2. Dynamic Programming

This problem will walk you through the steps of designing a dynamic program. The problem we are solving is the longest palindrome problem: given a string, S, what is the length of the longest palindrome that is a substring of S? A palindrome is a string that reads the same forwards as backwards. For example, "racecar", "eve", and "I", are all palindromes. We also consider the empty string to be a palindrome (of length 0).

(a) First we need to figure out what our subproblems are. Since we are working with strings, a natural subproblem to use is substrings. Let $OPT(i, n)$ denote the length of the longest palindrome in the substring of length n starting at index i. Write an expression for the recursive case of $OPT(i, n)$. (Hint: All palindromes above a certain size have palindromes as substrings).

$$OPT(i, n) = \begin{cases} 2 + OPT(i+1, n-2) & \text{if } S_i = S_{i+n-1} \\ MAX(OPT(i, n-1), OPT(i+1, n-1)) & \text{otherwise} \end{cases}$$

(b) Next we need a base case for our OPT recurrence. Write an expression for the base case(s) of this recurrence. (Hint: Which size strings are always palindromes?)

$OPT(i, 0) = 0$ $(n=0)$ // empty string

$OPT(i, 1) = 1$ $(n=1)$ // at least 1 character

(c) Now that we have a complete recurrence, we need to figure out which order to solve the subproblems in. Which subproblems does the recursive case $OPT(i, n)$ require to be calculated before it can be solved?

$OPT(i, n) = 2 + OPT(i+1, n-2)$ if $S_i = S_{i+n-1}$

// the first character = the last character

(d) Given these dependencies, what order should we loop over the subproblem in?

1. $OPT(i, n) = 0$ $n = 0$
2. $OPT(i, n) = 1$. $n = 1$
3. $OPT(i, n) = 2 + OPT(i+1, n-2)$ if $S_i = S_{i+n-1}$
4. $OPT(i, n) = MAX(OPT(i, n-1), OPT(i+1, n-1))$ otherwise

(e) We have all of the pieces required to put together a dynamic program now. Write psuedocode for the dynamic program that computes the length of the longest palindromic substring of S.

```
OPT(i, n) {
    if (n = 0 ), return 0          // empty string
else if (n = 1) , return 1         // at least 1 character
else if (S[i] = S[i+n-1]), return 2+ OPT(i+1, n-2)
else    MAX( OPT(i, n-1), OPT(i+1, n-1))
}
```