Toán rời rạc và thuật toán
Đại học Khoa học Tự nhiên
Khoa Toán - Cơ - Tin học
Khoa học dữ liệu K4
Tháng 8 năm 2022

Bài tập số 1

Nguyễn Mạnh Linh, Nguyễn Thị Đông, Triệu Hồng Thúy

1 Bài 1

- **Bài toán** được cấu tạo bởi hai thành phần cơ bản:
  - Thông tin vào (input): Cung cấp cho ta các dữ liệu đã có.
  - Thông tin ra (output): Những yếu tố cần xác định.
- **Thuật toán** là một dãy hữu hạn các thao tác đơn giản được sắp xếp theo một trình tự xác định dùng để giải một bài toán.
- **Giải bài toán bằng thuật toán** là quá trình từ thông tin đầu vào (input), dùng một dãy hữu hạn các thao tác đơn giản được sắp xếp theo một trình tự xác định để tìm được thông tin đầu ra (output) theo yêu cầu của bài toán đề ra.

2 Bài 2

2.1 Chia để trị

2.1.1 Ý tưởng

Phương pháp chia để trị dựa trên 2 thao tác chính:

- Chia (*devide*): phân rã bài toán ban đầu thành các bài toán con có kích thước nhỏ hơn, có cùng cách giải.
- Trị (*conque*): giải từng bài toán con (theo cách tương tự bài toán đầu - đệ qui) rồi tổng hợp các lời giải để nhận kết quả của bài toán ban đầu.

Việc "Phân rã": thực hiện trên miền dữ liệu (chia miền dữ liệu thành các miền nhỏ hơn tương đương 1 bài toán con)

2.1.2 Mô hình và lược đồ

Xét bài toàn P trên miền dữ liệu R.
Gọi $D\_C(R)$ là thuật giải P trên miền dữ liệu R.
Nếu R có thể phân rã thành n miền con: $R = R_1 \cup R_2 \cup ... \cup R_n$
Với R_0 là miền đủ nhỏ để $D\_C(R)$ có lời giải, ta có lược đồ giải thuật chia để trị như sau:

```
Divide_Conque(R):
    if(R = R_0):
        solve Divide_Conque(R_0)
    else
        divide R to R_1, R_2, ..., R_n
        for (i = 1, 2, ..., n):
            Divide_Conque(R_i)
        Combine and get result
end
```

2.1.3 Phân tích và đánh giá

Để phân tích và đánh giá độ phức tạp của thuật toán, ta thực hiện 2 công đoạn

- Xây dựng công thức truy hồi đánh giá độ phức tạp thuật toán
- Giải công thức truy hồi xác định độ phức tạp thuật toán.
  - Phép thế liên tiếp
  - Sử dụng định lí chính

2.1.4 Ví dụ

Ta xét bài toán *tìm kiếm nhị phân trên một mảng được sắp xếp.*

- Cho dãy n phần tử được sắp theo thứ tự (*tăng dần*) và một giá trị x bất kỳ. Kiểm tra xem phần tử x có trong dãy không?
- Phân tích ý tưởng: so sánh giá trị x với phần tử giữa của dãy tìm kiếm. Dựa vào giá trị này sẽ quyết định giới hạn tìm kiếm ở bước kế tiếp là nửa trước hay nửa sau dãy.
- Lược đồ của thuật toán như sau:

```
BinarySearch(a, x, L, R):
    // Search element x in array a from position L to R
    if (L == R):
        return (x == a_L ? L : -1)
    else
        M = (L + R)/2
        if (x == a_M)
            return (M)
```

```
            else
                if (x < a_M)
                    BinarySearch(a, x, L, R)
                else
                    BinarySearch(a, x, M + 1, R)
                endif
            endif
        endif
    end
```

Tính đúng của thuật toán
Ta chứng mình bằng quy nạp như sau

- Cơ sở quy nạp: $n = R - L + 1 = 1$ (dãy có 1 phần tử)
  - Câu lệnh `return (x = a_L ? L : -1)` trả về giá trị L hoặc -1
- Giả thiết quy nạp: Thuật toán đúng với mọi dãy có độ dài $n = R - L + 1$. Hay hàm `BinarySearch(a, x, L, R)` trả về đúng kết quả tìm kiếm x với mọi dãy có đội dài $1 \leq n' \leq n = R - L + 1$
- Tổng quát: Chứng minh thuật toán đúng với $n + 1 = R - L + 2$
  - Đặt $M = (L + R + 1)/2$, ta có $L \leq M \leq R$
  - Nếu $x = a_M$ thì kết quả trả về là M: đúng
  - Nếu $x < a_M$ thì kết quả là của bài toán tìm x trong tập $a_L, ..., a_M$. Theo giả thiết quy nạp thì `BinarySearch(a, x, L, R)` đúng vì $1 \leq M - L + 1 = (R - L + 1)/2 + 1 \leq R - L + 1$
  - Tương tự với $x > a_M$

Độ phức tạp của thuật toán

$$T(n) = \begin{cases} 1 & \text{when } n = 1 \\ T(n/2) + 1 & \text{when } n > 1 \end{cases}$$

Do đó $T(n) = O(\log n)$
python code cho bài toán.

2.2 Quay lui

2.2.1 Ý tưởng

Ý tưởng của phương pháp Quay lui (Back tracking): Theo nguyên tắc vét cạn (xét qua tất cả các trường hợp có thể xảy ra để tìm kết quả), nhưng chỉ xét những trường hợp "khả quan".
Tại mỗi bước, nếu có một lựa chọn được chấp thuận thì ghi nhận lại lựa chọn này và tiến hành các bước thử tiếp theo. Còn ngược lại không có lựa chọn nào thích hợp thì quay lại bước trước.
Phương pháp quay lui được sử dụng để giải bài toán liệt kê các cấu hình:

- Mỗi cấu hình được xây dựng bằng cách xác định từng phần tử
- Mỗi phần tử được chọn bằng cách thử các khả năng ***có thể***
- Độ dài cấu hình tùy thuộc bài toán
  - Xác định trước: sinh dãy độ dài n
  - Không xác định trước: đường đi

2.2.2 Mô hình

Không gian nghiệm của bài toán (tập khả năng) $X = \{(x_1, x_2, \ldots, x_n)\}$ gồm các cấu hình liệt kê có dạng $(x_1, x_2, \ldots, x_n)$ cần được xây dựng:

- Cho x_1 nhận lần lượt các giá trị có thể. Với mỗi giá trị thử gán cho x_1 thì:
- Cho x_2 nhận lần lượt các giá trị có thể. Với mỗi giá trị thử gán cho x_2 thì xét khả năng chọn $x_3, \ldots, x_n$, => cấu hình tìm được $(x_1, x_2, \ldots\ldots, x_n)$.
- **Tổng quát:** Tại mỗi bước i: Xây dựng thành phần x_i
  - Xác định x_i theo khả năng v.
  - Nếu gặp điều kiện dừng (i = n hoặc x_i thỏa mãn điều kiện dừng) thì ta có được một lời giải, ngược lại thì tiến hành bước i+1 để xác định x_{i+1}
  - Nếu không có một khả năng nào chấp nhận được cho x_i thì lại lùi lại bước trước để xác định lại thành phần x_{i-1} – sự quay lui

2.2.3 Lược đồ

Lược đồ phương pháp quay lui như sau:

```
Try(i) ≡ //Sinh thành phần thứ i của cấu hình
    for (v thuộc tập khả năng thành phần nghiệm x_i)
        if (x_i chấp nhận được giá trị v)
            x_i = v;
            <Ghi nhận trạng thái chấp nhận v>;
            if (gặp điều kiện dừng) //i=n hoặc x_i thỏa mãn điều kiện dừng
                <xử-lí-nghiệm>;
            else      //lời gọi sinh thành phần tiếp theo của cấu hình
                Try (i + 1)
            <Khôi phục trạng thái chưa chấp nhận v>;
        endif;
    endfor;
End.                                   Lời gọi ban đầu: Try(1)
```

2.2.4 Phân tích và đánh giá

Ta thực hiện 2 công đoạn phân tích và đánh giá thuật toán như sau:

- Công thức truy hồi

$$T(n) = \begin{cases} 1 & \text{khi} \quad n \leq 1 \\ dT(n-1)+1 & \text{khi} \quad n > 1 \end{cases}$$

  d: max (hoặc giá trị trung bình) lực lượng của tập khả năng của các thành phần nghiệm x_i

- Giải công thức truy hồi: $T(n) = O(d^n)$
  - Là trường hợp xấu nhất (vét cạn)
  - Trong triển khai thuật toán thời gian thực tế có thể giảm xuống

    for (v thuộc tập khả năng thành phần nghiệm x_i)

    if (x_i chấp nhận được giá trị v)

Tính đúng của thuật toán: Sự hiển nhiên trong tính đúng của thuật toán quay lui:

- Mỗi phần tử trong cấu hình nghiệm được chọn trong tập khả năng

  for (v thuộc tập khả năng thành phần nghiệm x_i)

  if (x_i chấp nhận được giá trị v)

  x_i = v;

- Tính dừng của lời gọi đệ quy và cho ra kết quả đúng

  if (gặp điều kiện dừng) *//i=n hoặc xi thỏa mãn đk dừng*

  <xử-lí-nghiệm>;

  else *//lời gọi sinh thành phần tiếp theo của cấu hình*

  Try (i + 1)

- Đệ quy tiến Try(1)/x_1 → Try(2)/x_2 → ... → Try(m)/x_m
  - Đủ thành phần nghiệm m=n (n - hữu hạn)
  - Đến thành phần thỏa mãn điều kiện tức là x_m thỏa mãn điều kiện nghiệm

2.2.5 Ví dụ

Ta xét bài toán *Liệt kê dãy k_ ary có độ dài n*

- Phân tích
  - Input: n, k
  - Output: Các dãy X = $(x_1, x_2, \ldots, x_n)$ trong đó x_i =0,1,..k-1
  - Dùng giải thuật Try(i) để sinh giá trị x_i
  - Nếu i=n thì in giá trị nghiệm X, ngược lại sinh tiếp x_{i+1} bằng Try(i+1)

- Lược đồ giải thuật:

```
Try(i) ≡
    for (v=0..k-1) //v nhận giá trị từ 0 đến k-1
        x_i = v ;
        if (i=n) printResult (X) ;
        else Try(i+1) ;
    endfor;
End.                                    Lời gọi ban đầu Try(1)
```

python code cho bài toán

2.3 Quy hoạch động

2.3.1 Ý tưởng

- Quy hoạch động (Dynamic programing) là phương pháp giải các bài toán bằng cách kết hợp lời giải của các bài toán con theo nguyên tắc:
  - Giải tất cả các bài toán con (một lần)
  - Lưu lời giải của các bài toán vào một bảng
  - Phối hợp các bài toán con để nhận lời giải bài toán ban đầu
- Cách phát biểu khác: Một bài toán giải bằng quy hoạch động được phân rã thành các bài toán con và bài toán lớn sẽ được giải quyết thông qua các bài toán con này (bằng các phép truy hồi)
- Phương pháp quy hoạch động thường được dùng cho các bài toán tìm giá trị (hoặc giá trị tối ưu)

2.3.2 Các bước giải bài toán

1. Nhận dạng bài toán

   Các bài toán có các đặc trưng sau thì có thể giải bằng QHĐ
   - Bài toán có thể giải bằng đệ qui và tìm lời giải tối ưu.
   - Bài toán có thể phân rã thành nhiều bài toán con mà sự phối hợp lời giải của các bài toán con sẽ cho ta lời giải của bài toán ban đầu.
   - Quá trình tìm ra lời giải của bài toán ban đầu từ các bài toán con đơn giản hơn được thực hiện qua một số hữu hạn các bước có tính truy hồi.
2. Xây dựng công thức truy hồi
   - Đưa bài toán về 1 dạng cơ bản, và triển khai ý tưởng của dạng bài toán đó để nhanh chóng nhận ra hướng thiết lập công thức truy hồi.
   - Đây là bước khó nhất và cũng quan trọng nhất trong toàn bộ quá trình thiết kế thuật toán cho bài toán.

3. Xác định và xây dựng cơ sở QHĐ
   - Dựa vào công thức truy hồi để nhận ra các bài toán cơ sở.
   - Dựa vào ý nghĩa của công thức truy hồi để thiết lập giá trị cho cơ sở.
4. Dựng bảng phương án
   - Dựa vào công thức truy hồi để tính giá trị các ô trong bảng phương án.
   - Chú ý: bảng phương án có thể 1 chiều, 2 chiều hoặc nhiều hơn
5. Tìm kết quả tối ưu
   - Xác định vị trí chứa kết quả tối ưu của bài toán trên bảng phương án.
   - Chú ý: ngoài kết quả tối ưu, ô chứa kết quả tối ưu còn là điểm bắt đầu cho quá trình truy vết tìm nghiệm => lưu tọa độ của ô đó.
6. Truy vết liệt kê thành phần nghiệm
   - Từ điểm bắt đầu là vị trí chứa kết quả tối ưu
   - Truy ngược lại về điểm bắt đầu của nghiệm: có thể là những ô đầu tiên trong bảng phương án (bài toán cơ sở), có thể là ô của bảng phương án đạt giá trị đầu.

2.3.3 Ví dụ

1. Bài toán "Dãy con đơn điệu tăng dài nhất"

   Bài toán: Tìm dãy con dài nhất của một dãy đã cho. Các phần tử có thể không liên tiếp.

   Phân tích:
   - Input: (a,n)
   - Output: số lớn nhất các phần tử của dãy theo thứ tự tăng dần
   - Hàm tối ưu L(i): Độ dài dãy con đơn điệu tăng dài nhất đến phần tử i
     Là độ dài các dãy con dài nhất đến j cộng 1 khi ghép thêm ai vào sau, với điều kiện j<i, aj<ai
   - Công thức truy hồi: L(i) = maxL(j)+1 với j<i, aj<ai
   - Cơ sở của thuật toán: L(0) = 0; L(1) = 1

```
def lis(_list):
    longest = [_list[0]]
    current = [_list[0]]
    for i in _list[1:]:
        if i >= current[-1]:
            current.append(i)
        else:
            if len(longest) < len(current):
                longest = current
            current = [i]
    if len(longest) < len(current):
```

```
                longest = current
        return longest
```

2. Bài toán "Xếp balo 0-1"

   Bài toán: Có N đồ vật với trọng lượng và giá trị tương ứng(wi,vi). Tìm cách cho các vật vào balo có trọng lượng W sao cho đạt giá trị cao nhất. Mỗi vật chỉ được chọn 1 lần

   Phân tích:

   - Input: n đồ vật, (wi,vi) i=1..n, Túi có trọng lượng tối đa W
   - Output: V=tổng giá trị lớn nhất của các đồ vật vào balo
   - Hàm tối ưu dp[i,j]: Giá trị lớn nhất khi chọn đồ vật từ 1 tới i với trọng lượng balo j (i=0...n,j=0...W)
     Nếu không chọn đồ vật thứ i thì: dp[i,j]=dp[i-1,j]
     Nếu chọn đồ vật thứ i thì: dp[i,j]=dp[i-1,j-wi]+vi (Điều kiện wi<=j)
   - Công thức truy hồi:
     dp[i,j]=Max(dp[i-1,j], dp[i-1,j-wi]+vi)
   - Cơ sở quy hoạch động
     dp[i,0]=0
     dp[0,j]=0

```
def solve(weights, values, capacity):
    items = list(zip(weights, values))
    def dp(i, cp):
        if i == len(items):
            return 0.0
        w, v = items[i]
        ans = dp(i + 1, cp)
        if cp >= w:
            ans = max(ans, dp(i + 1, cp - w) + v)
        return ans
    return int(dp(0, capacity))
```

3 Bài 3

Trong bài này chúng ta sẽ xem xét bài toán dóng hàng toàn cục 2 chuỗi DNA sử dụng phương pháp quy hoạch động với giải thuật Needleman-Wunsch.

3.1 Bài toán dòng hàng toàn cục

Có nhiều cách để phát biểu bài toán dóng hàng trình tự (cho DNA, RNA hoặc 2 chuỗi kí tự). Hãy cùng điểm qua vài khái niệm.

- Dóng hàng trình tự: là một cách sắp xếp trình tự của DNA, RNA hoặc protein để xác định các vùng giống nhau có thể là hệ quả của mối quan hệ chức năng, cấu trúc hoặc tiến hóa giữa các trình tự.
- Một cách dóng hàng của 2 chuỗi được thiết lập bằng cách thêm các khoảng trống (dấu cách) vào các vị trí bất kì trên 2 chuỗi này để chúng có cùng độ dài và không có 2 khoảng trống nào có cùng vị trí trên 2 chuỗi
- Chuỗi con chung dài nhất (longest common subsequence, LCS): là chuỗi trình tự chứa nhiều kí tự giống nhau nhất của hai hay nhiều chuỗi.

Ví dụ một cách dóng hàng với 2 chuỗi S (`interestingly`) và T (`bioinformatics`) như sau

```
-i--nterestingly
bioinformatics--
```

Một cách tổng quát, chúng ta có thể chấm điểm cho mỗi cặp kí tự được dóng hàng. Gọi Σ là tập các kí tự và "-" là kí tự đặc biệt kí hiệu cho dấu cách. Sự tương tự của các cặp kí tự trong 2 chuỗi có thể được biểu diễn thông qua một ma trận δ mà $\delta(x, y)$ là điểm của cặp x và y với $x, y \in \Sigma \cup \{-\}$.
Bài toán dóng hàng toàn cục có thể mô hình hóa bằng cách tìm một các dóng hàng A nào đó để cực đại $\sum_{\{x,y\}\in A} \delta(x, y)$. Cách dóng hàng này được gọi là cách tối ưu (*optimal alignment*).
Khi một cặp kí tự được sắp xếp và giống nhau, liên hệ giữa chúng được gọi là *match*, ngược lại gọi là *mismatch*. Khi dấu cách được thêm vào chuỗi thứ nhất mối liên hệ là *insert*, khi được thêm vào chuỗi thứ 2 thì được gọi là *delete*.
Trong ví dụ trên chúng ta có 5 *matches*, 6 *mismatchs*, 3 *inserts* và 2 *deletes*.
Chúng ta cùng xem xét mooyj ví dụ về dóng hàng toàn cục. Xét ma trận điểm cho tập $\{-, A, C, G, T\}$ với $\delta(x, y) = 2, -1, -1, -1$ lần lượt cho match, mismatch, delete và insert.
Xét 2 chuỗi DNA $S = ACAATCC$ và $T = AGCATGC$, một cách dóng hàng khả dĩ như sau:

```
S = A-CAATCC
T = AGCA-TGC
```

Cách dóng hàng trên có 5 matches, 1 mismatches, 1 insert và 1 delete. Vậy điểm tương tự của cách dòng hàng này là 7. Có thể kiểm tra được đây là điểm cực đại thế nên cách dóng hàng này là một cách tối ưu. Cần chú ý là có thể có nhiều hơn một cách dóng hàng tối ưu. Ví dụ 1 cách khác cũng trả về điểm tương tự cực đại.

```
S = A-CAATCC
T = AGC-ATGC
```

	-	A	C	G	T
-		-1	-1	-1	-1
A	-1	2	-1	-1	-1
C	-1	-1	2	-1	-1
G	-1	-1	-1	2	-1
T	-1	-1	-1	-1	2

Hình 1: Ví dụ ma trận điểm tương tự

3.2 Giải thuật Needleman-Wunsch

3.2.1 Giải thuật

Xét 2 chuỗi kí tự $S[1...n]$ và $T[1...m]$. Để tìm các dóng hàng toàn cục tối ưu cho bài toán này chúng ta có thể nghĩ tới giải thuật vét cạn bằng cách sinh ra mọi cách dóng hàng và tìm xem cách nào có điểm cao nhất. Tuy nhiên cách tiếp cận này có thời gian tính toán theo hàm mũ. Trong phần này, chúng ta sẽ xem xét một giải thuật hiệu quả áp dụng *quy hoạch đông* có tên là *Needleman-Wunsch*.
Chúng ta thiết kế một hàm đệ quy (công thức truy hồi) $V(i,j)$ với 2 trường hợp: (1) $i=0$ hoặc $j=0$ và (2) cả $i>0$ và $j>0$.
Với trường hợp (1) khi $i=0$ hoặc $j=0$ chúng ta dóng hàng chuỗi bằng 1 chuỗi rỗng. Hay nói cách khác là ta chỉ cần thêm hoặc xóa kí tự. Chúng ta có các phương trình:

$$\begin{aligned} V(0,0) &= 0 \\ V(0,j) &= V(0,j-1) + \delta(-,T[j]) \quad \text{thêm j lần} \\ V(i,0) &= V(i-1,0) + \delta(S[i],-) \quad \text{xóa i lần} \end{aligned} \tag{1}$$

Với trường hợp (2) khi cả $i>0$ và $j>0$, ta thấy rằng với một cách dóng hàng nào đó của chuỗi $S[1...i]$ và $T[1...j]$ cặp kí tự cuối cùng phải thuộc 1 trong 3 loại match/mismatch (cả 2 đều là kí tự), insert hoặc delete. Để thu được điểm tối ưu ta chọn cách có điểm cực đại trong 3 trường hợp trên. Nghĩa là:

$$V(i,j) = \max \begin{cases} V(i-1,j-1) + \delta(S[i],T[j]) & \text{match/mismatch} \\ V(i-1,j) + \delta(S[i],-) & \text{delete} \\ V(i,j-1) + \delta(-,T[j]) & \text{insert} \end{cases} \tag{2}$$

Điểm dóng hàng tối ưu của $[1...n]$ và $T[1...m]$ là $V(n,m)$. Điểm này có thể được tính bằng cách điền đầy từng hàng vào bảng $V(1...n,1...m)$ bằng các công thức truy hồi trên.

	_	A	G	C	A	T	G	C
_	0	-1	-2	-3	-4	-5	-6	-7
A	-1	2	1	0	-1	-2	-3	-4
C	-2	1	1	3	2	1	0	-1
A	-3	0	0	2	5	4	3	2
A	-4	-1	-1	1	4	4	3	2
T	-5	-2	-2	0	3	6	5	4
C	-6	-3	-3	0	2	5	5	7
C	-7	-4	-4	-1	1	4	4	7

Hình 2: Bảng điểm dóng hàng 2 chuỗi $S = ACAATCC$ và $T = AGCATGC$ bằng giải thuật quy hoạch động Needleman-Wunsch

Hình trên là bảng V cho 2 chuỗi $S = ACAATCC$ và $T = AGCATGC$. Chúng ta có thể tính lần lượt giá trị của các ô trong bảng, ví dụ $V(1,1)$ là giá trị lớn nhất trong tập $\{0+2, -1-1, -1-1\}$. Sau khi điền đầy các ô ta có $V(7,7)=7$ là giá trị của ô dưới cùng bên phải hay chính là điểm của cách dóng hàng tối ưu.

Để tìm ngược lại cách dóng hàng tối ưu, với mỗi ô trong bảng ta vẽ các mũi tên để biểu diễn sự tương quan giữa các cặp. Ta vẽ mũi tên chéo, ngang và dọc lần lượt cho các trường hợp match, insert và delete. Ví dụ giá trị tại $V(1,1)$ thu được bởi $V(0,0)+2$, chúng ta vẽ 1 mũi tên chéo. Với ô $V(3,2)$, có 2 cách có giá trị điểm bằng điểm lớn nhất nên ta vẽ cả 2 mũi tên chéo và dọc. Để thu được cách dóng hàng tối ưu ta cần quay lui từ ô $V(7,7)$ về ô $V(0,0)$. Nếu mũi tên là chéo, 2 kí tự được dóng hàng. Với mũi tên ngang và dọc thì lần lượt là delete và insert. Ta thu được đường quay lui là đường mũi tên in đậm trong hình 2 theo đường $7 \to 5 \to 6 \to 4 \to 5 \to 3 \to 1 \to 0$. Tương ứng ta thu được cách dóng hàng tối ưu:

```
A-CAATCC
AGCA-TGC
```

3.2.2 Thời gian tính toán

Như đã trình bày trong phần trước, thuật toán Needleman-Wunsch có thời gian tính toán là $O(nm)$ khi giải bài toán dóng hàng toàn cục do chúng ta cần điền vào toàn bộ các ô trong bảng điểm dòng hàng có kích thước $n \times m$

3.3 Triển khai thuật toán

Trong phần này chúng ta sẽ đưa ra mã giả cài đặt thuật toán Needleman-Wunsch để lập bảng điểm dóng hàng và trình bày cách quay lui để đưa ra kết quả dóng hàng 2 chuỗi DNA.

Thay vì gọi là *bảng* như phần trước, chung ta sẽ xây dựng một *ma trận* V gọi là ma trận *score*. Ở đây ta kí hiệu $v[i,j]$ là thành phần $v_{i,j}$ của ma trận V, $s[i]$ là kí tự tại chỉ số i của chuỗi s (lưu ý chỉ số đánh từ 0).

Trước hết là mã giả để xây dựng ma trận V hay nói cách khác là điền đầy bảng điểm dóng hàng của 2 chuỗi s và t.

```
// alias for backtracking arrows
left, up, dia = 1, 2, 3

// define score for characters status
match, mismatch, insert, delete = 2, -1, -1, -1

delta(a, b):
    // define score
    if a == b:
        return match
    if '-' == a:
        return insert
    if '-' == b:
        return delete
    return mismatch
end

node_score(s, t, node, i, j):
    // calculate score of node[i, j]
    score_match_mismatch = node[i-1, j-1] \
```

```
            + delta(s[i], t[j])
        score_delete = node[i-1, j] + delta(s[i], '-')
        score_insert = node[i, j-1] + delta(s[i], '-')
        return max([score_match_mismatch,
                    score_delete,
                    score_insert])
    end


    lsc(s, t):
        // fill the score table
        n = length of s
        m = length of t
        backtrack = zero matrix with size (n+1) × (m+1)
        v = zero matrix with size (n+1) × (m+1)
        s = '-' + s
        t = '-' + t

        for i from 0 to n:
            v[i, 0] = -i

        for j from 0 to m:
            v[0, j] = -j

        for i from 1 to n:
            for j from 0 to m:
                v[i, j] = node_score(s, t, v, i, j)
                if v[i, j] == v[i-1, j] + delta(s[i], '-'):
                    backtrack[i, j] = up
                elif v[i, j] == v[i, j-1] + delta('-', t[j]):
                    backtrack[i, j] = left
                else:
                    backtrack[i, j] = dia

        return v, backtrack
    end
```

Tiếp theo ta có mã giả cho quá trình quay lui để đưa ra kết quả dòng hàng

```
    output(s, t, backtrack):
        // return 2 aligned sequences
        n = length of s
        m = length of t
        source, target = s, t
        i, j = n, m
        while i > 0 and j > 0:
            if backtrack[i, j] == left:
                inserting '-' to source at index (j − 1)
                j = j - 1
                continue
            if backtrack[i, j] == up:
```

```
                inserting '-' to target at index (i - 1)
                i = i - 1
                continue
            i = i - 1
            j = j - 1
        return source, target
    end
```

Cuối cùng là python code cài đặt thuật toán.