

Bài 3: SỰ TƯƠNG ĐỒNG VÀ CÁC KHOẢNG CÁCH (TT)

I. Mục tiêu:

Sau khi thực hành xong, sinh viên nắm được:

- Khoảng cách thay đổi
- Khoảng cách dãy con chung dài nhất
- Khoảng cách biến đổi thời gian động

II. Tóm tắt lý thuyết:

1. Khoảng cách thay đổi (Edit distance):

Khoảng cách thay đổi là số ký tự nhỏ nhất mà khi thêm vào, xóa hoặc thay thế ký tự được yêu cầu để thay đổi 1 chuỗi thành 1 chuỗi khác.

Cho 2 chuỗi $\bar{X} = (x_1 \dots x_m)$ và $\bar{Y} = (y_1 \dots y_n)$, cho $Edit(\bar{X}, \bar{Y})$ là sự thay đổi được thực thi trong chuỗi \bar{X} để biến đổi thành chuỗi \bar{Y} , I_{ij} là bộ phận chỉ nhị phân được xác định như sau

$$I_{ij} = \begin{cases} 0 & \text{nếu } x_i = y_i \\ 1 & \text{ngược lại} \end{cases}$$

Xét \bar{X}_i là i kí hiệu đầu tiên của \bar{X} , \bar{Y}_j là j kí hiệu đầu tiên của \bar{Y} . Chi phí thích ứng tối ưu của 2 đoạn này là $Edit(i, j)$. Mục tiêu là để xác định phép toán gì để thực thi phần tử cuối cùng của \bar{X}_i sao cho nó hoặc là match một phần tử trong \bar{Y}_j hoặc nó bị xóa. Có 3 khả năng xuất hiện:

- Phần tử cuối cùng của \bar{X}_i bị xóa và chi phí cho điều này là $[Edit(i - 1, j) + Deletion Cost]$. (Phần tử cuối cùng của đoạn bị chặt cụt \bar{X}_{i-1} có thể hoặc không phù hợp với phần tử cuối của \bar{Y}_j tại điểm này).
- Một phần tử được thêm vào cuối của \bar{X}_i để phù hợp với phần tử cuối của \bar{Y}_j và chi phí của điều này là $[Edit(i, j - 1) + Insertion Cost]$.

- Phần tử cuối cùng của \overline{X}_i được lật (flip) thành phần tử của \overline{Y}_j nếu nó khác nhau và chi phí cho điều này là $[Edit(i-1, j-1) + I_{ij} \cdot (Replacement\ Cost)]$.

Khi đó, việc phù hợp tối ưu được xác định bằng đệ quy như sau:

$$Edit(i, j) = \begin{cases} Edit(i-1, j) + Deletion\ Cost \\ Edit(i, j-1) + Insertion\ Cost \\ Edit(i-1, j-1) + I_{ij} \cdot (Replacement\ Cost) \end{cases}$$

Ví dụ: Cho 2 chuỗi $\overline{X} = INTENTION$ và $\overline{Y} = EXECUTION$. Chương trình quy hoạch động tính $dp(n, m)$ (hay $Edit(n, m)$) theo dạng bảng

- Giải bài toán bằng việc kết hợp lời giải tới các bài toán con.
- Khởi tạo:

$$dp(i, 0) = i$$

$$dp(0, j) = j$$

- Đệ quy:

For each $i = 1 \dots m$

For each $j = 1 \dots n$

$$dp(i, j) = \min \begin{cases} dp(i-1, j) + 1 \\ dp(i, j-1) + 1 \\ dp(i-1, j-1) + a \end{cases}$$

với

$$a = \begin{cases} 2 & \text{nếu } x_i \neq y_j \\ 0 & \text{ngược lại} \end{cases}$$

- Kết thúc: $dp(n, m)$ là khoảng cách Edit.

Tính

$$dp(1, 1) = \min(dp(0, 1) + 1, dp(1, 0) + 1, dp(0, 0) + 2) = 2$$

$$dp(1, 2) = \min(dp(0, 2) + 1, dp(1, 1) + 1, dp(0, 1) + 2) = 3$$

$$dp(1, 3) = \min(dp(0, 3) + 1, dp(1, 2) + 1, dp(0, 2) + 2) = 4$$

$$dp(1, 4) = \min(dp(0, 4) + 1, dp(1, 3) + 1, dp(0, 3) + 2) = 5$$

$$dp(1, 5) = \min(dp(0, 5) + 1, dp(1, 4) + 1, dp(0, 4) + 2) = 6$$

$$dp(1, 6) = \min(dp(0, 6) + 1, dp(1, 5) + 1, dp(0, 5) + 2) = 7$$

$$dp(1, 7) = \min(dp(0, 7) + 1, dp(1, 6) + 1, dp(0, 6) + 0) = 6$$

$$dp(1, 8) = \min(dp(0, 8) + 1, dp(1, 7) + 1, dp(0, 7) + 2) = 7$$

$$dp(1, 9) = \min(dp(0, 9) + 1, dp(1, 8) + 1, dp(0, 8) + 2) = 8$$

N	9									
O	8									
I	7									
T	6									
N	5									
E	4									
T	3									
N	2									
I	1	2	3	4	5	6	7	6	7	8
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

Tương tự, ta có

N	9	8	9	10	11	12	11	10	9	8
O	8	7	8	9	10	11	10	9	8	9
I	7	6	7	8	9	10	9	8	9	10
T	6	5	6	7	8	9	8	9	10	11
N	5	4	5	6	7	8	9	10	11	10
E	4	3	4	5	6	7	8	9	10	9
T	3	4	5	6	7	8	7	8	9	8
N	2	3	4	5	6	7	8	7	8	7
I	1	2	3	4	5	6	7	6	7	8
#	0	1	2	3	4	5	6	7	8	9
	#	E	X	E	C	U	T	I	O	N

Quay lui

- Các điều kiện cơ sở:

$$dp(i, 0) = i$$

$$dp(0, j) = j$$

- Định quy:

For each $i = 1 \dots m$

For each $j = 1 \dots n$

$$dp(i, j) = \min \begin{cases} dp(i-1, j) + 1 & \text{deletion} \\ dp(i, j-1) + 1 & \text{insertion} \\ dp(i-1, j-1) + a & \text{substitution} \end{cases}$$

với

$$a = \begin{cases} 2 & \text{nếu } x_i \neq y_j \\ 0 & \text{ngược lại} \end{cases}$$

$$ptr(i, j) = \begin{cases} LEFT & \boxed{\text{deletion}} \\ DOWN & \boxed{\text{insertion}} \\ DIAG & \boxed{\text{substitution}} \end{cases}$$

n	9	↓ 8	↖ ↗ 9	↖ ↗ 10	↖ ↗ 11	↖ ↗ 12	↓ 11	↓ 10	↓ 9	↖ 8
o	8	↓ 7	↖ ↗ 8	↖ ↗ 9	↖ ↗ 10	↖ ↗ 11	↓ 10	↓ 9	↖ 8	← 9
i	7	↓ 6	↖ ↗ 7	↖ ↗ 8	↖ ↗ 9	↖ ↗ 10	↓ 9	↖ 8	← 9	← 10
t	6	↓ 5	↖ ↗ 6	↖ ↗ 7	↖ ↗ 8	↖ ↗ 9	↖ 8	← 9	← 10	↖ 11
n	5	↓ 4	↖ ↗ 5	↖ ↗ 6	↖ ↗ 7	↖ ↗ 8	↖ 9	↖ 10	↖ 11	↖ 10
e	4	↖ 3	← 4	↖ 5	← 6	← 7	← 8	↖ 9	↖ 10	↓ 9
t	3	↖ 4	↖ 5	↖ 6	↖ 7	↖ 8	↖ 7	← 8	↖ 9	↓ 8
n	2	↖ 3	↖ 4	↖ 5	↖ 6	↖ 7	↖ 8	↓ 7	↖ 8	↖ 7
i	1	↖ 2	↖ 3	↖ 4	↖ 5	↖ 6	↖ 7	↖ 6	← 7	← 8
#	0	1	2	3	4	5	6	7	8	9
	#	e	x	e	c	u	t	i	o	n

Ta có

I	N	T	E	*	N	T	I	O	N
*	E	X	E	C	U	T	I	O	N

2. Khoảng cách dãy con chung dài nhất (Longest Common Subsequence-LCSS):

Cho 2 chuỗi $\bar{X} = (x_1 \dots x_m)$ và $\bar{Y} = (y_1 \dots y_n)$. Xét X_i là i kí hiệu đầu tiên của \bar{X} , \bar{Y}_j là j kí hiệu đầu tiên của \bar{Y} và $LCSS(i, j)$ biểu diễn các giá trị LCSS tối ưu giữa 2 đoạn này. Mục tiêu là để phù hợp phần tử cuối của \bar{X}_i và \bar{Y}_j hoặc là để xóa phần tử cuối của một trong hai chuỗi. Hai khả năng xuất hiện:

- Phần tử cuối của \bar{X}_i phù hợp với \bar{Y}_j . Giá trị tương đồng $LCSS(i, j)$ có thể được biểu diễn đệ quy $LCSS(i-1, j-1)$.
- Phần tử cuối không phù hợp. Trong trường hợp này, phần tử cuối của 1 trong 2 chuỗi cần được xóa. Giá trị của $LCSS(i, j)$ hoặc là $LCSS(i, j-1)$ hoặc $LCSS(i-1, j)$ phụ thuộc vào chuỗi được chọn để xóa.

Khi đó, $LCSS(i, j)$ được xác định như sau:

$$LCSS(i, j) = \max \begin{cases} LCSS(i-1, j-1) & \text{if } x_i = y_i \\ LCSS(i-1, j) & \text{otherwise (no match on } x_i) \\ LCSS(i, j-1) & \text{otherwise (no match on } y_i) \end{cases}$$

Ví dụ: Cho 2 chuỗi $x = \text{ACADB}$ và $y = \text{CBDA}$, sử dụng chương trình động để tìm LCSS của 2 chuỗi.

```
X and Y be two given sequences
Initialize a table LCS of dimension X.length * Y.length
X.label = X
Y.label = Y
LCS[0][] = 0
LCS[][0] = 0
Start from LCS[1][1]
Compare X[i] and Y[j]
  If X[i] = Y[j]
    LCS[i][j] = 1 + LCS[i-1, j-1]
    Point an arrow to LCS[i][j]
  Else
    LCS[i][j] = max(LCS[i-1][j], LCS[i][j-1])
    Point an arrow to max(LCS[i-1][j], LCS[i][j-1])
```

- Khởi tạo bảng $(m+1) \times (n+1)$ chiều với m, n tương ứng là chiều dài của chuỗi \overline{X} và chuỗi \overline{Y} . Dòng đầu tiên và cột đầu tiên được điền vào các số 0.

		C	B	D	A
	0	0	0	0	0
A	0				
C	0				
A	0				
D	0				
B	0				

- Điền vào mỗi ô của bảng sử dụng: Nếu ký tự tương ứng với dòng hiện tại và cột hiện tại đang phù hợp thì điền ô hiện tại bằng việc cộng thêm 1 thành phần từ đường chéo. Chỉ mũi tên tới ô đường chéo. Ngược lại lấy giá trị lớn nhất từ phần tử cột trước và dòng trước cho việc điền vào ô hiện tại. Chỉ mũi tên tới ô với giá trị lớn nhất. Nếu chúng bằng nhau thì vẽ tới giá trị bất kỳ của chúng.

		C	B	D	A
	0	0	0	0	0
A	0	0	0	0	1
C	0				
A	0				
D	0				
B	0				

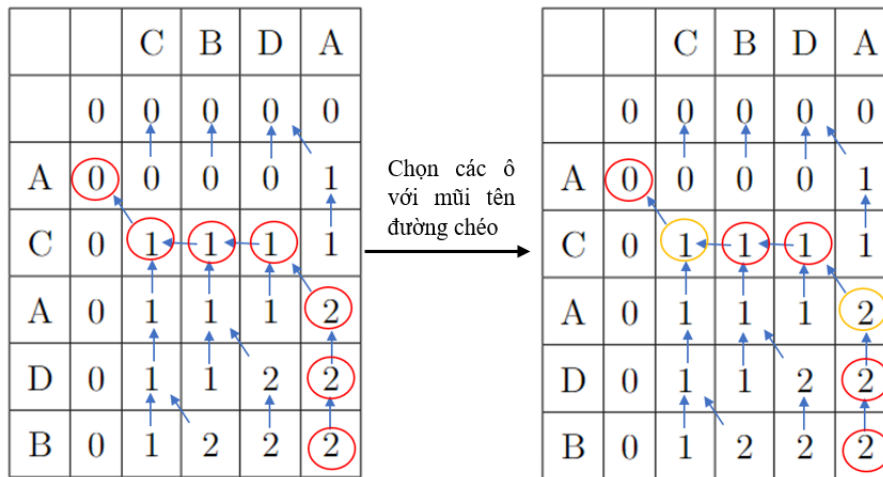
- Lặp lại bước trên cho tới khi bảng được điền vào đầy đủ

		C	B	D	A
	0	0	0	0	0
A	0	0	0	0	1
C	0	1	1	1	1
A	0	1	1	1	2
D	0	1	1	2	2
B	0	1	2	2	2

- Giá trị nằm trong dòng cuối và cột cuối là chiều dài của LCSS.

		C	B	D	A
	0	0	0	0	0
A	0	0	0	0	1
C	0	1	1	1	1
A	0	1	1	1	2
D	0	1	1	2	2
B	0	1	2	2	2

- Để tìm LCSS, ta bắt đầu từ phần tử cuối và theo hướng của mũi tên



- Do đó, LCSS là CA

3. Khoảng cách biến đổi thời gian động (Dynamic Time Warping - DTW):

DTW kéo dãn chuỗi thời gian theo chiều thời gian một cách linh động tại mỗi vị trí.

DTW xác định được khoảng cách giữa 2 chuỗi có chiều dài khác nhau.

Xét metric Manhattan L_1 , $M(\overline{X}_i, \overline{Y}_i)$ được xác định như sau:

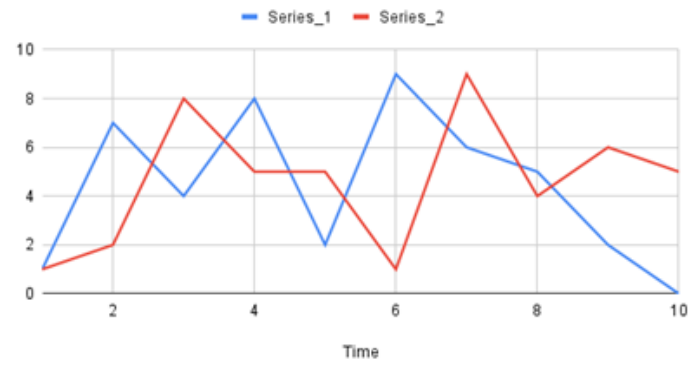
$$M(\overline{X}_i, \overline{Y}_i) = |x_i - y_i| + M(\overline{X}_{i-1}, \overline{Y}_{i-1})$$

Xét $DTW(i, j)$ là khoảng cách tối ưu giữa i phần tử đầu tiên của chuỗi $(x_1 \dots x_m)$ và j phần tử đầu tiên của chuỗi $(y_1 \dots y_n)$. Khi đó, $DTW(i, j)$ được xác định như sau:

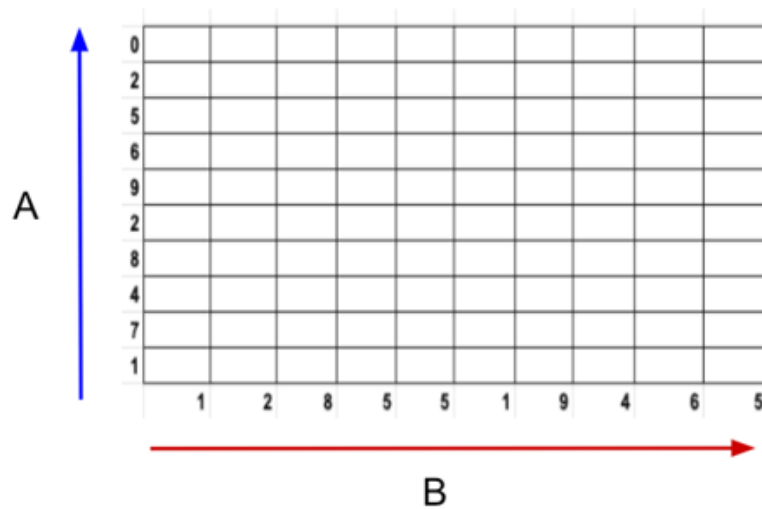
$$DTW(i, j) = distance(x_i, y_j) + \min \begin{cases} DTW(i, j-1) & \text{repeat } x_i \\ DTW(i-1, j) & \text{repeat } y_j \\ DTW(i-1, j-1) & \text{repeat neither} \end{cases}$$

Ví dụ: Cho 2 chuỗi thời gian A và B

Time	Series_1	Series_2
1	1	1
2	7	2
3	4	8
4	8	5
5	2	5
6	9	1
7	6	9
8	5	4
9	2	6
10	0	5



- Khởi tạo ma trận chi phí rộng



- Tính chi phí: sử dụng công thức

$$DTW(i, j) = distance(x_i, y_j) + \min \begin{cases} DTW(i, j - 1) & \text{repeat } x_i \\ DTW(i - 1, j) & \text{repeat } y_j \\ DTW(i - 1, j - 1) & \text{repeat neither} \end{cases}$$

với $distance(A_i, B_j) = |A_i - B_j|$.

Ta tính:

$$DTW(1, 1) = |1 - 1| = 0$$

$$DTW(1, 2) = |1 - 2| + \min(0) = 1$$

$$DTW(1, 3) = |1 - 8| + \min(1) = 8$$

$$DTW(1, 4) = |1 - 5| + \min(8) = 12$$

$$DTW(1, 5) = |1 - 5| + \min(12) = 16$$

$$DTW(1, 6) = |1 - 1| + \min(16) = 16$$

$$DTW(1, 7) = |1 - 9| + \min(16) = 24$$

$$DTW(1, 8) = |1 - 4| + \min(24) = 27$$

$$DTW(1, 9) = |1 - 6| + \min(27) = 32$$

$$DTW(1, 10) = |1 - 5| + \min(32) = 36$$

$$DTW(2, 1) = |7 - 1| + \min(0) = 6$$

$$DTW(3, 1) = |4 - 1| + \min(6) = 9$$

$$DTW(4, 1) = |8 - 1| + \min(9) = 16$$

$$DTW(5, 1) = |2 - 1| + \min(16) = 17$$

$$DTW(6, 1) = |9 - 1| + \min(17) = 25$$

$$DTW(7, 1) = |6 - 1| + \min(25) = 30$$

$$DTW(8, 1) = |5 - 1| + \min(30) = 34$$

$$DTW(9, 1) = |2 - 1| + \min(34) = 35$$

$$DTW(10, 1) = |0 - 1| + \min(35) = 36$$

$$DTW(2, 2) = |7 - 2| + \min(0, 1, 6) = 5$$

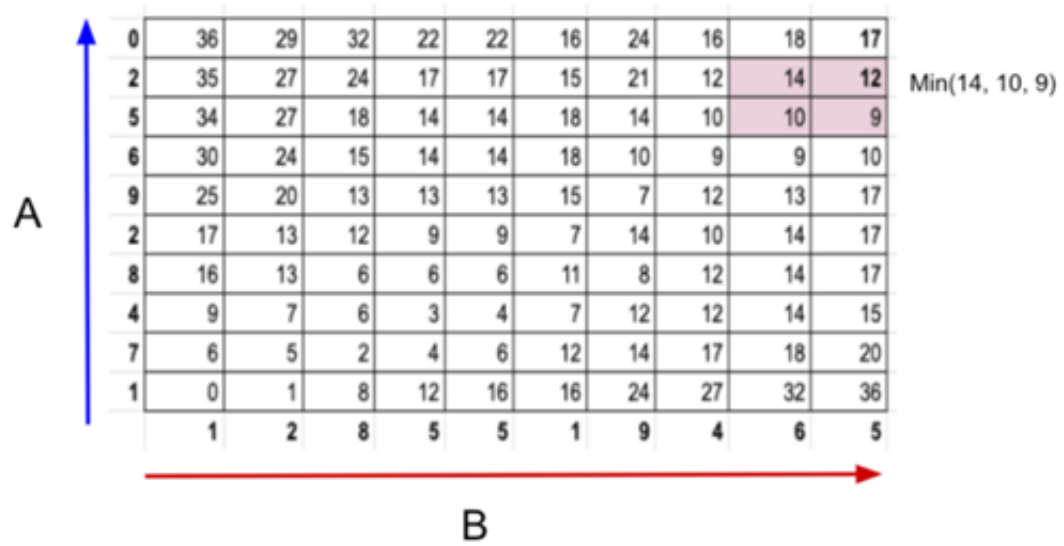
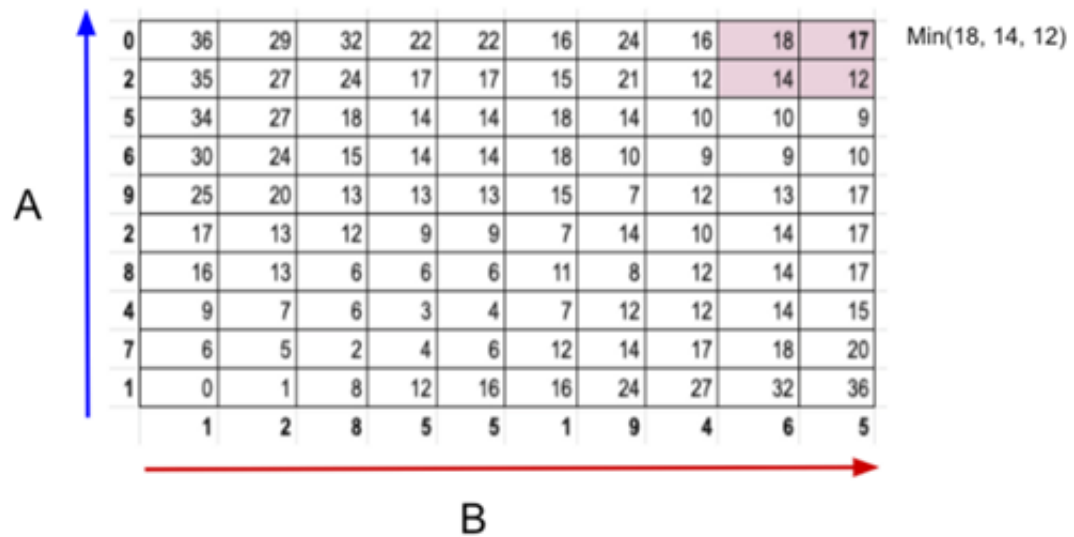
...

0	36	29							
2	35	27							
5	34	27							
6	30	24							
9	25	20							
2	17	13							
8	16	13	6	6					
4	9	7	6	3					
7	6	5	2	4	6	12	14	17	18
1	0	1	8	12	16	16	24	27	32
	1	2	8	5	5	1	9	4	6

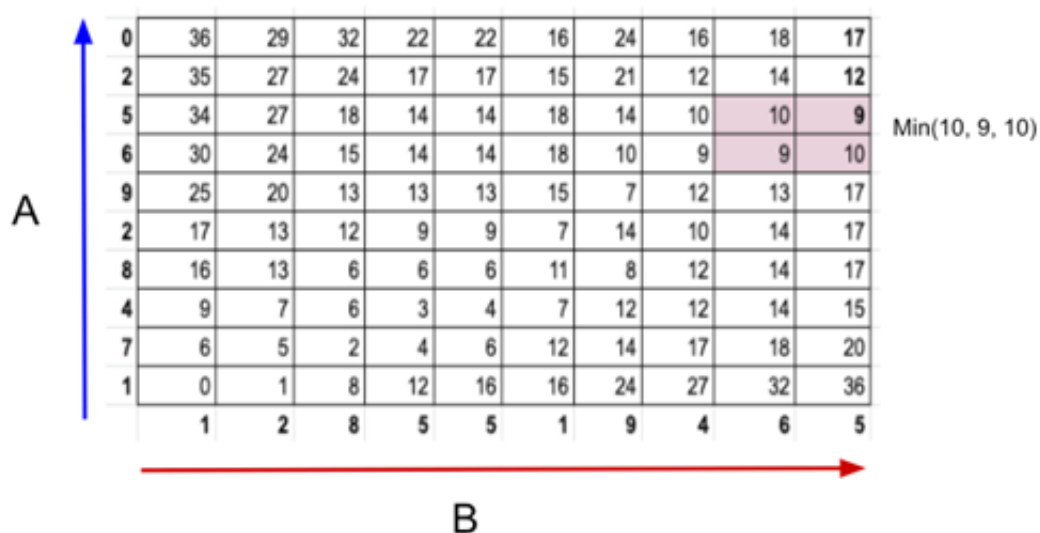
- Tương tự, ta điền đầy đủ vào bảng

0	36	29	32	22	22	16	24	16	18
2	35	27	24	17	17	15	21	12	14
5	34	27	18	14	14	18	14	10	10
6	30	24	15	14	14	18	10	9	9
9	25	20	13	13	13	15	7	12	13
2	17	13	12	9	9	7	14	10	14
8	16	13	6	6	6	11	8	12	14
4	9	7	6	3	4	7	12	12	14
7	6	5	2	4	6	12	14	17	18
1	0	1	8	12	16	16	24	27	32
	1	2	8	5	5	1	9	4	6

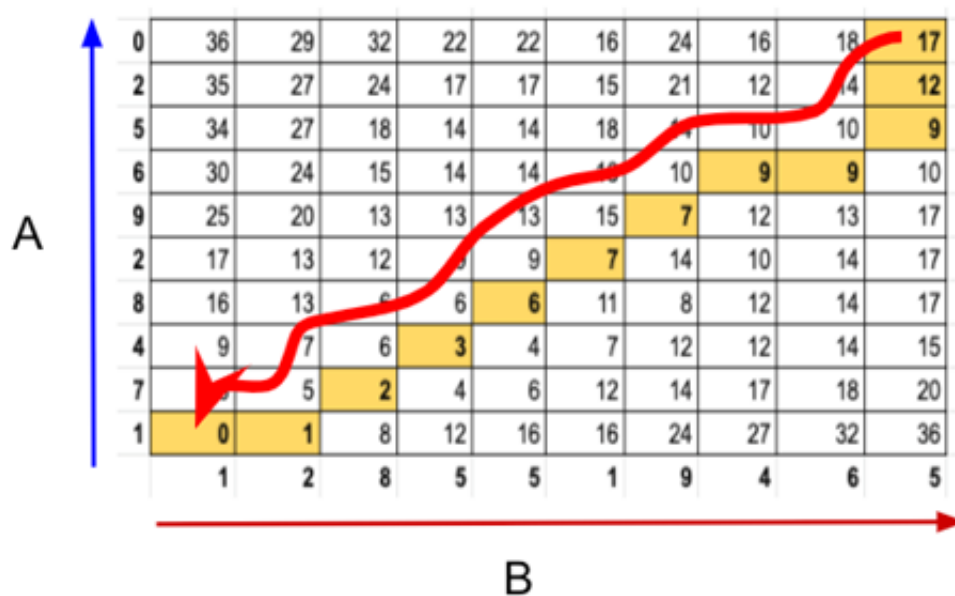
- Sự đồng nhất đường đi wrapping: bắt đầu từ ở góc phải phía trên của ma trận và đi ngang qua bên trái ở phía dưới. Đường đi ngang qua được dựa vào láng giềng với giá trị nhỏ nhất. Ví dụ: bắt đầu với 17 và tìm giá trị nhỏ nhất giữa 18, 14 và 12.



- Số tiếp theo trong đường đi là 9. Quá trình này tiếp tục cho tới chúng ta đi tới phía dưới bên trái của bảng.



- Đường đi cuối cùng



- Khi đó chuỗi đường đi wrapping là [17, 12, 9, 9, 9, 7, 7, 6, 3, 2, 1, 0].

III. Nội dung thực hành:

1. Viết chương trình tính khoảng cách edit

```
def find_minimum_edit_distance(source_string, target_string) :  
  
    # Create a dp matrix of dimension (source_string + 1) x (destination_matrix + 1)  
    dp = [[0] * (len(source_string) + 1) for i in range(len(target_string) + 1)]  
  
    # Initialize the required values of the matrix  
    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  
  
    # Maintain the record of operations done  
    # Record is one tuple. Eg : (INSERT, 'a') or (SUBSTITUTE, 'e', 'r') or (DELETE, 'j')  
    operations_performed = []  
  
    # Build the matrix following the algorithm  
    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)  
  
    # Initialization for backtracking  
    i = len(target_string)  
    j = len(source_string)  
  
    # Backtrack to record the operation performed  
    while (i != 0 and j != 0) :  
        # If the character of the source string is equal to  
        # the character of the destination string,  
        # no operation is performed  
        if target_string[i - 1] == source_string[j - 1] :  
            i -= 1  
            j -= 1  
        else :  
            # Check if the current element is derived from the upper-left diagonal element  
            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
    else :
        # Check if the current element is derived from the upper-left diagonal element
        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
        # Check if the current element is derived from the upper element
        elif dp[i][j] == dp[i - 1][j] + 1 :
            operations_performed.append(('INSERT', target_string[i - 1]))
            i -= 1
        # Check if the current element is derived from the left element
        else :
            operations_performed.append(('DELETE', source_string[j - 1]))
            j -= 1

    # If we reach top-most row of the matrix
    while (j != 0) :
        operations_performed.append(('DELETE', source_string[j - 1]))
        j -= 1

    # If we reach left-most column of the matrix
    while (i != 0) :
        operations_performed.append(('INSERT', target_string[i - 1]))
        i -= 1

    # Reverse the list of operations performed as we have operations in reverse
    # order because of backtracking
    operations_performed.reverse()
    return [dp[len(target_string)][len(source_string)], operations_performed]

if __name__ == "__main__":

    # Get the source and target string
    print("Enter the source string :")
    source_string = input().strip()
    print("Enter the target string :")
    target_string = input().strip()

```

```

# Find the minimum edit distance and the operation performed
distance, operations_performed = find_minimum_edit_distance(source_string, target_string)

# Count the number of individual operations
insertions, deletions, substitutions = 0, 0, 0
for i in operations_performed :
    if i[0] == 'INSERT' :
        insertions += 1
    elif i[0] == 'DELETE' :
        deletions += 1
    else :
        substitutions += 1

# Print the results
print("Minimum edit distance : {}".format(distance))
print("Number of insertions : {}".format(insertions))
print("Number of deletions : {}".format(deletions))
print("Number of substitutions : {}".format(substitutions))
print("Total number of operations : {}".format(insertions +
                                                deletions + substitutions))

print("Actual Operations :")
for i in range(len(operations_performed)) :

    if operations_performed[i][0] == 'INSERT' :
        print("{} {} : {}".format(i + 1, operations_performed[i][0],
                                    operations_performed[i][1]))
    elif operations_performed[i][0] == 'DELETE' :
        print("{} {} : {}".format(i + 1, operations_performed[i][0],
                                    operations_performed[i][1]))
    else :
        print("{} {} : {} by {}".format(i + 1, operations_performed[i][0],
                                         operations_performed[i][1],
                                         operations_performed[i][2]))

```

2. Viết chương trình tính khoảng cách dãy con chung dài nhất

3. Viết chương trình tính khoảng cách biến đổi thời gian động

4. Yêu cầu:

- Cài đặt và thực thi mục 1 trên máy tính.
- Làm bài tương tự ở mục 2 và mục 3.
- Viết file báo cáo trình bày tóm lược lại phần code em đã làm trong mục 2 và 3.