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 14

Выполнил: Мурашов Н.А.

К3141

Проверил: Афанасьев А.В.

Санкт-Петербург 2024 г.

# Задачи по варианту:

# Задача 4:

# Текст задачи.

# 

# Листинг кода.

*"""Модуль для вычисления длины наибольшей общей подпоследовательности (LCS) двух последовательностей."""*class LCSFinder:  
 *"""  
 Класс для нахождения длины наибольшей общей подпоследовательности (LCS).  
 """* @staticmethod  
 def longest\_common\_subsequence(A, B):  
 *"""  
 Вычисляет длину LCS для последовательностей A и B.  
  
 :param A: Первая последовательность (список чисел).  
 :param B: Вторая последовательность (список чисел).  
 :return: Длина LCS.  
 """* n, m = len(A), len(B)  
 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])  
  
 return dp[n][m]

**Текстовое объяснение решения.**

**Для нахождения длины наибольшей общей подпоследовательности (LCS) используется метод динамического программирования. Создаётся двумерный массив** dp**, где** dp[i][j] **хранит длину LCS первых** i **элементов первой последовательности и первых** j **элементов второй. Если символы совпадают, текущая ячейка обновляется как** dp[i-1][j-1] + 1**. Иначе берётся максимум из соседних значений (**dp[i-1][j] **или** dp[i][j-1]**). Итоговое значение находится в** dp[n][m]**, где** n **и** m **— длины последовательностей.**

|  |  |  |
| --- | --- | --- |
|  | 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 эффективно решает задачу для любых двух последовательностей. Он помогает понять, сколько общих элементов можно упорядоченно выделить, что полезно в задачах сравнения строк, биоинформатике или текстовом анализе.**

## Задача 5:

# Текст задачи.

# 

# Листинг кода.

*"""Модуль для вычисления длины наибольшей общей подпоследовательности для трёх последовательностей."""*class LCS3Finder:  
 *"""  
 Класс для нахождения длины LCS для трёх последовательностей.  
 """* @staticmethod  
 def longest\_common\_subsequence\_3(a, b, c):  
 *"""  
 Вычисляет длину LCS для трёх последовательностей a, b и c.  
  
 :param a, b, c: Списки чисел.  
 :return: Длина LCS для трёх последовательностей.  
 """* n, m, l = len(a), len(b), len(c)  
 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])  
  
 return dp[n][m][l]

**Текстовое объяснение решения.**

Для трёх последовательностей используется аналогичный подход с динамическим программированием, но в трёхмерном массиве dp. Элемент dp[i][j][k] хранит длину LCS для первых i, j и k элементов трёх последовательностей. Если текущие символы всех трёх совпадают, добавляется единица к значению dp[i-1][j-1][k-1]. Если не совпадают, берётся максимум из значений, исключающих один из символов (по одной из трёх осей).

|  |  |  |
| --- | --- | --- |
|  | 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.**  
Это решение демонстрирует, как можно расширить методы для двух последовательностей на более сложные задачи. Хотя вычислительная сложность выше, задача наглядно показывает силу динамического программирования при решении многомерных проблем.

# Дополнительные задачи:

## Задача 6:

# Текст задачи.

# 

# Листинг кода.

# Lab7/Task6/src/LISFinder.py  
  
*"""Модуль для нахождения наибольшей возрастающей подпоследовательности (LIS)."""*from bisect import bisect\_left  
  
class LISFinder:  
 *"""  
 Класс для поиска наибольшей возрастающей подпоследовательности.  
 """* @staticmethod  
 def find\_lis(sequence):  
 *"""  
 Находит длину LIS и саму подпоследовательность.  
  
 :param sequence: Список чисел.  
 :return: Кортеж (lis\_length, lis\_sequence)  
 """* n = len(sequence)  
 lis = []  
 parent = [-1] \* n  
 indices = []  
  
 for i in range(n):  
 pos = bisect\_left(lis, sequence[i])  
 if pos == len(lis):  
 lis.append(sequence[i])  
 indices.append(i)  
 else:  
 lis[pos] = sequence[i]  
 indices[pos] = i  
  
 if pos > 0:  
 parent[i] = indices[pos - 1]  
  
 lis\_length = len(lis)  
 lis\_sequence = []  
 current\_index = indices[-1] if indices else -1  
  
 for \_ in range(lis\_length):  
 lis\_sequence.append(sequence[current\_index])  
 current\_index = parent[current\_index]  
  
 lis\_sequence.reverse()  
 return lis\_length, lis\_sequence

**Текстовое объяснение решения.**

**Для нахождения наибольшей возрастающей подпоследовательности применяется метод с использованием двоичного поиска. Сохраняется массив** lis**, в котором поддерживаются текущие элементы подпоследовательности. Для каждого нового элемента проверяется, можно ли заменить существующий элемент или добавить новый с помощью** bisect\_left **из библиотеки** bisect**. Таким образом, достигается эффективность 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.**  
Это решение показывает, как задача на упорядоченные данные может быть эффективно решена с использованием структур данных и алгоритмов поиска. LIS важна в анализе данных, где требуется найти упорядоченные тренды или последовательности.

## Задача 7:

# Текст задачи.

# 

# Листинг кода.

*"""Модуль для проверки совпадения строки с шаблоном, содержащим '?' и '\*'. """*class PatternMatcher:  
 *"""  
 Класс для проверки совпадения строки с шаблоном.  
 Шаблон может содержать:  
 - Буквы  
 - '?', соответствующий любому одиночному символу  
 - '\*', соответствующий нулю или более символов  
 """* @staticmethod  
 def matches\_pattern(pattern: str, string: str) -> str:  
 *"""  
 Проверяет, соответствует ли string шаблону pattern.  
  
 :param pattern: Шаблон.  
 :param string: Проверяемая строка.  
 :return: "YES" если соответствует, иначе "NO".  
 """* n, m = len(pattern), len(string)  
 dp = [[False]\*(m+1) for \_ in range(n+1)]  
 dp[0][0] = True  
  
 # Инициализация для '\*'  
 for i in range(1, n+1):  
 if pattern[i-1] == '\*':  
 dp[i][0] = dp[i-1][0]  
 else:  
 break  
  
 for i in range(1, n+1):  
 for j in range(1, m+1):  
 if pattern[i-1] == string[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 "YES" if dp[n][m] else "NO"

**Текстовое объяснение решения.**

**Для проверки соответствия строки шаблону используется динамическое программирование. Создаётся двумерный массив** dp**, где** dp[i][j] **указывает, возможно ли сопоставить первые** i **символов шаблона с первыми** j **символами строки. Символ** ? **совпадает с любым символом,** \* **может соответствовать пустой строке или любому количеству символов. Логика проверки основана на предыдущих значениях в массиве** dp**.**

|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений | 0.00782 сек | 0.48 МБ |
| входных данных из текста задачи |  |  |
| Пример из задачи | 0.06548 сек | 1.48 МБ |
| Пример из задачи | 0.25642 сек | 1.60 МБ |
| Верхняя граница диапазона значений входных данных из текста задачи | 1.10294 сек | 1.91 МБ |

**Вывод по задаче.  
Этот алгоритм полезен для задач, связанных с фильтрацией, поиском и обработкой текстовых данных. Его универсальность позволяет работать с широким спектром шаблонов, включая сложные случаи с использованием** \* **и** ?**.**

# Вывод

В ходе выполнения лабораторной работы были реализованы и протестированы алгоритмы для решения различных задач, включая поиск наибольшей общей подпоследовательности, работу с шаблонами и нахождение возрастающих подпоследовательностей. Эти задачи продемонстрировали практическое применение методов динамического программирования, структур данных и алгоритмов поиска, что позволило эффективно решать проблемы с различными ограничениями.