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

Вариант 11

Выполнила:

Жмачинская Д.С.

К3141

Проверил: Афанасьев А.В.

Санкт-Петербург 2024 г.

# **Задачи по варианту:**

## Задача 5:

# Текст задачи.

# 

# Листинг кода.

*"""Module for calculating the length of the Longest Common Subsequence (LCS) for three sequences."""*from typing import List  
  
  
class LCS3Finder:  
 *"""  
 A class to find the length of the Longest Common Subsequence (LCS) among three sequences.  
 """* @staticmethod  
 def longest\_common\_subsequence\_3(  
 sequence\_a: List[int], sequence\_b: List[int], sequence\_c: List[int]  
 ) -> int:  
 *"""  
 Calculates the length of the LCS for three sequences.  
  
 :param sequence\_a: The first sequence (list of integers).  
 :param sequence\_b: The second sequence (list of integers).  
 :param sequence\_c: The third sequence (list of integers).  
 :return: The length of the LCS for the three sequences.  
 """* len\_a: int = len(sequence\_a)  
 len\_b: int = len(sequence\_b)  
 len\_c: int = len(sequence\_c)  
 dp: List[List[List[int]]] = [  
 [[0] \* (len\_c + 1) for \_ in range(len\_b + 1)] for \_\_ in range(len\_a + 1)  
 ]  
  
 for i in range(1, len\_a + 1):  
 for j in range(1, len\_b + 1):  
 for k in range(1, len\_c + 1):  
 if (  
 sequence\_a[i - 1] == sequence\_b[j - 1]  
 and sequence\_a[i - 1] == sequence\_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],  
 )  
  
 return dp[len\_a][len\_b][len\_c]

**Текстовое объяснение решения.**

**Для нахождения LCS для трёх последовательностей используется метод динамического программирования с трёхмерной таблицей** dp**. Каждый элемент** dp[i][j][k] **хранит длину LCS для первых** i**,** j **и** k **элементов трёх последовательностей. Если все три символа совпадают, то значение в таблице обновляется как** dp[i-1][j-1][k-1] + 1**. В противном случае выбирается максимальное значение из предыдущих возможных вариантов.**

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.004003 сек | 0.30МБ |
| Пример из задачи | 0.05832 сек | 0.52 МБ |
| Пример из задачи | 0.09345 сек | 1.45 МБ |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.42455 сек | 1.99 МБ |

**Вывод по задаче.**

# Этот алгоритм показывает, как можно применить технику динамического программирования для решения более сложных задач, когда требуется работать с несколькими последовательностями.

## Задача 6:

# Текст задачи.

# 

# Листинг кода.

*"""Module for finding the Longest Increasing Subsequence (LIS) in a sequence."""*from bisect import bisect\_left  
from typing import List, Tuple  
  
  
class LISFinder:  
 *"""  
 A class to find the length and the actual Longest Increasing Subsequence (LIS) in a sequence.  
 """* @staticmethod  
 def find\_lis(sequence: List[int]) -> Tuple[int, List[int]]:  
 *"""  
 Finds the length of the LIS and the subsequence itself.  
  
 :param sequence: A list of integers representing the sequence.  
 :return: A tuple containing the length of the LIS and the LIS as a list of integers.  
 """* sequence\_length: int = len(sequence)  
 lis: List[int] = []  
 predecessors: List[int] = [-1] \* sequence\_length  
 lis\_indices: List[int] = []  
  
 for current\_index in range(sequence\_length):  
 current\_value: int = sequence[current\_index]  
 insertion\_position: int = bisect\_left(lis, current\_value)  
  
 if insertion\_position == len(lis):  
 lis.append(current\_value)  
 lis\_indices.append(current\_index)  
 else:  
 lis[insertion\_position] = current\_value  
 lis\_indices[insertion\_position] = current\_index  
  
 if insertion\_position > 0:  
 predecessors[current\_index] = lis\_indices[insertion\_position - 1]  
  
 lis\_length: int = len(lis)  
 lis\_sequence: List[int] = []  
 current\_position: int = lis\_indices[-1] if lis\_indices else -1  
  
 for \_ in range(lis\_length):  
 lis\_sequence.append(sequence[current\_position])  
 current\_position = predecessors[current\_position]  
  
 lis\_sequence.reverse()  
 return lis\_length, lis\_sequence

**Текстовое объяснение решения.**

**Для нахождения наибольшей возрастающей подпоследовательности (LIS) используется алгоритм с двоичным поиском. В массиве** lis **хранятся текущие элементы возрастающей подпоследовательности. Для каждого нового элемента с помощью функции** bisect\_left **определяется его позиция в массиве** lis**. Если элемент больше всех в** lis**, он добавляется в конец, в противном случае заменяет существующий элемент. Этот подход позволяет достичь времени работы O(nlog⁡n).**

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.01134 сек | 1.67 МБ |
| Пример из задачи | 0.02689 сек | 1.77 МБ |
| Пример из задачи | 0.07581 сек | 1.87 МБ |
| Верхняя граница диапазона значений входных данных из текста задачи | 1.02345 сек | 1.90 МБ |

**Вывод по задаче.**

**Задача на нахождение LIS является важной для анализа последовательностей, где нужно выявить упорядоченные данные. Данный алгоритм эффективно решает задачу с использованием бинарного поиска и позволяет быстро анализировать большие наборы данных.**

# Дополнительные задачи:

# **Задача 4:**

# Текст задачи.

# 

# Листинг кода.

*"""Module for calculating the length of the Longest Common Subsequence (LCS) of two sequences."""*from typing import List  
  
  
class LCSFinder:  
 *"""  
 A class to find the length of the Longest Common Subsequence (LCS) between two sequences.  
 """* @staticmethod  
 def longest\_common\_subsequence(sequence\_a: List[int], sequence\_b: List[int]) -> int:  
 *"""  
 Calculates the length of the LCS for two sequences.  
  
 :param sequence\_a: The first sequence (list of integers).  
 :param sequence\_b: The second sequence (list of integers).  
 :return: The length of the LCS.  
 """* len\_a: int = len(sequence\_a)  
 len\_b: int = len(sequence\_b)  
 dp: List[List[int]] = [[0] \* (len\_b + 1) for \_ in range(len\_a + 1)]  
  
 for i in range(1, len\_a + 1):  
 for j in range(1, len\_b + 1):  
 if sequence\_a[i - 1] == sequence\_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])  
  
 return dp[len\_a][len\_b]

**Текстовое объяснение решения.**

**Для вычисления длины наибольшей общей подпоследовательности (LCS) используется подход динамического программирования. Создаётся двумерная таблица** dp**, в которой элемент** dp[i][j] **хранит длину LCS для первых** i **элементов первой последовательности и первых** j **элементов второй. Если текущие элементы совпадают, то значение в таблице обновляется как** dp[i-1][j-1] + 1**. Если элементы не совпадают, то выбирается максимальное значение между** dp[i-1][j] **и** dp[i][j-1]**. Результат находится в ячейке** dp[n][m]**, где** n **и** m **— длины последовательностей.**

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.003465 сек | 0.35 МБ |
| Пример из задачи | 0.03958 сек | 0.56 МБ |
| Пример из задачи | 0.10345 сек | 1.59 МБ |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.4124 сек | 1.89 МБ |

**Вывод по задаче.**

# Данный алгоритм эффективно решает задачу нахождения общих элементов в двух последовательностях. Он находит применение в таких областях, как анализ строк, биоинформатика и обработка данных, где важно выявить сходства между двумя наборами информации.

## Задача 7:

# Текст задачи.

# 

# Листинг кода.

*"""Module for checking if a string matches a pattern containing '?' and '\*'."""*from typing import List  
  
  
class PatternMatcher:  
 *"""  
 A class to check if a string matches a given pattern.  
 The pattern may contain:  
 - Letters  
 - '?', which matches any single character  
 - '\*', which matches zero or more characters  
 """* @staticmethod  
 def matches\_pattern(pattern: str, string: str) -> str:  
 *"""  
 Checks if the given string matches the pattern.  
  
 :param pattern: The pattern string containing letters, '?', and '\*'.  
 :param string: The string to be matched against the pattern.  
 :return: "YES" if the string matches the pattern, otherwise "NO".  
 """* pattern\_length: int = len(pattern)  
 string\_length: int = len(string)  
 dp: List[List[bool]] = [[False] \* (string\_length + 1) for \_ in range(pattern\_length + 1)]  
 dp[0][0] = True  
  
 # Initialize dp for patterns starting with '\*'  
 for pattern\_idx in range(1, pattern\_length + 1):  
 if pattern[pattern\_idx - 1] == '\*':  
 dp[pattern\_idx][0] = dp[pattern\_idx - 1][0]  
 else:  
 break  
  
 for pattern\_idx in range(1, pattern\_length + 1):  
 for string\_idx in range(1, string\_length + 1):  
 if pattern[pattern\_idx - 1] == string[string\_idx - 1] or pattern[pattern\_idx - 1] == '?':  
 dp[pattern\_idx][string\_idx] = dp[pattern\_idx - 1][string\_idx - 1]  
 elif pattern[pattern\_idx - 1] == '\*':  
 dp[pattern\_idx][string\_idx] = dp[pattern\_idx - 1][string\_idx] or dp[pattern\_idx][string\_idx - 1]  
  
 return "YES" if dp[pattern\_length][string\_length] else "NO"

**Текстовое объяснение решения.**

Для проверки соответствия строки шаблону используется подход динамического программирования. Создаётся двумерный массив dp, где dp[i][j] определяет, возможно ли сопоставить первые i символов шаблона с первыми j символами строки. Символ ? сопоставляется с любым символом, а \* может заменять как пустую строку, так и любую последовательность символов. Проверка проводится по правилам шаблона, используя предшествующие значения из dp.

|  |  |  |
| --- | --- | --- |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.004577 сек | 0.25 МБ |
| Пример из задачи | 0.04566 сек | 0.44 МБ |
| Пример из задачи | 0.12357 сек | 1.69 МБ |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.4675 сек | 2.00 МБ |

**Вывод по задаче.**

**Алгоритм предоставляет мощный инструмент для работы с текстовыми шаблонами. Его применение охватывает задачи фильтрации, обработки пользовательских запросов и анализа текстовых данных, где используются паттерны с** \* **и** ?**.**

# Вывод

В данной лабораторной работе были решены задачи, связанные с динамическим программированием, поиском подпоследовательностей и шаблонным сопоставлением. Все задания демонстрируют важность структурирования данных и алгоритмов, оптимизирующих процесс вычислений.