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

Шпак А.П.

Перевірив

доцент кафедри КІЕ

Сидоренко В.М.

Кременчук 2025

**Тема.** Графи. Ациклічні графи

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

**Хід роботи**

**Завдання №5**

1. **Розв’язок**

Маємо дві короткі послідовності символів: «XYBAC» і «ABXC».

Створимо таблицю розміром (𝑙𝑒𝑛(𝑠1) + 1) 𝑥 (𝑙𝑒𝑛(𝑠2) + 1) та ініціалізуємо всі елементи матриці нулями:

'' A B X C

'' 0 0 0 0 0

X 0

Y 0

B 0

A 0

C 0

**Перший рядок (s1[0] = 'X'):**

dp[1][1] (X vs A): не рівні → max(0, 0) = 0

dp[1][2] (X vs B): не рівні → max(0, 0) = 0

dp[1][3] (X vs X): рівні → dp[0][2] + 1 = 0 + 1 = 1

dp[1][4] (X vs C): не рівні → max(0, 1) = 1

'' A B X C

'' 0 0 0 0 0

X 0 0 0 1 1

**Другий рядок (**s1[1] = 'Y'**):**

dp[2][1] (Y vs A): не рівні → max(0, 0) = 0

dp[2][2] (Y vs B): не рівні → max(0, 0) = 0

dp[2][3] (Y vs X): не рівні → max(1, 0) = 1

dp[2][4] (Y vs C): не рівні → max(1, 1) = 1

'' A B X C

'' 0 0 0 0 0

X 0 0 0 1 1

Y 0 0 0 1 1

**Третій рядок (**s1[2] = 'B'**):**

dp[3][1] (B vs A): не рівні → max(0, 0) = 0

dp[3][2] (B vs B): рівні → dp[2][1] + 1 = 0 + 1 = 1

dp[3][3] (B vs X): не рівні → max(1, 1) = 1

dp[3][4] (B vs C): не рівні → max(1, 1) = 1

'' A B X C

'' 0 0 0 0 0

X 0 0 0 1 1

Y 0 0 0 1 1

B 0 0 1 1 1

**Четвертий рядок (**s1[3] = 'A'**):**

dp[4][1] (A vs A): рівні → dp[3][0] + 1 = 0 + 1 = 1

dp[4][2] (A vs B): не рівні → max(1, 1) = 1

dp[4][3] (A vs X): не рівні → max(1, 1) = 1

dp[4][4] (A vs C): не рівні → max(1, 1) = 1

'' A B X C

'' 0 0 0 0 0

X 0 0 0 1 1

Y 0 0 0 1 1

B 0 0 1 1 1

A 0 1 1 1 1

#### ****П’ятий рядок (****s1[4] = 'C'****):****

dp[5][1] (C vs A): не рівні → max(1, 0) = 1

dp[5][2] (C vs B): не рівні → max(1, 1) = 1

dp[5][3] (C vs X): не рівні → max(1, 1) = 1

dp[5][4] (C vs C): рівні → dp[4][3] + 1 = 1 + 1 = 2

'' A B X C

'' 0 0 0 0 0

X 0 0 0 1 1

Y 0 0 0 1 1

B 0 0 1 1 1

A 0 1 1 1 1

C 0 1 1 1 2

dp[5][4] = 2

Починаємо з dp[5][4] = 2, рухаємось вгору та вліво:

s1[4]='C', s2[3]='C' → збігаються → додаємо 'C', переходимо до dp[4][3]

s1[3]='A', s2[2]='X' → не збігаються → переходимо туди, де max (ліворуч чи вгору)

Далі знайдемо, що s1[3]='A' і s2[0]='A' → збігаються → додаємо 'A'

1. **Відповідь:**

Отже, результатом для послідовностей «XYBAC» і «ABXC» буде найдовша спільна підпослідовність «AC».

1. **Висновок:**

Набув практичних навичок застосування базових алгоритмів на рядках та оцінювання їх асимптотичної складності.

# **Контрольні питання:**

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

**Задача LCS** – полягає у знаходженні найдовшої підпослідовності, яка зустрічається в обох рядках у тому ж порядку.

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

**Методи для знаходження LCS**:

**Динамічне програмування** – використовує таблицю для збереження проміжних результатів.

**Рекурсивний підхід** – шукає LCS за допомогою розбиття на підзадачі.

**Алгоритм Хаббарда** – покращений метод, який використовує спеціальні структури даних.

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

**Динамічне програмування для LCS**:

Створюється таблиця розміром m × n, де m і n – довжини рядків.

Заповнюється таблиця на основі порівняння символів.

Остаточний результат – найдовша спільна підпослідовність.

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

 **Алгоритм Хаббарда**:

Використовує структури даних для швидкого знаходження збігів.

Покращує швидкодію у порівнянні з класичним підходом.

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

**Переваги та недоліки**:

**Динамічне програмування**: точне, але може займати багато пам’яті.

**Хаббард**: швидший для великих рядків, але складніший в реалізації.

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

**Практичні застосування**:

**Порівняння текстів** (пошук змін між версіями документів).

**Генетичні дослідження** (порівняння ДНК-послідовностей).

**Стиснення даних** (оптимізація кодів і шаблонів).