

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

Отчет по лабораторной работе №1
по курсу «Алгоритмы и структуры данных»
Тема: Введение
Вариант 27

Выполнил:
Филиппов А.Э.
К3139

Проверила:
Артамонова В.Е.

Санкт-Петербург
2022 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3-17
Задача №1. Ввод-Вывод	3-8
Задача №2. Числа Фибоначчи	8-12
Задача №3. Еще про числа Фибоначчи	12-16
Задача №4. Тестирование ваших алгоритмов	16-17
Дополнительные задачи	18
Задача №4. Тестирование ваших алгоритмов	18
Вывод	19

Задачи по варианту

Задача №1 Ввод-Вывод

Вам необходимо выполнить 4 следующих задачи:

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 $a + b$.
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 $a + b^2$.
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

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

Код 1:

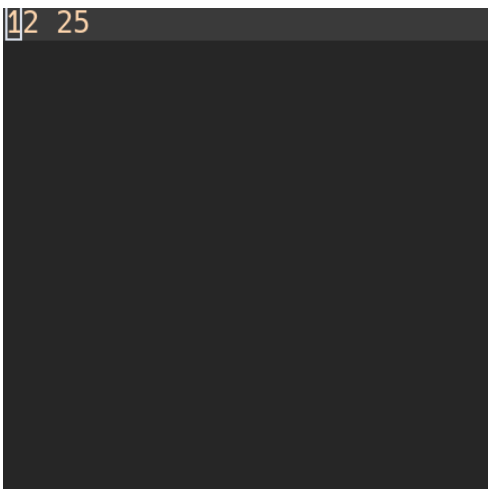
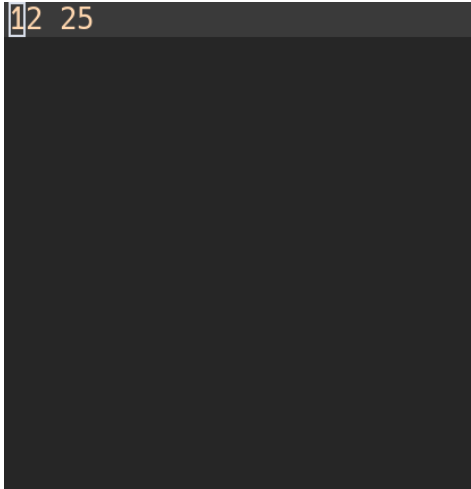
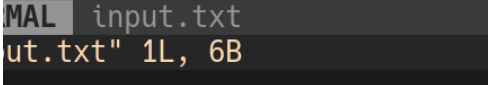
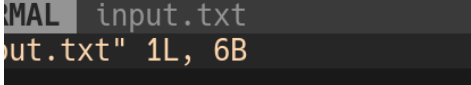
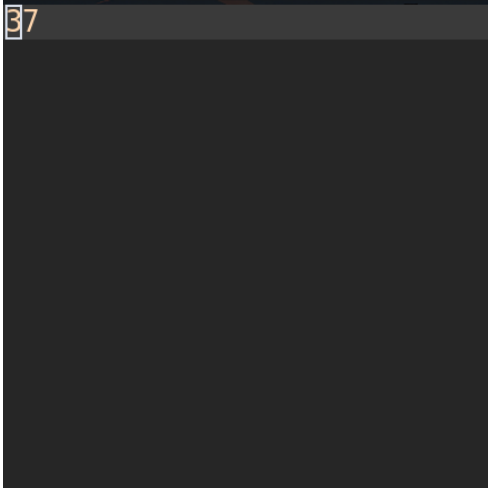
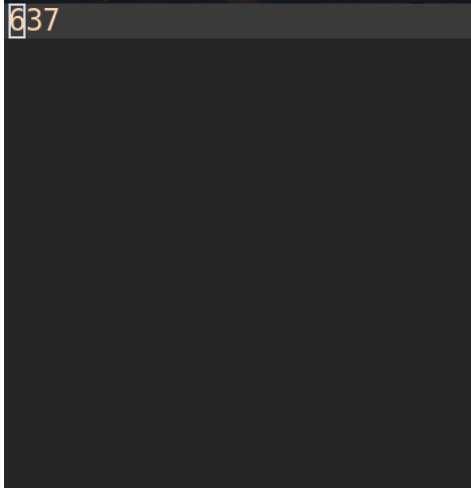
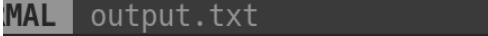
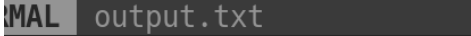
```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
a,b = map(int,open("input.txt", "r").readline().rstrip().split())
open("output.txt", 'w').write(str(a+b))
print("Время выполнения: " + str(time.perf_counter() -t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) +
" байт")
tracemalloc.stop()
```

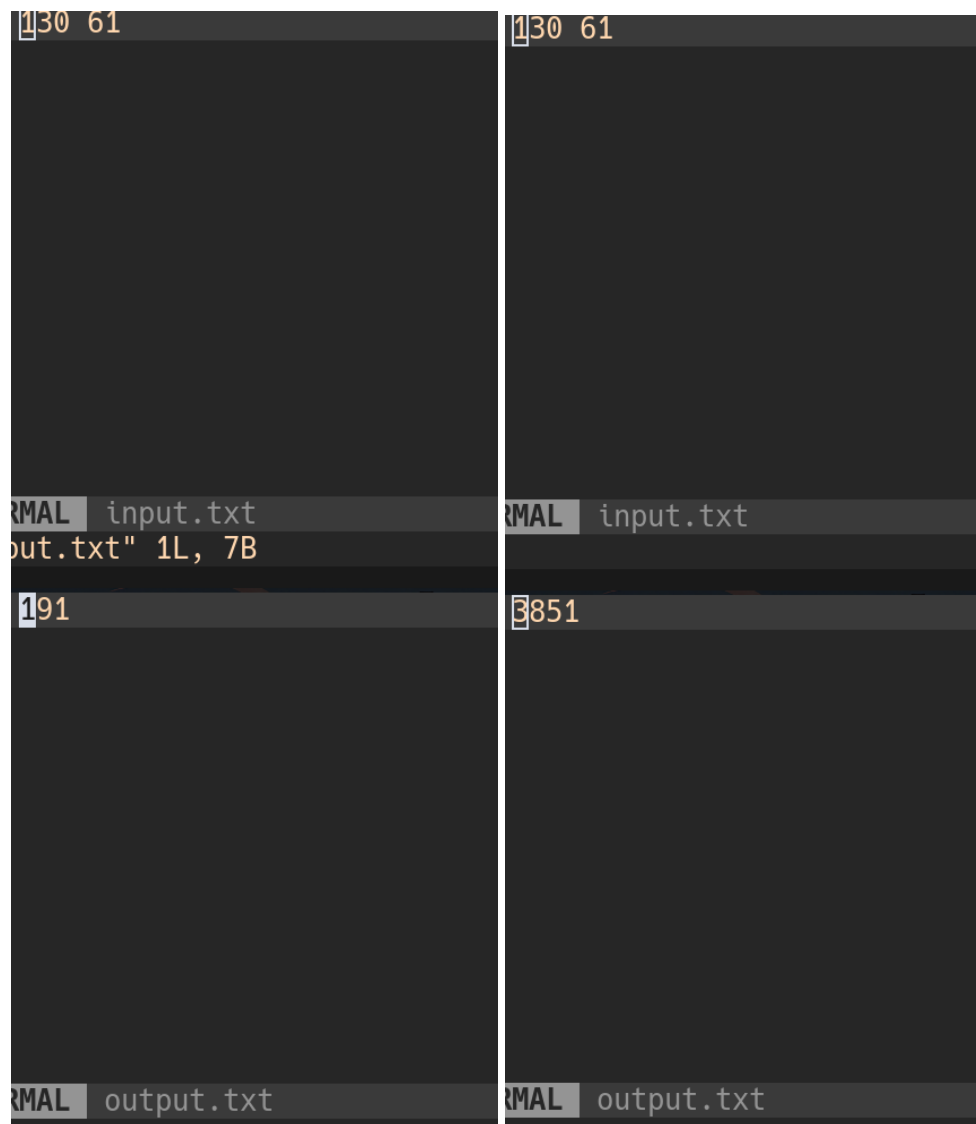
Код 2:

```
import time
t_start = time.perf_counter()
a,b = map(int,open("input.txt", "r").readline().rstrip().split())
open("output.txt", 'w').write(str(a+b**2))
print("Время выполнения: " + str(time.perf_counter() -t_start) + " секунд")
```

Для решения данного задания были использованы встроенные в язык Python операции плюс и возведение в квадрат. Для решения подпунктов была применена функция open.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:

<pre>1000000000 1000000000 NORMAL input.txt :qa and press <Enter> to exit Vim 2000000000 NORMAL output.txt</pre>	<pre>1000000000 1000000000 NORMAL input.txt projects/labs/sem1/0/Задачи по варианту/1/in 10000000001000000000 NORMAL output.txt</pre>
<pre>1000000000 -1000000000 NORMAL input.txt projects/labs/sem1/0/Задачи по варианту/1/in 2000000000 NORMAL output.txt</pre>	<pre>1000000000 -1000000000 NORMAL input.txt projects/labs/sem1/0/Задачи по варианту/1/in 999999990000000000 NORMAL output.txt</pre>

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.000212674999602313 62 0.000278329999673587 74	13773 13773
Пример из задачи	0.000251352000304905 23 0.000317578000249341 13	13773 13773
Пример из задачи	0.000271854999482457 06 0.006372893999468943	13773 13773
Верхняя граница диапазона значений входных данных из текста задачи	0.000272391999715182 46 0.000303370999972685 24	13773 13773

Задача №2 Числа Фибоначчи

Определение последовательности Фибоначчи:

$$F_0 = 0 \quad (1)$$

$$F_1 = 1$$

$$F_i = F_{i-1} + F_{i-2} \text{ для } i \geq 2.$$

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

Код:

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
f1 = 0
f2 = 1
n = int(open("input.txt", "r").readline().rstrip())
for i in range(2, n+1):
    f3 = f2 + f1
    f1 = f2
    f2 = f3
if n != 0:
    open("output.txt", "w").write(str(f2))
else:
    open("output.txt", "w").write("0")

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
str(tracemalloc.get_traced_memory()[1]) + " байт")

tracemalloc.stop()
```

Для того чтобы ускорить наивное решение был использован способ сохранения вычислений в память. Сложность алгоритма - $O(n)$

Результат работы кода на примере из текста задачи:

```
10
NORMAL input.txt
input.txt" 1L, 3B
55
```

Результат работы кода на максимальных и минимальных значениях:

<pre>0 NORMAL input.txt out.txt" 1L, 2B 0</pre>	<pre>45 NORMAL input.txt out.txt" 1L, 3B 1134903170</pre>
---	---

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.000245834000452305 2	13789
Пример из задачи	0.000298888999168411 83	13789
Верхняя граница диапазона значений входных данных из текста задачи	0.000312055999529548	13789

Задача №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 до статочно 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

$F_{331} =$
6689966153880050315310000812417454153067665172467745519645952921
86469.

• Пример 2.

input.txt	327305
output.txt	5

Код:

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
f1 = 0
f2 = 1
n = int(open("input.txt", "r").readline().rstrip())
for i in range(2, n+1):
    f3 = (f2 + f1) % 10
    f1 = f2
    f2 = f3
if n != 0:
    open("output.txt", "w").write(str(f2))
else:
    open("output.txt", "w").write("0")

print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) +
      " байт")

tracemalloc.stop()
```

Длинная арифметика требует от компьютера больших вычислительных мощностей. Так как для решения нужна только последняя цифра числа, то в памяти достаточно хранить остаток от деления числа на 10.

Результат работы кода на примерах из текста задачи:

331	327305
RMAL input.txt ut.txt" 1L, 4B	RMAL input.txt put.txt" 1L, 7B
9	5

Результат работы кода на максимальных и минимальных значениях:

<pre>10000000 RMAL input.txt out.txt" 1L, 9B 5</pre>	<pre>0 RMAL input.txt out.txt" 1L, 2B written 0</pre>
--	---

	Время выполнения (с)	Затраты памяти (байт)
Нижняя граница диапазона значений входных данных из текста задачи	0.0002665680003701709	13773
Пример из задачи 1	0.00033639299908827525	13773
Пример из задачи 2	0.11361844100065355	13773
Верхняя граница диапазона значений входных данных из текста задачи	3.1125326589990436	13773

Задача №4 Тестирование ваших алгоритмов

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

Код для задания 3:

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
f1 = 0
f2 = 1
n = int(open("input.txt", "r").readline().rstrip())
for i in range(2, n+1):
    f3 = (f2 + f1) % 10
    f1 = f2
    f2 = f3
if n != 0:
    open("output.txt", "w").write(str(f2))
else:
    open("output.txt", "w").write("0")
print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) +
" байт")
tracemalloc.stop()
```


Код для задания 2:

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
f1 = 0
f2 = 1
n = int(open("input.txt", "r").readline().rstrip())
for i in range(2, n+1):
    f3 = f2 + f1
    f1 = f2
    f2 = f3
if n != 0:
    open("output.txt", "w").write(str(f2))
else:
    open("output.txt", "w").write("0")

print("Время выполнения: " + str(time.perf_counter() - t_start) + "
секунд")
print("Использование памяти: " +
str(tracemalloc.get_traced_memory()[1]) + " байт")

tracemalloc.stop()
```

Для тестирования времени выполнения алгоритма была использована библиотека time. С ее помощью можно задать точку начала и конца отсчета времени.

```
Время выполнения: 0.00026724700001068413 секунд
Использование памяти: 13773 байт
```

```
Время выполнения: 3.1729739800000279 секунд
Использование памяти: 13773 байт
```

Дополнительные задачи

Задача №4. Тестирование ваших алгоритмов

Вы можете протестировать объем используемой памяти при выполнении вашего алгоритма.

Код для задания 3:

```
import time
import tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
f1 = 0
f2 = 1
n = int(open("input.txt", "r").readline().rstrip())
for i in range(2, n+1):
    f3 = (f2 + f1) % 10
    f1 = f2
    f2 = f3
if n != 0:
    open("output.txt", "w").write(str(f2))
else:
    open("output.txt", "w").write("0")
print("Время выполнения: " + str(time.perf_counter() - t_start) + " секунд")
print("Использование памяти: " + str(tracemalloc.get_traced_memory()[1]) +
      " байт")
```

Для подсчета использованного кол-ва памяти программой была использована библиотека tracemalloc. Метод get_traced_memory возвращает пиковое использование памяти программой в байтах.

Вывод

Существует множество различных методов оптимизации алгоритмов. Одним из способов является метод сохранения вычислений в память. Для того, чтобы сохранить посчитанные значения можно использовать массивы или переменные.