CS375 Notes: Dynamic Programming

4^{th} November, 2026

1. Palindrome

Given: a string $c_1c_2...c_n$

Output: a max length palindrome substring

Attempt 1

$OPT(i)$ = the max length palindrome from the first i characters

$OPT(i) = \max(2 + OPT(i-1)$ // if c_i is included and $c_j = c_i$, $OPT(i-1))$

Problem: no information about the string in between, not a good definition.

Attempt 2

$OPT(i,j)$ = the maximal length palindrome between $c_i, \ldots, c_j$

$$OPT(i,j) = \max \begin{cases} OPT(i+1, j-1) + 2 & \text{// only if } c_i = c_j \text{ and if } j - i - 1 = OPT(i+1, j-1) \\ OPT(i+1, j) \\ OPT(i, j-1) \end{cases}$$

Base cases:

If $j < i$: $OPT(i,j) = 0$

$OPT(i,i) = 1$ for all i

Pseudocode

```
Initialize W as an n x n array, where every entry is initialized to zero.
for i = 1 ... n:
    W[i, i] = 1

for j = i+1 ... n:
    W[i, j] = max(W[i, j-1], W[i+1, j])
    if c_i == c_j and j - i - 1 == W[i+1, j-1]:
        W[i, j] = max(W[i, j], W[i+1, j-1] + 2)
```

2. Longest common subsequence

Given: 2 sequences $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$

Output: the longest subsequence that is a (substring) subsequence of both

Example: A B C D and A D C B A D

Attempt 1

$OPT(i)$ = the longest common subsequence of the first i of each

$OPT(i) = \max(OPT(i-1) + 1$ // include i if $x_i = y_i$, $OPT(i-1)$ // don't include $i)$

Problem: no index tracking.

Attempt 2

$OPT(i, j)$ = the LCS of $(x_1, \ldots, x_i)$ and $(y_1, \ldots, y_j)$

$$OPT(i,j) = \max \begin{cases} OPT(i-1, j-1) + 1 & \text{// include if } x_i = y_j \\ OPT(i-1, j) & \text{// not include } i \\ OPT(i, j-1) & \text{// not include } j \end{cases}$$

Base case:

$OPT(0, j) = OPT(i, 0) = 0$

Pseudocode

```
Let W be an (n+1) x (m+1) array
Time: Theta(n x m)

for i = 0 ... n:
    W[i, 0] = 0
for j = 0 ... m:
    W[0, j] = 0

for i = 1 ... n:
    for j = 1 ... m:
        W[i, j] = max(W[i, j-1], W[i-1, j])
        if x_i == y_j:
            W[i, j] = max(W[i, j], W[i-1, j-1] + 1)
```

Backtracking

```
Output: []
i = n, j = m
while (j > 0 and i > 0):
    if W[i, j] == W[i, j-1]:
        j = j - 1       // ignore y_j
    else if W[i, j] == W[i-1, j]:
        i = i - 1       // ignore x_i
    else:
        add x_i (or y_j) to output
        i = i - 1
        j = j - 1

return output
```