Stack Construction Problem

This is meant to be a brief explanation of the recurrence for the StackConstruction problem, discussed in the seminar. The variable names here reflect the (very simple) names in the accompanying code.

Here we are given a string $w = w_1 w_2 \cdots w_n$ and want to find the fewest number of stack operations (*push*, *print*, and *pop*) to print the entire string as discussed in the problem description.

The subproblem, $s(i, j)$, will be the fewest number of stack operations to print $w_i w_{i+1} \cdots w_j$, as always starting with an empty stack and ending with an empty stack. Thus the desired output will be $s(1, n)$.

comments:

- A single character (alone) needs 3 operations (*push*, *print*, *pop*), so $s(i, i) = 3$.
- We also need a base case of $s(i + 1, i) = 0$ for the empty string.
- It is always possible to process the first character of the substring alone (3 operations) then handle the rest independently, so $s(i, j)$ has $3 + s(i + 1, j)$ as a possible solution.
- Suppose $w_i = a$ and another a appears later in the substring, say at position k (so $w_k = w_i = a$ with $i < k \leq j$) ...
- ... here think of the substring as $a w_{i+1} \cdots w_{k-1} a w_{k+1} \cdots w_j$, and now the idea is that we could reuse the a at position k ...
- ... that is, *push* a, *print* a, process $w_{i+1} \cdots w_{k-1}$ on top of the a, then deal with the a at position k.
- The key point for the recurrence is that we will not charge $w_i = a$ for a *push/pop*, but defer that to the last character a that is used to match with it.
- Therefore, another solution to $s(i, j)$ is
  1. 1 (to print w_i) plus,
  2. $s(i + 1, k - 1)$ (to process $w_{i+1} \cdots w_{k-1}$ on top of the a) plus,
  3. $s(k, j)$, to process $a w_{k+1} \cdots w_j$ (the *push/pop* costs for a would be paid here or deferred to an even later a)

  Combining the above we get

$$
s(i,j) = \begin{cases} 0 & (i > j) \\ 3 & (i = j) \\ & \\ \textit{minimum of} & (i < j) \\ 3 + s(i+1, j) & \textit{and} \\ 1 + s(i+1, k-1) + s(k, j) & \text{for all } k \ (i < k \leq j) \text{ where } w_i = w_k \end{cases}
$$

To think about coding this, note that the subproblems $s(i, j)$ get harder as $d = j - i$ get larger (the substrings get longer). Therefore we should compute them in that order.

Pseudo-pseudo-Code:

```
create array s

initialize all s[i,i]=3, s[i+1,i]=0 as the base cases

for d=1 to n-1
     for i=1 to n-d
          j=i+d
          s(i,j) = minimum of the last two lines in the recurrence above

return s[1,n]
```

Time is going to be $O(n^3)$ using $O(n^2)$ space.