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 9

Выполнил:
Дудкин Константин Александрович
2 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и
вычислительная техника»
направление «Программное
обеспечение средств вычислительной
техники и автоматизированных
систем»,
очная форма обучения

_____
(подпись)

Руководитель практики:
Кандидат технических наук, доцент
кафедры инфокомуникаций, доцент
Воронкин Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Тема: Рекурсия в языке Python

Цель: Приобретение навыков с рекурсивными функциями при написании программ с помощью языка программирования Python версии 3.x

Порядок выполнения работы

1. Провел исследование стандартного пакета Python timeit и lru_cache:

timeit — пакет, предназначенный для измерения времени выполнения функции. Декортатор lru_cache из модуля functools предназначен для кэширования выполнения функций и предотвращения повторного их выполнения при одних и тех же входных данных. Это позволяет намного сильнее увеличить скорость выполнения функций

Для того, чтобы изучить действие lru_cache провел исследование на основе факториальной функции и функции Фибоначчи:

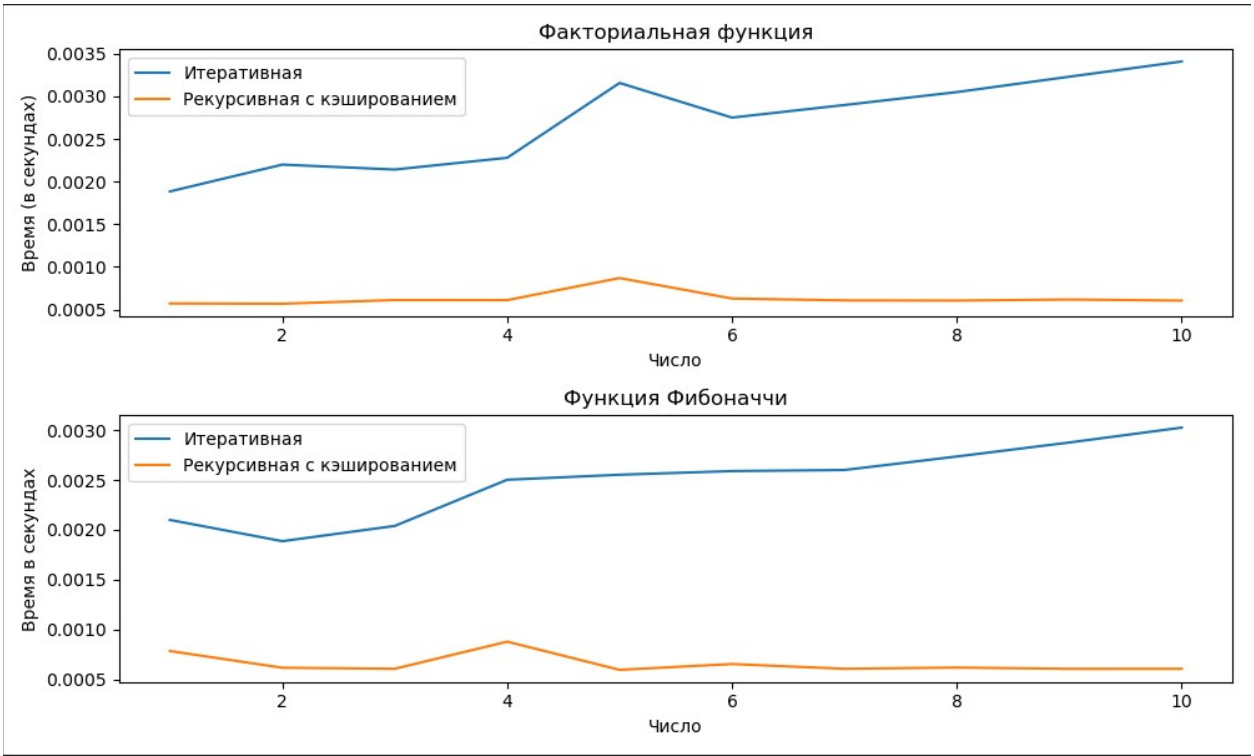


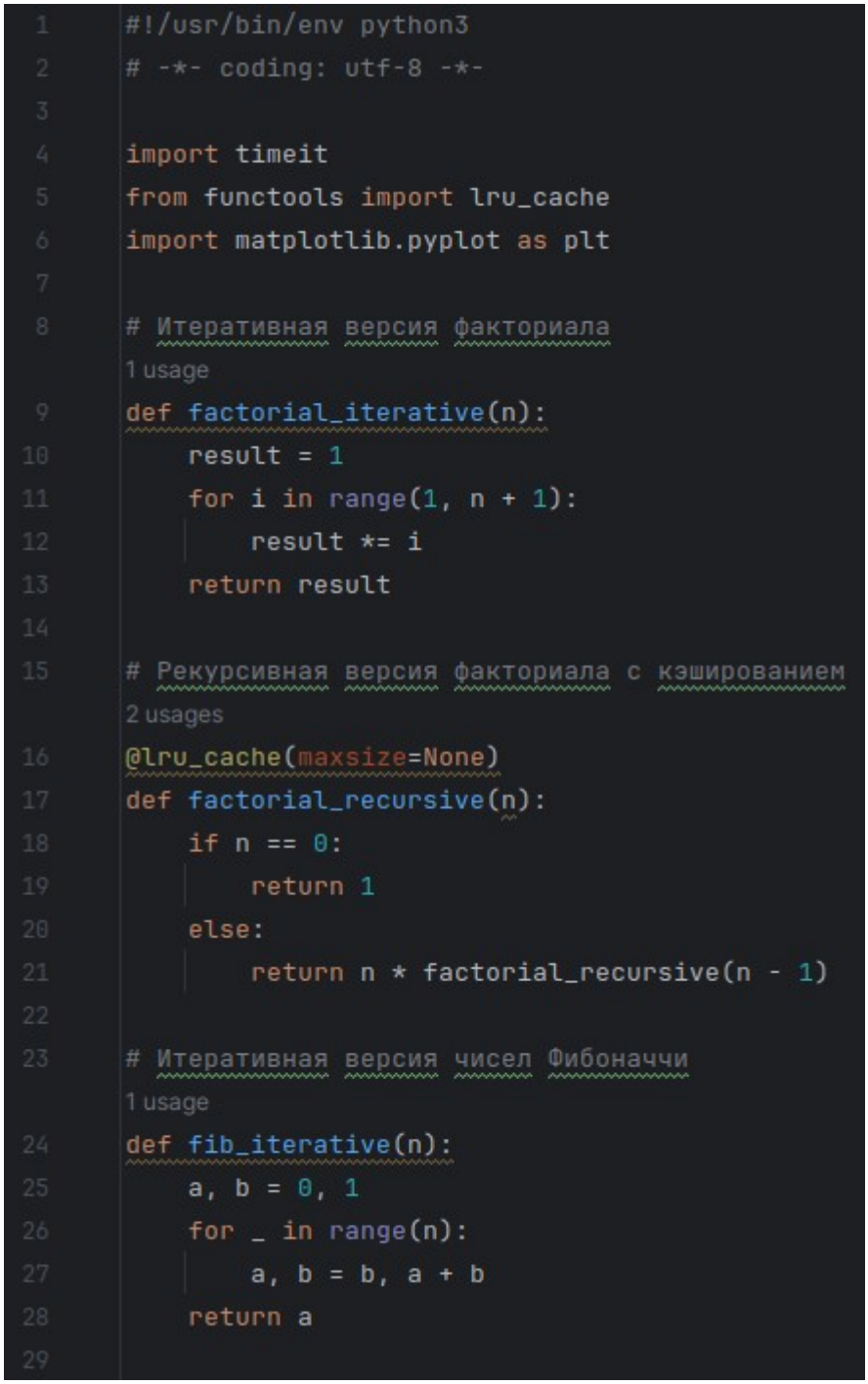
Рисунок 1. Результаты исследования. Сравнение итеративной и рекурсивной факториальной функции и функции Фибоначчи

Как видно из графика, итеративный метод (без использования lru_cache) постоянно увеличивает время своего поиска и выполнения по мере увеличения числа, в то время как в рекурсивный (с использованием

lru_cache) наоборот, почти не изменяется и остается на одном, намного менее затратном по времени уровне.

Итог: Рекурсивный метод при использовании lru_cache является наиболее эффективным

Для исследования был использован следующий код:



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

import timeit
from functools import lru_cache
import matplotlib.pyplot as plt

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

# Рекурсивная версия факториала с кэшированием
2 usages
@lru_cache(maxsize=None)
def factorial_recursive(n):
    if n == 0:
        return 1
    else:
        return n * factorial_recursive(n - 1)

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

```

Рисунок 2. Код программы исследования (часть 1)

```python
# Рекурсивная версия чисел Фибоначчи с кэшированием
3 usages
@lru_cache(maxsize=None)
def fib_recursive(n):
    if n <= 1:
        return n
    else:
        return fib_recursive(n - 1) + fib_recursive(n - 2)

# Функция для оценки скорости выполнения функций
4 usages
def evaluate_speed(func, *args):
    return timeit.timeit(lambda: func(*args), number=10000)

if __name__ == '__main__':
    # Списки для хранения данных о скорости выполнения функций
    factorial_iterative_times = []
    factorial_recursive_times = []
    fib_iterative_times = []
    fib_recursive_times = []

    # Вычисление времени выполнения функций для различных значений
    for n in range(1, 11):
        factorial_iterative_times.append(evaluate_speed(factorial_iterative, *args: n))
        factorial_recursive_times.append(evaluate_speed(factorial_recursive, *args: n))
        fib_iterative_times.append(evaluate_speed(fib_iterative, *args: n))
        fib_recursive_times.append(evaluate_speed(fib_recursive, *args: n))

    # Построение графиков
    plt.figure(figsize=(10, 6))
```

Рисунок 3. Код программы исследования (часть 2)

```python
    # График для факториала
    plt.subplot(*args: 2, 1, 1)
    plt.plot(*args: range(1, 11), factorial_iterative_times, label='Итеративная')
    plt.plot(*args: range(1, 11), factorial_recursive_times, label='Рекурсивная с кэшированием')
    plt.xlabel('Число')
    plt.ylabel('Время (в секундах)')
    plt.title('Факториальная функция')
    plt.legend()

    # График для чисел Фибоначчи
    plt.subplot(*args: 2, 1, 2)
    plt.plot(*args: range(1, 11), fib_iterative_times, label='Итеративная')
    plt.plot(*args: range(1, 11), fib_recursive_times, label='Рекурсивная с кэшированием')
    plt.xlabel('Число')
    plt.ylabel('Время в секундах')
    plt.title('Функция Фибоначчи')
    plt.legend()

    plt.tight_layout()
    plt.show()
```

Рисунок 4. Код программы исследования (часть 3)

2. Выполнил индивидуальное задание: Даны целые числа, где 0 <= m <= n, вычислить, используя рекурсию, число сочетаний C_n^m по формуле $C_n^0=C_n^n=1, C_{n-1}^m+C_{n-1}^{m-1}$ при 0 <= m <= n. Воспользовавшись формулой можно проверить правильность результата

$$C_n^m = \frac{n!}{m!(m-n)!}$$

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



# Функция для вычисления факториала
4 usages  new *
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n - 1)

# Функция для вычисления числа сочетаний C^m_n с использованием рекурсии
3 usages  new *
def combinations(m, n):
    if m == 0 or m == n:
        return 1
    else:
        return combinations(m - 1, n - 1) + combinations(m, n - 1)

# Функция для вычисления числа сочетаний C^m_n с использованием формулы
1 usage  new *
def combinations_formula(m, n):
    return factorial(n) / (factorial(m) * factorial(n - m))

# Основная часть
if __name__ == '__main__':
    m = int(input('Введите число m: '))
    n = int(input('Введите число n: '))
    if m > n:
        print('Число m не может быть больше числа n')
    else:
        print(f"Число сочетаний C^{m}_{n}, вычисленное с помощью рекурсии:", combinations(m, n))
        print(f"Число сочетаний C^{m}_{n}, вычисленное с помощью формулы:", combinations_formula(m, n))
```

Рисунок 5. Код программы задания

```
/usr/bin/python3.11 /home/code_ra1der/git/Python_LW12/Python Programs/Individual.py
Введите число m: 5
Введите число n: 10
Число сочетаний C^5_10, вычисленное с помощью рекурсии: 252
Число сочетаний C^5_10, вычисленное с помощью формулы: 252.0

Process finished with exit code 0
```

Рисунок 6. Результат работы программы

Ответы на вопросы

1. Рекурсия имеет несколько преимуществ по сравнению с итеративным методом. Она занимает меньше времени, чем если данные циклически вписывались по порядку. Она может выполняться в обратную сторону, в то время как тот же цикл for работает строго вперед. Это очевидно при реализации обращения связанного списка

2. Базой рекурсии называют случай, когда происходит возврат значения без обращения к самой функции. Если бы не было базового случая в рекурсии, она бы была бесконечной

3. Стек программы — структура данных, используемая компьютерном программой для управления вызовами функций во время их выполнения. При вызове функций текущее состояние программы, включая локальные переменные и адрес возврата, помещаются в стек. Стек программы обеспечивает управление выполнением функций в порядке их вызова и позволяет программе возвращаться к предыдущим состояниям после завершения работы

4. Чтобы получить значение максимальной глубины рекурсии, нужно воспользоваться функцией sys.getrecursionlimit()

5. В Python при попытке превысить указанный лимит рекурсий программа выдаст ошибку Maximum Recursion Depth Exceeded и прервет ее выполнение. Конечно, лимит рекурсий можно увеличить, чтобы предотвратить данную ситуацию, но это не рекомендуется делать

6. Чтобы изменить лимит рекурсий в программе, необходимо воспользоваться sys.setrecursionlimit(limit)

7. Декоратор lru_cache используется для кэширования результатов вызовов функций. LRU расшифровывается как Less Resently Used, что означает, что после вызова функции и выполнения манипуляций с данными, они сохраняются в программе, и когда потребуется что-то вычислить заново, вместо повторных вычислений будут взяты кэшированные данные, из-за чего выполнение программы сильно может ускориться

8. Хвостовая рекурсия — случай рекурсии, при котором любой рекурсивный вызов является последней операцией перед возвратом из функции. Подобный вид рекурсии примечателен тем, что может быть легко заменен на итерацию путем формальной и гарантированно корректной перестройки кода функции. Оптимизация хвостовой рекурсии путем преобразования ее в плоскую итерацию реализована во многих оптимизирующих компиляторах. В некоторых функциональных языках программирования спецификация гарантирует обязательную оптимизацию хвостовой рекурсии

Вывод. В ходе выполнения работы были приобретены навыки по работе с рекурсивными функциями при написании программ с помощью языка программирования Python версии 3.x