Алгоритмы и структуры данных

Задание к лабораторной работе №0. Введение

0 Проверьте, установлен ли Python на вашем компьютере и установите его, если это требуется. Посетите страницу http://www.python.org/download/ и загрузите последнюю версию.

Для справки: Swaroop C.H. A Byte of Python (Russian). Вы можете использовать любой удобный для вас редактор.

Запустите редактор Python или IDLE.

Задание 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.

Небольшая справка по работе с файлами. (стр. 121 - A Byte of Python):

Рано или поздно возникают ситуации, когда программа должна взаимодействовать с пользователем. Например, принять какие-нибудь данные от пользователя, а затем вывести результаты. Для этого применяются функции `input()` и `print()` соответственно

Одним из типом ввода/вывода является работа с файлами. Возможность создавать, читать и записывать в файлы является ключевой для многих программ.

Открывать и использовать файлы для чтения или записи можно путём создания объекта класса `file`, а читать/записывать в файл – при помощи его методов `read, readline` или `write` соответственно. Возможность читать или записывать в файл зависит от режима, указанного при открытии файла. По окончании работы с файлом, нужно вызвать метод `close`, чтобы указать Python, что файл больше не используется.

Пример:

```
poem = '''\
Программировать весело.
Если работа скучна,
Чтобы придать ей весёлый тон -
используй Python!
'''

f = open('poem.txt', 'w') # открываем для записи (writing)
f.write(poem) # записываем текст в файл
f.close() # закрываем файл

f = open('poem.txt') # если не указан режим, по умолчанию подразумевается
                # режим чтения ('r'eading)
while True:
        line = f.readline()
        if len(line) == 0: # Нулевая длина обозначает конец файла (EOF)
                break
        print(line, end='')

f.close() # закрываем файл
```

Как это работает: Сперва мы открываем файл при помощи встроенной функции open с указанием имени файла и режима, в котором мы хотим его открыть. Режим может быть для чтения (`'r'`), записи (`'w'`) или добавления (`'a'`). Можно также указать, в каком виде мы будем считывать, записывать или добавлять данные: в текстовом (`'t'`) или бинарном (`'b'`). На самом деле существует много других режимов, и `help(open)` даст вам их детальное описание. По умолчанию `open()` открывает файл как текст в режиме для чтения.

В нашем примере мы сначала открываем файл в режиме записи текста и используем метод `write` файлового объекта для записи в файл, после чего закрываем файл при помощи `close`.

Далее мы открываем тот же самый файл для чтения. В этом случае нет нужды указывать режим, так как режим «чтения текстового файла» применяется по

умолчанию. Мы считываем файл построчно методом `readline` в цикле. Этот метод возвращает полную строку, включая символ перевода строки в конце. Когда же он возвращает пустую строку, это означает, что мы достигли конца файла, и мы прерываем цикл при помощи `break`.

По умолчанию функция `print()` выводит текст, автоматически добавляя символ перевода строки в конце. Мы подавляем этот символ, указывая `end=''`, поскольку строки, считанные из файла, и без того оканчиваются символом перевода строки. И, наконец, мы закрываем файл с помощью `close`.

Теперь проверяем содержимое файла `poem.txt`, чтобы убедиться, что программа действительно записала текст в него и считала из него.

Задание 2. Число Фибоначчи

Определение последовательности Фибоначчи:

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_i &= F_{i-1} + F_{i-2} \text{ для } i \geq 2. \end{aligned} \tag{1}$$

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

Задание 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 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. $0 \leq n \leq 10^7$.
- Формат выходного файла. Одна последняя цифра числа F_n.
- Пример 1.

input.txt	331
output.txt	9

  $F_{331} = 668996615388005031531000081241745415306766517246774551964595292186469$.

- Пример 2.

input.txt	327305
output.txt	5

  Это число не влезет в страницу, но оканчивается действительно на 5.

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

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.

Небольшая справка:

Что делать, если ваш алгоритм не работает? Как правило, проблема в следующем:

- Неправильный ответ
- Превышен предел времени / памяти

- Ошибка (ошибка времени выполнения)

Для полноценной проверки работоспособности вашего алгоритма вы можете пройтись по следующим шагам (в зависимости от ситуации):

- Проверьте, не забыли ли вы какой-нибудь крайний случай. Например, когда $n = 1$ или n - максимально возможное число, удовлетворяющее ограничениям. Проверьте, правильно ли ведет себя ваша программа, учитывая, что входные данные наталкиваются на ограничения во всех возможных смыслах.
- Сделайте несколько общих тестов, на которые вы знаете правильный ответ. Не смотрите на ответ вашего кода для этих тестов, пока не поймете, какой ответ должен быть при помощи ручки и бумаги. В противном случае легко убедить себя в том, что это правильно, и не увидеть какой-нибудь глупой ошибки.
- Если превышен лимит времени, измерьте, как долго ваша программа работает для больших значений входных данных. Следующие функции помогают измерить процессорное время с момента запуска программы:
  - **C++** `(double)clock() / CLOCKS PER SEC` with `ctime` included.
  - **python** `time.perf_counter()` returns floating-point value in seconds.
  - **Java** `System.nanoTime()` returns long value in nanoseconds

  Использование:

```
import time
t_start = time.perf_counter()
# ваш код
print("Время работы: %s секунд " %  (time.perf_counter() - t_start))
```

  Измерьте время для небольших значений входных данных, а также средних и больших значений. Вы можете столкнуться с одним из возможных результатов:

  - Ваша программа выполняется быстро для малых и средних значений, но для больших значений она работает более чем в 10 раз медленнее, чем необходимо (или просто зависает). В этом случае у вас, вероятно, есть проблемы со сложностью. Вы можете:
    * Измерить время, которое части программы занимают отдельно (например, сколько времени занимает чтение ввода / печать вывода)
    * Вычислить фактическое количество операций, выполняемых вашим алгоритмом и его частями, и посмотрите, соответствует ли оно ожидаемому.

  - Ваша программа зависает при небольших или средних входных значениях. Можно проверить наличие бесконечного цикла / рекурсии. Добавьте Assertions (утверждения) (`assert` в c ++, python и java) в предусловия и постусловия ваших циклов и функций и посмотрите, не сработают ли они. Используйте отладчик, чтобы узнать, какая часть кода приводит к зависанию.

- Если вы получили ошибку во время выполнения, то это наиболее информативный вердикт, который означает, что ваша программа вылетает из-за одного из следующих факторов:
  - Вы пытаетесь получить доступ к области памяти, которая не принадлежит вашей программе. Это может принимать две формы: вы пытаетесь получить доступ к несуществующему элементу массива, вы пытаетесь оценить нулевой указатель или указатель, указывающий на место, которое не принадлежит вашей программе.
  - Вы допустили арифметическую ошибку: деление на ноль, переполнение числа с плавающей запятой.
- Для проверки правильности выполнения вашего алгоритма, можно использовать альтернативное решение, которое может быть неэффективно (и даже не работать при некоторых значениях входных данных), но которое можно использовать для проверки правильности ответов вашего решения.

Возможно, вы захотите ознакомиться:

♡ *Еще раз:* Swaroop C.H. A Byte of Python (Russian).

♡ Глава 3.2: Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы. Построение и анализ. 2013

♡ Раздел 0.3: S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani. Algorithms. 2016

♡ Alternative testing guide