Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Мегафакультет трансляционных информационных технологий

Факультет инфокоммуникационных технологий

Дисциплина: Алгоритмы и структуры данных

**Отчет по Лабораторной работе №1**

Выполнила: Микулина Алиса Романовна

Группа: K3143, 1 курс

Преподаватель: Харьковская Татьяна Александровна

Санкт-Петербург

10.09.2021

**Описание задания**

**Задание 1.**

1.1. Задача a + b. Необходимо вычислить сумму двух заданных чисел.

1.2. Задача a + b2. Необходимо вычислить значение a + b2

1.3. Задача a + b с использованием входных/выходных файлов. Необходимо вычислить сумму двух заданных чисел используя внешние файлы.

1.4. Задача a + b2 с использованием входных/выходных файлов. Необходимо вычислить сумму двух заданных чисел используя внешние файлы.

**Задание 2.**

Число Фибоначчи. Требуется разработать эффективный алгоритм для подсчета чисел Фибоначчи с использованием внешних входных/выходных файлов. На вход поступает файл с порядковым номером необходимого числа Фибоначчи, на выходе необходимо получить файл с этим числом.

**Задание 3.**

Необходимо определить последнюю цифру какого-либо большого числа Фибоначчи по порядковому номеру этого числа Фибоначчи с помощью эффективного алгоритма. При этом ограничение по времени работы алгоритма: 5сек; ограничение по памяти: 512 мб.

**Задание 4.**

Необходимо протестировать время выполнения и объем используемой памяти алгоритма в Задании 2 и Задании 3.

**Описание решения и исходный код**

**Задача 1.**

**1.1.** Решение данного задания занимает всего две строки и является самым логичным и оптимальным:

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

В первой строке с помощью стандартного ввода **input()** мы получаем значение типа **str**, которое разделяем на две части в помощью метода **.split()**. С помощью функции **map** применяем преобразование в тип **int** к каждому из полученных двух значений. При этом переменным **a** и **b** соответственно присваиваются эти два значения.

Вторая строка выводит результат сложения чисел **a** и **b**.

**1.2.** Данное задание отличается от предыдущего только тем, что необходимо вывести не сумму двух данных чисел, а сумму первого числа с квадратом второго. Соответственно, алгоритм крайне схож с алгоритмом из предыдущего задания.

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

Первая строка полностью идентична первой строке из первой части задания.

Во второй строке функция print() выводит сумму числа **a** с квадратом числа **b (b\*\*2)**, используя оператор **\*\***.

**1.3.** В данном задании необходимо взять два числа для сложения не из стандартного ввода, а из заранее заготовленного файла.

f = open("input.txt", "r")  
line = f.read()  
f.close()  
  
a,b = map(int, line.split())  
  
d = open("output.txt", "w")  
d.write(str(a+b))  
d.close()

Для корректной работы данного кода файл с числами **(input.txt)** должен находиться в той же папке, где находится запущенный проект python.

Для начала работы необходимо открыть файл с числами. Для этого используем функцию **open()**, в качестве аргументов указывая название файла, который требуется открыть и способ открытия данного файла (**“r”** значит, что файл открывается для чтения).

Далее переменной line присваивается значение, записанное в файле, и файл закрывается: **f.close()**.

Далее, с помощью функции **map**, строка **line** преобразуется в два числа, значения которых присваиваются переменным.

Далее мы снова открываем файл, но уже не ранее существовавший, а новый. Теперь, в качестве аргумента, отвечающего за способ открытия файла, я использую **“w”**, что дает мне возможность записывать что-либо в файле.

Далее, с помощью метода **.write()**, сумма **a** и **b** (преобразованная в тип **str**, так как .**write()** работает исключительно с типом **str**) вписывается в файл, после чего файл закрывается.

**1.4.** Данное задание снова практически идентично предыдущему, ведь единственное различие — необходимо получить файл не с суммой **a** и **b**, а с суммой **a** и **b2**.

f = open("input.txt", "r")  
line = f.read()  
f.close()  
  
a,b = map(int, line.split())  
  
d = open("output.txt", "w")  
d.write(str(a+(b\*\*2)))  
d.close()

Принцип работы кода точно такой же, как в задании 1.3., меняется лишь аргумент у метода **.write()**, где вместо **a+b** записано a**+(b\*\*2)**.

**Задача 2.**

Спустя некоторое врямя я смогла разработать более быстрый, а соответственно и эффективный алгоритм расчета чисел Фибоначчи. Он основан на принцие предыдущего кода, но используются не массивы, а три переменные. Переменные **fib1** и **fib2** хранят числа, которые необходимо сложить, а переменная **fibsum** используется для подсчета суммы двух чисел, хранящихся в вышеупомянутых переменных. После каждой итерации, **fib1** присваивается значение **fib2**, а **fib2** присваивается значение **fibsum**. Соответственно, **fibsum** снова можно использовать для подсчета суммы **fib1** и **fib2**. Цикл замыкается.

Также в данном коде я рассмотрела варианты **n = 1** и **n = 2**, так как в предыдущем коде данные точки были безжалостно мною проигнорированны (каюсь и клянусь впредь быть более внимательной к крайним точкам).

f = open("input.txt", "r")

n = int(f.read())

f.close()

fib1 = 0

fib2 = 1

fibsum = 0

if n == 1:

   fibsum = fib1

elif n == 0:

   fibsum = 0

elif n == 2:

   fibsum = fib2

else:

   for i in range(n - 1):

       fibsum = fib1 + fib2

       fib1 = fib2

       fib2 = fibsum

d = open("output.txt", "w")

d.write(str(fibsum))

d.close()

**Old Version:**

Для разработки эффективного алгоритма для подсчета чисел Фибоначчи, очевидно, рекурсивный метод совершенно не эффективен. Мало того, что для каждого нового числа весь цикл рекурсии запускается заново, так еще и для каждого отдельного числа некоторые значения будут подсчитываться десятки, если не сотни раз. Подобное повторение занимает огромное количество времени и перегружает память.

Вследствие этого, я разработала более длинный, но гораздо более быстрый код, занимающий меньше времени и памяти.

f = open("input.txt", "r")  
n = int(f.read())  
f.close()  
  
fiblist = [0,1]  
uselist = [0,1]  
  
for i in range (n-1):  
    uselist[0]=(fiblist[1])  
    uselist[1]=(fiblist[0] + fiblist[1])  
    for i in range(2):  
        fiblist[i-1] = uselist[i-1]  
  
d = open("output.txt", "w")  
d.write(str(uselist[-1]))  
d.close()

Получив номер числа Фибоначчи, которое необходимо найти, из файла, можно начать расчет такого числа.

Первым делом я пыталась создать код, используя единственный массив **fiblist**. На начальном этапе **fiblist** присваивалось значение **[0, 1]**. Далее, с помощью итератора и метода **.append()** я поочередно добавляла каждое последующее число, используя при этом предыдущие два записанные в массиве.

Минусы данного подхода были налицо: в массиве хранятся все числа Фибоначчи, даже те, что уже были использованы для подсчета последующих и никогда больше использоваться не будут. При подсчете числа Фибоначчи с порядковым номером больше **1000** компьютер начинал зависать из-за перегруженной памяти. Это был уже шаг вперед, ведь при использовании рекурсивного метода компьютер замедлял работу начиная с порядкового номера больше **40**.

Несмотря на значительные преимущества уже написанного кода, неспокойной душе программиста необходимо было создать такой код, чтобы он быстро работал, и при этом не перегружал память компьютера. Как избежать перегрузки памяти и зависания? — Нужно сделать так, чтобы те значения, которые никогда уже не будут использоваться при расчете, уничтожались и не засоряли кэш.

Для осуществления данной идеи мне потребовалось добавить второй массив в код. Массив **fiblist** является основным хранителем для подсчета последующих чисел. Массив **uselist** используется как хранилище только что подсчитанных чисел.

Два вышеупомянутых массива я поместила в цикл **for**, который производится **n-1** раз из-за того, что первая пара чисел уже и так находится в массивах.

В строке **uselist[0]=(fiblist[1])** мы присваиваем последнее значение из массива-счетчика первой ячейке массива-хранителя. Далее, во вторую ячейку массива-хранителя вносится число, равное сумме обоих значений из массива-счетчика: **uselist[1]=(fiblist[0] + fiblist[1])**. После этого значения ячеек массива-счетчика обновляются и приравниваются к значениям ячеек массива-хранителя. Цикл повторяется, пока не достигнет необходимого числа.

*Sidenote: в процессе я поняла, что если я хочу, чтобы значения ячеек массивов приравнивались к значениям ячеек другого массива, то невозможно просто приравнять два массива. В случае приравнивания этих массивов они начинают ссылаться на один и тот же объект, и работать с каждым из них раздельно после подобного приравнивания невозможно. Чтобы каждый из массивов оставался самостоятельным и ссылался на собственный объект, нужно приравнивать значения ячеек массивов, а не сами массивы.*

В результате мы получили алгоритм, с помощью которого можно быстро и эффективно найти нужное число Фибоначчи. Такой код работает быстро и, помимо этого, он не перегружает компьютер лишней информацией даже при вводе числа больше **1000000**.

**Задача 3.**

После внесения корректировок в код из предыдущего задания, я отредактировала и Задание 3. По прежнему в данном коде с целью наименьшей перегрузки памяти я “обрезаю” все цифры чисел, кроме последней на всех этапах. Код функцинирует быстрее примерно в 7-8 раз, чем предыдущий.

f = open("input.txt", "r")

n = int(f.read())

f.close()

fib1 = 0

fib2 = 1

fibsum = 0

if n == 1:

   fibsum = fib1

elif n == 0:

   fibsum = 0

elif n == 2:

   fibsum = fib2

else:

   for i in range(n - 1):

       fibsum = fib1 + fib2

       fib1 = fib2

       fib2 = fibsum % 10

d = open("output.txt", "w")

d.write(str(fibsum % 10))

d.close()

**Old Version#1:**

В данном задании нужно создать код, с помощью которого по порядковому номеру можно узнать не все число Фибоначчи, а лишь его последнюю цифру.

В теории, можно бы было оставить полностью код из предыдущего задания. Тогда лишь в самом конце строчка

d.write(str(uselist[-1]))  заменилась бы на

d.write(str(uselist[-1] % 10)).

Такой код, несомненно, был бы и быстрым, и затрачивающим небольшое количество памяти компьютера. Однако, если размышлять логически, можно проследить, что только последние цифры предыдущих чисел Фибоначчи влияют на то, какой будет последняя цифра последующего числа. Таким образом, можно совершенно избавиться от всех цифр чисел Фибоначчи, кроме последней цифры каждого числа.

Казалось бы, что это совершенно незначительная корректировка кода, и это действительно так. Однако, если мы стремимся к максимальной эффективности кода, то нам необходимо помнить, что каждый символ занимает определенное количество памяти. Избавляясь от бессмысленных в данной ситуации лишних цифр, мы освобождаем пусть и небольшое, но количество памяти, которое в некоторых ситуациях может являться критическим.

После размышлений и корректировки алгоритма из предыдущего задания получаем такой код:

f = open("input.txt", "r")  
n = int(f.read())  
f.close()  
  
fiblist = [0,1]  
uselist = [0,1]  
  
for i in range (n-1):  
    uselist[0]= (fiblist[1]) % 10  
    uselist[1]=(fiblist[0] + fiblist[1]) % 10  
    for i in range(2):  
        fiblist[i-1] = uselist[i-1]  
  
d = open("output.txt", "w")  
d.write(str(uselist[-1]))  
d.close()

В результате алгоритм подсчитывает последние цифры чисел Фибоначчи, при этом отбрасывая лишние цифры в процессе каждой итерации.

**Задание 4.**

Требовалось протестировать время выполнения алгоритма в заданиях 2 и 3.

В обоих заданиях это возможно сделать с помощью импортирования модуля **time** и использования метода **.perf\_counter()**.

import time  
t\_start = time.perf\_counter()

\* код, который необходимо протестировать \*

print("Время работы: %s секунд " % (time.perf\_counter() - t\_start))

С помощью этого кода я отслеживала различия в эффективности кодов на протяжении всего процесса написания. Этот же код помог вычислить наиболее логичную траекторию мысли для сокращения количества затрачиваемого времени.

**Описание проведенных тестов.**

**1.1**

ввод: -1000000000 1

вывод: -999999999

ввод: 1 1000000000

вывод: 1000000001

ввод: 12 25

вывод: 37

ввод: 130 61

вывод: 191

**1.2**

ввод: 1000000000 2

вывод: -999999996

ввод: 1 1000000000

вывод: 1000000000000000001

ввод: 12 25

вывод: 637

ввод: 130 61

вывод: 3851

**1.3**

ввод: -1000000000 1

вывод: -999999999

ввод: 1 1000000000

вывод: 1000000001

ввод: 12 25

вывод: 37

ввод: 130 61

вывод: 191

**1.4**

ввод: 1000000000 2

вывод: -999999996

ввод: 1 1000000000

вывод: 1000000000000000001

ввод: 12 25

вывод: 637

ввод: 130 61

вывод: 3851

**2**

ввод: 0

вывод: 0

Время работы: 0.0007320999999999994 секунд

ввод: 45

вывод: 1134903170

Время работы: 0.0007541000000000006 секунд

ввод: 10

вывод: 55

Время работы: 0.0011368000000000003 секунд

**3**

ввод: 0

вывод: 0

Время работы: 0.0010020999999999988 секунд

ввод: 10000000

вывод: 5

Время работы: 1.5236531 секунд

ввод: 331

вывод: 9

Время работы: 0.0011002000000000026 секунд

ввод: 3273055

вывод: 5

Время работы: 0.4411897 секунд

**Выводы по проделанной работе.**

При выполнении данной лабораторной работы я вспомнила, как работать с простейшими алгоритмами Python, вспомнила принцип работы рекурсии, способы взаимодействия с файлами на ввод/вывод, импорт модулей, а также научилась отслеживать и модифицировать эффективность написанного кода.