Dynamic Programming

- Define subproblems.
- Define recursive formula to solve subproblems.
- (Remember & Reuse) Memoization $\leftrightarrow$ Table.
  - Evaluate formula by memoization or Bottom-up using the table.

Ex 0: Evaluate, given $n$, $0 \leq k \leq n$

$$C(n, k) = C(n-1, k-1) + C(n-1, k) \quad \text{if } 0 < k < n$$

$$= 1 \quad \text{if } k = 0 \text{ or } k = n$$

Naive Implementation:

$$T(n) = 2T(n-1) + O(1)$$

$$\Rightarrow O(2^n)$$

$\# \text{distinct subprob} = O(1+2+3\ldots+n-1)$

$$= O(n^2)$$

$\begin{array}{c}
\text{Directed acyclic graph (DAG)} \downarrow \\
C(n, k) \quad \text{if } 0 \leq k \leq n \\
\end{array}$

$$C(n, 0) \leftarrow 1$$

$$C(n, n) \leftarrow 1$$

$$C(n-1, k) \leftarrow 2$$

$$C(n-2, k-2) \leftarrow 3$$

$$\vdots$$

$$n=4, \quad k=2 \quad C(4, 2)$$

$\begin{array}{cccc}
0 & 1 & 2 & 3 & 4 \\
C(4, 0) & C(4, 1) & C(4, 2) & C(4, 3) & C(4, 4) \\
\end{array}$
Pascal's Triangle.

Evaluation order smallest to largest value from each $n$ to $K$.

**Pseudo code:**

For $i = 1$ to $n$
  $c[i, 0] = 1$, $c[i, i] = 1$

For $j = 1$ to $i - 1$
  $c[i, j] = c[i-1, j-1] + c[i-1, j]$

Return $c[n, k]$.

$O(n^3)$ time.

$O(n^2)$ space. Can be improved $O(n)$ by only storing $(i-1)k$ to $ik$ row.

"Real" Ex. 1: Longest Common Subsequence (LCS).

Given two seq. $A = a_1 a_2 \ldots a_m$
$B = b_1 b_2 \ldots b_m$

$C(n, m)$
Find longest subseq. of A that is also a subseq of B.

**eg.**

```
ALGORITHM
LOGARITHM
```

```
LORITHM 8 size 7
LGRITHM 9 size 7
```

**Applying check similarity of two string (number of deletions/insertions)**

UNIX `diff`

⇒ Define subproblems.

```
C(i,j) = length of LCS of
        a_i,..a_j & b_i,..b_j
```

Where:
```
i = 0..n, j = 0..m
```

Answer: 
```
C(n,m) = 
```

Base case:
```
i = 0 or j = 0 & B = φ
```
```
C(0,j) = 0, C(i,0) = 0
```
```
A = φ
```

⇒ Recursive Formula: 

```
C(i,j)
```

3 cases:
```
1. Not take a_i → C(i-1,j)
2. Not take b_i → C(i,j-1)
3. Take a_i & b_j when a_i = b_j
   → C(i-1,j-1) + 1
```
```
     a_1...a_i
     b_1...b_j
```
```
   ...  •  s  (i,j)  C(i,j) = 2
```
```
         a_i ≠ b_j
```

⇒ Intuition: suppose \( a_m = b_m \).
Then LCS(a_1...a_{m-1}, b_1...b_{m-1}) + 1.
\[ C(i, j) = \begin{cases} \max \{ C(i-1, j), C(i, j-1) \} & \text{if } a_i \neq b_j \\ \max \{ C(i-1, j), C(i, j-1), C(i-1, j-1) \} & \text{if } a_i = b_j \end{cases} \]

Pseudo code:

\textbf{Exe.}

Analysis:

Each \( O(1) \) time \( \Rightarrow O(mn) \) time

\( O(mn) \) space

\( \Rightarrow \) can be improved to \( O(n) \)

by storing two rows at a time when only want length of NOT the actual LCS.

\textbf{Ex2: Given string } \( x = a_1 a_2 \ldots a_n \)

split \( x \) into max # of palindromes.

\[ \overline{AAAAA} \cdot \overline{1} \]
Define subproblem:

\[ C(i) = \min_{q_i} \# \text{palindromes} \]

for splitting \( a_1 \ldots a_i \)

Answer: \( C(n) \)

Base case: \( C(0) = 0 \)

Recursive Formula:

\[
\begin{align*}
\text{cases for opt sol'n} \\
\text{it is last palindrome } a_j \ldots a_i \\
\Rightarrow C(j-1) + 1
\end{align*}
\]

\[ C(i) = \min_{j=1 \to i} \left( C(j-1) + 1 \right). \]

Since \( a_j \ldots a_i \) is a palindrome.

Evaluate in increasing order of \( i \).

Pseudocode:

\[
C[0] = 0
\]

\[
\text{for } i = 1 \to n \\
\quad C[i] = 0
\]

\[
\text{for } j = 1 \to i \\
\quad \text{if } a_j \ldots a_i \text{ is a palindrome} \\
\quad \quad \text{or } \quad C[i] + 1 < C[j-1] + 1 \\
\quad \quad \text{then } C[i] = C[j-1] + 1.
\]

\( O(n^3) \) time.
\[ O(n^3) \text{ time.} \]
\[ \text{Return } C[i] . \]

\[ \text{Then } \]
\[ C[i] = C[i-1] + 1 ; \]
\[ \text{Pred}[i] = j ; \]

→ Analysis:
\[ O(n) \text{ subproblems.} \]  \[ \Rightarrow \text{Total } O(n^3) \text{ time.} \]
\[ \text{each takes } O(n^2) \text{ time.} \]
\[ O(n) \text{ space.} \]

→ Psedocode to output the answer:

\[
\text{OutputAns}(i) \\
\text{if } i=0 \text{ then return;} \\
\text{i = Pred[i]} \\
\text{OutputAns}(i-1) \\
\text{Print aj...ai // palindrome}
\]