Scribe Notes: Dynamic Programming

Topics: Longest Palindromic Subsequence & Longest Common Subsequence

1 Palindrome Problem

1.1 Problem Statement

- **Given:** A string $c_1 c_2 \dots c_n$.
- **Output:** The maximum length of a palindromic substring / subsequence.

1.2 Attempt 1 (1D Dynamic Programming)

We first try defining subproblems using a single index:

$$\text{Opt}(i) = \text{“The maximal length palindrome from just the first } i \text{ characters”}$$

Attempted recurrence:

$$\text{Opt}(i) = \max \begin{cases} \text{Opt}(i-1) \\ 2 + \text{Opt}(i-1) \quad \text{if } c_1 = c_i \text{ (Doesn't work)} \end{cases}$$

Note: This formulation fails because checking palindromic structure requires knowing both boundaries (start and end) of the substring.

1.3 Correct Formulation (2D Dynamic Programming)

We introduce two indices (i, j) to represent intervals:

$$\text{Opt}(i,j) = \text{“The maximal length palindrome between } c_i, \dots, c_j\text{”}$$

1.3.1 Recurrence Relation

$$\text{Opt}(i,j) = \max \begin{cases} \text{Opt}(i, j-1) \\ \text{Opt}(i+1, j) \\ \text{Opt}(i+1, j-1) + 2 \quad \text{if } c_i = c_j \text{ and } (j-i-1) = \text{Opt}(i+1, j-1) \end{cases}$$

1.3.2 Base Cases

- $\text{Opt}(i,j) = 0$ if $j < i$
- $\text{Opt}(i,i) = 1$ for all $1 \le i \le n$

1.4 Algorithm (Iterative DP Table Construction)

Initialize W as an $n \times n$ array where every entry is initialized to 0.

```
W[n, n] ← 1
for i = n − 1 down to 1 do
    W[i, i] ← 1
    for j = i + 1 to n do
        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] then
            W[i, j] ← max(W[i, j], W[i + 1, j − 1] + 2)
        end if
    end for
end for
```

---

2 Longest Common Subsequence (LCS)

2.1 Problem Statement

- **Given:** Two sequences $x_1, \ldots, x_n$ and $y_1, \ldots, y_m$.
- **Output:** The longest sequence that is a subsequence of both.

2.2 Formulating Subproblems

First Try: Opt(i) = The LCS of the first i of each.

$$\text{Opt}(i) = \max\{\ldots\} \quad \text{Doesn't work (not flexible enough).}$$

Correct Formulation: Let Opt(i, j) be the LCS of prefixes $x_1, \ldots, x_i$ and $y_1, \ldots, y_j$:

$$\text{Opt}(i,j) = \max \begin{cases} \text{Opt}(i-1, j) \\ \text{Opt}(i, j-1) \\ \text{Opt}(i-1, j-1) + 1 \quad \text{if } x_i = y_j \end{cases}$$

2.2.1 Base Cases

$$\text{Opt}(0, j) = 0, \quad \text{Opt}(i, 0) = 0 \quad \forall\, i, j$$

Note: We store these base cases by allocating a W matrix of size $(n+1) \times (m+1)$ to store the $[i, 0]$ and $[0, j]$ entries.

2.3 Traceback (Reconstructing the Solution)

To compute the actual sequence after filling table W:

```
output ← []
i ← n,   j ← m
while i > 0 and j > 0 do
    if W[i, j] == W[i − 1, j] then
        i ← i − 1
    else if W[i, j] == W[i, j − 1] then
```

```
        j ← j − 1
    else
        Add x_i to output                                   ▷ Or equivalently y_j
        i ← i − 1
        j ← j − 1
    end if
end while
return output
```

Complexity: Total running time for LCS is $\Theta(nm)$.