САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №7

по курсу «Алгоритмы и структуры данных»

Тема: Тема работы

Вариант 23

Выполнил:

Юшков А.М.

К3139

Санкт-Петербург

2025г.

# Содержание отчета

[**Содержание отчета**](#_10udeakjagvs) **2**

[**Задачи по варианту**](#_gf7yxvsg0zb) **3**

[Задача №2.](#_pgzbaj56cf0o) Телефонная книга 3

[Задача №4.](#_pgzbaj56cf0o) Прошитый ассоциативный массив 4

[**Вывод**](#_fu90fuyk873) **6**

# Задачи по варианту

## Задача №2. Телефонная книга

from lab7.utils import measure,file\_write,file\_read  
import os, sys  
  
abspath = os.path.abspath(\_\_file\_\_)  
dname = os.path.dirname(abspath)  
sys.path.append(dname)  
os.chdir(dname)  
  
def optimal\_sequence(n):  
operations = [0] \* (n + 1)  
  
 for i in range(2, n + 1):  
 min\_ops = operations[i - 1] + 1  
 if i % 2 == 0:  
 min\_ops = min(min\_ops, operations[i // 2] + 1)  
 if i % 3 == 0:  
 min\_ops = min(min\_ops, operations[i // 3] + 1)  
 operations[i] = min\_ops  
  
 sequence = []  
 while n > 0:  
 sequence.append(n)  
 if n % 3 == 0 and operations[n] == operations[n // 3] + 1:  
 n //= 3  
 elif n % 2 == 0 and operations[n] == operations[n // 2] + 1:  
 n //= 2  
 else:  
 n -= 1  
  
 return operations[-1], sequence[::-1]  
  
  
def primitive\_calculator\_main():  
 n = file\_read()[0][0]   
 min\_ops, sequence = optimal\_sequence(n)  
 file\_write([min\_ops, sequence])  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 measure(primitive\_calculator\_main)

Программа эффективно решает задачу поиска оптимальной последовательности операций с использованием динамического программирования. Это позволяет минимизировать количество шагов и восстановить последовательность чисел, которая приводит к заданному числу **n**.

Алгоритм:

**Входные данные**:

Программа получает число **n**, которое нужно достичь.

**Динамическое программирование**:

Используется массив operations, где operations[i] хранит минимальное количество операций для получения числа **i**.

Для каждого числа от **2** до **n** вычисляется минимальное количество операций на основе трёх возможных действий:

* + - Увеличение на **1**.
    - Умножение на **2** (если число делится на **2**).
    - Умножение на **3** (если число делится на **3**).

**Восстановление последовательности**:

Начиная с числа **n**, программа восстанавливает последовательность чисел, выбирая на каждом шаге оптимальную операцию (умножение на **3**, умножение на **2** или вычитание **1**).

**Результат**:

Программа возвращает:

* + - Минимальное количество операций.
    - Последовательность чисел, которая приводит к числу **n**.

Результат работы кода на примере из текста задачи

Input.txt

5

Output.txt

3  
1 2 4 5

## Задача №4. Прошитый ассоциативный массив

from lab7.utils import measure, file\_write, file\_read  
  
  
def sample\_length(seq1, seq2):  
 n, m = len(seq1), len(seq2)  
 dp = [[0] \* (m + 1) for \_ in range(n + 1)]  
  
 for i in range(1, n + 1):  
 for j in range(1, m + 1):  
 if seq1[i - 1] == seq2[j - 1]:  
 dp[i][j] = dp[i - 1][j - 1] + 1  
 else:  
 dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])  
  
 return dp[n][m]  
  
  
def sample\_main(seq1=None, seq2=None):  
 if seq1 is None or seq2 is None:  
 data = file\_read()  
 n, seq1 = data[0][0], data[1]  
 m, seq2 = data[2][0], data[3]  
  
 result = sample\_length(seq1, seq2)  
 file\_write([result])  
 else:  
 result = sample\_length(seq1, seq2)  
 return result  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 measure(sample\_main)

Программа эффективно решает задачу нахождения длины наибольшей общей подпоследовательности с использованием динамического программирования. Это позволяет находить общие элементы двух последовательностей, сохраняя их порядок. Алгоритм:

### Алгоритм:

**Входные данные**:

Две последовательности: seq1 и seq2.

**Динамическое программирование**:

Используется двумерный массив dp, где dp[i][j] хранит длину наибольшей общей подпоследовательности для первых i элементов seq1 и первых j элементов seq2.

Для каждого элемента последовательностей:

* + - Если элементы seq1[i-1] и seq2[j-1] совпадают, то dp[i][j] = dp[i-1][j-1] + 1.
    - Если элементы не совпадают, то dp[i][j] = max(dp[i-1][j], dp[i][j-1]).

**Результат**:

Программа возвращает длину наибольшей общей подпоследовательности, которая находится в dp[n][m], где n и m — длины последовательностей.

Результат работы кода на примере из текста задачи

Input.txt

3  
2 7 5  
2  
2 5

Output.txt

2

# Вывод

В ходе выполнения лабораторной работы были разработаны и протестированы две программы, каждая из которых решает уникальную задачу с использованием методов динамического программирования.