BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN ĐHQG-HCM
KHOA TOÁN – TIN HỌC
---o0o---

BTTH TUẦN 3: KHAI THÁC DỮ LIỆU

Sinh viên thực hiện: Nguyễn Công Hoài Nam

Mã số sinh viên: 21280099

Tp. Hồ Chí Minh, ngày 6 tháng 4 năm 2024

I. Minimum Edit Distance

Thuật toán tìm khoảng cách thay đổi nhỏ nhất (minimum edit distance) để biến chuỗi source_string thành chuỗi target_string

```
def find_minimum_edit_distance(source_string, target_string)
```

Khởi tạo ma trận 0 dp cỡ (m +1 x n +1) với m là độ dài chuỗi targert_string và n là độ dài chuỗi source_string

```
dp = [[0] * (len(source_string) + 1) for i in range(len( target_string) + 1)]
```

Điền cột/hàng đầu tiên bằng cách lấy giá trị cột/hàng trước đó tăng lên 1. Làm cơ sở để tính toàn bộ ma trận

```
for i in range(1, len(target_string) + 1) :
      dp[i][0] = dp[i - 1][0] + 1
for i in range(1, len(source_string) + 1) :
      dp[0][i] = dp[0][i - 1] + 1
```

Tạo list operations_performed gồm tuple lưu lại các phép biến đổi.
Ví dụ (INSERT, 'a') hoặc (SUBSTITUTE, 'e', 'r') hoặc (DELETE, 'j')

```
operations_performed = []
```

Duyệt tính giá trị của ma trận dp, so sánh ký tự tại targert_string[i-1] và source_string[j-1]. Nếu hai ký tự bằng nhau, lấy giá trị ở ô chéo. Nếu khác nhau, lấy giá trị min của ba ô bên trên, bên trái và chéo, cộng thêm 1 nếu là phép DELETE/INSERT, cộng thêm 2 nếu là SUBTITUTE

```
for i in range(1, len(target_string) + 1) :
    for j in range(1, len(source_string) + 1) :
        if source_string[j - 1] == target_string[i - 1] :
            dp[i][j] = dp[i - 1][j - 1]
        else :
            dp[i][j] =  min(dp[i - 1][j] + 1, \
                            dp[i - 1][j - 1] + 2, \
                            dp[i][j - 1] + 1)
```

Gán i, j lần lượt là độ dài chuỗi target và source

```
i = len(target_string)
j = len(source_string)
```

Tiến hành quay lui để ghi lại các thao tác:

Duyệt lần lượt ký tự i-1 và j-1 của chuỗi target và chuỗi source (tuỳ theo trường hợp mà đi chéo, trên hay phải bằng cách giảm i,j)

```
while (i != 0 and j != 0) :
```

Nếu hai ký tự i-1 và j-1 bằng nhau: không có gì thay đổi

```
    if target_string[i - 1] == source_string[j - 1] :
        i -= 1
        j -= 1
```

Trường hợp hai ký tự i-1 và j-1 khác nhau:

- Nếu dp[i][j] == dp[i - 1][j - 1] + 2 (bằng ô chéo cộng 2) thực hiện phép SUBTITUTE, giảm i,j
- Nếu dp[i][j] == dp[i - 1][j] + 1 (bằng ô trên cộng 1) thực hiện phép INSERT, giảm i
- Nếu dp[i][j] == dp[i][j-1] + 1 (bằng ô trái cộng 1) thực hiện phép DELETE

```
    else :
        if dp[i][j] == dp[i - 1][j - 1] + 2 :
            operations_performed.append(('SUBSTITUTE', source_string[j - 1],
            target_string[i - 1]))
            i -= 1
            j -= 1
        elif dp[i][j] == dp[i - 1][j] + 1 :
            operations_performed.append(('INSERT', target_string[i - 1]))
            i -= 1
        else :
            operations_performed.append(('DELETE', source_string[j - 1]))
            j -= 1
```

Nếu đã duyệt hết chuỗi target, ta DELETE các ký tự còn lại của chuỗi source

```
while (j != 0) :
    operations_performed.append(('DELETE', source_string[j - 1]))
    j -= 1
```

Nếu đã duyệt hết chuỗi source, ta INSERT các ký tự còn lai của chuỗi target

```
while (i != 0) :
    operations_performed.append(('INSERT', target_string[i - 1]))
    i -= 1
```

Đảo ngược lại danh sách các phép biến đổi
Trả về minimum edit distance và danh sách cách phép biến đổi

```
operations_performed.reverse()
return [dp[len(target_string)][len(source_string)], operations_performed]
```

Kết quả:
Phép biến đổi chuỗi "INTENTION" thành "EXECUTION"

```
distance, operations_performed = find_minimum_edit_distance(source_string,
target_string)
```

```
Minimum edit distance : 8
Number of insertions : 1
Number of deletions : 1
Number of substitutions : 3
Total number of operations : 5
Actual Operations :
1) DELETE : I
2) SUBSTITUTE : N by E
3) SUBSTITUTE : T by X
4) INSERT : C
5) SUBSTITUTE : N by U
```

II. Longest Common Subsequence – LCSS

Thuật toán tìm dãy con chung lớn nhất của chuỗi source và chuỗi target

```
def find_longest_common_subsequence(source_string, target_string):
```

Gán độ dài chuỗi source và target lần lượt cho m và n

```
m = len(source_string)
n = len(target_string)
```

Khởi tạo ma trận 0 dp cấp (m+1 x n+1) và biến subsequence lưu chuỗi con (ban đầu rỗng)

```
dp = [[0]*(n + 1) for i in range(m + 1)]
subsequence = ''
```

Áp dụng thuật toán để tính ma trận dp, duyệt lần lượt từng cặp ký tự của hai chuỗi
Nếu ký tự i-1 bằng ký tự j-1 của cặp chuỗi ta gán cho nó giá trị ô chéo + 1
Nếu hai cặp ký tự không trùng ta lấy max ô trên và ô trái của ma trận

```
for i in range(1, m + 1):
    for j in range(1, n + 1):
        if source_string[i - 1] == target_string[j - 1]:
            dp[i][j] = dp[i - 1][j - 1]+1
        else:
            dp[i][j] = max(dp[i-1][j], dp[i][j-1])
```

Gán i,j = m,n (độ dài cặp chuỗi) và tạo danh sách path là tuple lưu lại đường đi
Ví dụ ('UNMATCH',[i,j]) nếu hai ký tự match nhau ngược lại UNMATCH

```
i = m
```

```
j = n
path = []
```
Tiến hành quay lui

- Nếu hai ký tự của cặp chuỗi bằng nhau thì thêm vào subsequence vào lưu path với MATCH
- Nếu hai ký tự không bằng nhau:
    - Nếu dp[i][j-1] == dp[i-1][j] + 1 tức ô bên trái lớn hơn ô trên nên ta lấy ô trái
    - Ngược lại ô trên lớn hơn nên ta lấy ô trên (trường hợp ô trái bằng ô trên thì lấy ô trên cho hợp kết quả của trên file KDL_Tuan3.pdf, nếu lấy ô trái cũng không sai)
    - Lưu path với UNMATCH

```
while i != 0 and j != 0:
    if source_string[i-1] == target_string[j-1]:
        subsequence += source_string[i-1]
        path.append(('MATCH',[i,j]))
        i -= 1
        j -= 1
    elif dp[i][j-1] == dp[i-1][j] + 1:
        path.append(('UNMATCH',[i,j]))
        j -= 1

    else:
        path.append(('UNMATCH',[i,j]))
        i -= 1
```

Thêm vị trí cuối là UNMATCH (so sánh với NULL) và đảo ngược lại path

```
path.append(('UNMATCH',[i,j]))
path.reverse()
```

Trả về độ lcss, lcs, ma trận dp và path (để visual)

```
return dp[m][n], subsequence[::-1], dp, path
```

Kết quả

Ta tìm chuỗi con lớn nhất của chuỗi "ACADB" với chuỗi "CBDA"

```
source_string = "ACADB"
target_string = "CBDA"
distance, subsequence, dp, path =
find_longest_common_subsequence(source_string, target_string)
print("LCSS: ", subsequence)
print("Độ dài:", distance)
```

Output:

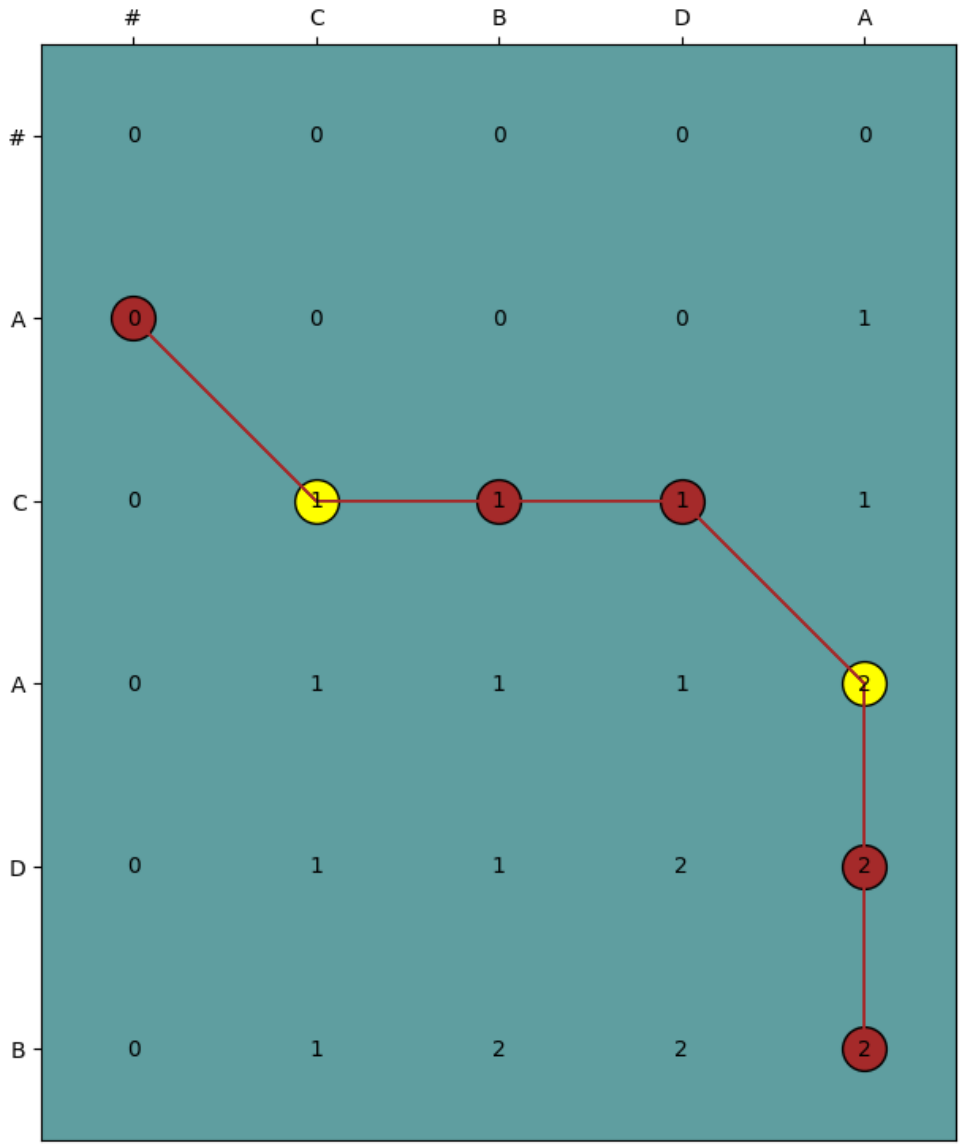
```
LCSS:  CA
Độ dài: 2
```

Bonus: trực quan thuật toán

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

def show_result_lcss(dp, path, source_string, target_string):
    dp = np.array(dp)

    plt.figure(figsize=(16, 9))

    cmap = ListedColormap('cadetblue')

    plt.imshow(dp, cmap=cmap)

    for i in range(dp.shape[0]):
        for j in range(dp.shape[1]):
            plt.text(j, i, str(dp[i, j]), ha='center', va='center',
color='black')

    for action, coord in path:
        color = 'yellow' if action == 'MATCH' else 'brown'
        plt.scatter(coord[1], coord[0], color=color, edgecolors='black',
s=400)


    for i in range(len(path)-1):
        x1, y1 = path[i][1][0], path[i][1][1]
        x2, y2 = path[i+1][1][0], path[i+1][1][1]
        plt.arrow(y1, x1, y2-y1, x2-x1, ec='brown')


    x_labels = ['#']
    for char in target_string:
        x_labels.append(char)
    plt.gca().xaxis.tick_top()

    plt.xticks(np.arange(len(x_labels)), x_labels)
     y_labels = ['#']
```

```
    for char in source_string:
        y_labels.append(char)
    plt.yticks(np.arange(len(y_labels)), y_labels)

    plt.show()
```

```
show_result_lcss(dp, path, source_string, target_string)
```



III. Dynamic Time Warping – DTW

Tính khoảng cách biến đổi thời gian động từ hai chuỗi đầu vào

```
def find_DTW(series_1, series_2)
```

Lấy độ dài hai chuỗi lưu vào `m,n`

```
m = len(series_1)
n = len(series_2)
```

Nếu một trong hai chuỗi rỗng return

```
if m == 0 or n == 0:
    return [],[]
```

Khởi tạo ma trận 0 dp với kích thước `(m x n)`

```
dp = [[0]*n for i in range(m)]
```

Tính giá trị cho dp[0][0], tính dp cho hàng/cột đầu tiên bằng trị tuyệt sự khác biệt giữa các cặp phần tử của 2 chuỗi + giá trị ô trước hàng/cột đó (là min nhưng trong trường hợp này chỉ có một phần tử)

```
dp[0][0]=abs(series_1[0] - series_2[0])
for i in range(1, m):
    dp[i][0] = abs(series_1[i]-series_2[0])+dp[i-1][0]
for i in range(1, n):
    dp[0][i] = abs(series_2[i]-series_1[0])+dp[0][i-1]
```

Tính toán dp cho các phần tử còn lại bằng trị truyệt đối chênh lệch giữa hai cặp phần tử của hai chuỗi + min(ô trên, ô trái, ô chéo)
Đảo ngược dp để quay lui, gán i,j bằng phần tử cuối cùng và lưu nó vào dtw_path

```
dp = dp[::-1]
i = 0
j = n-1
dtw_path = [(i, j)]
```

Tiến hành quay lui
Xét dp[i][j] lần lượt với ô chéo, ô trên, ô trái. Lấy ô có giá trị nhỏ nhất thêm vào dtw_path và tiếp tục duyệt

```
while i != m-1 and j != 0:
    if dp[i+1][j-1] == min(dp[i+1][j], dp[i][j-1], dp[i+1][j-1]):
        dtw_path.append((i+1, j-1))
        i += 1
        j -= 1
    elif dp[i+1][j] == min(dp[i+1][j], dp[i][j-1], dp[i+1][j-1]):
        dtw_path.append((i+1, j))
        i += 1
    else:
        dtw_path.append((i, j-1))
        j -= 1
```

Thêm điểm cuối cùng vào dtw_path và return ma trận dp và toạ độ path

```
dtw_path.append((m-1, 0))
return dp, dtw_path
```

Kết quả

Tính khoảng cách biến đổi thời gian động của chuỗi series 1 và 2

```
series_1 = [1, 7, 4, 8, 2, 9, 6, 5, 2, 0]
series_2 = [1, 2, 8, 5, 5, 1, 9, 4, 6, 5]
```

```
dp, path = find_DTW(series_1, series_2)
print("DTW Path: ", end = "")
for i in range(len(path) - 1):
    x, y = path[i]
    print(dp[x][y], end="")
    if i != len(path) - 2:
        print("->", end="")
```

Output

```
DTW Path: 17→12→9→9→9→7→7→6→3→2→1
```

Bonus: trực quan thuật toán

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

def show_result_dtw(dp, path, series_1, series_2):
    dp = np.array(dp)

    plt.figure(figsize=(16, 9))

    cmap = ListedColormap('cadetblue')

    plt.imshow(dp, cmap=cmap)

    for coord in path:
        plt.scatter(coord[1], coord[0], color='yellow', edgecolors = 'black',
s=400)

    for i in range(len(path)-1):
        plt.arrow(path[i][1], path[i][0], path[i+1][1]-path[i][1],
path[i+1][0]-path[i][0], ec='brown')

    for i in range(dp.shape[0]):
        for j in range(dp.shape[1]):
            plt.text(j, i, str(dp[i, j]), ha='center', va='center',
color='black')

     x_labels = []
    for char in series_2:
        x_labels.append(char)
    plt.gca().xaxis.tick_top()
```

```
    plt.xticks(np.arange(len(x_labels)), x_labels)
    y_labels = []
    for char in series_1:
        y_labels.append(char)
    plt.yticks(np.arange(len(y_labels)), y_labels)

    plt.show()
```

Output

```
show_result_dtw(dp,path, series_1, series_2)
```