Практична робота №7. Графи. Найкоротші шляхи

2025.11.23,м.Кременьчук	Створив: Огоновський О.Є.

Мета: набути практичних навичок розв’язання задач пошуку найкоротших шляхів у графі та оцінювання їх асимптотичної складності.

Задача для самостійного розв’язання

16. Маємо дві короткі послідовності символів: «AABBCC» і «ABABAB».Знайти найдовшу спільну підпослідовності символів, використовуючи алгоритм Хаббарда

Будуємо таблицю

	0	A	B	A	B	A	B
0	0	0	0	0	0	0	0
A	0	1	1	1	1	1	1
A	0	1	1	2	2	2	2
B	0	1	2	2	3	3	3
B	0	1	2	2	3	3	4
C	0	1	2	2	3	3	4
C	0	1	2	2	3	3	4

Максимальне значення - **4**.

Відновлення LCS (рухаємось назад за «стрілками» Хаббарда)

Починаємо з `L[6][6] = 4`:

- S1[3] = **B**, S2[6] = **B** -> в LCS
- S1[2] = **A**, S2[5] = **A** -> в LCS
- S1[1] = **A**, S2[3] = **A** -> в LCS

- S1[1] = **A**, S2[1] = **A** -> (але вже не додаємо, бо збіг по індексах)

Так отримуємо послідовність у зворотному напрямку:

B B A A -> перевертаємо -> A A B B

Результат: Найдовша спільна підпослідовність: AABB

Контрольні питання

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

   Задача LCS полягає в тому, щоб знайти найдовшу послідовність символів, яка:

   зустрічається в обох рядках, у тому самому порядку, але не обов'язково підряд.

   Тобто LCS - це максимальна спільна підпослідовність двох (або більше) послідовностей.

2. **Які головні методи можна використовувати для знаходження LCS?**

   Основні методи:

   Рекурсивний метод Пряме визначення через порівняння останніх символів (експоненційна складність).

   Алгоритм динамічного програмування (DP) Використовує таблицю (матрицю) для пошуку оптимального результату. Складність: O(n·m).

   Алгоритм Хаббарда Покращений варіант DP з оптимізацією пам'яті та використанням напрямків руху для відновлення LCS.

   Алгоритм Гірша—Берґера (для великих послідовностей) Зменшує пам'ять до O(min(n, m)).

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

   Динамічний алгоритм створює таблицю:

   L[i][j] - довжина LCS для перших i символів першого рядка і перших j символів другого рядка

   Рекурентні формули:

   Якщо символи збігаються: L[i][j] = L[i-1][j-1] + 1

Якщо не збігаються: L[i][j] = max(L[i-1][j], L[i][j-1])

Після заповнення таблиці - результат у L[n][m].

Для відновлення самої підпослідовності рухаються назад з правого нижнього кута таблиці.

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

   Алгоритм Хаббарда використовує динамічне програмування, але з оптимізацією для економії пам'яті. На відміну від класичного підходу, який вимагає O(m×n) пам'яті, алгоритм Хаббарда може працювати з O(min(m,n)) пам'яті.

   Кроки алгоритму:

   - Ініціалізація: Створюється масив довжиною min(m,n) + 1, де m та n — довжини вхідних рядків
   - Заповнення таблиці: Алгоритм проходить по рядках і стовпцях, оновлюючи значення в масиві
   - Відстеження змін: Зберігається лише поточний рядок таблиці, а не вся таблиця

5. **Переваги та недоліки DP та алгоритму Хаббарда Динамічне програмування**

   Переваги:

   - гарантує оптимальний результат,
   - зрозумілий і широко використовується,
   - легко реалізується.

   Недоліки:

   - потребує O(n·m) пам’яті,
   - не найшвидший для дуже великих текстів.

   Алгоритм Хаббарда

   Переваги:

   - компактніше зберігає інформацію про напрямки,
   - пришвидшує відновлення LCS,

- може оптимізувати використання пам’яті.

Недоліки:

- складніша реалізація,
- все ще потребує O(n·m) часу,
- менш універсальний, ніж класичний DP.

6. **Практичні застосування задачі LCS**

Задача знаходження LCS використовується у:

Системах контролю версій (Git, SVN)

- для пошуку відмінностей у файлах.

Порівнянні текстів

- визначення схожості документів, пошук змін.

Біоінформатиці

- порівняння ДНК, білкових послідовностей.

Редакторах коду

- підсвітка різниць (diff).

Компресії даних

- пошук повторюваних структур.

Виявлення плагіату

- оцінка подібності текстів.