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

Вариант 14

Выполнила:
Скокова А.В.
К34422

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту 3
Задача №1. Ввод-вывод 3
Задача №2. Число Фибоначчи 8
Задача №3. Еще про числа Фибоначчи 11
Задача №4. Тестирование алгоритмов 14
Вывод 15

Задачи по варианту

Задача №1. Ввод-вывод

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 ≤ a, b ≤ 10^9. Выход: единственное целое число — результат сложения a + b.

```
def sum_ab():
    k = 0
    restr1 = -10**9
    restr2 = 10**9
    while k != 1:
        nums_input = input('Введите два целых числа через пробел: ')
        nums_list = nums_input.split(' ')
        try:
            a = int(nums_list[0])
            b = int(nums_list[1])
            if (a >= restr1) and (a <= restr2) and (b >= restr1) and (b
<= restr2):
                k = 1
            else:
                print('Введены числа вне интервала [-10^9, 10^9]')
        except ValueError:
            print('Числа введены неверно')
    print(f'{a + b}')

if __name__ == '__main__':
    sum_ab()
```

Просим пользователя ввести два числа через пробел. Проверяем, введены ли целые числа, введены ли числа в правильном формате и введены ли числа, соответствующие условию задачи. Выводим результат.

Результат работы кода на максимальных и минимальных значениях:

```
Введите два целых числа через пробел: 1000000000 -1000000000
0

Введите два целых числа через пробел: 1000000000 1000000000
2000000000
```

```
Введите два целых числа через пробел: -1000000000 -1000000000
-2000000000
Введите два целых числа через пробел: -1000000000 1000000000
0
```

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 ≤ a, b ≤ 10^9. Выход: единственное целое число — результат сложения (a + b)^2.

```
def sq_sum_ab():
    k = 0
    restr1 = -10**9
    restr2 = 10**9
    while k != 1:
        nums_input = input('Введите два целых числа через пробел: ')
        nums_list = nums_input.split(' ')
        try:
            a = int(nums_list[0])
            b = int(nums_list[1])
            if (a >= restr1) and (a <= restr2) and (b >= restr1) and (b
<= restr2):
                k = 1
            else:
                print('Введены числа вне интервала [-10^9, 10^9]')
        except ValueError:
            print('Числа введены неверно')
    print(f'{(a + b) ** 2}')

if __name__ == '__main__':
    sq_sum_ab()
```

Просим пользователя ввести два числа через пробел. Проверяем, введены ли целые числа, введены ли числа в правильном формате и введены ли числа, соответствующие условию задачи. Выводим результат.

Результат работы кода на максимальных и минимальных значениях:

```
Введите два целых числа через пробел: -1000000000 1000000000
0
Введите два целых числа через пробел: 1000000000 1000000000
4000000000000000000
Введите два целых числа через пробел: -1000000000 -1000000000
4000000000000000000
```

```
Введите два целых числа через пробел: -1000000000 1000000000
0
```

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 $(-10)^9 \le a, b \le 10^9$.
- Формат выходного файла. Выходной файл единственное целое число — результат сложения a + b.

  Примеры:

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

```
def sum_fromFile(filepath):
    restr1 = -10**9
    restr2 = 10**9
    with open(filepath) as f:
        nums_input = f.read()
    nums_list = nums_input.split(' ')
    try:
        a = int(nums_list[0])
        b = int(nums_list[1])
        if (a >= restr1) and (a <= restr2) and (b >= restr1) and (b <=
restr2):
            with open('output.txt', 'w') as f:
                f.write(f'{a + b}')
        else:
            print('Введены числа вне интервала [-10^9, 10^9]')
    except ValueError:
        print('Числа введены неверно')

if __name__ == '__main__':
    sum_fromFile('input.txt')
```

Считываем данные из входного файла. Проверяем, введены ли целые числа, введены ли числа в правильном формате и введены ли числа,

соответствующие условию задачи. Записываем в отдельный файл результат.

Результат работы кода на примерах из текста задачи:

lab0 > task1_3 > input.txt

```
1 12 25
```

lab0 > task1_3 > output.txt

```
1 37
```

lab0 > task1_3 > input.txt

```
1 130 61
```

lab0 > task1_3 > output.txt

```
1 191
```

Результат работы кода на максимальных и минимальных значениях:

lab0 > task1_3 > input.txt

```
1 1000000000 1000000000
```

lab0 > task1_3 > output.txt

```
1 2000000000
```

lab0 > task1_3 > input.txt

```
1 -1000000000 1000000000
```

lab0 > task1_3 > output.txt

```
1 0
```

lab0 > task1_3 > input.txt

```
1 -1000000000 -1000000000
```

lab0 > task1_3 > output.txt

```
1 -2000000000
```

lab0 > task1_3 > input.txt

```
1 1000000000 -1000000000
```

lab0 > task1_3 > output.txt

```
1    0
```

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

```
def sq_sum_fromFile(filepath):
    restr1 = -10**9
    restr2 = 10**9
    with open(filepath) as f:
        nums_input = f.read()
    nums_list = nums_input.split(' ')
    try:
        a = int(nums_list[0])
        b = int(nums_list[1])
         if (a >= restr1) and (a <= restr2) and (b >= restr1) and (b <=
restr2):
            with open('output.txt', 'w') as f:
                f.write(f'{(a + b) ** 2}')
        else:
            print('Введены числа вне интервала [-10^9, 10^9]')
    except ValueError:
        print('Числа введены неверно')

if __name__ == '__main__':
    sq_sum_fromFile('input.txt')
```

Считываем данные из входного файла. Проверяем, введены ли целые числа, введены ли числа в правильном формате и введены ли числа, соответствующие условию задачи. Записываем в отдельный файл результат.

Результат работы кода на примерах из текста задачи:

lab0 > task1_4 > input.txt

```
1    12 25
```

lab0 > task1_4 > output.txt

```
1    1369
```

lab0 > task1_4 > input.txt

```
1    130 61
```

lab0 > task1_4 > output.txt

```
1    36481
```

Результат работы кода на максимальных и минимальных значениях:

lab0 > task1_4 > input.txt

```
1    1000000000 1000000000
```

lab0 > task1_4 > output.txt

```
1    4000000000000000000
```

lab0 > task1_4 > input.txt

```
1    -1000000000 1000000000
```

lab0 > task1_4 > output.txt

```
1    0
```

lab0 > task1_4 > input.txt

```
1    1000000000 -1000000000
```

lab0 > task1_4 > output.txt

```
1    0
```

lab0 > task1_4 > input.txt

```
1    -1000000000 -1000000000
```

lab0 > task1_4 > output.txt

```
1    4000000000000000000
```

Вывод по задаче: были выполнены задачи на ввод-вывод с использованием взаимодействия с пользователем и файлов.

Задача №2. Число Фибоначчи

Каждое число Фибоначчи представляет собой сумму двух предыдущих, что дает последовательность

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

Необходимо разработать эффективный алгоритм для подсчета чисел Фибоначчи. 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. $0 \le n \le 45$.
- Формат выходного файла. Число Fn.

  Пример:

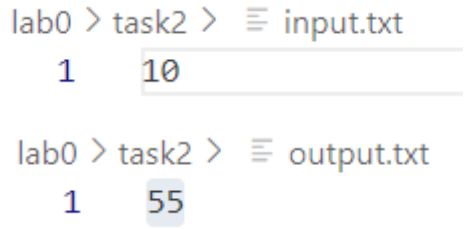
input.txt	10
output.txt	55

```
import time
import tracemalloc

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


if __name__ == '__main__':
    try:
        tracemalloc.start()
        t_start = time.perf_counter()
        with open('input.txt') as f:
            n_str = f.read()
        n = int(n_str)
        if (n >= 0) and (n <= 45):
            with open('output.txt', 'w') as f:
                f.write(f'{calc_fib(n)}')
            print(f'Время работы: {time.perf_counter() - t_start}
сек.')
            size, peak = tracemalloc.get_traced_memory()
            print(f'Текущие и пиковые затраты памяти: {size} и {peak}
байт')
```

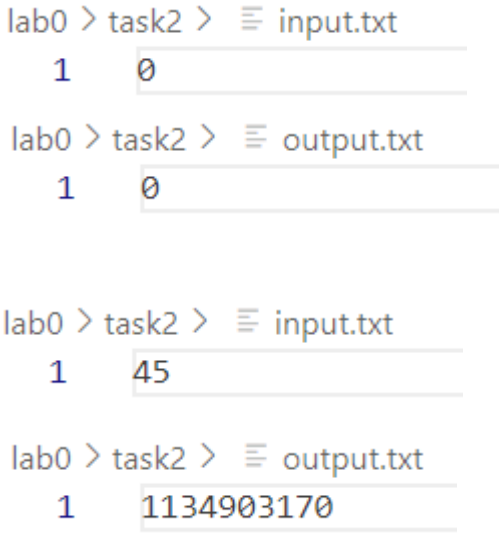
```
        else:
            print('Введены числа вне интервала [0, 45]')
    except ValueError:
        print('Число введено неверно')
```

Наивный рекурсивный алгоритм занимает много времени, поскольку функция возвращает сумму двух предыдущих значений, поэтому, например, и при нахождении значения n, и при нахождении значения n-1 находится значение n-2. Повторные вычисления одного и того же числа можно сократить, собирая найденные один раз числа, например, в словарь.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:



	Время выполнения	Затраты памяти (пиковые)
Нижняя граница диапазона значений входных данных из текста задачи	0.0008154999995895196 сек.	10506 байт

Пример из задачи	0.0005337000002327841 сек.	10549 байт
Верхняя граница диапазона значений входных данных из текста задачи	0.001109500000893604 сек.	13751 байт

Вывод по задаче: был предложен алгоритм для подсчета чисел Фибоначчи, который работает быстрее наивного рекурсивного алгоритма.

Задача №3. Еще про числа Фибоначчи

Определить последнюю цифру большого числа Фибоначчи.

- 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 Fn.

Пример 1:

input.txt	331
output.txt	9

Пример 2:

input.txt	327305
output.txt	5

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

```
import time
import tracemalloc

def fib_lastDigit(n):
    if n < 2:
        return n
    prev = 0
    curr = 1
```

```
    for _ in range(n):
        fib = (curr + prev) % 10
        prev = curr
        curr = fib
    return curr


if __name__ == '__main__':
    try:
        tracemalloc.start()
        t_start = time.perf_counter()
        with open('input.txt') as f:
            n_str = f.read()
        n = int(n_str)
        if (n >= 0) and (n <= (10 ** 7)):
            with open('output.txt', 'w') as f:
                f.write(f'{fib_lastDigit(n)}')
            print(f'Время работы: {time.perf_counter() - t_start}
сек.')
            size, peak = tracemalloc.get_traced_memory()
            print(f'Текущие и пиковые затраты памяти:
{size/(1024*1024)} и {peak/(1024*1024)} мб')
        else:
            print('Введены числа вне интервала [0, 10^7]')
    except ValueError:
        print('Число введено неверно')
```

Последняя цифра числа Фибоначчи всегда будет последней цифрой суммы последних цифр двух предшествующих чисел Фибоначчи. Дополнительно можно не сохранять результаты всех вычислений, поскольку уходим от рекурсии, так как для расчета больших чисел Фибоначчи превышается максимальная глубина рекурсии.

Результат работы кода на примерах из текста задачи:

lab0 > task3 > input.txt

```
1   331
```

lab0 > task3 > output.txt

```
1   9
```

lab0 > task3 > input.txt

```
1    327305
```

lab0 > task3 > output.txt

```
1    5
```

Результат работы кода на максимальных и минимальных значениях:

lab0 > task3 > input.txt

```
1    10000000
```

lab0 > task3 > output.txt

```
1    5
```

lab0 > task3 > input.txt

```
1    0
```

lab0 > task3 > output.txt

```
1    0
```

	Время выполнения, сек.	Затраты памяти (пиковые), мб
Нижняя граница диапазона значений входных данных из текста задачи	0.001023500008159317	0.009981155395507812
Пример из задачи	0.0014730999973835424	0.010049819946289062
Пример из задачи	0.15786340000340715	0.010052680969238281
Верхняя граница диапазона значений входных данных из текста задачи	3.417891499993857	0.010054588317871094

Вывод по задаче: был предложен алгоритм для определения последней цифры большого числа Фибоначчи, соответствующий ограничениям по времени выполнения и по затратам памяти.

Задача №4. Тестирование алгоритмов

Необходимо протестировать время выполнения алгоритма в Задаче №2 и Задаче №3. Дополнительно можно протестировать объем используемой памяти при выполнении алгоритма.

```
import time
import tracemalloc

# функция для вычисления числа Фибоначчи (последней цифры числа
Фибоначчи)

if __name__ == '__main__':
    tracemalloc.start()
    t_start = time.perf_counter()
    # чтение входных данных из файла
    # использование функции и запись результата в файл
    print(f'Время работы: {time.perf_counter() - t_start} сек.')
    size, peak = tracemalloc.get_traced_memory()
    print(f'Текущие и пиковые затраты памяти: {size/(1024*1024)} и
{peak/(1024*1024)} мб')
```

Для тестирования времени выполнения алгоритма была использована функция perf_counter() модуля time. Было зафиксировано время начала выполнения алгоритма и затем вычислена разница между временем окончания выполнения алгоритма и временем начала выполнения алгоритма.

Для тестирования объема используемой памяти при выполнении алгоритма использовался модуль tracemalloc. Перед началом выполнения алгоритма была вызвана функция start(), которая запускает отслеживание выделения памяти. Функция get_traced_memory() возвращает в том числе максимальный объем памяти, используемый во время вычислений.

Вывод по задаче: был описан возможный вариант тестирования времени выполнения алгоритма и объема используемой памяти при выполнении алгоритма.

Вывод

Были выполнены задачи на ввод-вывод с использованием результата ввода пользователя и входных данных из файла. Были разработаны эффективные алгоритмы для подсчета чисел Фибоначчи и последней цифры большого числа Фибоначчи, которые затем были протестированы с точки зрения времени выполнения и объема используемой памяти. Оказалось, что более эффективно применение цикла, а не рекурсии.