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: Динамическое программирование №1

Вариант 9

Выполнила:

Коновалова Кира Романовна

К3139

Проверил:

Афанасьев А.В

Санкт-Петербург

2024 г.

# Содержание отчета

**[Задачи по варианту. Задачи по выбору 3](#_Toc26361)**

[Задача №4. Наибольшая общая подпоследовательность двух последовательностей (по выбору) 3](#_Toc25424)

[Задача №5. Наибольшая общая подпоследовательность трех последовательностей (по выбору) 5](#_Toc23850)

[Задача №6. Наибольшая возрастающая подпоследовательность 8](#_Toc18987)

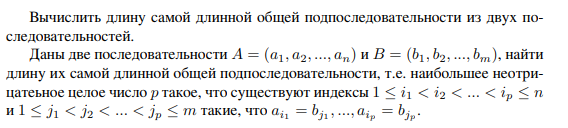
[Задача №7. Шаблоны 12](#_Toc15849)

**[Вывод 16](#_Toc4895)**

# Задачи по варианту. Задачи по выбору

## Задача №4. Наибольшая общая подпоследовательность двух последовательностей (по выбору)

Текст задачи:



Листинг кода:

def main(input\_data, output\_file):  
 # Чтение входных данных  
 lines = input\_data.strip().split("\n")  
 n = int(lines[0])  
 A = list(map(int, lines[1].split()))  
 m = int(lines[2])  
 B = list(map(int, lines[3].split()))  
 # Динамическое программирование для нахождения LCS  
 dp = [[0] \* (m + 1) for \_ in range(n + 1)]  
 for i in range(1, n + 1):  
 for j in range(1, m + 1):  
 if A[i - 1] == B[j - 1]:  
 dp[i][j] = dp[i - 1][j - 1] + 1  
 else:  
 dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])  
  
 # Результат в правом нижнем углу  
 output\_file.write(f"{dp[n][m]}\n")  
if \_\_name\_\_ == '\_\_main\_\_':  
 import sys  
 input\_path = sys.argv[1] if len(sys.argv) > 1 else "../txtf/input.txt"  
 output\_path = sys.argv[2] if len(sys.argv) > 2 else "../txtf/output.txt"  
 try:  
 with open(input\_path, "r") as input\_file:  
 input\_data = input\_file.read()  
 with open(output\_path, "w") as output\_file:  
 main(input\_data, output\_file)  
 print(f"Результат успешно записан в файл '{output\_path}'")  
 except Exception as e:  
 print(f"Ошибка при обработке: {e}")

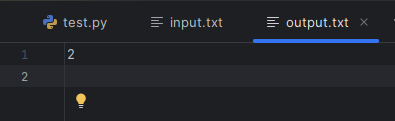
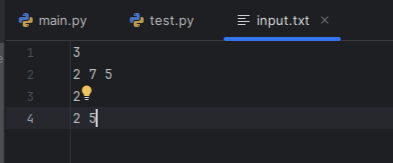
Текстовое объяснение задачи:

Задача заключается в нахождении длины самой длинной общей подпоследовательности (LCS) двух последовательностей. Требуется найти наибольшее неотрицательное целое число p, такое что существует последовательность индексов в обеих последовательностях, где элементы на этих позициях совпадают.Мною сспользуется метод динамического программирования для вычисления LCS. Мы строим таблицу dp, где dp[i][j] это длина наибольшей общей подпоследовательности для первых i элементов первой последовательности и первых j элементов второй. Если элементы на текущих позициях совпадают, то увеличиваем значение на 1. В противном случае берем максимум из значений в предыдущих ячейках

Тесты:

import unittest  
from io import StringIO  
from unittest.mock import patch  
from lab7.task4.src.main import main  
class TestLCS(unittest.TestCase):  
  
 def run\_main(self, input\_data):  
 with patch("sys.stdout", new\_callable=StringIO) as output\_file:  
 main(input\_data, output\_file)  
 return output\_file.getvalue().strip()  
  
 def test\_lcs(self):  
 input\_data = """5  
 1 2 3 4 5  
 4  
 2 3 4 5"""  
 expected\_output = "4"  
 result = self.run\_main(input\_data)  
 self.assertEqual(result, expected\_output)  
  
 def test\_lcs\_single\_element(self):  
 input\_data = """1  
 1  
 1  
 1"""  
 expected\_output = "1"  
 result = self.run\_main(input\_data)  
 self.assertEqual(result, expected\_output)  
  
 def test\_lcs\_no\_common\_elements(self):  
 input\_data = """3  
 1 2 3  
 3  
 4 5 6"""  
 expected\_output = "0"  
 result = self.run\_main(input\_data)  
 self.assertEqual(result, expected\_output)  
  
 def test\_lcs\_edge\_case(self):  
 input\_data = """5  
 1 2 3 4 5  
 5  
 1 2 3 4 5"""  
 expected\_output = "5"  
 result = self.run\_main(input\_data)  
 self.assertEqual(result, expected\_output)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

Результат работы кода на примерах из текста задачи:

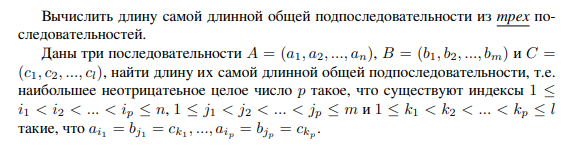


Вывод по задаче:

Решение задачи с помощью динамического программирования позволяет эффективно вычислять длину LCS двух последовательностей с временной сложностью O(n\*m), где n и m это длины последовательностей. Решение корректно обрабатывает все возможные варианты входных данных

## Задача №5. Наибольшая общая подпоследовательность трех последовательностей (по выбору)

Текст задачи:



Листинг кода:

def main(input\_data, output\_file):  
 # Чтение входных данных  
 lines = input\_data.strip().split("\n")  
 n = int(lines[0])  
 A = list(map(int, lines[1].split()))  
 m = int(lines[2])  
 B = list(map(int, lines[3].split()))  
 l = int(lines[4])  
 C = list(map(int, lines[5].split()))  
 # Динамическое программирование для нахождения LCS  
 dp = [[[0] \* (l + 1) for \_ in range(m + 1)] for \_ in range(n + 1)]  
 for i in range(1, n + 1):  
 for j in range(1, m + 1):  
 for k in range(1, l + 1):  
 if A[i - 1] == B[j - 1] == C[k - 1]:  
 dp[i][j][k] = dp[i - 1][j - 1][k - 1] + 1  
 else:  
 dp[i][j][k] = max(dp[i - 1][j][k], dp[i][j - 1][k], dp[i][j][k - 1])  
 # Результат в правом верхнем углу  
 output\_file.write(f"{dp[n][m][l]}\n")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 import sys  
 input\_path = sys.argv[1] if len(sys.argv) > 1 else "../txtf/input.txt"  
 output\_path = sys.argv[2] if len(sys.argv) > 2 else "../txtf/output.txt"  
 try:  
 with open(input\_path, "r") as input\_file:  
 input\_data = input\_file.read()  
 with open(output\_path, "w") as output\_file:  
 main(input\_data, output\_file)  
 print(f"Результат успешно записан в файл '{output\_path}'")  
 except Exception as e:  
 print(f"Ошибка при обработке: {e}")

Текстовое объяснение задачи:

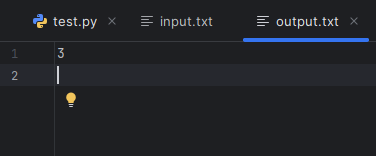
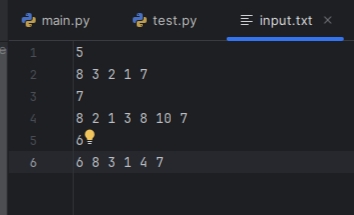
Задача усложняется на добавление третьей последовательности, и теперь требуется найти длину самой длинной общей подпоследовательности для трех последовательностей. Задача аналогична предыдущей, но для трех строк

Здесь используется трехмерный массив dp, где dp[i][j][k] это длина LCS для первых i элементов первой последовательности, первых j элементов второй и первых k элементов третьей. Для вычисления значений в таблице применяется аналогичный принцип, как в задаче для двух последовательностей

Тесты:

import unittest  
from io import StringIO  
from unittest.mock import patch  
from lab7.task5.src.main import main  
class TestLCS3(unittest.TestCase):  
  
 def run\_main(self, input\_data):  
 with patch("sys.stdout", new\_callable=StringIO) as output\_file:  
 main(input\_data, output\_file)  
 return output\_file.getvalue().strip()  
  
 def test\_lcs3(self):  
 input\_data = """5  
1 2 3 4 5  
4  
2 3 4 5  
3  
3 4 5"""  
 expected\_output = "3"  
 result = self.run\_main(input\_data)  
 self.assertEqual(result, expected\_output)  
  
 def test\_lcs3\_single\_element(self):  
 input\_data = """1  
1  
1  
1  
1  
1"""  
 expected\_output = "1"  
 result = self.run\_main(input\_data)  
 self.assertEqual(result, expected\_output)  
  
 def test\_lcs3\_no\_common\_elements(self):  
 input\_data = """3  
1 2 3  
3  
4 5 6  
3  
7 8 9"""  
 expected\_output = "0"  
 result = self.run\_main(input\_data)  
 self.assertEqual(result, expected\_output)  
  
 def test\_lcs3\_edge\_case(self):  
 input\_data = """5  
1 2 3 4 5  
5  
1 2 3 4 5  
5  
1 2 3 4 5"""  
 expected\_output = "5"  
 result = self.run\_main(input\_data)  
 self.assertEqual(result, expected\_output)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

Результат работы кода на примерах из текста задачи:

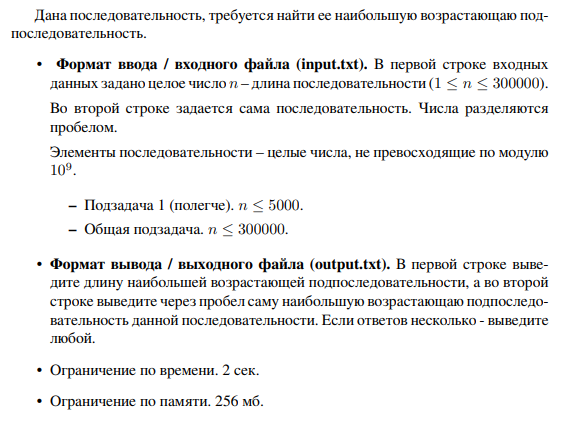


Вывод по задаче:

Динамическое программирование в данной задаче также эффективно решает задачу с временной сложностью O(n\*m\*l), где n, m, и l это длины трех последовательностей. Этот подход позволяет находить наибольшую общую подпоследовательность для трех строк

## Задача №6. Наибольшая возрастающая подпоследовательность

Текст задачи:



Листинг кода:

import sys  
import os  
import bisect  
  
  
def lis(sequence):  
 *"""  
 Функция для нахождения наибольшей возрастающей подпоследовательности (LIS).  
 :param sequence: Список целых чисел (последовательность).  
 :return: Кортеж из длины LIS и самой LIS.  
 """* n = len(sequence)  
 if n == 0:  
 return 0, []  
  
 # Массив для хранения минимальных концов возрастающих подпоследовательностей  
 # длины i+1 в arr[i]  
 lis\_ends = []  
 # Массив для восстановления самой последовательности  
 prev\_idx = [-1] \* n  
 indices = [-1] \* n  
  
 for i in range(n):  
 # Используем бинарный поиск для нахождения места вставки  
 pos = bisect.bisect\_left(lis\_ends, sequence[i])  
  
 # Если pos == len(lis\_ends), то добавляем новый элемент  
 if pos == len(lis\_ends):  
 lis\_ends.append(sequence[i])  
 else:  
 lis\_ends[pos] = sequence[i]  
  
 # Сохраняем индекс для восстановления последовательности  
 indices[pos] = i  
 if pos > 0:  
 prev\_idx[i] = indices[pos - 1]  
  
 # Восстанавливаем саму последовательность  
 lis\_seq = []  
 idx = indices[len(lis\_ends) - 1]  
 while idx != -1:  
 lis\_seq.append(sequence[idx])  
 idx = prev\_idx[idx]  
  
 # Лист нужно перевернуть, так как мы восстанавливаем его с конца  
 lis\_seq.reverse()  
  
 return len(lis\_ends), lis\_seq  
  
  
def main(input\_data, output\_file):  
 *"""  
 Основная функция для запуска задачи.  
 :param input\_data: Строка с содержимым входного файла.  
 :param output\_file: Объект файла для записи вывода.  
 """* lines = input\_data.strip().split("\n")  
 n = int(lines[0])  
 sequence = list(map(int, lines[1].split()))  
  
 # Получаем результат  
 length, subsequence = lis(sequence)  
  
 # Записываем результат в выходной файл  
 output\_file.write(f"{length}\n")  
 output\_file.write(" ".join(map(str, subsequence)) + "\n")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Указываем пути для входного и выходного файлов  
 input\_path = sys.argv[1] if len(sys.argv) > 1 else "../txtf/input.txt"  
 output\_path = sys.argv[2] if len(sys.argv) > 2 else "../txtf/output.txt"  
  
 # Проверка, существуют ли файлы в указанной директории  
 if not os.path.exists(input\_path):  
 print(f"Ошибка: файл '{input\_path}' не найден!")  
 sys.exit(1)  
  
 try:  
 # Чтение входных данных  
 with open(input\_path, "r") as input\_file:  
 input\_data = input\_file.read()  
  
 # Запись результата  
 with open(output\_path, "w") as output\_file:  
 main(input\_data, output\_file)  
  
 print(f"Результат успешно записан в файл '{output\_path}'")  
 except Exception as e:  
 print(f"Ошибка при обработке: {e}")

Текстовое объяснение задачи:

Необходимо найти наибольшую возрастающую подпоследовательность в заданной последовательности чисел. Под возрастающей подпоследовательностью понимается такая подпоследовательность, в которой каждый следующий элемент больше предыдущего

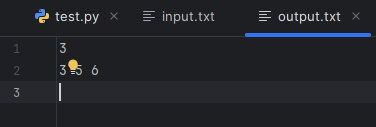
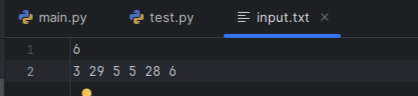
Мною используется метод динамического программирования с использованием бинарного поиска. Я строю массив lis\_endslis, который хранит минимальные возможные значения концов возрастающих подпоследовательностей для каждой длины. Восстановление самой последовательности выполняется через массив индексов

Тесты:

import unittest  
from io import StringIO  
from lab7.task6.src.main import main

class TestLIS(unittest.TestCase):  
 def setUp(self):  
 # Метод setUp() будет запускаться перед каждым тестом  
 self.input\_data = ""  
 self.output = StringIO()  
 def run\_main(self, input\_data):  
 *"""Запуск основного алгоритма и возврат результата"""* self.input\_data = input\_data  
 output\_file = StringIO()  
 main(self.input\_data, output\_file)  
 return output\_file.getvalue().strip()  
 def test\_basic(self):  
 input\_data = """6  
1 3 2 3 4 5"""  
 expected\_output = """5  
1 2 3 4 5"""  
 result = self.run\_main(input\_data)  
 self.assertEqual(result, expected\_output)  
  
 def test\_edge\_case\_single\_element(self):  
 input\_data = """1  
5"""  
 expected\_output = """1  
5"""  
 result = self.run\_main(input\_data)  
 self.assertEqual(result, expected\_output)  
 def test\_edge\_case\_all\_equal(self):  
 input\_data = """5  
5 5 5 5 5"""  
 expected\_output = """1  
5"""  
 result = self.run\_main(input\_data)  
 self.assertEqual(result, expected\_output)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

Результат работы кода на примерах из текста задачи:

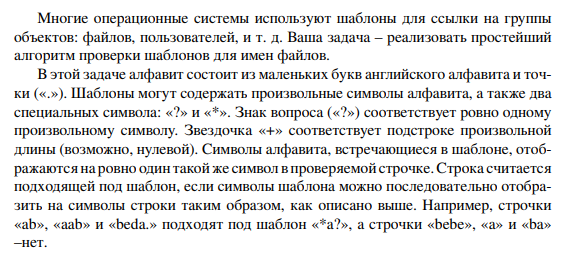


Вывод по задаче:

Алгоритм работает с временной сложностью O(nlog n), что значительно улучшает эффективность по сравнению с наивным методом. Это решение позволяет найти наибольшую возрастающую подпоследовательность в последовательности чисел за приемлемое время

## Задача №7. Шаблоны

Текст задачи:



Листинг кода:

def is\_match(pattern, s):  
 m, n = len(pattern), len(s)  
  
 # dp[i][j] будет True, если первые i символов pattern могут соответствовать первым j символам s  
 dp = [[False] \* (n + 1) for \_ in range(m + 1)]  
  
 # Пустая строка соответствует пустому шаблону  
 dp[0][0] = True  
  
 # Инициализация для шаблона, начинающегося с '\*'  
 for i in range(1, m + 1):  
 if pattern[i - 1] == '\*':  
 dp[i][0] = dp[i - 1][0]  
  
 # Заполнение таблицы dp  
 for i in range(1, m + 1):  
 for j in range(1, n + 1):  
 if pattern[i - 1] == s[j - 1] or pattern[i - 1] == '?':  
 dp[i][j] = dp[i - 1][j - 1]  
 elif pattern[i - 1] == '\*':  
 dp[i][j] = dp[i - 1][j] or dp[i][j - 1]  
 return dp[m][n]  
  
  
def main(input\_data, output\_file):  
 lines = input\_data.strip().split("\n")  
 pattern = lines[0]  
 s = lines[1] if len(lines) > 1 else ''  
  
 if is\_match(pattern, s):  
 output\_file.write("YES\n")  
 else:  
 output\_file.write("NO\n")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 import sys  
 import os  
  
 input\_path = sys.argv[1] if len(sys.argv) > 1 else "../txtf/input.txt"  
 output\_path = sys.argv[2] if len(sys.argv) > 2 else "../txtf/output.txt"  
  
 if not os.path.exists(input\_path):  
 print(f"Ошибка: файл '{input\_path}' не найден!")  
 sys.exit(1)  
  
 try:  
 with open(input\_path, "r") as input\_file:  
 input\_data = input\_file.read()  
  
 with open(output\_path, "w") as output\_file:  
 main(input\_data, output\_file)  
  
 print(f"Результат успешно записан в файл '{output\_path}'")  
 except Exception as e:  
 print(f"Ошибка при обработке: {e}")

Текстовое объяснение задачи:

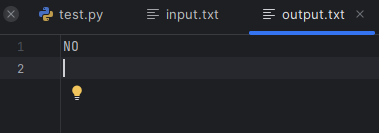
Задача заключается в проверке, подходит ли строка под заданный шаблон. Шаблон может содержать обычные символы, а также символы "?" (один произвольный символ) и "\*" (произвольная длина подстроки)

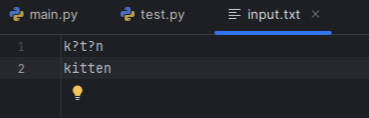
Для решения задачи используется динамическое программирование. Создается таблица dp, где dp[i][j] показывает, можно ли сопоставить первые i символов шаблона с первыми j символами строки. Таблица заполняется с учетом возможных совпадений символов и работы с символами "?" и "\*"

Тесты:

import unittest  
from io import StringIO  
  
from lab7.task7.src.main import main  
  
class TestPatternMatching(unittest.TestCase):  
  
 def setUp(self):  
 self.input\_data = ""  
 self.output = StringIO()  
  
 def run\_main(self, input\_data):  
 *"""Запуск основного алгоритма и возврат результата"""* self.input\_data = input\_data  
 output\_file = StringIO()  
 main(self.input\_data, output\_file)  
 return output\_file.getvalue().strip()  
  
 def test\_exact\_match(self):  
 input\_data = """abc  
abc"""  
 expected\_output = "YES"  
 result = self.run\_main(input\_data)  
 self.assertEqual(result, expected\_output)  
  
 def test\_wildcard\_question\_mark(self):  
 input\_data = """a?c  
abc"""  
 expected\_output = "YES"  
 result = self.run\_main(input\_data)  
 self.assertEqual(result, expected\_output)  
  
 def test\_wildcard\_star(self):  
 input\_data = """a\*c  
abc"""  
 expected\_output = "YES"  
 result = self.run\_main(input\_data)  
 self.assertEqual(result, expected\_output)  
  
 def test\_star\_matches\_empty\_string(self):  
 input\_data = """\*  
"""  
 expected\_output = "YES"  
 result = self.run\_main(input\_data)  
 self.assertEqual(result, expected\_output)  
  
 def test\_no\_match(self):  
 input\_data = """a?c  
ab"""  
 expected\_output = "NO"  
 result = self.run\_main(input\_data)  
 self.assertEqual(result, expected\_output)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

Результат работы кода на примерах из текста задачи:

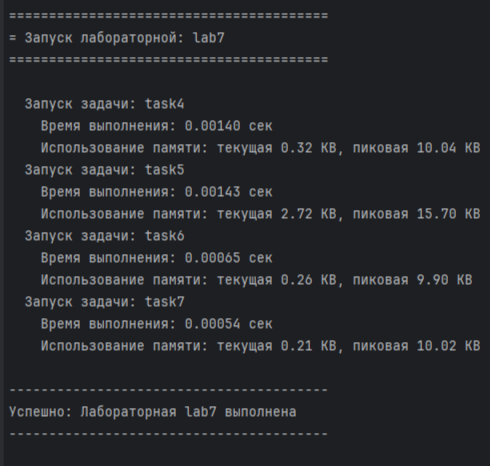




Вывод по задаче:

Решение задачи с помощью динамического программирования эффективно обрабатывает шаблоны с символами "?" и "\*", имея временную сложность O(m\*n), где m это длина шаблона, а n длина строки

**Запуск всех задач**



# Вывод

В ходе выполнения лабораторной работы были реализованы решения задач с использованием метода динамического программирования. Каждая из задач требовала применения различных подходов и оптимизаций для работы с подстроками и подпоследовательностями. Алгоритмы, основанные на динамическом программировании, продемонстрировали свою эффективность и позволяют решать задачи с оптимальной сложностью, что важно для работы с большими объемами данных. В целом, лабораторная работа показала важность и мощь динамического программирования для решения задач оптимизации и поиска наибольших общих подпоследовательностей