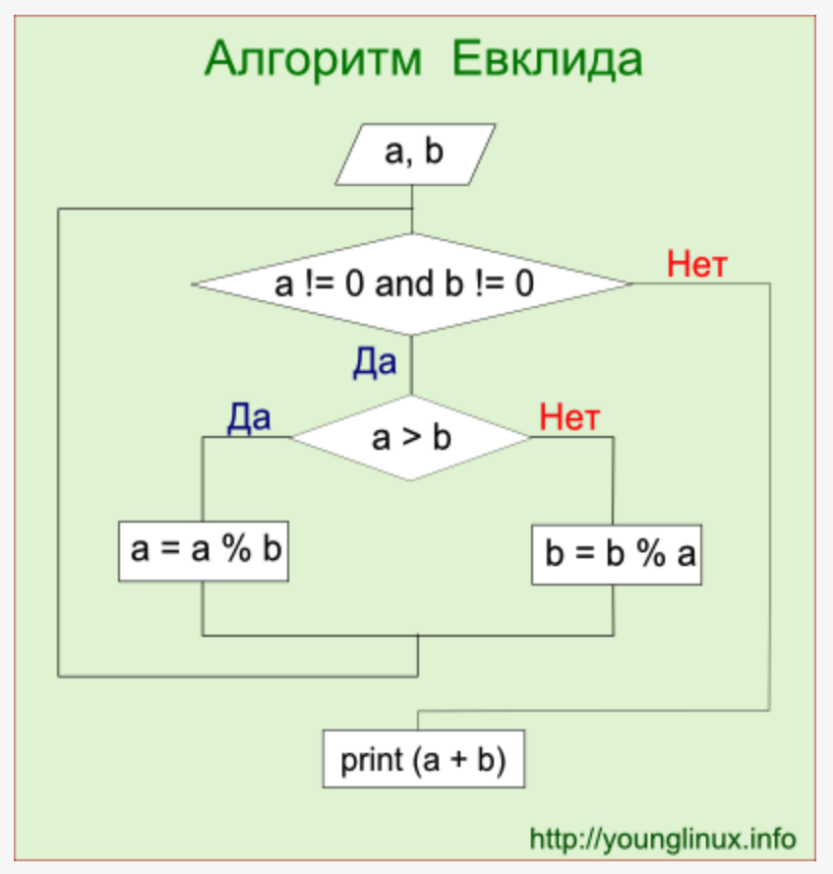

Figure 3.1: Блок-схема алгоритма Евклида

Бинарный алгоритм Евклида — метод нахождения наибольшего общего делителя двух целых чисел. Данный алгоритм “быстрее” обычного алгоритма Евклида, т.к. вместо медленных операций деления и умножения используются сдвиги. Возможно, алгоритм был известен еще в Китае 1-го века, но опубликован был лишь в 1967 году израильским физиком и программистом Джозефом Стайном.

Алгоритм реализации алгоритма достаточно хорошо раскрыт в описании лабораторной работы, представленном на ТУИС.

Расширенные алгоритмы основаны на “уменьшенной версии” алгоритмов. Подробное описание представлено в описании лабораторной работы, представленном на ТУИС.

4 Выполнение лабораторной работы

Примечание: комментарии по коду представлены на скриншотах к каждому из проделанных заданий.

В соответствии с заданием, были написаны программы реализации алгоритмов нахождения наибольшего общего делителя. Нами были рассмотрены следующие алгоритмы: 1. Алгоритм Евклида; 2. Бинарный алгоритм Евклида; 3. Расширенный алгоритм Евклида; 4. Расширенный бинарный алгоритм Евклида.

Программный код и результаты выполнения программ представлен ниже.

4.1 Алгоритм Евклида

```
#ввели числа для поиска НОДа
a=27
b=9
```

Figure 4.1: Входные данные для реализации алгоритмов по нахождению НОД

```python
#алгоритм Евклида
def algorithm_Evklida(a,b):
  '''
  Расписываем пункты 1-4 для алгоритма Евклида
  '''
  r=[]
  r.append(a)
  r.append(b)
  i=1
  while r[i]!=0:
    i+=1
    r.append(r[i-2]%r[i-1])
  d=r[i-1]
  print('НОД(',a,',',b,')=',d)
algorithm_Evklida(a,b)
```

Figure 4.2: Реализация алгоритма Евклида для нахождения НОД

Результаты выполнения программы представлены ниже.

Figure 4.3: Результат реализации алгоритма Евклида для нахождения НОД

4.2 Бинарный алгоритм Евклида. 1 способ

```
#бинарный алгоритм Евклида
'''
Выпишем два случая реализации программы для бинарного алгоритма Евклида
'''
#1 случай (с использованием алгоритма Евклида и ограничивающих условий)
def bin1_algorithm_Evklida(a,b):
  def NOD(a,b):
    '''
    Расписываем пункты 1-4 для алгоритма Евклида
    '''
    r=[]
    r.append(a)
    r.append(b)
    i=1
    while r[i]!=0:
      i+=1
      r.append(r[i-2]%r[i-1])
    d=r[i-1]
    return d
```

Figure 4.4: 1 часть программного кода реализации бинарного алгоритма Евклида 1 способом для нахождения НОД

```
  if ((b>0)and(a>=b)):
    '''проверка на обязательное условие'''
    if ((a%2==0)and(b%2==0)):
      '''проверка на четность'''
      print('НОД(',a,',',b,')=',2*NOD(a//2,b//2))
    if ((a%2!=0)and(b%2==0)):
      '''проверка на то, что a нечетное, а b четное'''
      print('НОД(',a,',',b,')=',NOD(a,b//2))
    if ((a%2!=0)and(b%2!=0)and(a>b)):
      '''проверка на то, что оба нечетные'''
      print('НОД(',a,',',b,')=',NOD(a-b,b))
    if (a==b):
      '''если числа равны друг другу'''
      print('НОД(',a,',',b,')=',a)
  else:
    print('Перепроверьте входные данные! Должно быть: (0<b<=a)')
bin1_algorithm_Evklida(a,b)
```

Figure 4.5: 2 часть программного кода реализации бинарного алгоритма Евклида 1 способом для нахождения НОД

Результаты выполнения программы представлены ниже.

Figure 4.6: Результат реализации бинарного алгоритма Евклида для нахождения НОД (1 способ)

4.3 Бинарный алгоритм Евклида. 2 способ

```
#бинарный алгоритм Евклида
#2 случай (по алгоритму)
def bin2_algorithm_Evklida(a,b):
  '''
  Выполняем пункты 1-6 по алгоритму
  '''
  g=1
  u=a
  v=b
  while ((a%2==0)and(b%2==0)):
    '''
    пока четные выполнять до получения хотя бы одного нечетного
    '''
    a=a//2
    b=b//2
    g=2*g
```

Figure 4.7: 1 часть программного кода реализации бинарного алгоритма Евклида 2 способом для нахождения НОД

```
  while (u!=0):
    '''
    выполняем пункт 4 алгоритма
    '''
    if (u%2==0):
      u=u//2
    else:
      u=u
    if (v%2==0):
      v=v//2
    else:
      v=v
    if u>=v:
      u=u-v
    else:
      v=v-u
  d=g*v
  return d
if ((b>0)and(a>=b)):
  print('НОД(',a,',',b,')=',bin2_algorithm_Evklida(a,b))
else:
  print('Перепроверьте входные данные! Должно быть: (0<b<=a)')
```

Figure 4.8: 2 часть программного кода реализации бинарного алгоритма Евклида 2 способом для нахождения НОД

Результаты выполнения программы представлены ниже.

```
НОД( 27 , 9 )= 9
```

Figure 4.9: Результат реализации бинарного алгоритма Евклида для нахождения НОД (2 способ)

4.4 Расширенный алгоритм Евклида

```
#расширенный алгоритм Евклида
def rassh_algorithm_Evklida(a,b):
  '''
  выполняем пункты 1-4 по расширенному алгоритму Евклида
  '''
  #пункт 1
  r=[]
  x=[]
  y=[]
  r.append(a)
  r.append(b)
  x.append(1)
  x.append(0)
  y.append(0)
  y.append(1)
  i=1
  while r[i]!=0:
    '''
    пункт 2
    '''
    i+=1
    r.append(r[i-2]%r[i-1])
```

Figure 4.10: 1 часть программного кода реализации расширенного алгоритма Евклида для нахождения НОД

```
    if r[i]==0:
      '''
      пункт 3
      '''
      d=r[i-1]
      x=x[i-1]
      y=y[i-1]
    else:
      x.append(x[i-2]-((r[i-2]//r[i-1])*x[i-1]))
      y.append(y[i-2]-((r[i-2]//r[i-1])*y[i-1]))
  return d,x,y
if ((b>0)and(a>=b)):
  print('НОД(',a,',',b,')=',rassh_algorithm_Evklida(a,b)[0])#тк три значения в return пишем индекс ответа, который нужен
  print('x=',rassh_algorithm_Evklida(a,b)[1])#тк три значения в return пишем индекс ответа, который нужен
  print('y=',rassh_algorithm_Evklida(a,b)[2])#тк три значения в return пишем индекс ответа, который нужен
else:
  print('Перепроверьте входные данные! Должно быть: (0<b<=a)')
```

Figure 4.11: 2 часть программного кода реализации расширенного алгоритма Евклида для нахождения НОД

Результаты выполнения программы представлены ниже.

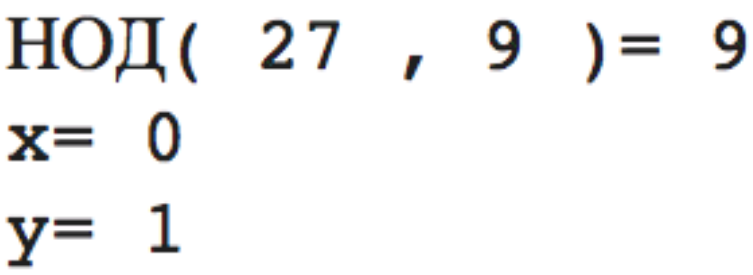

Figure 4.12: Результат реализации расширенного алгоритма Евклида для нахождения НОД

4.5 Расширенный бинарный алгоритм Евклида

```
#расширенный бинарный алгоритм Евклида
def rassh_bin_algorithm_Evklida(a,b):
  '''
  6 пунктов алгоритма к выполнению
  '''
  g=1
  u=a
  v=b
  A=1
  B=0
  C=0
  D=1
  while ((a%2==0)and(b%2==0)):
    '''
    пока четные выполнять до получения хотя бы одного нечетного
    '''
    a=a//2
    b=b//2
    g=2*g
  '''
  пока u не равно 0
  '''
```

Figure 4.13: 1 часть программного кода реализации расширенного бинарного алгоритма Евклида для нахождения НОД

```
  while (u!=0):
    #пока u четное
    if (u%2==0):
      u=u//2
      #если оба A и B четные
      if ((A%2==0)and(B%2==0)):
        A=A//2
        B=B//2
      else:
        A=(A+b)//2
        B=(B-a)//2
    else:
      u=u
    #пока v четное
    if (v%2==0):
      v=v//2
      #если оба C и D четные
      if ((C%2==0)and(D%2==0)):
        C=C//2
        D=D//2
      else:
        C=(C+b)//2
        D=(D-a)//2
    else:
      v=v
```

Figure 4.14: 2 часть программного кода реализации расширенного бинарного алгоритма Евклида для нахождения НОД

```
    #пункт 4.3
    if u>=v:
      u=u-v
      A=A-C
      B=B-D
    else:
      v=v-u
      C=C-A
      D=D-B
  #пункт 5
  d=g*v
  x=C
  y=D
  return d,x,y
if ((b>0)and(a>=b)):
  print('НОД(',a,',',b,')=',rassh_bin_algorithm_Evklida(a,b)[0])
  print('x=',rassh_bin_algorithm_Evklida(a,b)[1])
  print('y=',rassh_bin_algorithm_Evklida(a,b)[2])
else:
  print('Перепроверьте входные данные! Должно быть: (0<b<=a)')
```

Figure 4.15: 3 часть программного кода реализации расширенного бинарного алгоритма Евклида для нахождения НОД

Результаты выполнения программы представлены ниже.

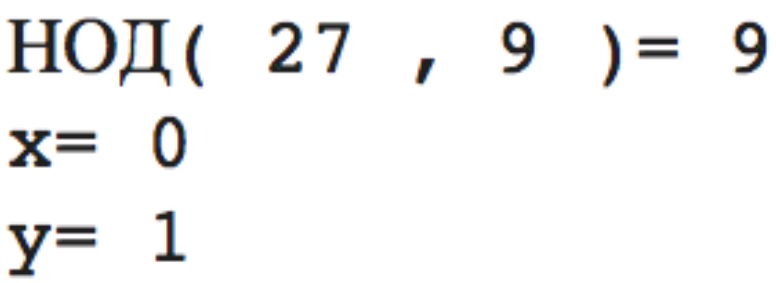

Figure 4.16: Результат реализации расширенного бинарного алгоритма Евклида для нахождения НОД

5 Выводы

Таким образом, была достигнута цель, поставленная в начале лабораторной работы: я ознакомилась с алгоритмами вычисления наибольшего общего делителя, – а так же реализовала данные алгоритмы на языке программирования Python 3.

Список литературы

1. Найденов А. Наибольший общий делитель [Электронный ресурс]. URL: https://blog.tutoronline.ru/naibolshij-obshhij-delitel.
2. Википедия. Наибольший общий делитель [Электронный ресурс]. Википедия, свободная энциклопедия. URL: https://en.wikipedia.org/wiki/Greatest_common_divisor.
3. Феникс. Алгоритм Евклида [Электронный ресурс]. Феникс. URL: https://wiki.fenix.help/matematika/algoritm-evklida.
4. Линуксоида Л. Алгоритм Евклида - нахождение наибольшего общего делителя [Электронный ресурс]. Лаборатория Линуксоида. URL: https://younglinux.info/algorithm/euclidean.