BỘ CÔNG THƯƠNG

TRƯỜNG ĐẠI HỌC CÔNG THƯƠNG TP.HỒ CHÍ MINH

KHOA CÔNG NGHỆ THÔNG TIN

-----o0o----

***Đề tài*:**

MẢNG PHẦN TỬ CHUNG DÀI NHẤT CỦA HAI MẢNG

Nhóm: 09
Lớp học phần: 13DHTH01
GVHD: Phan Thị Ngọc Mai

Thành phố Hồ Chí Minh, tháng 10 năm 2024

BỘ CÔNG THƯƠNG

TRƯỜNG ĐẠI HỌC CÔNG THƯƠNG TP.HỒ CHÍ MINH

KHOA CÔNG NGHỆ THÔNG TIN

-----o0o----

Đề tài

Mảng phần tử chung dài nhất của hai mảng

Nhóm: 08 Thành viên: 1. Võ Trường Danh 2. Nguyễn Vương Hồng Đào 3. Hồ Thái Dinh 4. Trần Nguyễn Như Huỳnh	**Giảng viên hướng dẫn:** **Phan Thị Ngọc Mai**

Thành phố Hồ Chí Minh, tháng 10 năm 2024

Lời cam đoan

Em/chúng em xin cam đoan đề tài tiểu luận: **Mảng phần tử chung dài nhất của hai mảng** do cá nhân/nhóm 09 nghiên cứu và thực hiện.

Em/ chúng em đã kiểm tra dữ liệu theo quy định hiện hành.

Kết quả bài làm của đề tài **Mảng phần tử chung dài nhất của hai mảng** là trung thực và không sao chép từ bất kỳ bài tập của nhóm khác.

Các tài liệu được sử dụng trong tiểu luận có nguồn gốc, xuất xứ rõ ràng.

(Ký và ghi rõ họ tên)

Nguyễn Vương Hồng Đào

Lời cảm ơn

Lời đầu tiên, em/ chúng em xin được gửi lời cảm ơn chân thành nhất đến cô **Phan Thị Ngọc Mai.** Trong quá trình học tập và tìm hiểu môn **Phân tích thiết kế thuật toán**, em/ chúng em đã nhận được rất nhiều sự quan tâm, giúp đỡ, hướng dẫn tâm huyết và tận tình của cô. Cô đã giúp em/ chúng em tích lũy thêm nhiều kiến thức về môn học này để có thể hoàn thành được bài tiểu luận về đề tài: **Mảng phần tử chung dài nhất của hai mảng**

Trong quá trình làm bài chắc chắn khó tránh khỏi những thiếu sót. Do đó, em/ chúng em kính mong nhận được những lời góp ý của cô để bài tiểu luận của em ngày càng hoàn thiện hơn.

Em/ chúng em xin chân thành cảm ơn!

MỤC LỤC

I. Giới thiệu về Quy hoạch động và Chuỗi con chung dài nhất (LCS)

1 Quy hoạch động (Dynamic Programming - DP)

1.1 Khái niệm

Quy hoạch động (dynamic programming) là một phương pháp được sử dụng trong toán học và khoa học máy tính để giải quyết các vấn đề phức tạp bằng cách chia chúng thành các bài toán con đơn giản hơn. Bằng cách giải mỗi bài toán con chỉ một lần và lưu trữ kết quả, nó tránh được các phép tính dư thừa, dẫn đến giải pháp hiệu quả hơn cho nhiều bài toán.

1.2 Nguyên lý cơ bản

- Chia bài toán thành các bài toán con nhỏ hơn: Bài toán ban đầu được chia thành các bài toán con mà mỗi bài toán con là một phần của bài toán lớn hơn. Các bài toán con này thường có tính chất lặp lại.
- Lưu trữ kết quả của các bài toán con: Để tránh việc tính toán lặp lại các bài toán con nhiều lần, kết quả của chúng được lưu trữ trong một bảng (thường là mảng hoặc ma trận).
- Sử dụng lại kết quả đã lưu: Khi cần kết quả của một bài toán con nào đó, chương trình sẽ kiểm tra bảng lưu trữ để xem kết quả đã được tính toán trước đó chưa. Nếu đã có, nó sẽ sử dụng lại kết quả đó thay vì tính toán lại từ đầu.

1.3 Cách tiếp cận trong Quy hoạch động

- **Top-down (Memoization):** Bắt đầu từ bài toán lớn nhất và chia nhỏ dần, lưu trữ kết quả của các bài toán con để sử dụng lại khi cần.
- **Bottom-up (Tabulation):** Bắt đầu từ các bài toán con nhỏ nhất và kết hợp chúng để giải quyết bài toán lớn hơn.

1.4 Các ví dụ điển hình và ứng dụng về Quy hoạch động

- **Bài toán Fibonacci:** Tính số Fibonacci thứ n bằng cách lưu trữ kết quả của các số Fibonacci trước đó.

- **Bài toán balo (Knapsack Problem):** Tìm cách chọn các vật phẩm để tối đa hóa giá trị mà không vượt quá trọng lượng cho phép.
- **Bài toán chuỗi con chung dài nhất (LCS):** Tìm chuỗi con dài nhất xuất hiện trong cả hai chuỗi cho trước.

1.5 Ưu và nhược điểm của Quy hoạch động

- **Ưu điểm:**
    - Giảm thời gian tính toán: Quy hoạch động lưu trữ kết quả của các bài toán con đã được tính toán trước đó, giúp giảm thiểu đáng kể số lượng phép tính cần thực hiện. Điều này đặc biệt hiệu quả trong các bài toán có tính chất lặp lại, như dãy Fibonacci hoặc bài toán tối ưu hóa.
    - Giải quyết được các bài toán phức tạp: DP cho phép giải quyết nhiều bài toán tối ưu hóa và lập lịch phức tạp mà các phương pháp khác không thể làm được hiệu quả, chẳng hạn như bài toán balo, chuỗi con chung dài nhất (LCS), và nhiều bài toán đồ thị.
    - Tránh được các tính toán lặp lại không cần thiết: Bằng cách lưu trữ kết quả của các bài toán con, DP tránh được việc tính toán lặp lại những phần đã giải quyết, giúp tiết kiệm tài nguyên và tăng hiệu suất.
    - Độ chính xác cao: Kỹ thuật DP đảm bảo tìm ra giải pháp tối ưu cho bài toán bằng cách xây dựng các giải pháp từ các bài toán con tối ưu.

- **Nhược điểm:**
    - Tốn bộ nhớ: DP yêu cầu lưu trữ kết quả của tất cả các bài toán con, điều này có thể tốn rất nhiều bộ nhớ, đặc biệt đối với các bài toán có không gian trạng thái lớn. Điều này có thể gây ra vấn đề khi áp dụng DP cho các bài toán có quy mô lớn.
    - Phức tạp trong việc triển khai: Hiểu và triển khai một giải pháp DP đòi hỏi người lập trình phải hiểu rõ cấu trúc của bài toán và cách chia nhỏ nó thành các bài toán con. Điều này có thể phức tạp và tốn nhiều thời gian để thiết kế và debug.

- Không phải lúc nào cũng khả thi: Không phải bài toán nào cũng có thể áp dụng được DP. Các bài toán cần có tính chất con tối ưu (optimal substructure) và tính chất lặp lại của bài toán con (overlapping subproblems). Nếu bài toán không có các tính chất này, DP sẽ không hiệu quả.
- Tiêu tốn thời gian cho việc lưu trữ và truy xuất: Mặc dù giảm thời gian tính toán, việc lưu trữ và truy xuất kết quả từ bộ nhớ cũng tiêu tốn thời gian, đặc biệt khi kích thước của bảng lưu trữ lớn.

2 Bài toán Chuỗi con chung dài nhất (LCS)

Dãy con chung dài nhất (LCS) được định nghĩa là dãy con dài nhất chung cho tất cả các chuỗi đã cho, miễn là các phần tử của dãy con không bắt buộc phải chiếm các vị trí liên tiếp trong các chuỗi ban đầu**.**

II. Nguyên lý của Quy hoạch động trong bài toán LCS

Cho hai dãy ký hiệu X và Y, dãy con chung dài nhất của X và Y là dãy các ký hiệu nhận được từ X bằng cách xóa đi một số các phần tử và cũng nhận được từ Y bằng cách xóa đi một số phần tử.

Ví dụ: cho X = ABCDCAE; Y = DACDBA

X =		**A**	B	**C**	**D**	C		**A**	E
Y = D	**A**		**C**	**D**		B	**A**	

Dãy con chung dài nhất: ACDA.

III. Công thức Quy hoạch động cho LCS

- Ta ký hiệu :
  - Xi = x1x2…xi được gọi là tiền tố thứ i của X.
  - Yj = y1y2…yj là tiền tố thứ j của Y.
- Dùng mảng c[0..n, 0..m] để lưu giá trị độ dài của các dãy con chung dài nhất của các cặp tiền tố.

- Gọi c[i, j] là độ dài dãy con chung dài nhất của Xi và Yj .
- Khi đó độ dài dãy con chung dài nhất của X và Y sẽ là c[n,m].
- Trường hợp đơn giản nhất: độ dài dãy con chung dài nhất của của một dãy rỗng và một dãy bất kỳ luôn bằng 0 do đó c[i, 0] = 0 và c[0, j] = 0 với mọi i, j.

Vậy ta có:

$$c[i,j] = \begin{cases} 0 & i = 0 \text{ hay } j = 0 \\ c[i-1, j-1] + 1 & i, j > 0, x_i = y_j \\ max(c[i-1, j],\ c[i, j-1] & i, j > 0, x_i \neq y_j \end{cases}$$

IV. Thuật toán Chuỗi con chung dài nhất (LCS)

1 Thuật toán quy hoạch động :

```
Procedure LCS-length(X, Y)
Begin
    // Bước 1: Xác định độ dài của hai chuỗi X và Y và khởi tạo ma trận c có kích thước (n+1) x (m+1)
    n = length(X)        // Độ dài chuỗi X
    m = length(Y)        // Độ dài chuỗi Y
  // Bước 2: Thiết lập hàng và cột đầu tiên
    For i from 0 to n do
        c[i, 0] = 0      // Chuỗi con của Y là rỗng
    Endfor
    For j from 0 to m do
        c[0, j] = 0      // Chuỗi con của X là rỗng
    Endfor
    // Bước 3: Tính toán độ dài LCS
    For i from 1 to n do
```

```
        For j from 1 to m do
            If X[i] == Y[j] then
                c[i, j] = c[i-1, j-1] + 1
            Else
                c[i, j] = max(c[i-1, j], c[i, j-1])
            Endif
        Endfor
    Endfor
    // Bước 4: Trả về kết quả
    Return c
End
```

2 Độ phức tạp thuật toán :

Thuật toán tính mảng n x m phần tử bởi hai vòng lặp lồng nhau: độ phức tạp O(nm).

3 Ví dụ minh họa:

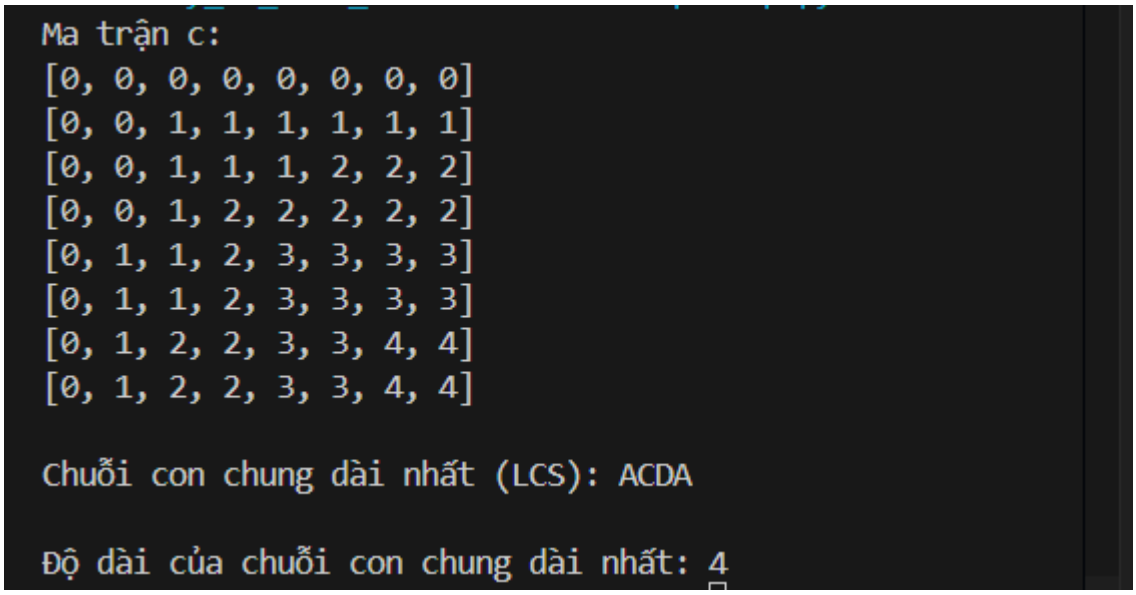
- **Chuỗi X:** ABCDCAE
- **Chuỗi Y:** DACDBA
- **Kết quả LCS:** ACDA (Độ dài LCS = 4)

Cài đặt bằng Python:

```python
def lcs_length(X, Y):
    # Bước 1: Xác định độ dài của hai chuỗi X và Y
    n = len(X)  # Độ dài chuỗi X
    m = len(Y)  # Độ dài chuỗi Y

    # Bước 2: Khởi tạo ma trận c với kích thước (n+1) x (m+1)
    c = [[0 for _ in range(m+1)] for _ in range(n+1)]
```

```
    # Bước 3: Tính toán độ dài LCS
    for i in range(1, n+1):
        for j in range(1, m+1):
            if X[i-1] == Y[j-1]:
                c[i][j] = c[i-1][j-1] + 1
            else:
                c[i][j] = max(c[i-1][j], c[i][j-1])

    # Bước 4: Trả về ma trận c và chuỗi con chung dài nhất
    lcs = []
    i, j = n, m
    while i > 0 and j > 0:
        if X[i-1] == Y[j-1]:
            lcs.append(X[i-1])  # Thêm ký tự vào chuỗi LCS
            i -= 1
            j -= 1
        elif c[i-1][j] >= c[i][j-1]:
            i -= 1
        else:
            j -= 1

    # Vì chuỗi con chung dài nhất được tìm theo thứ tự ngược, ta đảo lại chuỗi
    lcs.reverse()

    # In ma trận LCS
    print("Ma trận c:")
    for row in c:
        print(row)

    # In chuỗi con chung dài nhất
    print("\nChuỗi con chung dài nhất (LCS): " + "".join(lcs))
    lcs_length = c[n][m]
    print(f"\nĐộ dài của chuỗi con chung dài nhất: {lcs_length}")


# Ví dụ sử dụng
X = "ABCDCAE"
Y = "DACDBA "

# Gọi hàm và tính toán LCS
lcs_length(X, Y)
```

Kết quả nhận được:



V. Ứng dụng của thuật toán LCS :

- **So sánh chuỗi**:
  - LCS có thể được sử dụng để xác định sự tương đồng giữa hai chuỗi. Trong các ứng dụng như kiểm tra chính tả, tìm kiếm văn bản, hoặc phân tích văn bản, LCS giúp phát hiện các phần giống nhau giữa hai đoạn văn bản.
- **Khôi phục dữ liệu:**
  - Trong các hệ thống khôi phục dữ liệu hoặc quản lý phiên bản (version control), LCS giúp xác định các thay đổi giữa hai phiên bản của một tài liệu. Nó cho phép xác định những phần nào đã thay đổi, thêm hoặc xóa.
- **Lập trình di truyền (Bioinformatics):**
  - Trong sinh học, LCS được sử dụng để so sánh chuỗi DNA, RNA hoặc protein. Nó giúp tìm kiếm các chuỗi gen hoặc cấu trúc tương đồng trong các loài khác nhau.
- **Nhận diện ngữ nghĩa (Natural Language Processing)**:

- Trong xử lý ngôn ngữ tự nhiên, LCS có thể được áp dụng để so sánh các câu hoặc đoạn văn để phát hiện các ý nghĩa tương đồng, phục vụ cho việc phân loại hoặc trích xuất thông tin.

- **Công cụ so sánh tài liệu:**
  - Các ứng dụng như kiểm tra đạo văn sử dụng LCS để phát hiện nội dung giống nhau giữa các tài liệu. Điều này rất hữu ích trong giáo dục và nghiên cứu.
- **Xây dựng trình biên dịch:**
  - Trong lập trình, LCS có thể được sử dụng để phân tích cú pháp hoặc kiểm tra tương thích giữa các đoạn mã nguồn trong các ngôn ngữ lập trình.
- **Phân tích dữ liệu:**
  - LCS cũng có thể được sử dụng trong phân tích dữ liệu để tìm kiếm các mẫu và xu hướng trong các tập dữ liệu lớn.
- **Thiết kế phần mềm:**
  - Trong phát triển phần mềm, LCS có thể hỗ trợ trong việc so sánh mã nguồn, giúp lập trình viên dễ dàng phát hiện sự khác biệt giữa các phiên bản mã nguồn.

➢ Nhìn chung, LCS là một công cụ mạnh mẽ trong việc xử lý chuỗi và tìm kiếm sự tương đồng, với nhiều ứng dụng trong các lĩnh vực khoa học, công nghệ, và quản lý dữ liệu.

VI. Kết luận

❖ Hiệu quả của quy hoạch động trong LCS

1. Giảm độ phức tạp thời gian

- **Độ phức tạp thời gian** của thuật toán LCS bằng phương pháp quy hoạch động là O(n×m) trong đó nnn và mmm là độ dài của hai chuỗi cần so sánh.

- Nếu không sử dụng quy hoạch động, phương pháp quay lui (brute force) có thể có độ phức tạp lên đến O(2^max(n,m))điều này có nghĩa là nó không khả thi cho các chuỗi có độ dài lớn.

2. Lưu trữ kết quả trung gian

- Quy hoạch động lưu trữ các kết quả trung gian trong một ma trận (hoặc bảng) để tránh việc tính toán lại những giá trị đã biết. Điều này giúp giảm thiểu số lần tính toán và tiết kiệm thời gian.
- Khi một giá trị LCS cho một cặp chỉ số (i, j) đã được tính, nó có thể được sử dụng lại cho các phép so sánh khác, làm tăng hiệu suất tính toán.

3. Giải quyết các bài toán con

- Quy hoạch động chia bài toán lớn thành các bài toán con nhỏ hơn, giải quyết từng bài toán con một cách độc lập và sau đó kết hợp các kết quả lại để tìm ra giải pháp cho bài toán lớn.
- Việc này không chỉ làm cho quá trình tính toán hiệu quả hơn mà còn dễ dàng trong việc theo dõi và kiểm soát quá trình giải quyết bài toán.

4. Tính khả thi với chuỗi lớn

- Do hiệu suất cao và khả năng lưu trữ kết quả trung gian, quy hoạch động cho phép xử lý các chuỗi có độ dài lớn (vài nghìn ký tự) mà vẫn đảm bảo kết quả chính xác trong thời gian hợp lý.
- Điều này làm cho phương pháp quy hoạch động trở thành lựa chọn tối ưu trong các ứng dụng thực tế, nơi dữ liệu thường rất lớn.

5. Dễ dàng mở rộng và điều chỉnh

- Quy hoạch động có thể được điều chỉnh để giải quyết các bài toán tương tự như LCS, chẳng hạn như tìm chuỗi con chung dài nhất với các điều kiện khác nhau (như cho phép xóa ký tự, chèn ký tự, v.v.).
- Phương pháp này dễ dàng áp dụng cho các bài toán khác trong lập trình và nghiên cứu, giúp tăng tính linh hoạt trong việc phát triển các giải pháp.

❖ Tiềm năng tối ưu hoá thêm

1. Giảm kích thước ma trận lưu trữ

- Chỉ sử dụng hai hàng: Thay vì lưu trữ toàn bộ ma trận ccc kích thước $(n+1)\times(m+1)(n+1)$, bạn có thể chỉ cần lưu trữ hai hàng hiện tại (hàng i và hàng $i-1$). Điều này giúp giảm sử dụng bộ nhớ từ $O(n\times m)$ xuống $O(\min(n,m))$.

2. Sử dụng các phương pháp nhớ lại (Memoization)

- Trong các trường hợp khi LCS có thể tính toán lặp lại cho nhiều cặp chỉ số khác nhau, bạn có thể sử dụng phương pháp nhớ lại để lưu trữ kết quả đã tính toán. Điều này có thể giúp giảm độ phức tạp tính toán trong những trường hợp có nhiều ký tự trùng lặp trong chuỗi.

3. Tối ưu hóa so sánh ký tự

- Sử dụng các kỹ thuật như hashing hoặc mapping để tối ưu hóa việc tìm kiếm và so sánh ký tự. Ví dụ, nếu bạn biết rằng các ký tự trong chuỗi có giới hạn nhất định (ví dụ: chỉ chứa chữ cái), bạn có thể sử dụng một bảng băm để theo dõi vị trí và số lượng của từng ký tự, từ đó tối ưu hóa tốc độ tìm kiếm.

4. Thay thế bằng thuật toán kết hợp

- Bạn có thể áp dụng các thuật toán kết hợp (hybrid algorithms) để kết hợp giữa quy hoạch động và các thuật toán khác (như thuật toán quay lui hoặc phân tách và chinh phục) để tận dụng ưu điểm của cả hai phương pháp, đặc biệt là khi xử lý các chuỗi có độ dài lớn.

5. Phân chia và trị liệu

- Nếu chuỗi có thể được chia thành các phần nhỏ hơn mà không làm mất đi tính chất của bài toán LCS, bạn có thể tính toán LCS cho các phần nhỏ trước và sau đó kết hợp kết quả để tìm ra LCS cho chuỗi lớn hơn.

6. Cải thiện khả năng song song

- Nếu xử lý trên hệ thống đa lõi, bạn có thể cải thiện hiệu suất bằng cách phân chia bài toán thành nhiều phần và thực hiện song song. Ví dụ, mỗi lõi có thể xử lý một phần của ma trận LCS.

7. Áp dụng các kỹ thuật học máy

- Trong một số trường hợp, bạn có thể sử dụng các mô hình học máy để dự đoán các phần của chuỗi có khả năng cao nhất sẽ tạo ra LCS, từ đó tối ưu hóa việc tìm kiếm các chuỗi con dài nhất.

8. Sử dụng chuỗi nhị phân hoặc các cấu trúc dữ liệu nâng cao

- Nếu bạn đang làm việc với chuỗi nhị phân (như chuỗi bit), có thể áp dụng các cấu trúc dữ liệu như Trie hoặc Segment Tree để cải thiện hiệu suất tìm kiếm và tính toán

TÀI LIỆU THAM KHẢO

[1] Dawson, Chris (25 tháng 7 năm 2014). “JavaScript's History and How it Led To ReactJS”. The New Stack (bằng tiếng Anh).

[2] React book, your beginner guide to react by Chris Noring (https://softchris.github.io/react-book/#/)

[3] React JS Notes for Professionals book nguồn tổng hợp từ Stack Overflow Documentation (https://goalkicker.com/ReactJSBook/)

[4] Nguồn tài liệu tham khảo từ VIBLO (https://viblo.asia/newest)

[5] Ví dụ tham khảo từ React (https://legacy.reactjs.org)