

Sửa bài WECODE

Bài 1. Palindrome:

```
Chương 10-DP2 > palindrone.py / ...
1  def minPalPartion(str):
2
3      n = len(str)
4      C = [[0 for i in range(n)]
5            for i in range(n)]
6      P = [[False for i in range(n)]
7            for i in range(n)]
8
9      j = 0
10     k = 0
11     L = 0
12     for i in range(n):
13         P[i][i] = True
14         C[i][i] = 0
15     for L in range(2, n + 1):
16         for i in range(n - L + 1):
17             j = i + L - 1
18             if L == 2:
19                 P[i][j] = (str[i] == str[j])
20             else:
21                 P[i][j] = ((str[i] == str[j]) and
22                           P[i + 1][j - 1])
23             if P[i][j] == True:
24                 C[i][j] = 0
25             else:
26                 C[i][j] = 100000000
27                 for k in range(i, j):
28                     C[i][j] = min (C[i][j], C[i][k] +
29                                     C[k + 1][j] + 1)
30
31     return C[0][n - 1]
32
33 # Driver code
34 str = input()
35 print (minPalPartion(str))
36
```

- giải thích đoạn mã:
 - o 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.
 - o 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.
 - o 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 ($2 \leq L \leq n$))
- + 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 $L > 2$:
- $P[i][j] = ((str[i] == str[j]) \text{ and } P[i + 1][j - 1])$** (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 2,3,5,8,9,12,13,15 : 4.5 điểm
- Các bạn có thể tham khảo code của Nhóm 6, Nhóm 7: 5 điểm

```
1  def min_cuts_palindrome(s):  
2      n = len(s)  
3  
4      dp = [float('inf')] * (n + 1)  
5      dp[0] = -1  
6  
7      is_palindrome = [[False] * n for _ in range(n)]  
8  
9      for i in range(n):  
10         for j in range(i, -1, -1):  
11             if s[i] == s[j] and (i - j < 2 or is_palindrome[j + 1][i - 1]):  
12                 is_palindrome[j][i] = True  
13                 dp[i + 1] = min(dp[i + 1], dp[j] + 1)  
14  
15         return dp[n]  
16  
17 s = input()  
18  
19 result = min_cuts_palindrome(s)  
20 print(result)
```

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

```

1 A = input()
2 B = input()
3 A = list(map(int, A.split()))
4 B = list(map(int, B.split()))
5
6 def lcs(x, y):
7     m = len(x)
8     n = len(y)
9
10    # Khởi tạo bảng lcs với giá trị 0
11    lcs_table = [[0 for j in range(n+1)] for i in range(m+1)]
12
13    # Tính toán độ dài chuỗi con chung dài nhất
14    for i in range(1, m+1):
15        for j in range(1, n+1):
16            if x[i-1] == y[j-1]:
17                lcs_table[i][j] = lcs_table[i-1][j-1] + 1
18            else:
19                lcs_table[i][j] = max(lcs_table[i-1][j], lcs_table[i][j-1])
20
21    # Trả về độ dài chuỗi con chung dài nhất
22    return lcs_table[m][n]
23
24 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: 5 điểm
- Nhóm 14 sai hướng: 3 điểm: [không áp dụng phương pháp dynamic

programming trực tiếp. Dù sử dụng một số bước lặp và tạo một ma trận lưu trữ thông tin, nhưng không có một quy luật truy hồi rõ ràng, bước truy hồi không được áp dụng để tính kết quả cuối cùng]