МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КРЕМЕНЧУЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ МИХАЙЛА ОСТРОГРАДСЬКОГО

Кафедра комп'ютерної інженерії та електроніки

ЗВІТ З ПРАКТИЧНИХ РОБІТ

з навчальної дисципліни
«Алгоритми та методи обчислень»

Тема «Алгоритми на рядках»

Студент гр. КІ-23-1 ПІБ Кобець О. О.

Кременчук 2024

Практична робота № 7

Тема. Алгоритми на рядках

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

Завдання

8. Маємо дві короткі послідовності символів: «HELLO» і «WORLD». Знайти найдовшу спільну підпослідовності символів, використовуючи алгоритм Хаббарда.

```python
A = "HELLO"
B = "WORLD"

n = len(A)
m = len(B)

C = [[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]:
            C[i][j] = C[i - 1][j - 1] + 1
        else:
            C[i][j] = max(C[i - 1][j], C[i][j - 1])

lcs = []
i, j = n, m
while i > 0 and j > 0:
    if A[i - 1] == B[j - 1]:
        lcs.append(A[i - 1])
        i -= 1
        j -= 1
    elif C[i - 1][j] > C[i][j - 1]:
        i -= 1
    else:
        j -= 1

lcs.reverse()

print("Матриця динамічного програмування:")
print("    ", "  ".join(B))
for i, row in enumerate(C):
    if i == 0:
        print("   ", row)
    else:
        print(A[i - 1], row)

print("\nНайдовша спільна підпослідовність (LCS):", ''.join(lcs))
```

```
    W  O  R  L  D
    [0, 0, 0, 0, 0, 0]
H [0, 0, 0, 0, 0, 0]
E [0, 0, 0, 0, 0, 0]
L [0, 0, 0, 0, 1, 1]
L [0, 0, 0, 0, 1, 1]
O [0, 0, 1, 1, 1, 1]

Найдовша спільна підпослідовність (LCS): O
```

Найдовша спільна підпослідовність: O

Контрольні питання

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

Задача LCS полягає в тому, щоб знайти найдовшу підпослідовність, яка є спільною для двох або більше послідовностей (рядків), зберігаючи порядок символів, але допускаючи пропуски (невключення деяких символів).

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

Основні методи:

1. **Динамічне програмування:**
    - Побудова матриці для порівняння всіх символів та відновлення LCS.
2. **Рекурсивний підхід:**
    - Використання розгалуженого перебору з відстеженням можливих збігів.
3. **Хаббардовий алгоритм (Hubbard's Algorithm):**
    - Підхід із побудовою спеціальних структур, таких як графи, для оптимізації пошуку.
4. **Евристичні методи:**
    - Застосовуються для приблизного знаходження LCS у великих послідовностях, наприклад, біологічних даних.

3. Як працює алгоритм динамічного програмування для знаходження LCS?

Побудова матриці:
Створюється таблиця розміром $(m+1)\times(n+1)$, де m і n — довжини послідовностей.
Кожна клітинка [i][j] зберігає довжину LCS для підрядків A[0:i] та B[0:j].

Заповнення таблиці:

- Якщо символи збігаються: $C[i][j]=C[i-1][j-1]+1$
- Якщо не збігаються: $C[i][j]=\max(C[i-1][j],C[i][j-1])$

Відновлення LCS:
Починаючи з кінця таблиці, рухаючись назад за збігами, відновлюємо LCS.

4. Як працює алгоритм Хаббарда для знаходження LCS?

Алгоритм Хаббарда застосовує графову модель:

1. Кожна вершина графа відповідає символу послідовності.
2. Ребра між вершинами позначають можливі збіги символів між послідовностями.
3. LCS знаходиться шляхом побудови найдовшого шляху в графі за умов, що зберігається порядок символів.

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

Динамічне програмування:

Переваги:

- Проста реалізація.
- Гарантовано знаходить оптимальне рішення.

Недоліки:

- Потребує $O(m\times n)$ пам'яті та часу.

Алгоритм Хаббарда:

Переваги:

- Підходить для великих послідовностей.
- Може бути швидшим у спеціальних задачах.

Недоліки:

- Складність реалізації.
- Може бути менш ефективним для малих послідовностей.

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

1. **Біоінформатика:**
   Вирівнювання ДНК, РНК або білкових послідовностей для виявлення схожості між геномами.
2. **Обробка тексту:**
   Пошук спільних фраз або схожості між документами.
3. **Контроль версій:**
   Порівняння та об'єднання змін у програмному коді.
4. **Розпізнавання мовлення:**
   Виявлення загальних шаблонів у звукових даних.
5. **Відновлення даних:**
   Пошук схожих структур у пошкоджених файлах.