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 ЛАБОРАТОРНОЙ РАБОТЕ №12
дисциплины «Программирование на Python»

Вариант 7

Выполнил:
Кулешов Олег Иванович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Руководитель практики:
Воронкин Р. А.
_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023

Тема: Рекурсия в языке Python

Цель работы: приобретение навыков по работе с рекурсивными функциями при написании программ с помощью языка программирования Python версии 3.x.

Задание. Самостоятельно изучите работу со стандартным пакетом Python timeit . Оцените с помощью этого модуля скорость работы итеративной и рекурсивной версий функций factorial и fib . Во сколько раз изменится скорость работы рекурсивных версий функций factorial и fib при использовании декоратора lru_cache? Приведите в отчет и обоснуйте полученные результаты.

Листинг:

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

import timeit
from functools import lru_cache


# Итеративная версия функции factorial
def factorial_iterative(n):
    result = 1
    for i in range(1, n+1):
        result *= i
    return result


# Рекурсивная версия функции factorial
def factorial_recursive(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial_recursive(n-1)


# Итеративная версия функции fib
def fib_iterative(n):
    a, b = 0, 1
    for _ in range(n):
        a, b = b, a + b
    return a


# Рекурсивная версия функции fib
def fib_recursive(n):
    if n <= 1:
        return n
    else:
        return fib_recursive(n-1) + fib_recursive(n-2)


# Рекурсивная версия функции factorial с использованием lru_cache
```

```python
@lru_cache
def factorial_recursive_lru(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial_recursive_lru(n-1)


# Рекурсивная версия функции fib с использованием lru_cache
@lru_cache
def fib_recursive_lru(n):
    if n <= 1:
        return n
    else:
        return fib_recursive_lru(n-1) + fib_recursive_lru(n-2)


if __name__ == '__main__':
    # Оценка скорости работы итеративной и рекурсивной версий функций
    print("Factorial iterative:", timeit.timeit(lambda:
factorial_iterative(10), number=100000))
    print("Factorial recursive:", timeit.timeit(lambda:
factorial_recursive(10), number=100000))
    print("Fib iterative:", timeit.timeit(lambda: fib_iterative(10),
number=100000))
    print("Fib recursive:", timeit.timeit(lambda: fib_recursive(10),
number=100000))

    # Оценка скорости работы рекурсивных версий функций с использованием
lru_cache
    print("Factorial recursive with lru_cache:", timeit.timeit(lambda:
factorial_recursive_lru(10), number=100000))
    print("Fib recursive with lru_cache:", timeit.timeit(lambda:
fib_recursive_lru(10), number=100000))
```

```
C:\Users\User\PycharmProjects\Python_laba_12\venv\Scripts\
Factorial iterative: 0.05619320000005246
Factorial recursive: 0.08988890000000538
Fib iterative: 0.04808029999912833
Fib recursive: 1.059677200000806
Factorial recursive with lru_cache: 0.008820700000796933
Fib recursive with lru_cache: 0.009150899999440298

Process finished with exit code 0
```

Рисунок 1. Результат программы

Пояснение. При запуске этого скрипта вы увидите результаты оценки скорости работы различных версий функций. Для оценки изменения скорости работы рекурсивных версий функций при использовании декоратора lru_cache, можно сравнить результаты оценки скорости работы рекурсивных функций до и после применения lru_cache.

Результаты могут показать значительное улучшение скорости работы рекурсивных функций после применения lru_cache, ***так как кэширование результатов предыдущих вызовов позволяет избежать повторных вычислений.***

number в функции timeit.timeit() - это количество выполнений, которые нужно сделать для оценки времени выполнения. Например, если мы установим number=100000, то функция будет выполнена 100000 раз, и затем будет измерено общее время выполнения. Это помогает получить более точные измерения времени выполнения, особенно для быстрых операций.

Индивидуальное задание. Создайте функцию, подсчитывающую сумму элементов массива по следующему алгоритму: массив делится пополам, подсчитываются и складываются суммы элементов в каждой половине. Сумма элементов в половине массива подсчитывается по тому же алгоритму, то есть снова путем деления пополам. Деления происходят, пока в получившихся кусках массива не окажется по одному элементу и вычисление суммы, соответственно, не станет тривиальным.

Листинг:

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

def recursive_sum(arr):
    if len(arr) == 1:
        return arr[0]
    else:
        mid = len(arr) // 2
        left_sum = recursive_sum(arr[:mid])
        right_sum = recursive_sum(arr[mid:])
        return left_sum + right_sum


if __name__ == '__main__':
    input_arr = [int(x) for x in input("Введите элементы массива через
пробел: ").split()]
    print("Сумма элементов массива:", recursive_sum(input_arr))
```

```
C:\Users\User\PycharmProjects\Python_laba_12\venv\Scripts\python.exe C
Введите элементы массива через пробел: 3 4 6 7 8 9 11 2 3 4 5 6 7 8 9
Сумма элементов массива: 92

Process finished with exit code 0
```

Рисунок 2. Результат программы

Вывод: в ходе выполнения данной лабораторной работы были приобретены навыки пользования рекурсивными функциями при написании программ с помощью языка программирования Python версии 3.x.

Ответы на контрольные вопросы

1. Рекурсия - это процесс, при котором функция вызывает саму себя. Она используется для решения задач, которые могут быть разбиты на более простые подзадачи того же типа. Рекурсия делает код более читаемым и понятным, особенно для задач, связанных с древовидными или иерархическими структурами данных

2. База рекурсии - это условие, при котором рекурсивные вызовы функции прекращаются. Без базы рекурсии функция будет вызывать саму себя бесконечно.

3. Стек программы - это структура данных, используемая для хранения информации о вызовах функций в программе. При вызове функции информация о текущем состоянии функции (аргументы, локальные переменные и адрес возврата) помещается в стек. Когда функция завершает свою работу, информация извлекается из стека.

4. Текущее значение максимальной глубины рекурсии в Python можно получить с помощью sys.getrecursionlimit().

5. Если число рекурсивных вызовов превысит максимальную глубину рекурсии в Python, будет возбуждено исключение RecursionError.

6. Максимальную глубину рекурсии в Python можно изменить с помощью функции sys.setrecursionlimit() из модуля sys. Однако изменение этого значения может повлиять на производительность и стабильность программы, поэтому следует быть осторожным при его изменении.

7. Декоратор lru_cache используется для кэширования результатов вызовов функции, чтобы избежать повторных вычислений при одинаковых аргументах. Это может значительно улучшить производительность функций, особенно в случае рекурсивных вызовов.

8. Хвостовая рекурсия - это тип рекурсии, при котором рекурсивный вызов является последней операцией в функции. Оптимизация хвостовых вызовов заключается в том, что компилятор или интерпретатор может заменить рекурсивные вызовы на циклы, что уменьшает использование памяти и улучшает производительность.