Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ
ПО ПРАКТИЧЕСКОЙ РАБОТЕ №2
дисциплины «Алгоритмизация»
Вариант 9

Выполнил:
Дудкин Константин Александрович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»
направление «Программное
обеспечение средств вычислительной
техники и автоматизированных
систем»,
очная форма обучения

_____
(подпись)

Руководитель практики:
доцент кафедры инфокоммуникаций,
кандидат технических наук
Воронкин Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Порядок выполнения работы

Написал два алгоритма расчета чисел Фиббоначи — наивный и оптимизированный, и сравнил время их выполнения. Сделал графики выполнения к каждому из них:

```
def F1(n):
    if n == 0: return 0
    elif n == 1: return 1
    else: return (F1(n-1)+F1(n-2))
```

Рисунок 1. Наивный алгоритм Фиббоначи

```
def F2(n, dic = {0: 0, 1: 1}):
    if n not in dic:
        dic[n] = F2(n-1, dic) + F2(n-2, dic)
    return dic[n]
```

Рисунок 2. Оптимизированный алгоритм Фиббоначи

```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import datetime
import timeit
import xlwt
from xlwt import Workbook

wb = Workbook()
sheet1 = wb.add_sheet("Sheet 1")

3 usages  new *
def F1(n):
    if n == 0: return 0
    elif n == 1: return 1
    else: return (F1(n-1)+F1(n-2))

3 usages  new *
def F2(n, dic = {0: 0, 1: 1}):
    if n not in dic:
        dic[n] = F2(n-1, dic) + F2(n-2, dic)
    return dic[n]

if __name__ == '__main__':
    measure1 = {0:0,1:0}
    measure2 = {0:0,1:0}

    for i in range(2, 35, 1):
        st = datetime.datetime.now()
        F1(i)
        ed = datetime.datetime.now()
        elapsed = ed-st
        measure1[i] = elapsed.total_seconds()
        sheet1.write(i,0,i)
        sheet1.write(i,1,elapsed.total_seconds())

        a = 1000000
        elapsed = timeit.timeit(lambda: F2(i),number=a)
        measure2[i] = elapsed/a
        sheet1.write(i,3,i)
        sheet1.write(i,4,elapsed/a)
```

Рисунок 3. Полный код расчета времени выполнения алгоритмов (часть 1)

```
40
41      print(measure1)
42      print(measure2)
43      wb.save("Fibbonachi.xls"
```

Рисунок 4. Полный код расчета времени выполнения алгоритмов (часть 2)

2	0,000009		2	1,0539E-07
3	0,000009		3	1,07572E-07
4	0,000017		4	1,09239E-07
5	0,000009		5	1,09237E-07
6	0,000009		6	1,09301E-07
7	0,000007		7	1,09426E-07
8	0,000008		8	1,0963E-07
9	0,000012		9	1,09373E-07
10	0,000012		10	1,09888E-07
11	0,000018		11	1,10439E-07
12	0,000028		12	1,10968E-07
13	0,000043		13	1,1124E-07
14	0,000062		14	1,09869E-07
15	0,000087		15	1,09748E-07
16	0,000137		16	1,11183E-07
17	0,000225		17	1,11888E-07
18	0,00036		18	1,1154E-07
19	0,000582		19	1,12436E-07
20	0,000939		20	1,12372E-07
21	0,001518		21	1,13179E-07
22	0,002465		22	1,16694E-07
23	0,004193		23	1,17966E-07
24	0,006903		24	1,15706E-07
25	0,010601		25	1,13073E-07
26	0,01742		26	1,12792E-07
27	0,02832		27	1,12536E-07
28	0,046082		28	1,11752E-07
29	0,074578		29	1,11684E-07
30	0,121474		30	1,12031E-07
31	0,208859		31	1,05239E-07
32	0,297272		32	1,05069E-07
33	0,482221		33	1,05306E-07
34	0,78397		34	1,05096E-07

Рисунок 5. Результат измерений

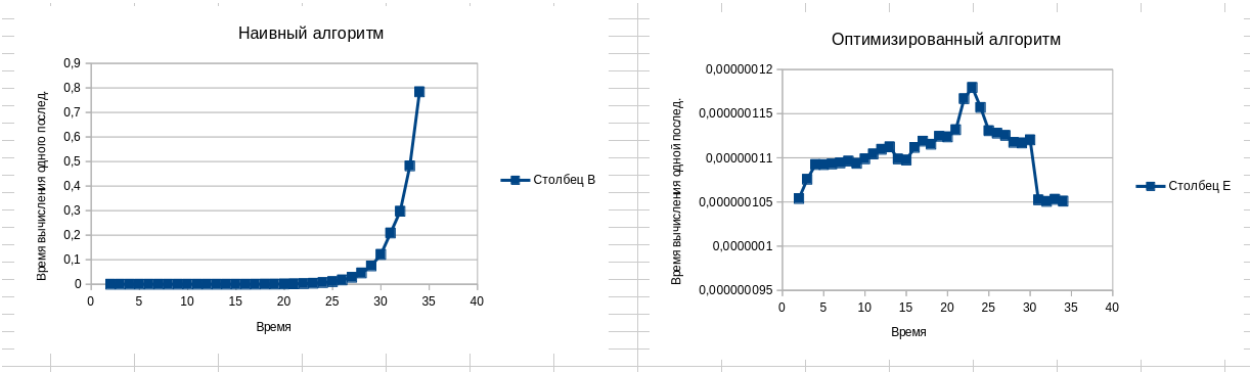


Рисунок 6. Графики измерения времени выполнения алгоритмов

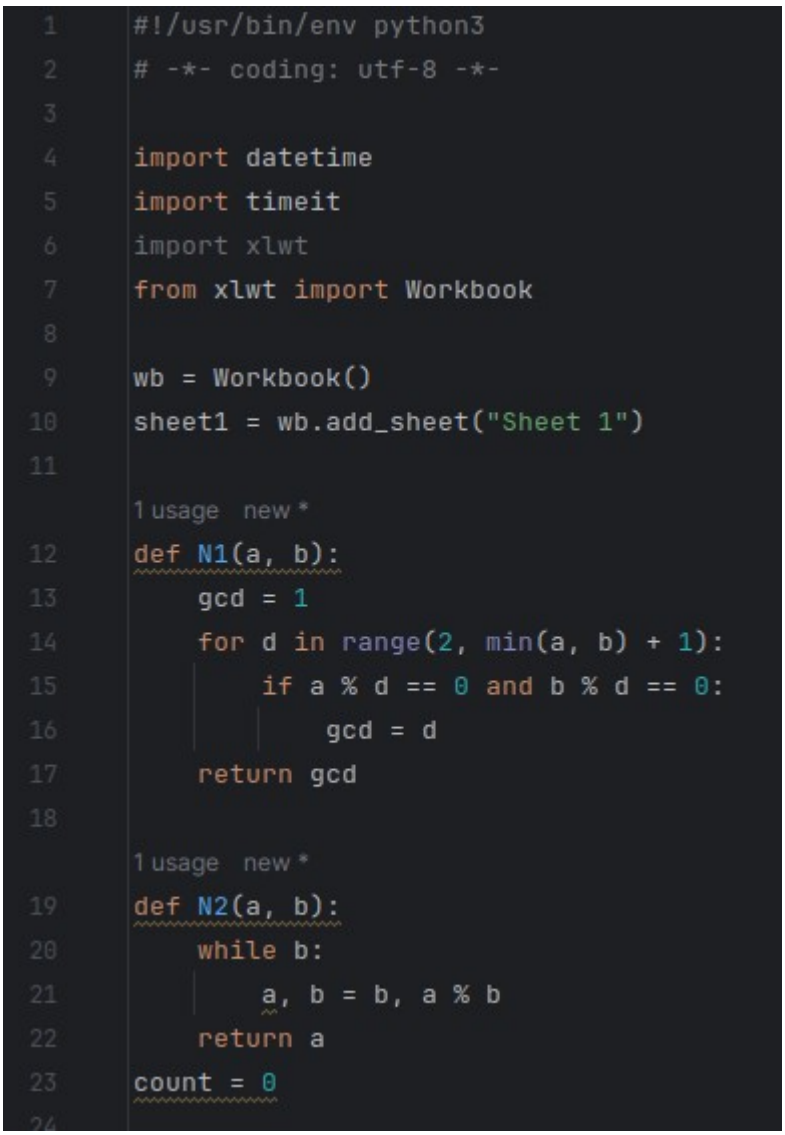
Написал два алгоритма вычисления НОД — наивный и оптимизированный, и измерил время их выполнения. Сделал графики выполнения к каждому из них:

```
def N1(a, b):
    gcd = 1
    for d in range(2, min(a, b) + 1):
        if a % d == 0 and b % d == 0:
            gcd = d
    return gcd
```

Рисунок 7. Наивный алгоритм НОД

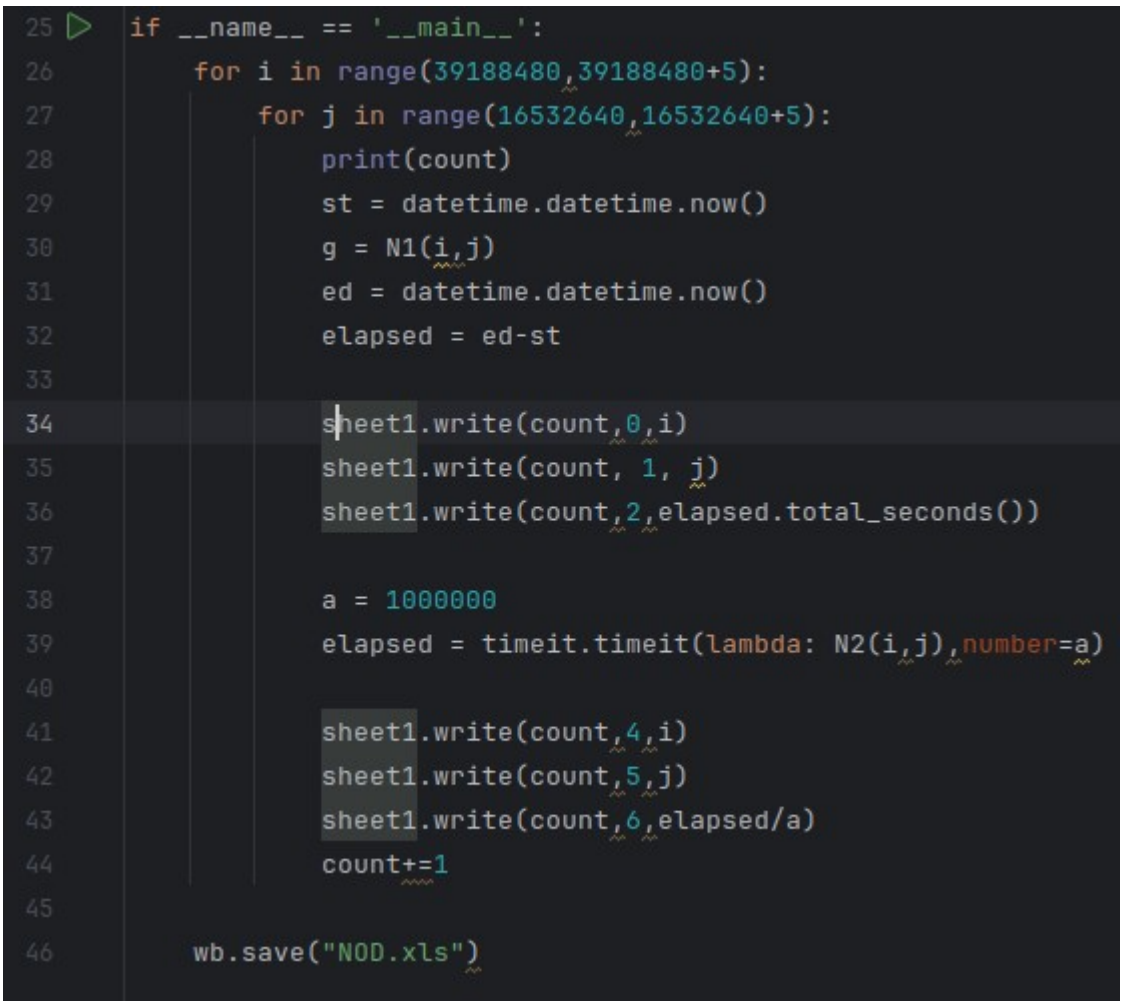
```
def N2(a, b):
    while b:
        a, b = b, a % b
    return a
count = 0
```

Рисунок 8. Оптимизированный алгоритм НОД



```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import datetime
import timeit
import xlwt
from xlwt import Workbook

wb = Workbook()
sheet1 = wb.add_sheet("Sheet 1")

1 usage  new *
def N1(a, b):
    gcd = 1
    for d in range(2, min(a, b) + 1):
        if a % d == 0 and b % d == 0:
            gcd = d
    return gcd

1 usage  new *
def N2(a, b):
    while b:
        a, b = b, a % b
    return a
count = 0
```

Рисунок 9. Полный код программы для рассчета времени выполнения алгоритмов (часть 1)



```
if __name__ == '__main__':
    for i in range(39188480,39188480+5):
        for j in range(16532640,16532640+5):
            print(count)
            st = datetime.datetime.now()
            g = N1(i,j)
            ed = datetime.datetime.now()
            elapsed = ed-st

            sheet1.write(count,0,i)
            sheet1.write(count, 1, j)
            sheet1.write(count,2,elapsed.total_seconds())

            a = 1000000
            elapsed = timeit.timeit(lambda: N2(i,j),number=a)

            sheet1.write(count,4,i)
            sheet1.write(count,5,j)
            sheet1.write(count,6,elapsed/a)
            count+=1

    wb.save("NOD.xls")
```

Рисунок 10. Полный код программы для рассчета времени выполнения алгоритмов (часть 2)

39188480	16532640	0,661043	39188480	16532640	1,64588E-07
39188480	16532641	0,637948	39188480	16532641	2,7247E-07
39188480	16532642	0,580424	39188480	16532642	2,37767E-07
39188480	16532643	0,589298	39188480	16532643	3,01423E-07
39188480	16532644	0,622065	39188480	16532644	2,70848E-07
39188481	16532640	0,572894	39188481	16532640	2,16579E-07
39188481	16532641	0,619599	39188481	16532641	2,3745E-07
39188481	16532642	0,572242	39188481	16532642	2,88097E-07
39188481	16532643	0,577056	39188481	16532643	2,35763E-07
39188481	16532644	0,560394	39188481	16532644	2,89437E-07
39188482	16532640	0,560606	39188482	16532640	2,17366E-07
39188482	16532641	0,570966	39188482	16532641	2,35708E-07
39188482	16532642	0,557612	39188482	16532642	2,54571E-07
39188482	16532643	0,566973	39188482	16532643	2,72176E-07
39188482	16532644	0,563272	39188482	16532644	2,90966E-07
39188483	16532640	0,554619	39188483	16532640	2,51172E-07
39188483	16532641	0,558643	39188483	16532641	2,35865E-07
39188483	16532642	0,562317	39188483	16532642	3,03044E-07
39188483	16532643	0,571147	39188483	16532643	3,04266E-07
39188483	16532644	0,556587	39188483	16532644	3,54535E-07
39188484	16532640	0,557221	39188484	16532640	2,00421E-07
39188484	16532641	0,560628	39188484	16532641	2,35046E-07
39188484	16532642	0,548929	39188484	16532642	2,40283E-07
39188484	16532643	0,552512	39188484	16532643	2,92821E-07
39188484	16532644	0,550968	39188484	16532644	2,73466E-07

Рисунок 11. Результаты времени выполнения алгоритмов в зависимости от значений (слева — наивный, справа - оптимизированный)

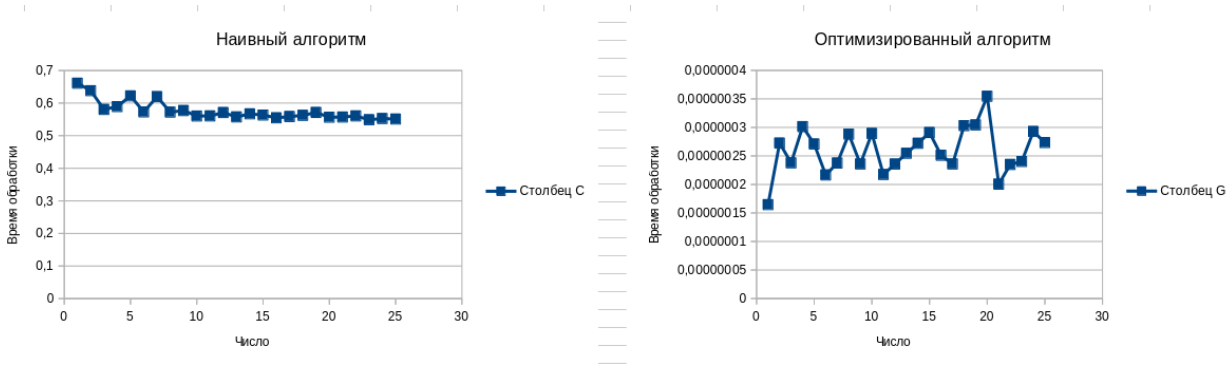


Рисунок 12. Графики выполнения алгоритмов

Вывод: В ходе работы были рассмотрены четыре различных алгоритма обработки чисел Фиббоначи и поиска НОД, было выполнено сравнение наивных и оптимизированных алгоритмов. По итогу можно сказать, что скорость оптимизированных алгоритмов во много раз превышает скорость обработки наивных алгоритмов