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 та методи обчислень

ЗВІТ

З ПРАКТИЧНИХ РОБІТ

Виконав:

студент групи КН-23-1

Сидоренко А.Ю.

Кременчук 2024

**Практична робота № 6**

**Тема.** Алгоритми на рядках

**Мета:** набути практичних навичок застосування базових алгоритмів на рядках та оцінювання їх асимптотичної складності.

**Хід роботи**

**Завдання:**

18. Маємо дві короткі послідовності символів: «XYZ» і «ABC». Знайти найдовшу спільну підпослідовності символів, використовуючи алгоритм Хаббарда.

Створимо матрицю розміром (m+1) × (n+1), де m - довжина першої послідовності, а n - довжина другої послідовності .

Ініціалізуємо перший рядок та перший стовпець матриці нулями.

Проходимо по всім елементам матриці, використовуючи наступний рекурентний вираз для обчислення dp[i][j]:

Якщо символи на позиціях (i-1) і (j-1) рівні, то dp[i][j] = dp[i-1][j-1] + 1.

В іншому випадку, dp[i][j] приймає значення максимуму з dp[i-1][j] та dp[i][j-1].

Після завершення обчислень, dp[m][n] буде містити довжину найдовшої спільної підпослідовності.

Для відновлення самої НСП можна використати матрицю dp, знаючи, що в кожній клітинці dp[i][j] зберігається довжина LCS для підпослідовностей s1 та s2.

def longest\_common\_subsequence(s1, s2):

m, n = len(s1), len(s2)

# Ініціалізуємо матрицю dp

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

# Заповнюємо матрицю dp

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])

# Повертаємо довжину найдовшої спільної підпослідовності та саму підпослідовність

return dp[m][n], reconstruct\_lcs(dp, s1, s2)

def reconstruct\_lcs(dp, s1, s2):

i, j = len(s1), len(s2)

lcs = []

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

return ''.join(reversed(lcs))

s1 = "XYZ"

s2 = "ABC"

length, lcs = longest\_common\_subsequence(s1, s2)

print(f"Довжина найдовшої спільної підпослідовності: {length}")

Довжина найдовшої спільної підпослідовності: 0

Перевірка вручну:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | A | B | C |
|  | 0 | 0 | 0 | 0 |
| X | 0 | 0 | 0 | 0 |
| Y | 0 | 0 | 0 | 0 |
| Z | 0 | 0 | 0 | 0 |

**Контрольні питання**

1. **У чому полягає задача знаходження найдовшої спільної підпослідовності (LCS)?**

Задача знаходження найдовшої спільної підпослідовності (LCS) визначає найдовшу послідовність елементів, яка з'являється в тому самому порядку в обох вихідних послідовностях, але не обов'язково підряд.

1. **Які головні методи можна використовувати для знаходження найдовшої спільної підпослідовності?**

* Динамічне програмування
* Алгоритм Хаббарда
* Рекурсивний алгоритм з використанням мемоізації
* Алгорим повного перебору

1. **Як працює алгоритм динамічного програмування для знаходження LCS?**

Алгоритм динамічного програмування для знаходження найдовшої спільної підпослідовності (LCS) працює шляхом обчислення і зберігання проміжних результатів у вигляді матриці.

Основна ідея полягає в тому, щоб рекурсивно порівнювати елементи двох вхідних послідовностей та обчислювати довжину LCS для підпослідовностей з префіксів цих послідовностей.

Кроки:

* Створення матриці (рядок і стовпець)+1 і ініціалізація першого рядка та стовпця по 0
* Обчилення значень шляхом проходження по всім елементам за допомогою рекурентного виразу
* Повертаємо результат у перші рядок і стовпець
* Відновлюємо LSC рухаючись до нульовоих рядків і стовпців

1. **Як працює алгоритм Хаббарда для знаходження LCS?**

Цей алгоритм є модифікацією динамічного програмування для знаходження найдовшої спільної підпослідовності (LCS). Він використовує менше пам'яті, ніж стандартний алгоритм динамічного програмування, і базується на техніці "розділяй та володарюй".

Алгоритм Хаббарда працює за принципом динамічного програмування, але робить це у рекурсивний спосіб.

Він спочатку розбиває задачі на підзадачі. Далі використовує динамичне програмування, а закінчує рекурсивним розподілом та володарюванням. І виводить канцевий результат

1. **Які переваги та недоліки алгоритмів динамічного програмування та Хаббарда для знаходження LCS?**

**Переваги динамічного програмування:**

Простота реалізації

Прямолінійність

Низька часова складність (O(n\*m))

**Недоліки:**

Потребує O(n\*m) для збереження матриці

**Переваги Хаббарда:**

Рекурсивний підхід

Ефективне використання пам’яті

**Недоліки:**

Складність реалізації

Часова складність для великих послідовностей

Відновлення LCS ( через використання додаткових викликів та розбиття на підзадачі)

1. **Які існують практичні застосування для задачі знаходження найдовшої спільної підпослідовності?**

Текстова обробка, порівняння текстів, відновлення текстві, аналіз природної мови, виявлення пошкодження даних, аналіз відео та зображень, використання для перевірки в навчальних системах і тд.