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: Тема работы

Вариант 1

Выполнил:

Говоров П. И.

К3140

Проверил:

Афанасьев А. В.

Санкт-Петербург

2024 г.

**Содержание отчета**

**Содержание отчета** 2

**Задачи по варианту** 3

Задача 1. Ввод-вывод 3

Задание 2. Число Фибоначчи 9

Задание 3. Еще про числа Фибоначчи 11

Задание 4. Тестирование ваших алгоритмов. 13

**Вывод** 16

# **Задачи по варианту**

Задача 1. Ввод-вывод

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.

Листинг кода

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

Текстовое объяснение решения.

С помощью функции *map* применяю к введенным переменным тип данных: *int,* затем вывожу сумму этих переменных.

Результат работы кода на примерах из текста задачи:

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях:Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 290 microseconds | 32 bytes |
| Пример из задачи | 150 microseconds | 28 bytes |
| Пример из задачи | 190 microseconds | 28 bytes |
| Верхняя граница диапазона значений входных данных из текста задачи | 200 microseconds | 32 bytes |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений незначительно увеличивает время работы программы. Используемая память практически константна.

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 .

Листинг кода

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

Текстовое объяснение решения.

С помощью функции *map* применяю к введенным переменным тип данных: *int,* затем вывожу сумму первой переменной и квадрата второй переменной.

Результат работы кода на примерах из текста задачи:Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описаниеРезультат работы кода на максимальных и минимальных значениях:Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 130 microseconds | 32 bytes |
| Пример из задачи | 230 microseconds | 28 bytes |
| Пример из задачи | 150 microseconds | 28 bytes |
| Верхняя граница диапазона значений входных данных из текста задачи | 300 microseconds | 32 bytes |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений незначительно увеличивает время работы программы. Используемая память практически константна.

1. 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.

1. f\_in **=** open('input.txt')
2. f\_out **=** open('output.txt', 'w')
3. a, b **=** map(int, f\_in.readline().split())
4. print(a **+** b, file**=**f\_out)
5. f\_in.close()
6. f\_out.close()

Текстовое объяснение решения.

Открываю файлы для чтения и записи. С помощью функции *map* применяю к считанным переменным тип данных: *int,* затем записываю в файл сумму этих переменных. Закрываю файлы.

Результат работы кода на примерах из текста задачи:Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, черный

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 340 microseconds | 32 bytes |
| Пример из задачи | 230 microseconds | 28 bytes |
| Пример из задачи | 250 microseconds | 28 bytes |
| Верхняя граница диапазона значений входных данных из текста задачи | 310 microseconds | 32 bytes |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений незначительно увеличивает время работы программы. Используемая память практически константна.

1. Выполните задачу a + b^2 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^2.

Листинг кода

1. f\_in **=** open('input.txt')
2. f\_out **=** open('output.txt', 'w')
3. a, b **=** map(int, f\_in.readline().split())
4. print(a **+** b**\*\***2, file**=**f\_out)
5. f\_in.close()
6. f\_out.close()

Текстовое объяснение решения.

Открываю файлы для чтения и записи. С помощью функции *map* применяю к считанным переменным тип данных: *int,* затем записываю в файл сумму первой переменной и квадрата второй переменной. Закрываю файлы.

Результат работы кода на примерах из текста задачи:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 260 microseconds | 32 bytes |
| Пример из задачи | 270 microseconds | 28 bytes |
| Пример из задачи | 250 microseconds | 28 bytes |
| Верхняя граница диапазона значений входных данных из текста задачи | 310 microseconds | 32 bytes |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений незначительно увеличивает время работы программы. Используемая память практически константна.

Задание 2. Число Фибоначчи

Определение последовательности Фибоначчи:

F0 = 0

F1 = 1

Fi = Fi−1 + Fi−2 для i ≥ 2.

Ваша цель – разработать эффективный алгоритм для подсчета чисел Фибоначчи.

• Имя входного файла: input.txt

• Имя выходного файла: output.txt

• Формат входного файла. Целое число n. 0 ≤ n ≤ 45.

• Формат выходного файла. Число Fn.

Листинг кода

1. n **=** int(input())
2. a1, a2 **=** 0, 1
4. **if** n < 0 **or** n > 45:
5. print('Err')
6. exit(0)

9. **if** n **==** 0: print(a1)
10. **elif** n **==** 1: print(a2)
11. **else**:
12. **for** i **in** range(n**-**1):
13. a1, a2 **=** a2, a1**+**a2
14. print(a2)

Текстовое объяснение решения.

При тривиальных значениях 0 и 1 сразу выводим ответ, иначе производим динамический подсчет следующего элемента через сумму двух предыдущих.

Результат работы кода на примерах из текста задачи:

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях:

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Автоматически созданное описание

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 110 microseconds | 28 bytes |
| Пример из задачи | 170 microseconds | 28 bytes |
| Верхняя граница диапазона значений входных данных из текста задачи | 230 microseconds | 32 bytes |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы значительней, чем время работы предыдущих задач. Используемая память практически константна.

Задание 3. Еще про числа Фибоначчи

Определение последней цифры большого числа Фибоначчи. Числа Фибоначчи

растут экспоненциально. Например,

F200 = 280571172992510140037611932413038677189525

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

• Имя входного файла: input.txt

• Имя выходного файла: output.txt

• Формат входного файла. Целое число n. 0 ≤ n ≤ 107

• Формат выходного файла. Одна последняя цифра числа Fn.

Листинг кода

1. n **=** int(input())
2. a1, a2 **=** 0, 1
4. **if** n < 0 **or** n > 10**\*\***7:
5. print('Err')
6. exit(0)
8. **if** n **==** 0: print(a1)
9. **elif** n **==** 1: print(a2)
10. **else**:
11. **for** i **in** range(n**-**1):
12. a1, a2 **=** a2**%**10, (a1**+**a2)**%**10
13. print(a2)

Текстовое объяснение решения.

При тривиальных значениях 0 и 1 сразу выводим ответ, иначе производим динамический подсчет следующего элемента через сумму двух предыдущих и применении к нему *mod* 10.

Результат работы кода на примерах из текста задачи:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описаниеИзображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях:

Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана, белый

Автоматически созданное описание

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 140 microseconds | 28 bytes |
| Пример из задачи | 300 microseconds | 28 bytes |
| Пример из задачи | 10970 microseconds | 28 bytes |
| Верхняя граница диапазона значений входных данных из текста задачи | 321770 microseconds | 28 bytes |

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы намного значительней, чем время работы предыдущих задач. Используемая память практически константна.

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.

Листинг кода

1. **import** time, sys
2. **from** random **import** randint
4. tests\_1 **=** [randint(0, 45) **for** i **in** range(5)]
5. tests\_2 **=** [randint(0, 10**\*\***7) **for** i **in** range(5)]
7. **def** fib\_1(n):
8. a1, a2 **=** 0, 1
10. **if** n **==** 0: **return** a1
11. **elif** n **==** 1: **return** a2
12. **else**:
13. **for** i **in** range(n**-**1):
14. a1, a2 **=** a2, a1**+**a2
15. **return** a2
17. **def** fib\_2(n):
18. a1, a2 **=** 0, 1
20. **if** n **==** 0: **return** a1
21. **elif** n **==** 1: **return** a2
22. **else**:
23. **for** i **in** range(n**-**1):
24. a1, a2 **=** a2**%**10, (a1**+**a2)**%**10
25. **return** a2
27. print('Test 2 task')
28. **for** i **in** range(5):
29. t\_start **=** time.perf\_counter()
30. ans **=** fib\_1(tests\_1[i])
31. print('INP NUM:',tests\_1[i], '| Elapsed time: %s sec' **%** round(time.perf\_counter() **-** t\_start, 8), '| Used memory: %s bytes' **%** sys.getsizeof(ans))
32. print()
33. print('Test 3 task')
34. **for** i **in** range(5):
35. t\_start **=** time.perf\_counter()
36. ans **=** fib\_2(tests\_2[i])
37. print('INP NUM:',tests\_2[i], '| Elapsed time: %s sec' **%** round(time.perf\_counter() **-** t\_start, 5), '| Used memory: %s bytes' **%** sys.getsizeof(ans))

Текстовое объяснение решения.

Подключаем библиотеки ***random, sys, time***

Генерируем с помощью метода *randint* из модуля *random* случайные тесты для 2-ой и 3-ей задач.

Создаем функцию fib\_1 выполняющую код из задачи 2

Создаем функцию fib\_2 выполняющую код из задачи 3

Обрабатываем каждый тест из набора через цикл *for*

С помощью методов *time.perf\_counter()* и *sys.getsizeof(ans),* вычисляем соответственно затраченное время и память каждого теста.

Результат работы кода на тестах:

Изображение выглядит как текст, снимок экрана, Шрифт, дизайн

Автоматически созданное описание

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из теста задачи | 141 microseconds | 28 bytes |
| Верхняя граница диапазона значений входных данных из теста задачи | 263970 microseconds | 32 bytes |

Вывод: с помощью методов *time.perf\_counter()* и *sys.getsizeof()* можно измерить затраченное время работы и памяти программы.

**Вывод**

1. Выполнение арифметических операций.

Сложение и возведение в квадрат в рамках ограничений

−10^9 ≤ n ≤ 10^9 выполняются практически моментально

Память используется для хранения числовых переменных практически константна, изменяется только при переходе между числовыми типами данных

1. Обычный ввод и вывод осуществляемый через терминал расходует больше времени, чем считывание и запись данных в файл
2. Линейный алгоритм O(n) выполняется прямопропорционально величине данных, затраты памяти неизменны
3. С помощью методов *time.perf\_counter()* и *sys.getsizeof()* можно отслеживать ресурсозатратность алгоритмов.