МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕЛЕКТРИЧНОЇ ІНЖЕНЕРІЇ
ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА АВТОМАТИЗАЦІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

НАВЧАЛЬНА ДИСЦИПЛІНА
«**АЛГОРИТМИ І СТРУКТУРИ ДАНИХ**»

ЗВІТ
З ПРАКТИЧНОЇ РОБОТИ №7

Виконав:
студент групи КН-24-1
Дон А.А.

Перевірив:
доцент кафедри АІС
Сидоренко В. М.

Кременчук 2025

Тема: Алгоритми на рядках

Мета роботи: Набути практичних навичок застосування базових алгоритмів на рядках та оцінювання їх асимптотичної складності.

Хід роботи

1. Теоретичні відомості

Рядок є однією з найпростіших структур даних, але вимагає знання низки важливих алгоритмів для їх ефективної обробки. Рядок - це послідовність символів із заданої множини.

У даній роботі розглядається задача знаходження найдовшої спільної підпослідовності (LCS - Longest Common Subsequence) двох рядків.

Підпослідовність Z рядка X - це рядок, отриманий із X шляхом вилучення деяких символів без зміни порядку інших символів.

Найдовша спільна підпослідовність двох рядків X і Y - це підпослідовність Z, яка є підпослідовністю і X, і Y одночасно, і має максимально можливу довжину.

Існує кілька алгоритмів для розв'язання цієї задачі:

Динамічне програмування (складність O(m·n))

Рекурсивний алгоритм з мемоізацією (складність O(m·n))

Алгоритм Хаббарда (Hirschberg's Algorithm) (складність O(m·n))

Алгоритм повного перебору (складність O(2^n))

2. Реалізація алгоритму динамічного програмування для знаходження LCS

```
def longest_common_subsequence(s1, s2):
    m = len(s1)
    n = len(s2)

    # Створення таблиці для зберігання проміжних результатів
    # dp[i][j] буде містити довжину найбільшої спільної підпослідовності для s1[:i] i s2[:j]
    dp = [[0] * (n + 1) for _ in range(m + 1)]
```

```
    # Заповнення таблиці знизу вгору
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if s1[i - 1] == s2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1] + 1
            else:
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

    # Відновлення найбільшої спільної підпослідовності
    lcs = []
    i, j = m, n
    while i > 0 and j > 0:
        if s1[i - 1] == s2[j - 1]:
            lcs.append(s1[i - 1])
            i -= 1
            j -= 1
        elif dp[i - 1][j] > dp[i][j - 1]:
            i -= 1
        else:
            j -= 1

    # Перевернення lcs, оскільки ми додавали символи з кінця
    lcs.reverse()
    return ''.join(lcs)
```

3. Приклад використання алгоритму

```
def print_dp_table(s1, s2, dp):
    """Функція для візуалізації таблиці динамічного програмування"""
    print("   ", end=" ")
    print("  ", end=" ")
    for char in s2:
        print(f"{char}", end=" ")
    print()

    for i in range(len(dp)):
        if i == 0:
            print(" ", end=" ")
        else:
            print(s1[i-1], end=" ")
        for j in range(len(dp[i])):
            print(f"{dp[i][j]}", end=" ")
```

```
        print()

def lcs_with_table(s1, s2):
    """Реалізація алгоритму LCS з виведенням таблиці динамічного програмування"""
    m = len(s1)
    n = len(s2)

    dp = [[0] * (n + 1) for _ in range(m + 1)]

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if s1[i - 1] == s2[j - 1]:
                dp[i][j] = dp[i - 1][j - 1] + 1
            else:
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])

    # Виведення таблиці
    print("Таблиця динамічного програмування:")
    print_dp_table(s1, s2, dp)

    # Відновлення найбільшої спільної підпослідовності
    lcs = []
    i, j = m, n
    while i > 0 and j > 0:
        if s1[i - 1] == s2[j - 1]:
            lcs.append(s1[i - 1])
            i -= 1
            j -= 1
        elif dp[i - 1][j] > dp[i][j - 1]:
            i -= 1
        else:
            j -= 1

    lcs.reverse()
    return ''.join(lcs)

# Приклад з документа
s1 = "ABCD"
s2 = "ACDB"
print(f"Рядок 1: {s1}")
print(f"Рядок 2: {s2}")
```

```
result = lcs_with_table(s1, s2)
print(f"Найдовша спільна підпослідовність: {result}")

# Самостійне завдання
print("\nСамостійне завдання:")
s1 = "ABCDF"
s2 = "ACEDB"
print(f"Рядок 1: {s1}")
print(f"Рядок 2: {s2}")
result = lcs_with_table(s1, s2)
print(f"Найдовша спільна підпослідовність: {result}")
```

4. Розв'язання самостійного завдання

Для рядків "ABCDF" і "ACEDB" застосуємо алгоритм динамічного програмування для знаходження найдовшої спільної підпослідовності:

Створюємо матрицю dp розміром (6×6), оскільки довжини рядків 5 і 5, і додаємо один рядок та один стовпець для ініціалізації.

Заповнюємо матрицю відповідно до алгоритму:

Якщо символи однакові, dp[i][j] = dp[i-1][j-1] + 1

Інакше, dp[i][j] = max(dp[i-1][j], dp[i][j-1])

Відновлюємо найдовшу спільну підпослідовність, рухаючись від правого нижнього кута матриці.

Результат для рядків "ABCDF" і "ACEDB" - підпослідовність "AB" або "AD" (обидва варіанти мають довжину 2). Конкретний варіант залежатиме від реалізації.

Висновки

У ході виконання практичної роботи:

Було розглянуто та проаналізовано алгоритми для знаходження найдовшої спільної підпослідовності (LCS) двох рядків.

Детально вивчено алгоритм динамічного програмування для розв'язання цієї задачі, який має часову складність O(m·n), де m і n - довжини вхідних рядків.

Реалізовано алгоритм мовою Python, який дозволяє ефективно знаходити найдовшу спільну підпослідовність.

Проведено тестування алгоритму на прикладах, включаючи приклад із завдання ("ABCD" і "ACDB") та самостійне завдання ("ABCDF" і "ACEDB").

На практиці переконались, що алгоритм динамічного програмування дозволяє знаходити оптимальне рішення за поліноміальний час, на відміну від методу повного перебору, який має експоненційну складність.

Отримано практичні навички роботи з алгоритмами на рядках та оцінювання їх асимптотичної складності, що є корисним для розв'язання широкого спектру задач, пов'язаних з обробкою текстових даних, біоінформатикою та іншими галузями.