МИНОБРНАУКИ РОССИИ
Федеральное государственное автономное
образовательное учреждение высшего образования
«Пермский государственный национальный
исследовательский университет»

Институт компьютерных наук и технологий

ОТЧЁТ

по индивидуальной работе №2
по дисциплине «Язык программирования Python»
Вариант 3

Работу выполнил
студент группы ИТ-3,4-2024 1 курса
Сайранов Эльдар Равилевич
«02» июня 2025 г.

Работу проверил
_____ Фамилия И.О.
«___» _____ 2025 г.

Пермь 2025

СОДЕРЖАНИЕ

Постановка задачи

Напечатать в порядке возрастания первых n натуральных чисел, в разложение которых на простые множители входят только числа 2,3,5. Идея решения: введем три очереди x2, x3, x5 в которых будем хранить элементы, которые соответственно в 2, 3, 5 раз больше напечатанных, но еще не напечатаны. Рассмотрим наименьший из ненапечатанных элементов: пусть это x. Тогда он делится нацело на одно из чисел 2, 3, 5; x находится в одной из очередей и является в ней первым элементом (меньшие его уже напечатаны, а элементы очередей не напечатаны). Напечатав x, нужно изъять его из очереди и добавить в очередь кратные ему элементы. Длины очередей не превосходят числа напечатанных элементов. Изначально в очередях хранится по одному числу.

Цель проекта — разработать программу, генерирующую последовательность чисел, все простые множители которых принадлежат множеству {2, 3, 5}. Эти числа называются «уродливыми».

Пользователь должен иметь возможность ввести количество требуемых чисел n, и программа должна вернуть первые n таких чисел в порядке возрастания.

Также необходимо реализовать юнит-тесты, проверяющие корректность генерации, отсутствие дубликатов и корректную обработку невалидного ввода.

Алгоритм решения

Было реализовано классовое решение, основанное на использовании трех очередей (queue2, queue3, queue5), в которые по мере генерации добавляются новые кандидаты на следующие “уродливые” числа.

Основная идея алгоритма

- Начать с числа 1
- Для каждого текущего числа x:
    - Добавить x*2 в очередь queue2, x*3 — в queue3, x*5 — в queue5
- Выбирать наименьшее из голов трех очередей, добавлять его в результат, и при этом удалять одинаковые минимумы (если совпадают)

Таким образом все числа формируются в правильном порядке и без повторений

Выбранные структуры и типы данных

1. **Очереди** — по условию
2. **Список** — используется для хранения результатов — уже сгенерированных «уродливых» чисел, которые возвращаются пользователю. Список сохраняет порядок добавления, удобен для индексации и вывода результата

Особенности реализации

- Для каждого вызова генерации очереди сбрасываются, чтобы избежать накопления старых данных
- Используется защита от нецелочисленного и некорректного ввода через raise ValueError

Тестирование

Для проверки корректности работы программы были написаны юнит-тесты с использованием библиотеки unittest.

Проверка первых значений

Убедиться, что get_numbers(n) возвращает правильные первые n «уродливых» чисел, например:

```
self.assertEqual(self.gen.get_numbers(10), [1, 2, 3, 4, 5, 6, 8, 9, 10, 12])
```

Проверка на дубликаты

```
nums = self.gen.get_numbers(50)
self.assertEqual(len(nums), len(set(nums)))
```

Проверка на ошибки при некорректном вводе

```
with self.assertRaises(ValueError):
    self.gen.get_numbers(-3)
```

Результат тестирования: все тесты выполнены успешно

Код программы

Код программы доступен на GitHub по ссылке.

```
class Node:
    """Узел связного списка для очереди"""
    def __init__(self, value):
        self.value = value
        self.next = None

class Queue:
    """Очередь (FIFO) на основе связного списка"""
    def __init__(self):
        self.head = None
        self.tail = None

    def is_empty(self):
        """Проверяет, пуста ли очередь"""
        return self.head is None

    def enqueue(self, value):
        """Добавляет элемент в конец очереди"""
        node = Node(value)
        if self.tail:
            self.tail.next = node
        self.tail = node
        if not self.head:
            self.head = node

    def dequeue(self):
        """Удаляет и возвращает элемент из начала очереди"""
        if self.is_empty():
            raise IndexError("Удаление из пустой очереди")
        value = self.head.value
        self.head = self.head.next
        if not self.head:
            self.tail = None
        return value

    def peek(self):
        """Возвращает начало очереди без удаления"""
        if self.is_empty():
            raise IndexError("Peek из пустой очереди")
        return self.head.value

class UglyNumbersGenerator:
    """Генератор 'уродливых' чисел (с множителями 2, 3, 5)"""
    def __init__(self):
        self.queue2 = Queue()
        self.queue3 = Queue()
        self.queue5 = Queue()
```

```
    def get_numbers(self, n):
        """Возвращает первые n чисел с множителями 2, 3, 5"""
        if not isinstance(n, int):
            raise ValueError("n должно быть целым числом")
        if n <= 0:
            raise ValueError("n должно быть положительным")

        self.queue2 = Queue()
        self.queue3 = Queue()
        self.queue5 = Queue()

        numbers = []
        current = 1
        numbers.append(current)

        # Инициализируем очереди первыми умноженными значениями
        self.queue2.enqueue(current * 2)
        self.queue3.enqueue(current * 3)
        self.queue5.enqueue(current * 5)

        # Генерируем оставшиеся n-1 чисел
        while len(numbers) < n:
            # Берем наименьшее число из трех очередей
            next_val = min(
                self.queue2.peek(),
                self.queue3.peek(),
                self.queue5.peek()
            )
            numbers.append(next_val)

                # Убираем совпадения из всех очередей и добавляем
новые элементы
            if next_val == self.queue2.peek():
                self.queue2.dequeue()
            if next_val == self.queue3.peek():
                self.queue3.dequeue()
            if next_val == self.queue5.peek():
                self.queue5.dequeue()

            # Генерируем новые кандидаты на основе next_val
            self.queue2.enqueue(next_val * 2)
            self.queue3.enqueue(next_val * 3)
            self.queue5.enqueue(next_val * 5)

        return numbers

def main():
    gen = UglyNumbersGenerator()
    try:
        n_input = input("Введите количество чисел n: ")
        n = int(n_input)
```

```
    except ValueError:
        print("Ошибка: введите целое число.")
        return

    if n <= 0:
        print("Ошибка: n должно быть положительным.")
        return

    try:
        result = gen.get_numbers(n)
            print("\nПервые", n, "чисел (с простыми множителями
только 2,3,5):")
        print(' '.join(map(str, result)))
    except Exception as e:
        print("Ошибка:", e)

if __name__ == '__main__':
    main()
```