Tutorial:

* Subtask 1: Với mỗi truy vấn, sử dụng biến ans để lưu kết quả cho truy vấn. Sử dụng vòng lặp để lấy các xâu con liên tiếp của xâu s[l,r], rồi sử dụng vòng lặp để kiểm tra xem các xâu con liên tiếp có phải là palindrome hay không. Nếu có thì ans + 1. In kết quả sau khi kiểm tra hết.

Độ phức tạp: O(n^4): Chúng ta có n truy vấn, mỗi truy vấn cần 2 vòng for để lấy các xâu con liên tiếp trong đoạn l, r mất n^2, và kiểm tra mỗi xâu con có phải là palindrome hay không mất n. n \* n ^ 2 \* n = n^4

Code: ngot\_task1.cpp

* Subtask 2: Mỗi truy vấn, sử dụng biến ans để lưu kết quả. Sử dụng vòng lặp để lấy các xâu con liên tiếp của xâu s[l,r], nhưng thay vì sử dụng vòng lặp để kiểm tra xem xâu con có phải là palindrome hay không thì chúng ta có thể sử dụng hashing để giảm thời gian.

Độ phức tạp: O(n^3): Chúng ta có n truy vấn, mỗi truy vấn cần 2 vòng for để lấy các xâu con liên tiếp trong đoạn l, r mất n^2, kiểm tra xâu palindrome bằng hash O(1). n \* n ^ 2 = n^3.

Code: ngot\_task2.cpp

* Subtask 3: Nên có một tiền xử lí để có thể trả lời truy vấn trong O(n^2). Đầu tiên tại mỗi vị trí của sâu s, chúng ta sử dụng một mảng đếm, để đếm xem từ vị trí i trở về sau có bao nhiêu xâu con là xâu palindrome bắt đầu từ i, để kiểm tra thì chúng ta sử dụng hashing. Với mỗi truy vấn khi chúng ta sử dụng vòng lặp, duyệt i từ l-> r. Với mỗi lần lặp chúng ta cộng vào biến ans số lượng xâu con liên tục là palindrome có vị trí bắt đầu là i trong khoảng [i,r].

Độ phức tạp : O(n^2)

Code: ngot\_task3.cpp

* Subtask 4: Sử một mảng dp[n][n]. Với ý nghĩa dp[i][j] là số lượng palindrome trong khoảng [i,j]. Công thức tính. Với i != j: dp[i][j] = dp[i + 1][j] + dp[i][j-1] – dp[i+1][j-1] + (s[i:j] là palidrome hay không). i == j: dp[i][j] = 1

Độ phức tạp: O(n^2) khi khởi taọ mảng dp[n][n]. Truy vấn O(n), mỗi truy vấn O(1).

Code: ngot\_task4.cpp