Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

ОТЧЕТ

По практической работе №2.9

Дисциплины «Программирование на Python»

Выполнил:

Пустяков Андрей Сергеевич

2 курс, группа ИВТ-б-о-22-1,

09.03.01 «Информатика и вычислительная техника (профиль) «Программное обеспечение средств вычислительной
техники и автоматизированных систем», очная форма обучения

_____

(подпись)

Руководитель практики:

Воронкин Р. А. кандидат технических наук, доцент, доцент кафедры инфокоммуникаций

_____

(подпись)

Ставрополь, 2023 г.

Тема: Рекурсия в языке Python.

Цель: приобрести навыки по работе с рекурсивными функциями при написании программ с помощью языка программирования Python версии 3.x.

Ход работы:

Проработка примеров лабораторной работы:

Задание 7.

Необходимо самостоятельно изучить работу с пакетом «timeit» и с помощью этого модуля оценить скорость работы итеративной и рекурсивной версии функций «fib» и «factorial». Необходимо определить во сколько раз изменится скорость работы рекурсивных версий обеих функций с использованием декоратора «lru_cache».

Код программы задания и результаты работы программы (рис. 1, 2).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import timeit
from functools import lru_cache


def recursion_factorial(n):
    """Рекурсивный алгоритм вычисления факториала."""

    if n == 0:
        return 1
    else:
        return n * recursion_factorial(n - 1)


def recursion_fib(n):
    """Рекурсивный алгоритм вычисление числа Фибоначчи."""

    if 1 >= n >= 0:
        return n
    else:
        return recursion_fib(n - 2) + recursion_fib(n - 1)


def iterative_factorial(n):
    """Итеративный алгоритм вычисления факториала."""

    value = 1
    while n > 1:
        value *= n
        n -= 1
    return value
```

```python
def iterative_fib(n):
    """Итеративный алгоритм вычисления числа Фибоначчи."""

    a, b = 0, 1
    while n > 0:
        a, b = b, a + b
        n -= 1
    return a


@lru_cache
def recursion_factorial_lru(n):
    """
    Рекурсивный алгоритм вычисления факториала с применением декоратора
lru_cache.
    """

    if n == 0:
        return 1
    else:
        return n * recursion_factorial_lru(n - 1)


@lru_cache
def recursion_fib_lru(n):
    """
    Рекурсивный алгоритм вычисления числа Фибоначчи с применением декоратора
lru_cache.
    """

    if 1 >= n >= 0:
        return n
    else:
        return recursion_fib_lru(n - 2) + recursion_fib_lru(n - 1)


if __name__ == '__main__':
    """
    Определение среднего времени работы алгоритмов нахождения факториалов
    чисел и чисел Фибоначчии рекурсивними алгоритмами и итеративными
    алгоритмами, а также с применением декоратора lru_cache и без него.
    """

    # Найдем среднее время, которое требуется для вычисления факториала числа
рекурсивным алгоритмом.
    for i in range(10, 30):
        str_i = str(i)
        time_work = timeit.timeit("recursion_factorial("+str_i+")",
globals=globals(), number=20)
        print(f'Время выполнения рекурсивного алгоритма вычисления '
              f'факториала для', i, '-го числа =', time_work, 'сек.')
    print('\n')

    # Найдем среднее время, которое требуется для вычисления факториала числа
итеративным алгоритмом.
    for i in range(10, 30):
        str_i = str(i)
        time_work = timeit.timeit("iterative_factorial("+str_i+")",
globals=globals(), number=20)
        print(f'Время выполнения итеративного алгоритма вычисления '
              f'факториала для', i, '-го числа =', time_work, 'сек.')
    print('\n')
```

```
    # Найдем среднее время, которое требуется для вычисления факториала
    # числа рекурсивным алгоритмом с применением декоратора lru_cache.
    for i in range(10, 30):
        str_i = str(i)
        time_work = timeit.timeit("recursion_factorial_lru("+str_i+")",
globals=globals(), number=20)
        print(f'Время выполнения рекурсивного алгоритма вычисления '
              f'факториала с lru_cache для', i, '-го числа =', time_work,
'сек.')
    print('\n')

    # Найдем среднее время, которое требуется для вычисления n-го числа
Фибоначчи рекурсивным алгоритмом.
    for i in range(15, 25):
        str_i = str(i)
        time_work = timeit.timeit("recursion_fib(" + str_i + ")",
globals=globals(), number=20)
        print(f'Время выполнения рекурсивного алгоритма вычисления',
              i, '-го числа Фибоначчи =', time_work, 'сек.')
    print('\n')

    # Найдем среднее время, которое требуется для вычисления n-го числа
Фибоначчи итеративным алгоритмом.
    for i in range(15, 25):
        str_i = str(i)
        time_work = timeit.timeit("iterative_fib(" + str_i + ")",
globals=globals(), number=20)
        print(f'Время выполнения итеративного алгоритма вычисления',
              i, '-го числа Фибоначчи =', time_work, 'сек.')
    print('\n')

    # Найдем среднее время, которое требуется для вычисления n-го числа
    # Фибоначчи рекурсивным алгоритмом с применением декоратора lru_cache.
    for i in range(15, 25):
        str_i = str(i)
        time_work = timeit.timeit("recursion_fib_lru(" + str_i + ")",
globals=globals(), number=20)
        print(f'Время выполнения рекурсивного алгоритма вычисления',
              i, '-го числа Фибоначчи с применением декоратора lru_cache =',
time_work, 'сек.')
    print('\n')
```

Рисунок 1 – Код программы определения времени работы функций.

```
Время выполнения рекурсивного алгоритма вычисления факториала для 10 -го числа = 2.5700000000003498e-05 сек.
Время выполнения рекурсивного алгоритма вычисления факториала для 11 -го числа = 2.5799999999999434e-05 сек.
Время выполнения рекурсивного алгоритма вычисления факториала для 12 -го числа = 2.769999999999856e-05 сек.
Время выполнения рекурсивного алгоритма вычисления факториала для 13 -го числа = 5.460000000000187e-05 сек.
Время выполнения рекурсивного алгоритма вычисления факториала для 14 -го числа = 3.290000000000237e-05 сек.
Время выполнения рекурсивного алгоритма вычисления факториала для 15 -го числа = 3.440000000000387e-05 сек.
Время выполнения рекурсивного алгоритма вычисления факториала для 16 -го числа = 3.689999999999943e-05 сек.
Время выполнения рекурсивного алгоритма вычисления факториала для 17 -го числа = 4.069999999999768e-05 сек.
Время выполнения рекурсивного алгоритма вычисления факториала для 18 -го числа = 4.350000000000187e-05 сек.
Время выполнения рекурсивного алгоритма вычисления факториала для 19 -го числа = 4.5900000000001495e-05 сек.
Время выполнения рекурсивного алгоритма вычисления факториала для 20 -го числа = 5.010000000000431e-05 сек.
Время выполнения рекурсивного алгоритма вычисления факториала для 21 -го числа = 5.180000000000462e-05 сек.
Время выполнения рекурсивного алгоритма вычисления факториала для 22 -го числа = 0.0001056000000000043 сек.
Время выполнения рекурсивного алгоритма вычисления факториала для 23 -го числа = 7.420000000000343e-05 сек.
Время выполнения рекурсивного алгоритма вычисления факториала для 24 -го числа = 6.47999999999968e-05 сек.
Время выполнения рекурсивного алгоритма вычисления факториала для 25 -го числа = 6.189999999999668e-05 сек.
Время выполнения рекурсивного алгоритма вычисления факториала для 26 -го числа = 6.559999999999899e-05 сек.
Время выполнения рекурсивного алгоритма вычисления факториала для 27 -го числа = 7.440000000000224e-05 сек.
Время выполнения рекурсивного алгоритма вычисления факториала для 28 -го числа = 7.199999999999568e-05 сек.
Время выполнения рекурсивного алгоритма вычисления факториала для 29 -го числа = 7.459999999999412e-05 сек.


Время выполнения итеративного алгоритма вычисления факториала для 10 -го числа = 1.6399999999999748e-05 сек.
Время выполнения итеративного алгоритма вычисления факториала для 11 -го числа = 1.7400000000000748e-05 сек.
Время выполнения итеративного алгоритма вычисления факториала для 12 -го числа = 1.8499999999997685e-05 сек.
Время выполнения итеративного алгоритма вычисления факториала для 13 -го числа = 2.0499999999999685e-05 сек.
Время выполнения итеративного алгоритма вычисления факториала для 14 -го числа = 2.2199999999999998e-05 сек.
Время выполнения итеративного алгоритма вычисления факториала для 15 -го числа = 2.4200000000001998e-05 сек.
```

Рисунок 2 – Результаты работы программы определения времени работы функций.

График сравнения времени работы алгоритмов нахождения факториала числа (рекурсивный, итеративный и рекурсивный с использованием декоратора lru_cache) (рис. 3).

Рисунок 3 – График времени работы функций нахождения факториалов чисел.

График сравнения времени работы алгоритмов нахождения чисел Фибоначчи (рекурсивный, итеративный и рекурсивный с использованием декоратора lru_cache) (рис. 4).

Рисунок 4 – График времени работы функций нахождения чисел Фибоначчи.

Исходя из результатов графиков видно, что итеративные алгоритмы в сравнении с рекурсивными работают гораздо быстрее, скорость роста рекурсивных алгоритмов больше скорости роста итеративных, таким образом рекурсия чаще всего не является лучшим способом решить те или иные задачи (при большом количестве рекурсивных вызовов возрастает время работы программы), однако рекурсивные алгоритмы с использованием декоратора lru_cache могут работать даже быстрее итеративных. Декоратор lru_cache позволяет не делать лишних вычислений в рекурсивных алгоритмах.

Выполнение индивидуального задания:

Дан список X из целых n вещественных чисел. Необходимо найти минимальный элемент списка, используя вспомогательную рекурсивную функцию, находящую минимум среди последних элементов списка X, начиная с n-го (Вариант 26(12)).

Код программы индивидуального задания с рекурсивной вспомогательной функцией и результаты работы программы (рис. 5, 6).

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import random


def min_recursive(list_x, n, border_n, min_element):
    """
    Рекурсивная функция поиска минимума среди элементов от n-го элемента до
конца списка X.
    """
    if list_x[border_n] < min_element:
        min_element = list_x[border_n]
    if border_n == n - 1:
        return min_element
    else:
        return min_recursive(list_x, n, border_n + 1, min_element)


if __name__ == '__main__':
    """
    Дан список X из целых n вещественных чисел. Необходимо найти минимальный
    элемент списка, используя вспомогательную рекурсивную функцию, находящую
    минимум среди последних элементов списка X, начиная с n-го.
    """

    n = int(input("Введите количество элементов в списке X: "))
    # Создадим список X указанной длины из случайных элементов.
    list_x = [random.randint(-1000, 1000) for i in range(n + 1)]
    border_n = int(input("Введите номер элемента с которого необходимо искать
минимальный элемент списка X: "))
    min_element = list_x[border_n]

    print("Минимальный элемент начиная с", border_n, "-го в списке
вещественных чисел = ",
          min_recursive(list_x, n, border_n, min_element))
```

Рисунок 5 – Код программы индивидуального задания.

```
Введите количество элементов в списке X: 100
Введите номер элемента с которого необходимо искать минимальный элемент списка X: 50
Минимальный элемент начиная с 50 -го в списке вещественных чисел =  -993

Process finished with exit code 0
```

Рисунок 6 – Результаты работы программы индивидуального задания.

Ответы на контрольные вопросы:

1. Рекурсии используются, когда задача может быть разбита на более мелкие подзадачи того же типа. При их использовании функция вызывает саму себя, передавая новому варианту иные входные значения.

2. База рекурсии – это условие, при котором рекурсивные вызовы функции прекращаются и начинается возврат из рекурсивных вызовов. Такие аргументы функции делают задачу настолько простой, что решение не требует дальнейших вложенных вызовов.

3. Стек программы является структурой данных, хранящей информацию о вызовах функций во время выполнения программы. Информация последовательно добавляется в «стопку», каждый новый объект помещается поверх предыдущего, а извлекаются объекты в обратном порядке, начиная с верхнего. При вызове функции, информация о текущем её состоянии (локальные переменные, адрес возврата), помещается в стек. Когда функция завершает выполнение, информация извлекается из стека, и управление передается обратно вызывающей функции. Это позволяет программе возвращаться к предыдущему состоянию после завершения выполнения вызванной функции.

4. Чтобы получить максимальную глубину рекурсии, достаточно использовать функцию «sys.getrecursionlimit()», возвращающую максимальное количество рекурсивных вызовов, которое может быть выполнено до возникновения ошибки «RecursionError».

5. При попытке превысить допустимую глубину рекурсий, будет выдана ошибка «RecursionError» или на старых версиях Python «RuntimeError».

6. Функция «setrecursionlimit()» позволяет изменить максимальную допустимую глубину рекурсий.

7. Декоратор lru_cache кэширует результаты функции в соответствии с последними аргументами вызова. Если функция вызывается с теми же аргументами, что и ранее, результат возвращается из кэша, вместо того чтобы

вычислять его снова. Это существенно улучшает производительность рекурсивных функций так как предотвращает лишние вычисления.

8. Хвостовая рекурсия - частный случай рекурсии, при котором любой рекурсивный вызов является последней операцией перед возвратом из функции. Для оптимизации хвостовой рекурсии компилятор или интерпретатор может заменить рекурсивный вызов на цикл, что позволяет избежать увеличения стека вызовов.

Вывод: в ходе лабораторной работы были рассмотрены принципы работы рекурсивных функций, применение рекурсии, ошибки, которые могут возникнуть в рекурсивных функциях (достижение предела максимальной глубины рекурсии), а также недостатки рекурсивных функций (большое время работы рекурсивных функций по сравнению с итеративными). Также был рассмотрен модуль языка Python «timeit», который позволяет определить среднее время работы функций или участков кода. Также был рассмотрен способ избежания лишних вычислений и сокращение времени работы рекурсивных функций, такое как использование декоратора «lru_cache».