Bài tập Thực hành môn Khai phá Dữ liệu

Họ và tên: Huỳnh Nguyễn Thế Dân

MSSV: 21110256

Lớp: 21TTH1

1. Viết chương trình tính khoảng cách Edit

Định nghĩa hàm tính độ đo chỉnh sửa tối thiểu

```
# Định nghĩa hàm tính độ đo chỉnh sửa tối thiểu
def Find_Mininum_Edit_Distance(source_string, target_string):

    # Tạo ma trận chỉnh sửa có kích thước (len(target_string) + 1) x
(len(source_string) + 1)
    Edit = [[0] * (len(source_string) + 1) for i in
range(len(target_string) + 1)]

    # Khởi tạo giá trị ban đầu của ma trận
    for i in range(1, len(target_string) + 1):
        Edit[i][0] = Edit[i-1][0] + 1
    for i in range(1, len(source_string) + 1):
        Edit[0][i] = Edit[0][i-1] + 1

    # Danh sách các thao tác được thực hiện
    operations_performed = []

    # Xây dựng ma trận chỉnh sửa theo thuật toán
    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]:
                Edit[i][j] = Edit[i-1][j-1]
            else:
                Edit[i][j] = min(Edit[i-1][j] + 1, Edit[i][j-1] + 1,
Edit[i-1][j-1] + 2)

    # Khởi tạo biến để backtrack
    i = len(target_string)
    j = len(source_string)

    # Backtrack để ghi lại các thao tác thực hiện
    while (i != 0 and j != 0):
        if target_string[i-1] == source_string[j-1]:
            i -= 1
            j -= 1
        else:
            if Edit[i][j] == Edit[i-1][j-1] + 2:
```

```
                operations_performed.append(("SUBSTITURE",
source_string[j-1], target_string[i-1]))
                i -= 1
                j -= 1
            elif Edit[i][j] == Edit[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
    while (j != 0):
        operations_performed.append(("DELETE", source_string[j-1]))
        j -= 1
    while (i != 0):
        operations_performed.append(("DELETE", target_string[i-1]))
        i -= 1

    # Đảo ngược danh sách thao tác vì đã backtrack
    operations_performed.reverse()

    # Trả về ma trận chỉnh sửa, khoảng cách tối thiểu và danh sách các
thao tác thực hiện
    return Edit, Edit[-1][-1], operations_performed
```

Chương trình chính

```
import pandas as pd  # Import thư viện pandas
import numpy as np   # Import thư viện numpy

# Nhập chuỗi nguồn và chuỗi đích
print("Nhập chuỗi nguồn: ")
source_string = input().strip()

print("Nhập chuỗi đích: ")
target_string = input().strip()

# Tìm độ đo chỉnh sửa tối thiểu và các thao tác thực hiện
Edit_matrix, distance, operations_performed =
Find_Mininum_Edit_Distance(source_string, target_string)

# Hiển thị ma trận chỉnh sửa
display(pd.DataFrame(Edit_matrix))

# Đếm số lượng các thao tác
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
```

```
# Hiển thị kết quả
print("Độ đo chỉnh sửa tối thiểu: {}".format(distance))
print("Số lượng thêm vào: {}".format(insertions))
print("Số lượng xóa bỏ: {}".format(deletions))
print("Số lượng thay thế: {}".format(substitutions))
print("Tổng số thao tác: {}".format(insertions + deletions +
substitutions))

print("Thao tác thực hiện: ")
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("{}) {} : {} - {}".format(i+1, operations_performed[i]
[0], operations_performed[i][1], operations_performed[i][2]))
```

Đoạn code trên thực hiện các công việc sau:

- Định nghĩa hàm Find_Mininum_Edit_Distance để tính độ đo chỉnh sửa tối thiểu giữa hai chuỗi.
- Tạo ma trận chỉnh sửa Edit với kích thước (len(target_string) + 1) x (len(source_string) + 1) và khởi tạo các giá trị ban đầu.
- Xây dựng ma trận chỉnh sửa Edit dựa trên thuật toán của độ đo chỉnh sửa tối thiểu (Minimum Edit Distance).
- Sử dụng backtracking để ghi lại các thao tác đã thực hiện để chuyển đổi từ chuỗi nguồn sang chuỗi đích.
- Tính và hiển thị ma trận chỉnh sửa Edit.
- Đếm số lượng các thao tác: thêm vào, xóa bỏ, thay thế.
- Hiển thị kết quả: độ đo chỉnh sửa tối thiểu và số lượng các thao tác.
- In ra danh sách các thao tác thực hiện để chuyển đổi từ chuỗi nguồn sang chuỗi đích.

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

Định nghĩa hàm tính khoảng cách dãy con chung dài nhất

```
def Find_Longest_Common_Subsequense(compare_string_1,
compare_string_2):

    # Tạo ma trận LCSS có kích thước (len(compare_string_1) + 1) x
(len(compare_string_2) + 1)
    LCSS = [[0] * (len(compare_string_2) + 1) for i in
range(len(compare_string_1) + 1)]

    # Danh sách các thao tác được thực hiện
```

```
    operations_performed = []

    # Xây dựng ma trận LCSS theo thuật toán
    for i in range(1, len(compare_string_1) + 1):
        for j in range(1, len(compare_string_2) + 1):
            if compare_string_2[j-1] == compare_string_1[i-1]:
                LCSS[i][j] = LCSS[i-1][j-1] + 1
            else:
                LCSS[i][j] = max(LCSS[i-1][j], LCSS[i][j-1])

    # Khởi tạo biến để backtrack
    i = len(compare_string_1)
    j = len(compare_string_2)

    # Backtrack để ghi lại các thao tác thực hiện
    while (i != 0 and j != 0):
        if compare_string_1[i-1] == compare_string_2[j-1]:
            operations_performed.append(("CROSS", compare_string_2[j-
1]))
            i -= 1
            j -= 1
        else:
            if LCSS[i][j] == LCSS[i-1][j]:
                operations_performed.append(("UP", 'None'))
                i -= 1
            elif LCSS[i][j] == LCSS[i][j-1]:
                operations_performed.append(("LEFT", 'None'))
                j -= 1

    # Nếu chúng ta đạt đến hàng trên cùng của ma trận
    while (j != 0):
        operations_performed.append(("LEFT", 'None'))
        j -= 1

    # Nếu chúng ta đạt đến cột bên trái của ma trận
    while (i != 0):
        operations_performed.append(("UP", 'None'))
        i -= 1

    # Đảo ngược danh sách thao tác vì đã backtrack
    operations_performed
    return [LCSS, LCSS[len(compare_string_1)][len(compare_string_2)],
operations_performed]
```

Chương trình chính

```
# Nhập chuỗi so sánh 1 và chuỗi so sánh 2
print("Nhập chuỗi so sánh 1: ")
compare_string_1 = input().strip()

print("Nhập chuỗi so sánh 2: ")
compare_string_2 = input().strip()

# Tìm dãy con dài nhất chung và các thao tác thực hiện
LCSS_matrix, distance, operations_performed =
```

```
Find_Longest_Common_Subsequense(compare_string_1, compare_string_2)

# Hiển thị ma trận LCSS
display(pd.DataFrame(LCSS_matrix))

# Hiển thị DataFrame chứa các thao tác thực hiện
display(pd.DataFrame(operations_performed))

# Đếm số lượng các thao tác
lefts, ups, cosses = 0, 0, 0
for i in operations_performed:
    if i[0] == 'LEFT':
        lefts += 1
    elif i[0] == 'UP':
        ups += 1
    else:
        cosses += 1

# Hiển thị kết quả
print("Độ đo chỉnh sửa tối thiểu: {}".format(distance))
print("Số lượng thao tác trái: {}".format(lefts))
print("Số lượng thao tác lên: {}".format(ups))
print("Số lượng thao tác chéo: {}".format(cosses))
print("Tổng số thao tác: {}".format(lefts + ups + cosses))

print("----------------------------------------------------------------
")

# In ra danh sách các thao tác thực hiện
print("Thao tác thực hiện: ")
for i in range(len(operations_performed)):
    if operations_performed[i][0] == "LEFT":
        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("{}) {} : {}".format(i+1, operations_performed[i][0],
operations_performed[i][1]))

print("----------------------------------------------------------------
")

# In ra dãy con dài nhất chung
print("Dãy con dài nhất chung: ")
for i in range(len(operations_performed)):
    if operations_performed[i][0] == "CROSS":
        print("{}".format(operations_performed[i][1]), end='')
```

Đoạn code trên thực hiện các công việc sau:

- Tạo ma trận LCSS (Longest Common Subsequence): Ma trận này có kích thước (len(compare_string_1) + 1) x (len(compare_string_2) + 1). Mỗi ô trong ma trận đại

diện cho một phần của chuỗi kết quả, trong đó hàng đầu tiên và cột đầu tiên đều có giá trị 0.

- Điền các giá trị vào ma trận LCSS: Sử dụng một thuật toán động để điền các giá trị vào ma trận này. Với mỗi ô trong ma trận, nếu ký tự tương ứng của chuỗi so sánh 1 bằng với ký tự của chuỗi so sánh 2, ta tăng giá trị ở ô đó lên 1 so với giá trị nằm ở đường chéo phía trên bên trái. Ngược lại, ta lấy giá trị lớn nhất từ ô phía trên hoặc ô bên trái.
- Backtracking để ghi lại các thao tác thực hiện: Sau khi điền đầy ma trận, ta thực hiện backtracking để ghi lại các thao tác thực hiện. Trong quá trình này, ta đi từ ô cuối cùng của ma trận về ô đầu tiên, và quan sát các giá trị để xác định xem ta đã thực hiện thao tác nào để đạt được giá trị tại ô hiện tại.
- Đếm số lượng các thao tác: Đếm số lượng các thao tác được thực hiện, bao gồm di chuyển sang trái, di chuyển lên, và di chuyển chéo.
- Hiển thị kết quả: In ra ma trận LCSS, số lượng các thao tác, danh sách chi tiết các thao tác thực hiện, cũng như dãy con dài nhất chung giữa hai chuỗi.

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

Định nghĩa hàm tính khoảng cách biến đổi thời gian động

```
def Find_Dynamic_Time_Wraping(time_seriesA, time_seriesB):

    # Tạo ma trận DTW_matrix với kích thước (len(time_seriesA) + 1) x
(len(time_seriesB) + 1)
    DTW_matrix = [[np.inf] * (len(time_seriesB) + 1) for i in
range(len(time_seriesA) + 1)]
    DTW_matrix[0][0] = 0

    # Danh sách ghi lại đường dẫn của quá trình thời gian
    wraping_path = []

    # Xây dựng ma trận theo thuật toán DTW
    for i in range(1, len(time_seriesA) + 1):
        for j in range(1, len(time_seriesB) + 1):
            DTW_matrix[i][j] = abs(time_seriesA[i-1] - time_seriesB[j-
1]) + min(DTW_matrix[i][j-1], DTW_matrix[i-1][j-1], DTW_matrix[i-1][j])

    # Khởi tạo cho việc backtracking
    i = len(time_seriesA)
    j = len(time_seriesB)

    # Backtrack để ghi lại đường dẫn của quá trình thời gian
    wraping_path.append(DTW_matrix[i][j])
    while (i != 0 and j != 0):
        if min(DTW_matrix[i][j-1], DTW_matrix[i-1][j-1], DTW_matrix[i-
1][j]) == DTW_matrix[i][j-1]:
            wraping_path.append(DTW_matrix[i][j-1])
            j -= 1
        elif min(DTW_matrix[i][j-1], DTW_matrix[i-1][j-1],
```

```
DTW_matrix[i-1][j]) == DTW_matrix[i-1][j-1]:
            wraping_path.append(DTW_matrix[i-1][j-1])
            i -= 1
            j -= 1
        else:
            wraping_path.append(DTW_matrix[i-1][j])
            i -= 1

    return [DTW_matrix, DTW_matrix[len(time_seriesA)]
[len(time_seriesB)], wraping_path[:-1]]
```

Chương trình chính

```
# Khởi tạo chuỗi thời gian A và B
time_seriesA = np.array([1, 7, 4, 8, 2, 9, 6, 5, 2, 0])
time_seriesB = np.array([1, 2, 8, 5, 5, 1, 9, 4, 6, 5])

# Tính toán DTW giữa hai chuỗi thời gian
DTW_matrix, distance, wraping_path =
Find_Dynamic_Time_Wraping(time_seriesA, time_seriesB)

# Hiển thị ma trận DTW
print("Ma trận DTW:")
for row in DTW_matrix:
    print(row)

# In ra đường dẫn của quá trình thời gian
print("\nĐường dẫn của quá trình thời gian:")
for i in wraping_path:
    print(i, end=', ')
```

Đoạn code trên thực hiện các công việc sau:

- Tạo ma trận DTW_matrix: Tạo ma trận DTW_matrix có kích thước (len(time_seriesA) + 1) x (len(time_seriesB) + 1). Mỗi ô trong ma trận này đại diện cho một phần của quá trình thời gian tối thiểu, với mục tiêu là làm cho hai chuỗi thời gian gần nhất có thể. Các giá trị ban đầu được đặt là vô cực (inf) ngoại trừ ô ở góc trên bên trái (0) đại diện cho điểm xuất phát.
- Tính toán giá trị của ma trận DTW: Sử dụng một vòng lặp kép để điền các giá trị vào ma trận DTW_matrix theo thuật toán Dynamic Time Warping (DTW). Cụ thể, giá trị tại mỗi ô (i, j) được tính bằng cách lấy giá trị tuyệt đối của sự khác biệt giữa các phần tử tương ứng của hai chuỗi thời gian, cộng với giá trị nhỏ nhất giữa các ô xung quanh (i, j).
- Backtracking để ghi lại đường dẫn của quá trình thời gian: Sau khi điền đầy ma trận, ta thực hiện backtracking từ ô cuối cùng (len(time_seriesA), len(time_seriesB)) về ô (0, 0) để ghi lại đường dẫn của quá trình thời gian tối thiểu. Trong quá trình này, ta lưu giữ giá trị của mỗi ô mà ta đi qua.
- Trả về kết quả: Trả về ma trận DTW, giá trị của quá trình thời gian tối thiểu và đường dẫn của quá trình thời gian.