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: Введение

Выполнил:
Тутубалин К.С.
Группа К3141

Проверила:
Артамонова В.Е.

Санкт-Петербург
2023 г.

Содержание отчета

Задачи по варианту

Задание №1. Ввод-вывод

Текст 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 $-10^9 \le a, b \le 10^9$. Выход: единственное целое число — результат сложения a + b.

```
a, b = map(int, input().split())
print(a+b)
```

Текстовое объяснение решения.

Получаем два числа в одной строке. Метод split() делит введенную строку на список подстрок. После этого map() выполняет функцию int() для каждого элемента списка. Выводим сумму 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 $-10^9 \le a, b \le 10^9$. Выход: единственное целое число — результат сложения a + b^2 .

```
a, b = map(int, input().split())
print(a+b**2)
```

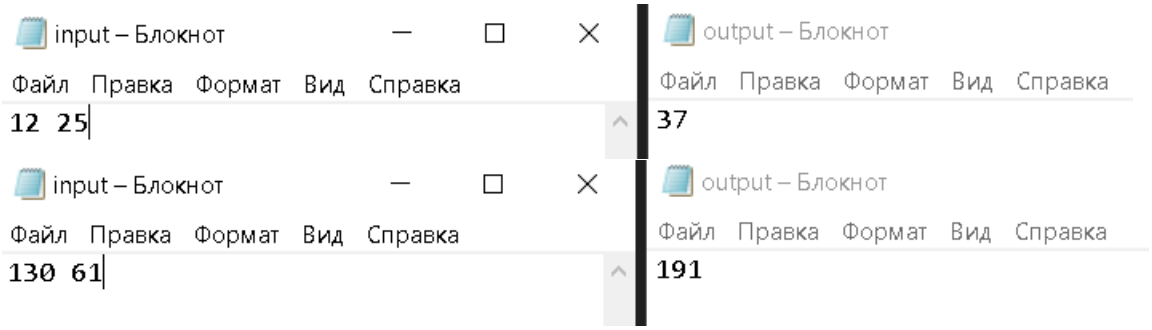
Текстовое объяснение решения.

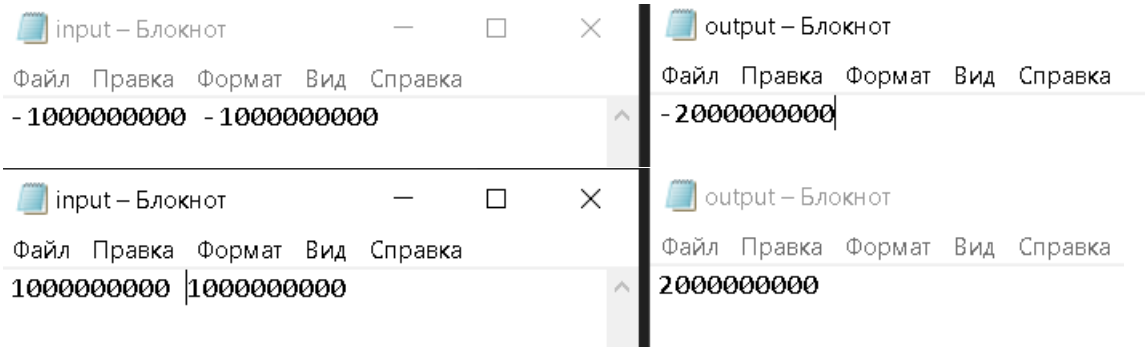
Получаем два числа в одной строке. Метод split() делит введенную строку на список подстрок. После этого map() выполняет функцию int() для каждого элемента списка. Выводим сумму a+b**2.

3) **Выполните задачу a + b с использованием файлов.**
   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 ≤ a, b ≤ 10^9
   • Формат выходного файла. Выходной файл единственное целое число — результат сложения a + b.

```
f = open("input.txt")
a, b = map(int, f.readline().split())
f.close()
res = str(a + b)
w = open("output.txt", 'w')
w.write(res)
w.close()
```

Текстовое объяснение решения.
Читаем 2 числа с “input.txt” в res записываем их сумму, выводим в “output.txt” сумму чисел.



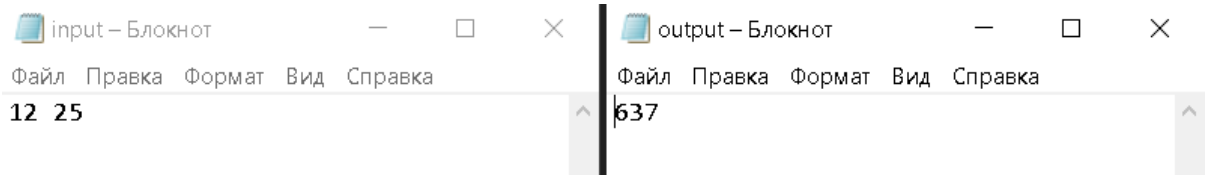


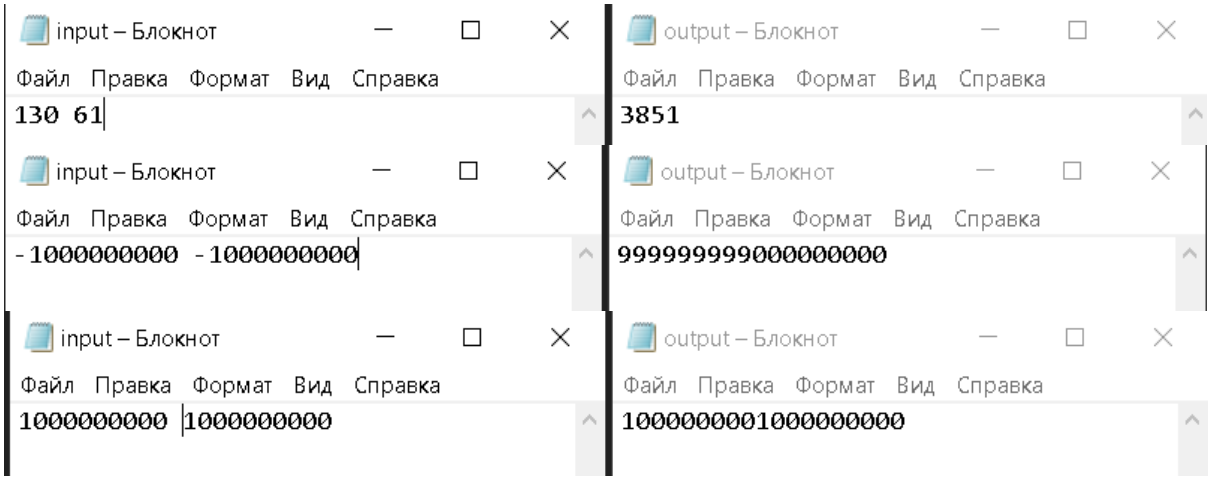
	Время выполнения	Затрата памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0005114000000503438	0.03353118896484375
Пример из задачи	0.00045860000000175205	0.03346538543701172
Пример из задачи	0.0005681999996340892	0.0334625244140625
Верхняя граница диапазона значений входных данных из текста задачи	0.0005436000001282082	0.033530235290527344

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

```
f = open("input.txt")
a, b = map(int, f.readline().split())
f.close()
res = str(a + b**2)
w = open("output.txt", 'w')
w.write(res)
w.close()
```

Текстовое объяснение решения.
Читаем 2 числа с “input.txt” в res записываем их сумму (a+b**2), выводим в “output.txt” результат.





	Время выполнения	Затрата памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00048100000003614696	0.03354167938232422
Пример из задачи	0.0004904000002170505	0.0334625244140625
Пример из задачи	0.0005384999994930695	0.033463478088378906
Верхняя граница диапазона значений входных данных из текста задачи	0.0005357999998523155	0.033538818359375

Вывод по задаче : вспомнил как работать с файлами, научился измерять время и память программы.

Задание №2.Число Фибоначчи

Текст задачи.

Разработать эффективный алгоритм для подсчета чисел Фибоначчи.

```
#оптимизация рекурсии
input = open('input.txt', 'r')
n=int(input.readline())
output = open('output.txt', 'w')
from functools import lru_cache
@lru_cache()
def calc_fib(n):
    if (n <= 1):
```

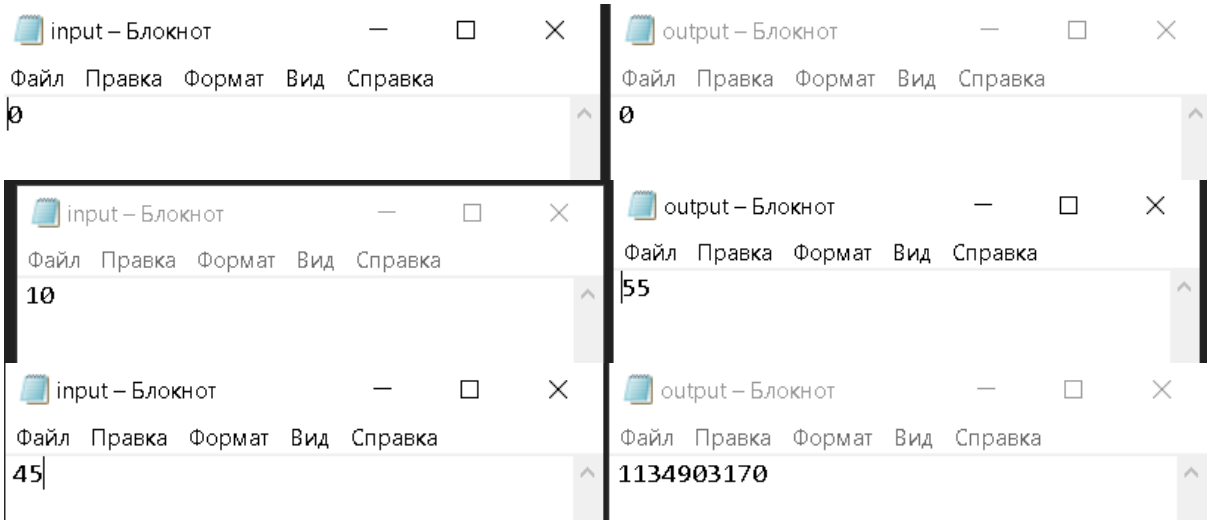
```
        return n
    return calc_fib(n - 1) + calc_fib(n - 2)
output.write(str(calc_fib(n)))
input.close()
output.close()
```

Текстовое объяснение для решения.

Как работает lru_cache? lru_cache запоминает результаты той функции, перед которой она записана. Перед вычислением очередного значения рекурсивная функция обращается к lru_cache и, если результат вычисления уже имеется в lru_cache, функция его сразу получает. Если же такого результата еще нет, функция его вычисляет и сохраняет результат в lru_cache для дальнейшего использования. Получается, что lru_cache выступает в роли "шпаргалки", в которую записаны все уже вычисленные раньше значения.

```
#решение без рекурсии
n=int(input())
f_1=0
f_2=1
for i in range(0,n):
    f_1,f_2=f_2,f_1+f_2
    print(f_1, end=' ')
```

Текстовое объяснение решения.
Пользователь вводит количество членов последовательности, f_1 и f_2 это 1 и 2 член последовательности. С помощью цикла вывожу текущий и следующий (сумма двух предыдущих), до n-го элемента.



	Время выполнения	Затрата памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0005263000002742046	0.034587860107421875
Пример из задачи	0.0006934000011824537	0.035785675048828125
Верхняя граница диапазона значений входных данных из текста задачи	0.0006078000005800277	0.041790008544921875

Вывод по задаче: вспомнил, как работать с циклами.

Задание №3. Еще про числа Фибоначчи.

Текст задачи.

Определение последней цифры большого числа Фибоначчи.

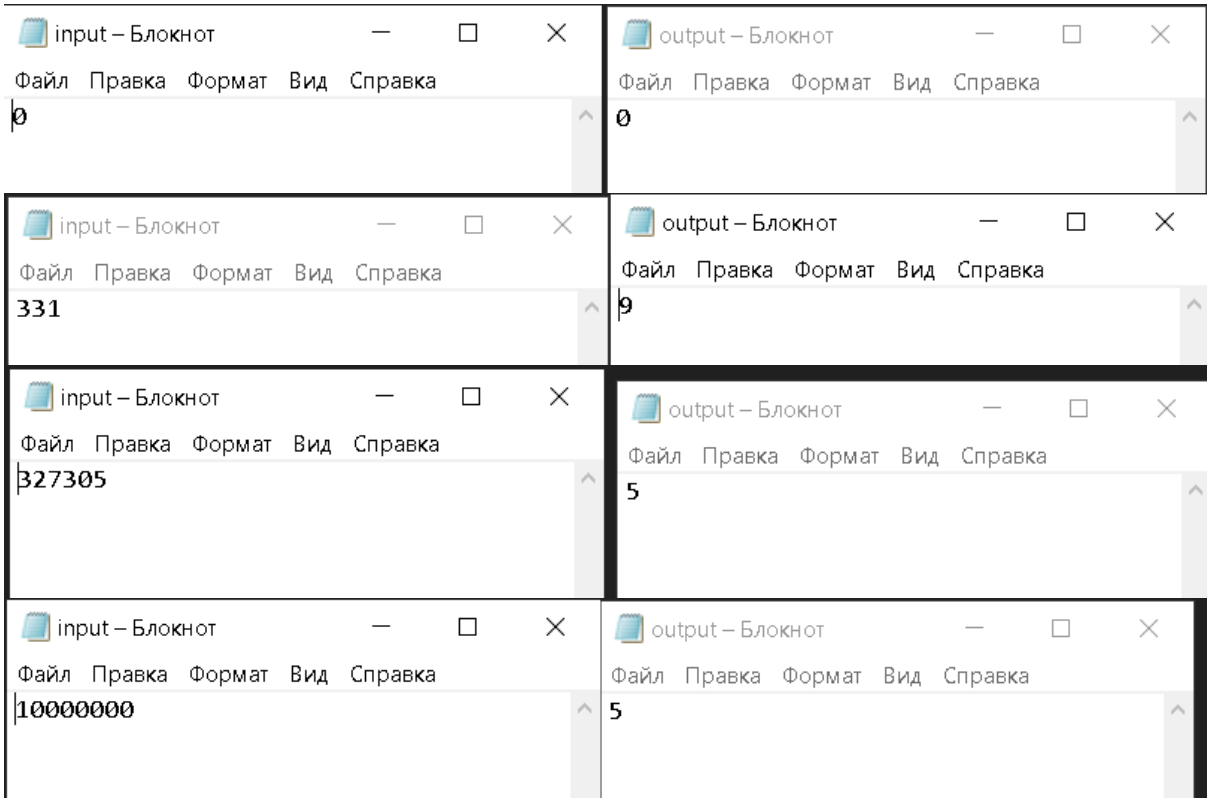
```
input = open('input.txt', 'r')
n=int(input.readline())
output = open('output.txt', 'w')

mas = [0,1]
for i in range(2, n+1):
    mas.append((mas[-1] + mas[-2]) % 10)

output.write(str(mas[n]))
input.close()
output.close()
```

Текстовое объяснение решения.

3-й вариант решения задания №2, через массив.

Каждый индекс, сумма предыдущих двух.

%10 - Каждый раз сохраняем последнюю цифру числа Фибоначчи.



	Время выполнения	Затрата памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0004923999986203853	0.033481597900390625
Пример из задачи	0.0006137999989732634	0.0362091064453125
Пример из задачи	0.2640075999988767	2.8248939514160156
Верхняя граница диапазона значений входных данных из текста задачи	8.099725000000035	85.00146484375

Вывод по задаче: чем больше член последовательности, тем дольше выполняется алгоритм и затрачивается большее количество памяти.

Задание №4. Тестирование ваших алгоритмов.

Текст задачи.

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

```
import time, tracemalloc
tracemalloc.start()
t_start = time.perf_counter()
#
#
#
print("Время работы: %s секунд " % (time.perf_counter() - t_start))
print("Max memory ", tracemalloc.get_traced_memory()[1] / 2 ** 20, "mb")
tracemalloc.stop()
```

Текстовое объяснение решения.
Для измерения времени использовал time. Для памяти tracemallov.

Вывод : вспомнил как работать с файлами, научился считать время выполнения алгоритма и рассчитывать затраты памяти. Освежил в памяти, как работать: с циклами, массивами, арифметическими операциями.