Let $S$ be a string of $n$ characters. Describe an $O(n^2)$ algorithm to compute the length of the longest substring that appears both forwards and backwards in $S$. The forward and backward substrings should not overlap.

For example, consider the following string

$$S = \mathsf{BOBSMAMASEESAUKULELE}.$$

The algorithm should return 3 with the substring ASE.

**Hint.** *This is very similar to the palindrome problem from tutorials...*

**Solution.** We use a Dynamic Programming approach by initialising an $n \times n$ array, $DP$, initialised as $0$, which still store the length of the longest common substring ending at index $i$ in the forward direction, and index $j$ in the backward direction. We will iterate through the list in two nested loops, $(i, j)$. For each:

- If $S[i] = S[j]$ and $i + j < n - 1$, set $DP[i][j] = 1 + DP[i-1][j+1]$. To ensure the substrings don't overlap, we check $i + j < n - 1$.
- Otherwise, set $DP[i][j]$ to $0$.

We then retrieve the maximum value from $DP$ and return the desired information once it is found. We iterate through the string over two nested loops, so clearly our time complexity is $O(n^2)$.