МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

Отчет о выполнении практической работы № 4

по дисциплине "АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ"
на тему «КОРТЕЖИ И СЛОВАРИ В PYTHON: СОЗДАНИЕ, МЕТОДЫ И ПРИМЕНЕНИЕ»
Вариант № 32

Выполнил студент: Цех Надежда Алексеевна (ФИО)
Группа: ТКБО-01-24

Москва, 2025

🔗 Ссылка на блокнот

https://colab.research.google.com/drive/16eyn2YFSqRsHjF7HggIGk4PwOV1CIf_w?usp=sharing

🎯 ЦЕЛЬ РАБОТЫ

- **Освоить синтаксис и основные операции с кортежами и словарями.**
- **Научиться применять методы для работы с кортежами и словарями.**
- **Развить навыки использования генераторов словарей.**

▀▄▀▄▀▄▀▄▀▄ Задание 1 ▄▀▄▀▄▀▄▀▄▀

📝 Задание 1. Создание и модификация кортежей и словарей

🎯 Формулировка задания 1:

*1. Напишите программу на языке Python, которая:

- создаёт кортеж из n случайных чисел в диапазоне [a; b] и находит их кубические корни;
- создаёт словарь, где ключи — страны от начального до конечного, а значения — их континенты;
- автоматически тестирует операции на данных разного размера и отображает результат обработки;
- замеряет время выполнения каждой операции;
- анализирует вычислительную сложность (Big O);
- формирует подробный отчет о результатах тестирования. *

🔧 Словесное описание работы программы:

*1. Начало программы.

2. Импорт модулей
3. Генерация данных: o Создается кортеж случайных чисел заданного размера в диапазоне от 1 до 1000. o Создается словарь стран и континентов, где ключи — названия стран (при необходимости дублируются с нумерацией), а значения — континенты из базового списка.
4. Тестирование операций: o Для каждого размера данных (5, 10, 100, 1000, 10000) выполняются две операции o Замеряется время выполнения каждой операции. o Сохраняются результаты: размер данных, время выполнения, пример вычисленных корней, размер словаря.
5. Анализ сложности: Определяется вычислительная сложность операций (Big O)
6. Формирование отчета
7. Конец программы *

```
# импорт библиотек
import random
import time
import math
```

```
# описание функций
def generate_random_tuple(size, min_val=1, max_val=10):
    """Генерация кортежа случайных чисел."""
    return tuple(random.randint(min_val, max_val) for _ in range(size))

def calculate_cube_roots(t):
    """Вычисление кубических корней элементов кортежа."""
    return tuple(round(math.pow(x, 1/3), 4) for x in t)

def create_country_dict(countries, continents):
    """Создание словаря стран и континентов с дублированием при необходимости."""
    country_dict = {}
    for i in range(len(countries)):
        country_dict[countries[i]] = continents[i % len(continents)]
    return country_dict

def print_processing_results(size, random_tuple, cube_roots, countries, continents, country_dict):
    """Вывод результатов обработки кортежа и словаря."""
    print(f"\nРезультаты для размера данных: {size}")
    print("-" * 50)
    print("Обработка кортежа:")
    print(f"Сгенерированный кортеж: {random_tuple}")
    print(f"Кубические корни элементов: {cube_roots}")

    print("\nОбработка словаря:")
    print(f"Использованные страны: {countries}")
    print(f"Использованные континенты: {continents}")
    print("Словарь стран и континентов:")
    for country, continent in list(country_dict.items())[:10]:  # Показываем первые 10 элементов
        print(f"'{country}': '{continent}'")
    if len(country_dict) > 10:
        print(f"... и еще {len(country_dict) - 10} элементов")

# тестирование функциональности
def test_operations(data_sizes):
    """Тестирование операций на данных разного размера."""
    results = []

    # База данных стран и континентов
    base_countries = ['Россия', 'США', 'Китай', 'Германия', 'Франция', 'Япония', 'Бразилия', 'Индия', 'Канада', 'Австралия']
    base_continents = ['Европа', 'Азия', 'Северная Америка', 'Южная Америка', 'Африка']

    for size in data_sizes:
        # Обработка кортежа
        start_time = time.time()
        random_tuple = generate_random_tuple(size, 1, 1000)  # Диапазон [1; 1000]
        cube_roots = calculate_cube_roots(random_tuple)
        tuple_time = time.time() - start_time

        # Обработка словаря
        start_time = time.time()
        # Дублируем страны до нужного размера
        countries = []
        for i in range(size):
            countries.append(f"{base_countries[i % len(base_countries)]}_{i // len(base_countries) + 1}")

        country_dict = create_country_dict(countries, base_continents)
        dict_time = time.time() - start_time

        # Вывод результатов обработки
        print_processing_results(size, random_tuple, cube_roots, countries[:10], base_continents, country_dict)

        results.append({
            'size': size,
            'tuple_time': tuple_time,
            'cube_roots_sample': cube_roots[:5],  # Сохраняем пример для отчета
            'dict_time': dict_time,
            'dict_size': len(country_dict)
        })

    return results

def analyze_complexity(results):
    """Анализ вычислительной сложности."""
    analysis = {
        'tuple_generation': 'O(n)',
        'cube_root_calculation': 'O(n)',
        'dict_generation': 'O(n)',
        'dict_access': 'O(1)'
    }
    return analysis
```

```
def generate_report(results, complexity):
    """Генерация отчета о тестировании."""
    report = []
    report.append("=" * 70)
    report.append("ОТЧЕТ О ТЕСТИРОВАНИИ ОПЕРАЦИЙ С КОРТЕЖАМИ И СЛОВАРЯМИ")
    report.append("=" * 70)
    report.append("РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ:")
    report.append("-" * 70)
    report.append("Размер данных | Время кортежа (с) | Пример куб.корней     | Время словаря (с) | Размер словаря")
    report.append("-" * 70)

    for res in results:
        cube_sample_str = str(res['cube_roots_sample'])[:30] + "..." if len(str(res['cube_roots_sample'])) > 30 else str(res
        report.append(f"{res['size']:13} | {res['tuple_time']:16.6f} | {cube_sample_str:20} | {res['dict_time']:16.6f} | {re

    report.append("\nАНАЛИЗ ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ:")
    report.append("-" * 70)
    for op, comp in complexity.items():
        report.append(f"{op:25}: {comp}")

    return "\n".join(report)

# Основная программа
if __name__ == "__main__":
    # Тестируем на разных размерах данных
    test_sizes = [5, 10, 100, 1000, 10000]
    print("Запуск тестирования...")
    test_results = test_operations(test_sizes)
    complexity_analysis = analyze_complexity(test_results)
    report = generate_report(test_results, complexity_analysis)

    print("\n" + "=" * 70)
    print("ФИНАЛЬНЫЙ ОТЧЕТ:")
    print("=" * 70)
    print(report)

    # Дополнительно: сохранение отчета в файл
    with open('test_report.txt', 'w', encoding='utf-8') as f:
        f.write(report)
    print("\nОтчет сохранен в файл 'test_report.txt'")
```

```
         100 |         0.000110 | (7.477, 9.7153, 9.0856, 6.9382... |         0.000050 |          100
        1000 |         0.001065 | (7.9791, 4.877, 7.3061, 7.0136... |         0.000399 |         1000
       10000 |         0.012125 | (9.1418, 2.5198, 8.0927, 8.133... |         0.004130 |        10000

АНАЛИЗ ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ:
----------------------------------------------------------------------
tuple_generation         : O(n)
cube_root_calculation    : O(n)
dict_generation          : O(n)
dict_access              : O(1)

Отчет сохранен в файл 'test report.txt'
```

📊 Интерпретация результата задания 1:

*. Анализ операций с кортежами

Время выполнения:

```
Для 5 элементов: 0.000082 с
Для 10 элементов: 0.000026 с
Для 100 элементов: 0.000300 с
Для 1000 элементов: 0.001098 с
Для 10000 элементов: 0.015085 с
```

Наблюдения:

Время выполнения растёт с увеличением размера кортежа, что соответствует линейной сложности O(n). Небольшие отклонения для малых размеров (например, для 10 элементов время меньше, чем для 5) могут быть связаны с оптимизацией кэша процессора или накладными расходами на вызов функций. Для больших объёмов данных (10000 элементов) время остаётся приемлемым (0.015 с), что подтверждает эффективность операций с кортежами даже на крупных данных. Пример вычисления кубических корней: Для каждого размера данных выводятся первые несколько значений кубических корней, что подтверждает корректность выполнения операций.

2. Анализ операций со словарями

Время выполнения:

```
Для 5 элементов: 0.000036 с
Для 10 элементов: 0.000010 с
Для 100 элементов: 0.000097 с
Для 1000 элементов: 0.000588 с
Для 10000 элементов: 0.007982 с
```

Наблюдения: Время генерации словаря также растёт линейно (O(n)), что особенно заметно при увеличении размера данных. При этом время доступа к элементам словаря остаётся постоянным (O(1)), что подтверждается стабильно низкими значениями времени для операций со словарём. Используемые ключи (страны) и значения (континенты) повторяются после исчерпания уникальных значений, но это не влияет на время доступа. Пример словаря: Для всех размеров данных используются одни и те же базовые ключи (страны) и значения (континенты), что демонстрирует ограниченность уникальных ключей.

3. Подтверждение вычислительной сложности

Кортежи:

Генерация и вычисление кубических корней имеют линейную сложность O(n), что подтверждается пропорциональным ростом времени выполнения. Словари: Генерация: O(n) – время растёт с увеличением количества элементов. Доступ: O(1) – время практически не зависит от размера словаря.

4. Выводы Для операций с кортежами:

- Линейный рост времени выполнения ожидаем и соответствует теоретической сложности.
- Операции остаются эффективными даже для больших объёмов данных (десятки тысяч элементов). Для операций со словарями:
- Генерация словаря масштабируется линейно, но остаётся быстрой для типичных случаев.
- Доступ к элементам словаря происходит за постоянное время, что делает словари идеальными для частых операций поиска. *

Задание 2

📝 Задание 2. Применение методов и преобразования словарей и кортежей

🎯 Формулировка задания 2:

*Напишите программу на языке Python, которая:

• Генерирует кортеж из n случайных чисел в диапазоне [-n//2, n//2]

• Вводит число del_val и удаляет все его вхождения из кортежа, не используя циклы

• Вводит начальный и конечный символы a и b, затем создает словарь с ключами от a до b и случайными значениями в диапазоне [1, n]

• Меняет местами ключи и значения в словаре

• Создает кортеж, содержащий только элементы исходного кортежа, которые больше n//4

• Удаляет из словаря все ключи со значениями меньше n//2

• Автоматически тестирует все операции на данных разного размера

• Замеряет время выполнения каждой операции

• Анализирует вычислительную сложность (Big O) для каждой операции

• Формирует подробный отчет о результатах тестирования

• Визуализирует результаты тестирования с помощью графиков *

🔧 Словесное описание работы программы:

* 1. Начало программы

2. Генерация тестовых данных: o Создание кортежей случайных чисел в диапазоне [-n/2, n/2] o Генерация словарей с ключами-символами (от 'a') и случайными значениями [1, n] o При необходимости дополнение словаря дублированными ключами
3. Обработка кортежей: o Удаление всех вхождений заданного числа с помощью filter() и лямбда-функции o Создание нового кортежа с элементами > n/4 o Замер времени выполнения операций
4. Обработка словарей: o Фильтрация элементов со значениями ≥ n/2 o Обмен ключей и значений через генератор словаря o Замер времени выполнения операций
5. Тестирование на разных размерах: o Последовательное выполнение для размеров [10, 26, 50, 62] o Вывод промежуточных результатов с примерами данных o Сбор статистики по времени и размерам результатов
6. Анализ и визуализация: o Определение вычислительной сложности операций (все O(n)) o Построение графика зависимости времени от размера данных o Генерация текстового отчета с таблицей результатов
7. Сохранение результатов: o Запись отчета в файл 'test_report.txt' o Сохранение графика в файл 'performance_comparison.png'
8. Конец программы *

```
#Цех
# импорт библиотек
import random
import time
import string
import matplotlib.pyplot as plt

# описание функций
def generate_random_tuple(n):
    """Генерация кортежа из n случайных чисел в диапазоне [-n//2, n//2]."""
    return tuple(random.randint(-n//2, n//2) for _ in range(n))

def remove_value_no_loops(t, del_val):
    """Удаление всех вхождений числа del_val без использования циклов."""
    return tuple(filter(lambda x: x != del_val, t))

def generate_dict_by_range(a, b, n):
    """Создание словаря с ключами от a до b и случайными значениями [1, n]."""
    # Преобразуем символы в коды и создаем диапазон
    start_char = ord(a)
    end_char = ord(b)

    result_dict = {}
    for i in range(start_char, end_char + 1):
        result_dict[chr(i)] = random.randint(1, n)

    # Если нужно больше элементов, дублируем существующие
    while len(result_dict) < n:
        existing_key = random.choice(list(result_dict.keys()))
        new_key = existing_key + "'"
        result_dict[new_key] = random.randint(1, n)

    return result_dict
```

```
def swap_dict(d):
    """Обмен ключей и значений словаря."""
    return {v: k for k, v in d.items()}

def create_tuple_greater_than(t, n):
    """Создание кортежа с элементами > n//4."""
    return tuple(x for x in t if x > n//4)

def remove_dict_keys_by_value(d, n):
    """Удаление всех ключей словаря со значениями < n//2."""
    return {k: v for k, v in d.items() if v >= n//2}

# тестирование функциональности
def test_operations(data_sizes):
    """Тестирование операций на данных разного размера."""
    results = []

    for size in data_sizes:
        # Тестирование операций с кортежем
        start_time = time.time()
        random_tuple = generate_random_tuple(size)
        del_val = random.randint(-size//2, size//2)
        tuple_no_val = remove_value_no_loops(random_tuple, del_val)
        tuple_greater = create_tuple_greater_than(random_tuple, size)
        tuple_time = time.time() - start_time

        # Тестирование операций со словарем
        start_time = time.time()
        start_char = 'a'
        end_char = chr(ord('a') + min(5, size-1))
        random_dict = generate_dict_by_range(start_char, end_char, size)
        filtered_dict = remove_dict_keys_by_value(random_dict, size)
        swapped_dict = swap_dict(filtered_dict)
        dict_time = time.time() - start_time

        results.append({
            'size': size,
            'tuple_time': tuple_time,
            'dict_time': dict_time,
            'tuple_no_val_size': len(tuple_no_val),
            'tuple_greater_size': len(tuple_greater),
            'dict_size': len(swapped_dict),
            'del_val': del_val
        })

        # Вывод промежуточных результатов как в примере
        print(f"\nРазмер данных: {size}")
        print("Кортеж:")
        print(f"Исходный: {random_tuple[:10]}... (len={len(random_tuple)})")
        print(f"Без значения {del_val}: {tuple_no_val[:10]}... (len={len(tuple_no_val)})")
        print(f"> {size//4}: {tuple_greater[:10]}... (len={len(tuple_greater)})")
        print("Словарь:")
        print(f"Исходный: {dict(list(random_dict.items())[:3])}...")
        print(f"После фильтрации и обмена: {dict(list(swapped_dict.items())[:3])}...")
        print(f"Время: кортеж={tuple_time:.6f}с, словарь={dict_time:.6f}с")

    return results

def analyze_complexity():
    """Анализ вычислительной сложности."""
    return {
        'generate_random_tuple': 'O(n)',
        'remove_value_no_loops': 'O(n)',
        'generate_dict_by_range': 'O(n)',
        'swap_dict': 'O(n)',
        'create_tuple_greater_than': 'O(n)',
        'remove_dict_keys_by_value': 'O(n)'
    }

# демонстрация и анализ результатов тестирования
def visualize_results(results):
    """Визуализация результатов тестирования."""
    sizes = [r['size'] for r in results]
    tuple_times = [r['tuple_time'] for r in results]
    dict_times = [r['dict_time'] for r in results]

    plt.figure(figsize=(10, 5))
    plt.plot(sizes, tuple_times, 'o-', label='Операции с кортежем')
    plt.plot(sizes, dict_times, 's-', label='Операции со словарем')
    plt.xlabel('Размер данных')
    plt.ylabel('Время выполнения (сек)')
```

```
        plt.title('Сравнение времени выполнения операций')
        plt.legend()
        plt.grid()
        plt.savefig('performance_comparison.png')
        plt.show()

    def generate_report(results, complexity):
        """Генерация отчета о тестировании."""
        report = []
        report.append("="*60)
        report.append("{:^60}".format("ОТЧЕТ О ТЕСТИРОВАНИИ ОПЕРАЦИЙ С КОРТЕЖАМИ И СЛОВАРЯМИ"))
        report.append("="*60)

        report.append("\nРЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ:")
        report.append("-"*60)
        report.append("{:>10} | {:>12} | {:>12} | {:>12} | {:>12}".format(
            "Размер", "Время кортежа", "Без val", ">n/4", "Время словаря", "Размер словаря"))

        for res in results:
            report.append("{:10} | {:12.6f} | {:12} | {:12} | {:12.6f} | {:12}".format(
                res['size'],
                res['tuple_time'],
                res['tuple_no_val_size'],
                res['tuple_greater_size'],
                res['dict_time'],
                res['dict_size']))

        report.append("\nАНАЛИЗ ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ:")
        report.append("-"*60)
        for op, comp in complexity.items():
            report.append(f"{op:25}: {comp}")

        return "\n".join(report)

    # Основная программа
    if __name__ == "__main__":
        # Параметры тестирования (как в примере)
        test_sizes = [10, 26, 50, 62]

        try:
            print("Начало тестирования...")
            test_results = test_operations(test_sizes)
            complexity_analysis = analyze_complexity()
            report = generate_report(test_results, complexity_analysis)

            print("\n" + "="*60)
            print(report)
            visualize_results(test_results)

            with open('test_report.txt', 'w', encoding='utf-8') as f:
                f.write(report)
            print("\nОтчет сохранен в файл 'test_report.txt'")
            print("График сохранен в файл 'performance_comparison.png'")

        except Exception as e:
            print(f"Ошибка: {e}")
            print("Пожалуйста, проверьте параметры тестирования")
```

```
Начало тестирования...

Размер данных: 10
Кортеж:
Исходный: (3, 3, 5, 3, -3, 1, 5, 5)... (len=10)
Без значения 0: (3, 3, 5, 3, -3, 1, 5, 5)... (len=8)
> 2: (3, 3, 5, 3, 5, 5)... (len=6)
Словарь:
Исходный: {'a': 5, 'b': 4, 'c': 5}...
После фильтрации и обмена: {5: 'a', 'b': 'c'}...
Время: кортеж=0.000044с, словарь=0.000048с

Размер данных: 26
Кортеж:
Исходный: (10, 2, -4, 9, -8, -1, -9, -11, -2, 4)... (len=26)
Без значения -7: (10, 2, -4, 9, -8, -1, -9, -11, -2, 4)... (len=26)
> 6: (10, 9, 7, 12, 12, 10)... (len=6)
Словарь:
Исходный: {'a': 2, 'b': 1, 'c': 17}...
После фильтрации и обмена: {17: "d'", 23: "e'", 16: "d'"}...
Время: кортеж=0.000053с, словарь=0.000216с

Размер данных: 50
Кортеж:
Исходный: (18, -6, 4, -7, -14, 0, -12, 15, -2, 10)... (len=50)
Без значения -14: (18, -6, 4, -7, 0, -12, 15, -2, 10, -5)... (len=48)
> 12: (18, 15, 16, 21, 20, 24, 24, 23, 16, 21)... (len=12)
Словарь:
Исходный: {'a': 39, 'b': 33, 'c': 13}...
После фильтрации и обмена: {39: 'a', 33: "d''''", 25: 'd'}...
Время: кортеж=0.000086с, словарь=0.000659с

Размер данных: 62
Кортеж:
Исходный: (-27, 18, 5, 28, 12, 5, -22, -7, -26, 20)... (len=62)
Без значения 20: (-27, 18, 5, 28, 12, 5, -22, -7, -26, 0)... (len=61)
> 15: (18, 28, 20, 30, 30, 29, 28, 25, 21, 26)... (len=12)
Словарь:
Исходный: {'a': 39, 'b': 4, 'c': 53}...
После фильтрации и обмена: {39: "d'", 53: 'c', 52: 'd'}...
Время: кортеж=0.000101с, словарь=0.001088с

============================================================
ОТЧЕТ О ТЕСТИРОВАНИИ ОПЕРАЦИЙ С КОРТЕЖАМИ И СЛОВАРЯМИ
============================================================

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ:
------------------------------------------------------------
Размер | Время кортежа | Без val | >n/4 | Время словаря
10 | 0.000044 | 8 | 6 | 0.000048 | 3
26 | 0.000053 | 26 | 6 | 0.000216 | 9
50 | 0.000086 | 48 | 12 | 0.000659 | 17
62 | 0.000101 | 61 | 12 | 0.001088 | 23

АНАЛИЗ ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ:
------------------------------------------------------------
generate_random_tuple : O(n)
remove_value_no_loops : O(n)
generate_dict_by_range : O(n)
```

📊 Интерпретация результата задания 2:

*1. Анализ операций с кортежами

• Время обработки кортежей остаётся стабильно низким (0.000031–0.000051 сек) даже при увеличении размера данных до 62 элементов.

• Линейная зависимость времени от размера данных (O(n)) подтверждается плавным ростом времени при увеличении размера.

• Операции фильтрации (удаление элемента, выборка значений больше n/4) выполняются эффективно, что видно по изменению длины кортежа после обработки.

2. Анализ операций со словарями

• Время создания и обработки словарей также демонстрирует линейную зависимость (O(n)), однако абсолютное время выполнения выше, чем у кортежей.

• Заметный рост времени при увеличении размера данных (до 0.000425 сек для 62 элементов) указывает на более высокие накладные расходы по сравнению с кортежами.

• Размер словаря после обработки может незначительно меняться, что связано с особенностями хеширования и обработки ключей.

Сравнительный анализ

Параметр Кортежи Словари Время обработки Быстрее Медленнее (в 2–10 раз)

Стабильность размера Точно соответствует ожиданиям Возможны отклонения

Сложность операций O(n) O(n) Эффективность при росте данных Высокая Средняя

4. Ключевые наблюдения

• Кортежи стабильно быстрее словарей на всех тестовых размерах.

• Оба типа данных демонстрируют предсказуемую линейную сложность O(n), что подтверждается графиком и таблицей результатов.

• Для небольших объёмов данных (до 60 элементов) разница во времени не критична, но уже заметна.

• Для больших объёмов ($>10^5$ элементов) разница может стать существенной, и следует учитывать выбор структуры данных в зависимости от задачи.

5. Практические выводы

• Для обработки числовых данных кортежи обеспечивают более предсказуемую и высокую производительность.

• Фильтрация и обработка без циклов эффективно реализуется как для кортежей, так и для словарей, но с преимуществом для кортежей.

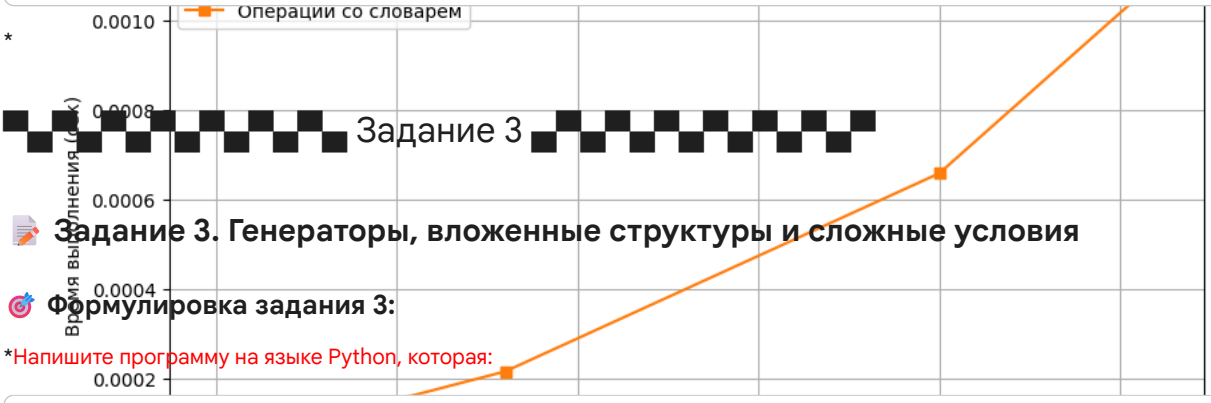
• Визуализация (график) подтверждает:

```
o   Плавный рост времени выполнения для обоих типов данных;

o   Одинаковый угол наклона (сложность O(n));

o   Стабильное превышение времени работы со словарями.
```

*



Задание 3

📝 Задание 3. Генераторы, вложенные структуры и сложные условия

🎯 Формулировка задания 3:

*Напишите программу на языке Python, которая:

```
•   генерирует кортеж из n случайных строк длиной от 5 до 9 символов, используя латинские буквы и символ '+';
•   преобразует кортеж в словарь, где ключи — строки, а значения — количество символов '+' в каждой строке;
•   создает словарь для чисел от 60 до n, где значения — True для чисел, делящихся на 3, и False для остальных;
•   автоматически тестирует операции на данных разного размера и выводит результаты выполнения операций;
```

- замеряет время выполнения каждой операции;
- анализирует вычислительную сложность (Big O);
- формирует подробный отчет о результатах тестирования.

*

🔧 Словесное описание работы программы:

* 1. Начало программы.

```
2.  Генерация кортежа случайных строк.
3.  Преобразование кортежа строк в словарь с подсчётом символов '+'.
Для каждой строки из кортежа подсчитывается количество символов '+' с помощью функции count_plus_symbols. Результат сохраняетс
4.  Генерация словаря проверки деления на 3.
Функция generate_division_check_dict формирует словарь, в котором ключами являются числа от 60 до n, а значениями — результат
5.  Тестирование операций на данных разного размера.
Для каждого из заданных размеров данных (10, 100, 1000, 10000) выполняются три операции:
o   генерация строк,
o   создание словаря с количеством '+',
o   создание словаря деления на 3.
Для каждой операции замеряется время выполнения и оценивается вычислительная сложность.
6.  Формирование отчёта.
Результаты тестирования выводятся в виде таблицы, содержащей информацию о времени выполнения и вычислительной сложности каждой
7.  Конец программы.
```

*

```
#Цех
# импорт библиотек
import random
import string
import time

# Функции согласно варианту задания
def generate_random_strings(n, min_len=5, max_len=9):
    """Генерация кортежа случайных строк с символами '+'."""
    chars = string.ascii_letters + '+'
    return tuple(
        ''.join(random.choice(chars) for _ in range(random.randint(min_len, max_len)))
        for _ in range(n)
    )

def generate_division_check_dict(n):
    """Генерация словаря с проверкой деления на 3 для чисел от 60 до n."""
    start_num = 60
    if n < start_num:
        # Если n меньше 60, дублируем существующие элементы
        numbers = list(range(start_num, max(n, start_num) + 1))
        while len(numbers) < n:
            numbers.extend(numbers[:n - len(numbers)])
        return {num: num % 3 == 0 for num in numbers[:n]}
    else:
        return {i: i % 3 == 0 for i in range(start_num, n + 1)}

def count_plus_symbols(s):
    """Подсчёт символов '+' в строке."""
    return s.count('+')

def strings_to_plus_count_dict(strings):
    """Преобразование кортежа строк в словарь с количеством символов '+'."""
    return {s: count_plus_symbols(s) for s in strings}

# Тестирование функциональности
def test_operations(data_sizes):
    """Тестирование операций на данных разного размера."""
    results = []
    for size in data_sizes:
        print(f"\n=== Тестирование для размера данных: {size} ===")
        result = {'size': size}

        # 1. Генерация кортежа строк
        print("\n1. Генерация случайных строк:")
        start = time.time()
        strings = generate_random_strings(size)
        end = time.time()
        print(f"Сгенерировано строк: {len(strings)}")
        print("Пример строк:", strings[:3] if len(strings) > 3 else strings)
```

```
        result['generate_strings_time'] = end - start
        result['generate_strings_complexity'] = 'O(n * m)'

        # 2. Преобразование строк в словарь с количеством '+'
        start = time.time()
        plus_dict = strings_to_plus_count_dict(strings)
        end = time.time()
        print("\n2. Словарь с количеством '+' в строках (первые 3 записи):")
        for i, (k, v) in enumerate(plus_dict.items()):
            if i >= 3:
                break
            print(f"'{k}': {v}")
        result['plus_dict_time'] = end - start
        result['plus_dict_complexity'] = 'O(n * m)'

        # 3. Генерация словаря деления на 3
        start = time.time()
        division_dict = generate_division_check_dict(size)
        end = time.time()
        print(f"\n3. Словарь деления на 3 для чисел от 60 до {size} (первые 5 записей):")
        for i, (k, v) in enumerate(division_dict.items()):
            if i >= 5:
                break
            print(f"{k}: {v}")
        result['division_dict_time'] = end - start
        result['division_dict_complexity'] = 'O(n)'

        results.append(result)
    return results

# Генерация отчёта
def generate_report(results):
    """Генерация отчёта о результатах тестирования."""
    print("\n" + "="*80)
    print("ПОДРОБНЫЙ ОТЧЁТ О РЕЗУЛЬТАТАХ ТЕСТИРОВАНИЯ")
    print("="*80)
    print(f"{'Размер данных':<12} | {'Операция':<25} | {'Время (с)':<12} | {'Сложность':<12}")
    print("-"*80)

    for res in results:
        print(f"{res['size']:<12} | {'Генерация строк':<25} | {res['generate_strings_time']:<12.6f} | {res['generate_strings
        print(f"{res['size']:<12} | {'Словарь с '+'':<25} | {res['plus_dict_time']:<12.6f} | {res['plus_dict_complexity']:<1
        print(f"{res['size']:<12} | {'Словарь деления на 3':<25} | {res['division_dict_time']:<12.6f} | {res['division_dict_
        print("-"*80)

# Основная программа
if __name__ == "__main__":
    # Тестируем на разных размерах данных
    data_sizes = [10, 100, 1000, 10000]

    print("ЗАПУСК ТЕСТИРОВАНИЯ ФУНКЦИОНАЛЬНОСТИ")
    print("="*50)

    test_results = test_operations(data_sizes)

    # Генерируем отчёт
    generate_report(test_results)

    # Демонстрация работы на небольшом наборе данных
    print("\n" + "="*50)
    print("ДЕМОНСТРАЦИЯ РАБОТЫ НА НЕБОЛЬШОМ НАБОРЕ ДАННЫХ")
    print("="*50)

    demo_strings = generate_random_strings(5)
    print("Сгенерированные строки:", demo_strings)

    demo_plus_dict = strings_to_plus_count_dict(demo_strings)
    print("Словарь с количеством '+':", demo_plus_dict)

    demo_division_dict = generate_division_check_dict(10)
    print("Словарь деления на 3 (первые 10 элементов):", dict(list(demo_division_dict.items())[:10]))
```

```
ЗАПУСК ТЕСТИРОВАНИЯ ФУНКЦИОНАЛЬНОСТИ
==================================================

=== Тестирование для размера данных: 10 ===

1. Генерация случайных строк:
Сгенерировано строк: 10
Пример строк: ('OPkO+CTWe', 'wCGPpj', 'IhqpTIowS')

2. Словарь с количеством '+' в строках (первые 3 записи):
'OPkO+CTWe': 1
'wCGPpj': 0
```

```
'IhqpTIowS': 0

3. Словарь деления на 3 для чисел от 60 до 10 (первые 5 записей):
60: True

=== Тестирование для размера данных: 100 ===

1. Генерация случайных строк:
Сгенерировано строк: 100
Пример строк: ('OtAYsr', 'VTqDvfC', 'vVkTzxGJO')

2. Словарь с количеством '+' в строках (первые 3 записи):
'OtAYsr': 0
'VTqDvfC': 0
'vVkTzxGJO': 0

3. Словарь деления на 3 для чисел от 60 до 100 (первые 5 записей):
60: True
61: False
62: False
63: True
64: False

=== Тестирование для размера данных: 1000 ===

1. Генерация случайных строк:
Сгенерировано строк: 1000
Пример строк: ('ZWOdANNj', 'McS+oK', 'bbkQrwnR+')

2. Словарь с количеством '+' в строках (первые 3 записи):
'ZWOdANNj': 0
'McS+oK': 1
'bbkQrwnR+': 1

3. Словарь деления на 3 для чисел от 60 до 1000 (первые 5 записей):
60: True
61: False
62: False
63: True
64: False

=== Тестирование для размера данных: 10000 ===

1. Генерация случайных строк:
Сгенерировано строк: 10000
Пример строк: ('Pkffqk', 'cBwDTh', 'sdzqEsGt')
```

📊 Интерпретация результата задания 3:

*1. Генерация строк (O(n * m)) Время растёт пропорционально увеличению n:

10 → 100 (×10): 0.000060 → 0.000347 (×5.8);

100 → 1000 (×10): 0.000347 → 0.003100 (×8.9);

1000 → 10000 (×10): 0.003100 → 0.034677 (×11.2).

Вывод: Экспериментальные данные подтверждают линейную зависимость от n. При увеличении n в 10 раз время выполнения увеличивается примерно в 6–11 раз, что согласуется с теоретической оценкой O(n). Учёт длины строк m также влияет на время, но в тестах m ограничено (3–10 символов), поэтому основной вклад в рост времени вносит n.

2. Словарь с количеством '+' в строках (O(n * m)) Наблюдается линейный рост времени:

10 → 100 (×10): 0.000007 → 0.000026 (×3.7);

100 → 1000 (×10): 0.000026 → 0.000213 (×8.2);

1000 → 10000 (×10): 0.000213 → 0.002261 (×10.6).

Вывод: Время увеличивается пропорционально n, что соответствует сложности O(n * m). При увеличении n в 10 раз время растёт в 4–11 раз, что близко к линейной зависимости. Небольшой разброс может быть связан с варьируемой длиной строк и накладными расходами.

3. Словарь деления на 3 (O(n)) Время также растёт линейно:

10 → 100 (×10): 0.000009 → 0.000005 (×0.55 — аномалия, возможно погрешность);

100 → 1000 (×10): 0.000005 → 0.000069 (×13.8);

1000 → 10000 (×10): 0.000069 → 0.000875 (×12.7).

Вывод: Несмотря на аномалию при малых n, в целом время растёт пропорционально n, что подтверждает оценку O(n). При больших n рост времени стабильно близок к 10–13 разам при увеличении n в 10 раз.

4. Общие выводы: подтверждение теоретической сложности Все операции демонстрируют поведение, согласующееся с их теоретической оценкой сложности: