

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
УНИВЕРСИТЕТ  
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ  
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №0  
по курсу «Алгоритмы и структуры данных»  
Тема: Работа с файлами. Тестирование

Выполнил:  
Нгуен Хыу Жанг  
К3140

Проверила:  
Афанасьев А.В

Санкт-Петербург  
2024 г

# Содержание

<b>Содержание .....</b>	<b>2</b>
<b>Задание 1: Ввод-вывод.....</b>	<b>3</b>
1.1 .....	3
1.2 .....	3
1.3 .....	4
1.4 .....	5
<b>Задание 2: Число Фибоначчи.....</b>	<b>6</b>
<b>Задание 3: Еще про числа Фибоначчи.....</b>	<b>7</b>
<b>Задание 4: Тестирование ваших алгоритмов.....</b>	<b>8</b>
4.1 .....	9
4.2 .....	10

## Задачи по варианту

### Задание 1 : Ввод-вывод

#### 1.1:

1. Задача  $a + b$ . В данной задаче требуется вычислить сумму двух заданных чисел. Вход: одна строка, которая содержит два целых числа  $a$  и  $b$ . Для этих чисел выполняются условия  $-10^9 \leq a, b \leq 10^9$ . Выход: единственное целое число — результат сложения  $a + b$ .

Visual studio code > 1.py > ...

```
1 a = int(input())
2 b = int(input())
3 if -10**9 <= a <= 10**9 and -10**9 <= b <= 10**9:
4     print("a + b = " + str(a+b))
5 else:
6     print("Ошибка: числа должны быть в диапазоне от -10^9 до 10^9.")
7
8
```

\* Например :

```
PROBLEMS 38 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\> & d:/venv/Scripts/python.exe "d:/Visual studio code/1.py"
12
25
a + b = 37
PS D:\>
```

- Код запрашивает ввод двух целых чисел от пользователя и проверяет, находятся ли они в диапазоне от  $-10^9$  до  $10^9$ .
- Если оба числа соответствуют условию, программа выводит их сумму.
- В противном случае программа выводит сообщение об ошибке, касающееся ограничения чисел.

## 1.2 :

2. Задача  $a + b^2$ . В данной задаче требуется вычислить значение  $a + b^2$ . Вход: одна строка, которая содержит два целых числа  $a$  и  $b$ . Для этих чисел выполняются условия  $-10^9 \leq a, b \leq 10^9$ . Выход: единственное целое число — результат сложения  $a + b^2$ .

Visual studio code > 2.py > ...

```
1 a = int(input())
2 b = int(input())
3 if -10**9 <= a <= 10**9 and -10**9 <= b <= 10**9:
4     result = a + b**2
5     print("a + b**2 = " + str(a+b**2))
6 else:
7     print("Ошибка: числа должны быть в диапазоне от -10^9 до 10^9.")
```

\* Например :

```
PS D:\> & d:/venv/Scripts/python.exe "d:/Visual studio code/2.py"
130
61
a + b**2 = 3851
PS D:\>
```

- Код запрашивает ввод двух целых чисел от пользователя и проверяет, находятся ли они в диапазоне от  $-10^9$  до  $10^9$ .
- Если условие выполняется, программа вычисляет сумму числа **a** и квадрата числа **b** ( $a + b^2$ ), а затем выводит результат.
- В противном случае программа выводит сообщение об ошибке, указывающее, что числа должны быть в пределах заданного диапазона.

## 1.3 :

3. Выполните задачу  $a + b$  с использованием файлов.

- Имя входного файла: input.txt
- Имя выходного файла: output.txt
- Формат входного файла. Входной файл состоит из одной строки, которая содержит два целых числа  $a$  и  $b$ . Для этих чисел выполняются условия  $-10^9 \leq a, b \leq 10^9$ .
- Формат выходного файла. Выходной файл единственное целое число — результат сложения  $a + b$ .

Примеры.

input.txt	12 25	130 61
output.txt	37	191

```
Visual studio code > try for task 1.py > ...  
1 with open("d:/Visual studio code/input.txt", "r") as input_file:  
2     a, b = map(int, input_file.read().split())  
3     sum = a + b  
4 with open("d:/Visual studio code/output.txt", "w") as output_file:  
5     output_file.write(str(sum))
```

\*Например :

```
Visual studio code > input.txt  
1 12 25
```



```
Visual studio code > output.txt  
1 37
```

```
Visual studio code > input.txt  
1 130 61
```



```
Visual studio code > output.txt  
1 191
```

- Откройте файл **input.txt**, чтобы прочитать данные.
- Прочитайте два целых числа **a** и **b** из файла и преобразуйте их в целые числа.
- Вычислить сумму  $a$  и  $b$
- Откройте файл **output.txt**, чтобы записать результаты.
- Запишите результат (сумму **a** и **b**) в файл.

## 1.4 :

4. Выполните задачу  $a + b^2$  с использованием файлов аналогично предыдущему пункту.

```
Visual studio code > try for task 2.py > ...
1 with open("d:/Visual studio code/input.txt", "r") as input_file:
2     a, b = map(int, input_file.read().split())
3     sum = a + b**2
4 with open("d:/Visual studio code/output.txt", "w") as output_file:
5     output_file.write(str(sum)) |
```

\* Например :

```
Visual studio code > input.txt
1 12 25
```



```
Visual studio code > output.txt
1 637
```

```
Visual studio code > input.txt
1 130 61
```



```
Visual studio code > output.txt
1 3851
```

- Откройте файл **input.txt** в режиме чтения ("r"). Переменная **input\_file** относится к этому файлу.
- Прочитайте содержимое файла и разделите его на два целых числа **a** и **b** с помощью функции **split()** (на основе пробелов) и преобразуйте элементы из строки в целое число с помощью **map(int, ...)**.
- Вычисление **sum = a + b\*\*2** представляет собой сумму целого числа **a** и квадрата **b** (**b\*\*2** — это оператор возведения в степень, то есть **b** умножается само на себя).
- Откройте (или создайте новый) файл **output.txt** в режиме записи ("w"). Если файл уже существует, он будет перезаписан.
- Записывает результат вычисления (**a + b\*\*2**) в файл в виде строки.

## Задание 2 : Число Фибоначчи

Определение последовательности Фибоначчи:

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_i &= F_{i-1} + F_{i-2} \text{ для } i \geq 2. \end{aligned} \quad (1)$$

Таким образом, каждое число Фибоначчи представляет собой сумму двух предыдущих, что дает последовательность

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Ваша цель – разработать эффективный алгоритм для подсчета чисел Фибоначчи. Вам предлагается начальный код на Python, который содержит наивный рекурсивный алгоритм:

```
def calc_fib(n):
    if (n <= 1):
        return n

    return calc_fib(n - 1) + calc_fib(n - 2)

n = int(input())
print(calc_fib(n))
```

- Имя входного файла: input.txt
- Имя выходного файла: output.txt
- Формат входного файла. Целое число  $n$ .  $0 \leq n \leq 45$ .
- Формат выходного файла. Число  $F_n$ .
- Пример.

input.txt	10
output.txt	55

Visual studio code > fi1.py > ...

```
1 def calc_fib(n):
2     if n <= 1:
3         return n
4     a, b = 0, 1
5     for _ in range(2, n + 1):
6         a, b = b, a + b
7     return b
8
9 n = int(input())
10 print(calc_fib(n))
11
```

```
PS D:\> & d:/venv/Scripts/python.exe "d:/Visual studio code/fi1.py"
10
55
PS D:\>
```

- Проверьте значение  $n$ : если  $n$  равно 0 или 1, верните  $n$ , поскольку  $F(0) = 0$  и  $F(1) = 1$ .
- Инициализируйте две переменные  $a$  и  $b$ :  $a$  равно  $F(0) = 0$ ,  $b$  равно  $F(1) = 1$ .
- Используйте цикл для вычисления чисел Фибоначчи: на каждой итерации обновляйте  $a$  и  $b$  следующим значением Фибоначчи, пока не будет достигнуто  $n$ .
- Возвращает конечный результат:  $b$  —  $n$ -е число Фибоначчи.
- Введите значение  $n$  от пользователя и распечатайте результат.

## Задание 3 : Еще про числа Фибоначчи

Определение последней цифры большого числа Фибоначчи. Числа Фибоначчи растут экспоненциально. Например,

$$F_{200} = 280571172992510140037611932413038677189525$$

Хранить такие суммы в массиве, и при этом подсчитывать сумму, будет достаточно долго. Найти последнюю цифру любого числа достаточно просто:  $F \bmod 10$ .

- Имя входного файла: input.txt
- Имя выходного файла: output.txt
- Формат входного файла. Целое число  $n$ .  $0 \leq n \leq 10^7$ .
- Формат выходного файла. Одна последняя цифра числа  $F_n$ .
- Пример 1.

input.txt	331
output.txt	9

$$F_{331} = 668996615388005031531000081241745415306766517246774551964595292186469.$$

- Пример 2.

input.txt	327305
output.txt	5

Это число не влезет в страницу, но оканчивается действительно на 5.

- Ограничение по времени: 5сек.
- Ограничение по памяти: 512 мб.

Visual studio code > fi3.py > last\_digit\_fibonacci

```
1 def last_digit_fibonacci(n):
2     period = 60
3     n = n % period
4
5     if n == 0:
6         return 0
7
8     a, b = 0, 1
9     for _ in range(n - 1):
10        a, b = b, (a + b) % 10
11
12    return b
13
14 n = int(input())
15 print(last_digit_fibonacci(n))
16
```

\* Например :

```
PS D:\> & d:/venv/Scripts/python.exe "d:/Visual studio code/fi3.py"
331
9
PS D:\>
```



```
PS D:\> & d:/venv/Scripts/python.exe "d:/Visual studio code/fi3.py"
327305
5
PS D:\> █
```

- Когда последовательность Фибоначчи берется по модулю 10 (то есть при делении на 10 берется только остаток), эта последовательность будет повторяться циклически. Период последовательности Фибоначчи по модулю 10 всегда равен **60**. Вам нужно найти число Фибоначчи в этом периоде, приняв  $n \% 60$ .
- Период 60: Задайте **period = 60**, чтобы использовать свойство period.
- Уменьшите количество вычислений: получите остаток **n** от деления на период ( $n = n \% \text{period}$ ), что ограничивает число n от 0 до 59.
- Если **n = 0**, то 0-е число Фибоначчи ( $F(0)$ ) равно 0, поэтому возвращается **0**.
- Инициализируйте первые два числа последовательности Фибоначчи: **a = 0** ( $F(0)$ ) и **b = 1** ( $F(1)$ ).
- Используйте цикл для вычисления **n**-го числа Фибоначчи, но заботьтесь только о последней цифре, используя операцию по модулю 10 на каждом шаге ( $(a + b) \% 10$ )
- После завершения цикла **b** будет содержать последнюю цифру **n**-го числа Фибоначчи.
- Введите число **n** от пользователя и распечатайте последнюю цифру **n**-го числа Фибоначчи.

## Задание 4 : Тестирование ваших алгоритмов

**Задача:** вам необходимо протестировать время выполнения вашего алгоритма в *Задании 2* и *Задании 3*.

*Дополнительно:* вы можете протестировать объем используемой памяти при выполнении вашего алгоритма.

## 4.1:

\* Код для Задания 2 :

```
Visual studio code > try fi1 a.py > calc_fib
1  from memory_profiler import memory_usage
2  import time
3
4  def calc_fib(n):
5      if n <= 1:
6          return n
7      return calc_fib(n - 1) + calc_fib(n - 2)
8
9  def measure_memory_usage():
10     start_time = time.time()
11     mem_usage = memory_usage((calc_fib, (n,)))
12     end_time = time.time()
13     print(f"Время выполнения: {end_time - start_time} секунд")
14     print(f"Используемая память: {max(mem_usage) - min(mem_usage)} МБ")
15
16  if __name__ == '__main__':
17     n = int(input("F(n): "))
18     print(calc_fib(n))
19     measure_memory_usage()
20
```

\* Результаты по заданию 2 :

```
PS D:\> & d:/venv/Scripts/python.exe "d:/Visual studio code/try fi1 a.py"
• F(n): 10
55
Время выполнения: 0.9751589298248291 секунд
Используемая память: 0.00390625 МБ
○ PS D:\>
```

### 1. Функция `calc_fib(n)`:

```
def calc_fib(n):
    if n <= 1:
        return n
    return calc_fib(n - 1) + calc_fib(n - 2)
```

- Это рекурсивная функция для вычисления n-го числа Фибоначчи.
- Если  $n \leq 1$ , функция возвращает n (при  $F(0) = 0$  и  $F(1) = 1$ ).
- Если  $n > 1$ , функция вычисляет число Фибоначчи, дважды рекурсивно вызывая  $n - 1$  и  $n - 2$ .

### 2. Функция `measure_memory_usage()`:

```
def measure_memory_usage():
    start_time = time.time()
    mem_usage = memory_usage((calc_fib, (n,)))
    end_time = time.time()
    print(f"Время выполнения: {end_time - start_time} секунд")
    print(f"Используемая память: {max(mem_usage) - min(mem_usage)} МБ")
```

\* Эта функция измеряет время и память, используемые при запуске функции `calc_fib(n)`:

- `start_time`: сохраняет время начала вычислений.

- `memory_usage()`: функция из библиотеки `memory_profiler` для измерения объема памяти, используемой при выполнении функции `calc_fib(n)`.
- `end_time`: сохраняет время окончания вычислений.
- `print`: печатает время выполнения и объем используемой памяти.

### 3. Основная часть программы :

```
if __name__ == '__main__':
    n = int(input("F(n): "))
    print(calc_fib(n))
    measure_memory_usage()
```

- Программа получает входные данные `n` от пользователя, затем вычисляет и печатает `n`-е число Фибоначчи с помощью функции `calc_fib(n)`.
- Программа вызывает функцию `measure_memory_usage()`, чтобы измерить и отобразить время и использование памяти во время выполнения.

## 4.2:

\* Код для Задания 3 :

```
Visual studio code > try fib a.py > ...
1  from memory_profiler import memory_usage
2  import time
3
4  def last_digit_fibonacci(n):
5      period = 60
6      n = n % period
7      if n == 0:
8          return 0
9      a, b = 0, 1
10     for _ in range(n - 1):
11         a, b = b, (a + b) % 10
12     return b
13
14 def measure_memory_usage():
15     start_time = time.time()
16     mem_usage = memory_usage((last_digit_fibonacci, (n,)))
17     end_time = time.time()
18     print(f"Время выполнения: {end_time - start_time} секунд")
19     print(f"Используемая память: {max(mem_usage) - min(mem_usage)} МБ")
20
21 if __name__ == '__main__':
22     n = int(input("F(n): "))
23     print(last_digit_fibonacci(n))
24     measure_memory_usage()
25
```

\* Результаты по заданию 3 :

```
PS D:\> & d:/venv/Scripts/python.exe "d:/Visual studio code/try fib a.py"
F(n): 331
9
Время выполнения: 0.9014909267425537 секунд
Используемая память: 0.0 МБ
PS D:\>
```

```
PS D:\> & d:/venv/Scripts/python.exe "d:/Visual studio code/try fi3 a.py"
F(n): 327305
5
Время выполнения: 0.8956344127655029 секунд
Используемая память: 0.0 МБ
PS D:\>
```

### 1. Функция `last_digit_fibonacci(n)`:

```
def last_digit_fibonacci(n):
```

```
    period = 60
    n = n % period
    if n == 0:
        return 0
    a, b = 0, 1
    for _ in range(n - 1):
        a, b = b, (a + b) % 10
    return b
```

- Цель: Вычислить последнюю цифру **n-го** числа Фибоначчи.

- Период: период последовательности Фибоначчи по модулю 10 равен 60, поэтому  **$n \% 60$**  поможет уменьшить количество необходимых вычислений.

- Алгоритм:

- Если  **$n = 0$** , вернуть 0.
- Инициализируйте первые два числа Фибоначчи:  **$a = 0$ ,  $b = 1$** .
- Используйте цикл для вычисления последней цифры n-го числа Фибоначчи, сохраняя только остаток при делении на 10 (по модулю 10).

### 2. Функция `measure_memory_usage()`:

```
def measure_memory_usage():
```

```
    start_time = time.time()
    mem_usage = memory_usage((last_digit_fibonacci, (n,)))
    end_time = time.time()
    print(f"Время выполнения: {end_time - start_time} секунд")
    print(f"Используемая память: {max(mem_usage) - min(mem_usage)} МБ")
```

- Цель: измерить время выполнения и использование памяти при запуске функции `last_digit_fibonacci(n)`.

- Используйте библиотеку `memory_profiler`:

- `memory_usage()`: измеряет объем памяти, используемый во время выполнения функции.
- `start_time` и `end_time`: измеряют время начала и окончания.
- Распечатайте время выполнения и объем используемой памяти.

### 3. Основная часть программы:

```
if __name__ == '__main__':
```

```
n = int(input("F(n): "))  
print(last_digit_fibonacci(n))  
measure_memory_usage()
```

- Получите значение **n** от пользователя.
- Вычислите и выведите последнюю цифру **n**-го числа Фибоначчи.
- Измерьте и распечатайте время и использование памяти.