Sửa bài WECODE

Bài 1. Palindrome:

```python
def minPalPartion(str):

    n = len(str)
    C = [[0 for i in range(n)]
            for i in range(n)]
    P = [[False for i in range(n)]
                for i in range(n)]
    j = 0
    k = 0
    L = 0
    for i in range(n):
        P[i][i] = True
        C[i][i] = 0
    for L in range(2, n + 1):
        for i in range(n - L + 1):
            j = i + L - 1
            if L == 2:
                P[i][j] = (str[i] == str[j])
            else:
                P[i][j] = ((str[i] == str[j]) and
                           P[i + 1][j - 1])
            if P[i][j] == True:
                C[i][j] = 0
            else:

                C[i][j] = 100000000
                for k in range(i, j):
                    C[i][j] = min (C[i][j], C[i][k] +
                                   C[k + 1][j] + 1)

    return C[0][n - 1]

# Driver code
str = input()
print (minPalPartion(str))
```

- giải thích đoạn mã:
    - Mảng C [n x n] được lưu số lần cắt ít nhất để tạo thành các đoạn con palindrome từ vị trí i đến vị trí j trong chuỗi str.
    - Mảng P [n x n] và được sử dụng để kiểm tra xem đoạn con từ vị trí i đến vị trí j có phải là một palindrome hay không.
    - Biến j, k, L được sử dụng để duyệt và lưu trữ các giá trị tạm thời trong quá trình tính toán.

- Sử dụng hai vòng lặp for để duyệt qua tất cả các đoạn con có độ dài từ 2 đến n.
- Trong mỗi vòng lặp, vị trí bắt đầu i và vị trí kết thúc j được xác định.
- Nếu đoạn con có độ dài là 2 (L == 2), kiểm tra xem hai ký tự ở hai đầu có giống nhau không (str[i] == str[j]).
- Nếu đoạn con có độ dài lớn hơn 2 (L > 2), kiểm tra xem hai ký tự ở hai đầu có giống nhau không và đoạn con bên trong (P[i + 1][j - 1]) cũng là một palindrome.
- Nếu đoạn con từ vị trí i đến vị trí j là một palindrome (P[i][j] == True), số lần cắt ít nhất tại vị trí đó sẽ là 0.
- Nếu không, sử dụng một vòng lặp for thứ ba để tìm vị trí k trong khoảng từ i đến j - 1 và tính toán số lần cắt ít nhất tại vị trí i và j bằng cách tìm số lần cắt ít nhất tại vị trí k cộng thêm 1.
- Lưu số lần cắt ít nhất tại vị trí i và j vào mảng C.
- Kết quả là giá trị tại vị trí C[0][n - 1], tức là số lần cắt ít nhất để tạo thành các đoạn con palindrome trong chuỗi str.

- Công thức truy hồi:

  (L là độ dài của đoạn con đang xét ($2 \le L \le n$))

  + Nếu L = 2:

  P[i][j] = (str[i] == str[j]) (Kiểm tra xem hai ký tự ở hai đầu có giống nhau không)

  + Ngược lại, nếu L > 2:

  P[i][j] = ((str[i] == str[j]) and P[i + 1][j - 1]) (Kiểm tra xem hai ký tự ở hai đầu có giống nhau không và đoạn con bên trong có là một palindrome không)

  + Sau khi xác định P[i][j], công thức truy hồi để tính C[i][j] là:

  + Nếu P[i][j] = True :

  C[i][j] = 0 (không cần cắt)

  + Ngược lại, P[i][j] = False:

  C[i][j] = min(C[i][k] + C[k + 1][j] + 1) (với k trong khoảng từ i đến j - 1)

  Trong công thức trên, C[i][j] được tính bằng cách tìm số lần cắt ít nhất tại vị trí k để tạo thành hai đoạn con palindrome C[i][k] và C[k + 1][j], và cộng thêm 1 lần cắt ở vị trí k. Công thức truy hồi này được áp dụng cho tất cả các giá trị của i và j trong phạm vi cho trước (từ i = 0 đến n - L và j = i + L - 1).

Nhận xét các nhóm:

- Nhóm làm đúng: Nhóm 2,3,5,6,7,8,9,12,13,15
- Tham khảo code Nhóm 6, Nhóm 7:

```python
def min_cuts_palindrome(s):
    n = len(s)

    dp = [float('inf')] * (n + 1)
    dp[0] = -1

    is_palindrome = [[False] * n for _ in range(n)]

    for i in range(n):
        for j in range(i, -1, -1):
            if s[i] == s[j] and (i - j < 2 or is_palindrome[j + 1][i - 1]):
                is_palindrome[j][i] = True
                dp[i + 1] = min(dp[i + 1], dp[j] + 1)

    return dp[n]

s = input()

result = min_cuts_palindrome(s)
print(result)
```

-
-
-

Bài 2. Build Bridges (Tương tự bài toán tìm xâu con chung dài nhất – longest common subsequence):

```python
A = input()
B = input()
A = list(map(int, A.split()))
B = list(map(int,B.split()))

def lcs(x, y):
    m = len(x)
    n = len(y)

    # Khởi tạo bảng lcs với giá trị 0
    lcs_table = [[0 for j in range(n+1)] for i in range(m+1)]

    # Tính toán độ dài xâu con chung dài nhất
    for i in range(1, m+1):
        for j in range(1, n+1):
            if x[i-1] == y[j-1]:
                lcs_table[i][j] = lcs_table[i-1][j-1] + 1
            else:
                lcs_table[i][j] = max(lcs_table[i-1][j], lcs_table[i][j-1])

    # Trả về độ dài xâu con chung dài nhất
    return lcs_table[m][n]

print(lcs(A,B))
```

- Sử dụng hai vòng lặp for để tính toán độ dài của LCS. Với mỗi vị trí (i, j) trong bảng lcs_table, nếu phần tử thứ i của danh sách x bằng phần tử thứ j của danh sách y, thì cập nhật theo công thức.
- Trả về giá trị lcs_table[m][n], đó chính là độ dài của LCS giữa hai danh sách A và B.

Công thức truy hồi:

if X[i-1] == Y[j-1]:

L[i][j] = L[i-1][j-1] + 1

else:

L[i][j] = max(L[i-1][j], L[i][j-1])

*) Nhận xét các nhóm:

- Nhóm 2,3,5,6,7,8,9,12,13 làm đúng
- Nhóm 14 sai hướng