

Bài tập thực hành-Khai thác dữ liệu-tuần 3

Phan Hồng Trâm - 21110414

April 2024

Mục lục

- 1 Tóm lượt lại phần code em đã làm trong mục 2: Khoảng cách dãy con chung dài nhất (Longest Common Subsequence-LCSS) 2
- 2 Tóm lượt lại phần code em đã làm trong mục 3: Khoảng cách biến đổi thời gian động (Dynamic Time Warping- DTW) 5

1 Tóm lượt lại phần code em đã làm trong mục 2: Khoảng cách dãy con chung dài nhất (Longest Common Subsequence-LCSS)

```
def lcsm_distance(X, Y):
    m = len(X)
    n = len(Y)

    # Create a DP table to store the lengths of LCS for prefixes of X and Y
    L = [[None]*(n+1) for i in range(m+1)]
    # Fill the first row and column of LCS with 0
    for i in range(m+1):
        L[i][0] = 0
    for j in range(n+1):
        L[0][j] = 0

    # Build the DP table
    for i in range(m + 1):
        for j in range(n + 1):
            # Compare X[i-1] and Y[j-1]
            # Start from LCS[1][1]
            if i == 0 or j == 0:
                L[i][j] = 0
            elif X[i - 1] == Y[j - 1]:
                L[i][j] = L[i - 1][j - 1] + 1
                # Point an arrow to LCS[i][j]
            else:
                L[i][j] = max(L[i - 1][j], L[i][j - 1])
                # Point an arrow to max(LCS[i-1][j], LCS[i][j-1])

    # Backtrack to find the LCSS
    lcsm = ""
    i = m
    j = n
    while i > 0 and j > 0:
        if X[i - 1] == Y[j - 1]:
            lcsm = X[i - 1] + lcsm
            i -= 1
            j -= 1
        else:
            if L[i - 1][j] > L[i][j - 1]:
                i -= 1
            else:
                j -= 1
    # L[m][n] contains the length of LCS of X[0..n-1] & Y[0..m-1]
    return L[m][n], lcsm

if __name__ == "__main__" :

    print("Enter the X string:")
    X = input().strip()
    print("Enter the Y string: ")
    Y = input().strip()

    distances, LCSS= lcsm_distance(X,Y)
    print("Distance : {}".format(distances))
    print("LCSS = {}".format(LCSS))
```

1 TÓM LƯỢC LẠI PHẦN CODE EM ĐÃ LÀM TRONG MỤC 2: KHOẢNG CÁCH DẪY CON CHUNG DÀI NHẤT (LONGEST COMMON SUBSEQUENCE-LCSS)

Giải thích code:

1. `def lcsm_distance(X, Y):` hàm có tên `lcsm_distance` nhận hai chuỗi `X` và `Y` làm đầu vào.
2. `m=len(X)` và `n=len(Y)` là chiều dài của 2 chuỗi `X` và `Y`.
3. `L = [[None]*(n+1) for i in range(m+1)]`: Dòng này tạo một bảng DP (Dynamic Programming) `L` có kích thước $(m+1) \times (n+1)$, các giá trị ban đầu đều bằng 0.
4. Điền giá trị 0 vào hàng và cột đầu tiên của `L`

```
# Fill the first row and column of LCS with 0
for i in range(m+1):
    L[i][0] = 0
for j in range(n+1):
    L[0][j] = 0
```

5. Bắt đầu tính toán giá trị của bảng `L`:

```
for i in range(m + 1):
    for j in range(n + 1):
        # Compare X[i-1] and Y[j-1]
        # Start from LCS[1][1]
        if i == 0 or j == 0:
            L[i][j] = 0
        elif X[i - 1] == Y[j - 1]:
            L[i][j] = L[i - 1][j - 1] + 1
            # Point an arrow to LCS[i][j]
        else:
            L[i][j] = max(L[i - 1][j], L[i][j - 1])
            # Point an arrow to max(LCS[i-1][j], LCS[i][j-1])
```

- (a) Dùng 2 vòng lặp để duyệt qua các phần tử của `L`, bắt đầu từ `L[1][1]` đến `L[m][n]`
- (b) `if i == 0 or j == 0`: Điều kiện này kiểm tra xem chúng ta đang ở hàng đầu tiên (`i=0`) hay cột đầu tiên (`j=0`) của bảng. Trong trường hợp này, độ dài LCS luôn bằng 0.
- (c) `elif X[i - 1] == Y[j - 1]`: Điều kiện này kiểm tra xem các ký tự tại chỉ số `i-1` trong `X` và `j-1` trong `Y` có bằng nhau hay không. Nếu bằng nhau thì độ dài LCS tại `L[i][j]` bằng độ dài LCS tại `L[i-1][j-1] + 1` (xem xét các ký tự trước đó)
- (d) `else`: Nếu các ký tự không bằng nhau, thì độ dài LCS tại `L[i][j]` bằng giá trị lớn nhất từ các hàng/cột trước đó, nghĩa là `L[i][j]=max(L[i - 1][j], L[i][j - 1])`.

6. Tìm LCSS:

```
lcsm = ""
i = m
j = n
while i > 0 and j > 0:
    if X[i - 1] == Y[j - 1]:
        lcsm = X[i - 1] + lcsm
        i -= 1
        j -= 1
    else:
        if L[i - 1][j] > L[i][j - 1]:
            i -= 1
        else:
            j -= 1
```

1 TÓM LƯỢC LẠI PHẦN CODE EM ĐÃ LÀM TRONG MỤC 2: KHOẢNG CÁCH DÂY CON CHUNG DÀI NHẤT (LONGEST COMMON SUBSEQUENCE-LCSS)

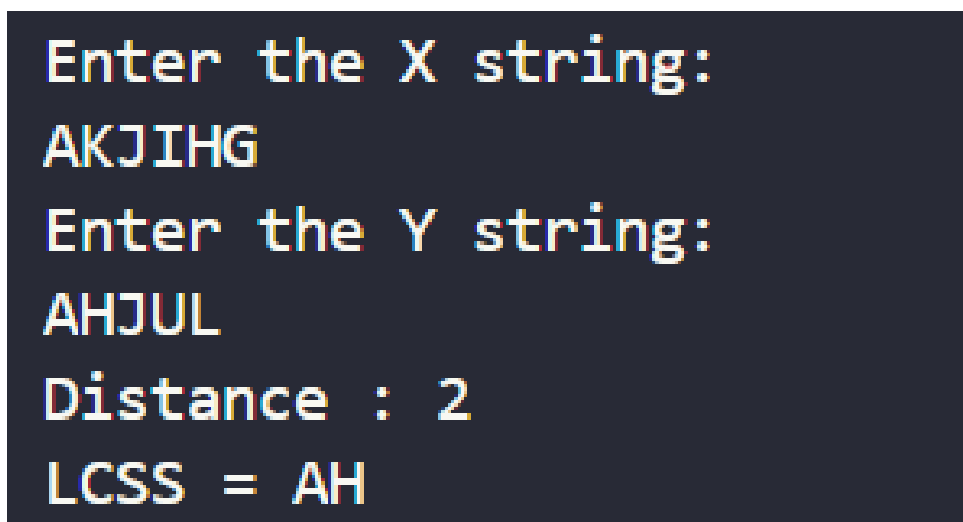
- (a) `lcscs = ""` khởi tạo một chuỗi rỗng `lcscs` để lưu trữ các ký tự LCSS thực tế.
 - (b) `while i > 0 and j > 0`: điều kiện vào vòng lặp, vòng lặp vẫn chạy miễn giá trị của `i` và `j` lớn hơn không (nói cách khác là chưa đến đầu chuỗi).
 - (c) `if X[i - 1] == Y[j - 1]`: kiểm tra xem các ký tự tại chỉ số `i-1` trong `X` và `j-1` trong `Y` có bằng nhau không. Nếu bằng nhau thì vị trí `[i-1]` trong chuỗi `X` được thêm vào chuỗi `lcscs`, nghĩa là `lcscs = X[i-1]+lcscs`.
`i -= 1` và `j -= 1` giảm `i` và `j` để di chuyển về các ký tự trước đó trong chuỗi `X` và `Y`.
 - (d) `else`: nếu các ký tự không bằng nhau, chúng ta so sánh giá trị `L` tại vị trí `L[i-1][j]` và `L[i][j-1]`, và di chuyển sang ô có giá trị `L` lớn hơn để duyệt.
Hai điều kiện bên trong `else` kiểm tra độ dài LCS nào trước đó (`L[i - 1][j]` hoặc `L[i][j - 1]`) lớn hơn và giảm chỉ số `i` hoặc `j` tương ứng.
7. Cuối cùng hàm trả về `L[m][n]` chứa độ dài LCS của chuỗi `X` và `Y` và `lcscs` chứa chuỗi LCSS thực tế.

```
return L[m][n], lcscs
```

8. Ta nhập chuỗi, chạy chương trình và như in ra kết quả.

```
if __name__ == "__main__" :  
  
    print("Enter the X string:")  
    X = input().strip()  
    print("Enter the Y string: ")  
    Y = input().strip()  
  
    distances, LCSS= lcscs_distance(X,Y)  
    print("Distance : {}".format(distances))  
    print("LCSS = {}".format(LCSS))
```

9. Ví dụ: Ta chạy chương trình nhập vào 2 chuỗi `X='AKJHIG'` và `Y=AHJUL`:



```
Enter the X string:  
AKJHIG  
Enter the Y string:  
AHJUL  
Distance : 2  
LCSS = AH
```

Hình 1: Kết quả khi chạy chương trình tìm khoảng cách dây con chung dài nhất

2 Tóm lược lại phần code em đã làm trong mục 3: Khoảng cách biến đổi thời gian động (Dynamic Time Warping- DTW)

```
import numpy as np

def dtw(x, y):

    m, n = len(x), len(y)
    dtw = np.zeros((m + 1, n + 1)) # Khoi tao ma tran chi phi rong

    # Tinh chi phi
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            cost = abs(x[i-1] - y[j-1])
            dtw[i, j] = cost + min(dtw[i-1, j], dtw[i, j-1], dtw[i-1, j-1])

    path = []
    i, j = m, n

    while i > 0 or j > 0:
        path.append(dtw[i, j])
        if i == 1 and j == 1:
            break
        elif i == 1:
            j -= 1
        elif j == 1:
            i -= 1
        else:
            if dtw[i-1, j] == min(dtw[i-1, j-1], dtw[i-1, j], dtw[i, j-1]):
                i -= 1
            elif dtw[i, j-1] == min(dtw[i-1, j-1], dtw[i-1, j], dtw[i, j-1]):
                j -= 1
            else:
                i -= 1
                j -= 1

    # Tao chuoai duong i wrapping
    path_str = ""

    for p in path:
        path_str += f"{p} -> "
    path_str = path_str[:-4]
    print("Warping Path:", path_str)
    return dtw[m, n]

if __name__ == "__main__" :

    # Nhap vao hai chuoai thoi gian x va y
    x = input("Input the time series x: ").split()
    x = [float(i) for i in x]
    y = input("Input the time series y: ").split()
    y = [float(i) for i in y]

    # Tinh va in ra khoang cach DTW giua hai chuoai thoi gian x va y
    distance = dtw(x, y)
    print("Dynamic Time Warping- DTW is:", distance)
```

2 TÓM LƯỢC LẠI PHẦN CODE EM ĐÃ LÀM TRONG MỤC 3: KHOẢNG CÁCH BIẾN ĐỔI THỜI GIAN ĐỘNG (DYNAMIC TIME WARPING- DTW)

Giải thích code:

1. `def dtw(x, y)`: Hàm nhận hai chuỗi thời gian `x` và `y` làm đầu vào và tính toán khoảng cách DTW giữa chúng.
2. `m, n = len(x), len(y)`: `m, n` lần lượt là các biến lưu trữ độ dài của `x` và `y`
3. `dtw = np.zeros((m + 1, n + 1))`: khởi tạo ma trận chi phí có kích thước $(m + 1) \times (n + 1)$, ban đầu tất cả các giá trị trong ma trận đều được khởi tạo bằng 0.
4. Tính ma trận chi phí:

```
for i in range(1, m + 1):
    for j in range(1, n + 1):
        cost = abs(x[i-1] - y[j-1])
        dtw[i, j] = cost + min(dtw[i-1, j], dtw[i, j-1], dtw[i-1, j-1])
```

- (a) Dùng hai vòng lặp để lặp qua các phần tử ($i=1$ đến m và $j=1$ đến n) để tính toán ma trận `dtw` theo công thức:

`dtw[i, j] = cost + min(dtw[i-1, j], dtw[i, j-1], dtw[i-1, j-1])`, với `cost` là khoảng cách giữa hai điểm tương ứng trong hai chuỗi, được tính bằng công thức `cost = abs(x[i-1] - y[j-1])`.

5. Tạo danh sách `path` để lưu trữ các giá trị trong đường đi wrapping:

```
path = []
i, j = m, n

while i > 0 or j > 0:
    path.append(dtw[i, j])
    if i == 1 and j == 1:
        break
    elif i == 1:
        j -= 1
    elif j == 1:
        i -= 1
    else:
        if dtw[i-1, j] == min(dtw[i-1, j-1], dtw[i-1, j], dtw[i, j-1]):
            i -= 1
        elif dtw[i, j-1] == min(dtw[i-1, j-1], dtw[i-1, j], dtw[i, j-1]):
            j -= 1
        else:
            i -= 1
            j -= 1
```

Ở mỗi bước lặp: `While i>0 or j>0`:

- (a) Chi phí hiện tại `dtw[i, j]` được thêm vào danh sách `path`.
- (b) `if i==1 and j==1`: vòng lặp sẽ ngắt khi chúng ta đạt đến điểm bắt đầu.
- (c) `elif`:
 - i. `if i==1`: (ở hàng đầu tiên) thì ta chỉ có thể di chuyển chéo lên trên (xóa ở `y`). Vì vậy, `j` bị giảm đi (`j -= 1`).
 - ii. `if j==1` (ở cột đầu tiên) thì chúng ta chỉ có thể di chuyển chéo sang trái (chèn vào `x`). Vì vậy, `i` bị giảm đi (`i -= 1`).
- (d) `else`: khi cả `i` và `j` đều lớn hơn 1, chúng ta cần chọn hướng có chi phí tối thiểu từ bước trước. Chi phí tối thiểu từ ba khả năng (đường chéo lên, đường chéo trái và đường chéo lên trái) được tính bằng cách sử dụng `min(dtw[i-1, j-1], dtw[i-1, j], dtw[i, j-1])`. Dựa trên chi phí tối thiểu, hướng tương ứng được chọn:

2 TÓM LƯỢC LẠI PHẦN CODE EM ĐÃ LÀM TRONG MỤC 3: KHOẢNG CÁCH BIẾN ĐỔI THỜI GIAN ĐỘNG (DYNAMIC TIME WARPING- DTW)

- i. if `dtw[i-1, j]==min(dtw[i-1, j-1], dtw[i-1, j], dtw[i, j -1])` (đường chéo lên): nó gợi ý sự thay thế hoặc xóa trong y. Vì vậy, i bị giảm đi (`i -= 1`).
- ii. if `dtw[i, j-1]==min(dtw[i-1, j-1], dtw[i-1, j], dtw[i, j -1])` (đường chéo bên trái): nó gợi ý một sự thay thế hoặc một sự chèn vào x. Vì vậy, j bị giảm đi (`j -= 1`).
- iii. else: Nếu không có điều nào ở trên thì chi phí tối thiểu phải đến từ `dtw[i-1, j-1]` (đường chéo lên trên bên trái), biểu thị sự thay thế. Cả i và j đều giảm dần (`i -= 1` và `j -= 1`).

6. Tạo chuỗi `path_str` để in ra đường đi wrapping:

```
path_str = ""

for p in path:
    path_str += f"{p} -> "
path_str = path_str[:-4]
print("Warping Path:", path_str)
```

7. Trả về giá trị `dtw[m, n]` khoảng cách DTW giữa hai chuỗi thời gian x và y.

```
return dtw[m, n]
```

8. Ta nhập chuỗi, chạy chương trình và in ra kết quả.

```
if __name__ == "__main__" :

    x = input("Input the time series x: ").split()
    x = [float(i) for i in x]
    y = input("Input the time series y: ").split()
    y = [float(i) for i in y]

    distance = dtw(x, y)
    print("Dynamic Time Warping- DTW is:", distance)
```

9. Ví dụ: Ta chạy chương trình nhập vào 2 chuỗi thời gian `x= '1 2 8 5 5 1 9 4 6 5'` và `y = '1 7 4 8 2 9 6 5 2 0'`

```
Input the time series x: 1 2 8 5 5 1 9 4 6 5
Input the time series y: 1 7 4 8 2 9 6 5 2 0
Warping Path: 17.0 -> 12.0 -> 9.0 -> 9.0 -> 9.0 -> 7.0 -> 7.0 -> 6.0 -> 3.0 -> 2.0 -> 1.0 -> 0.0
Dynamic Time Warping- DTW is: 17.0
```

Hình 2: Kết quả khi chạy chương trình tìm khoảng cách biến đổi thời gian động