Dynamic Programming

- define subproblems
- derive recursive formula to solve subproblems
- evaluate formula by memoization or bottom-up with table

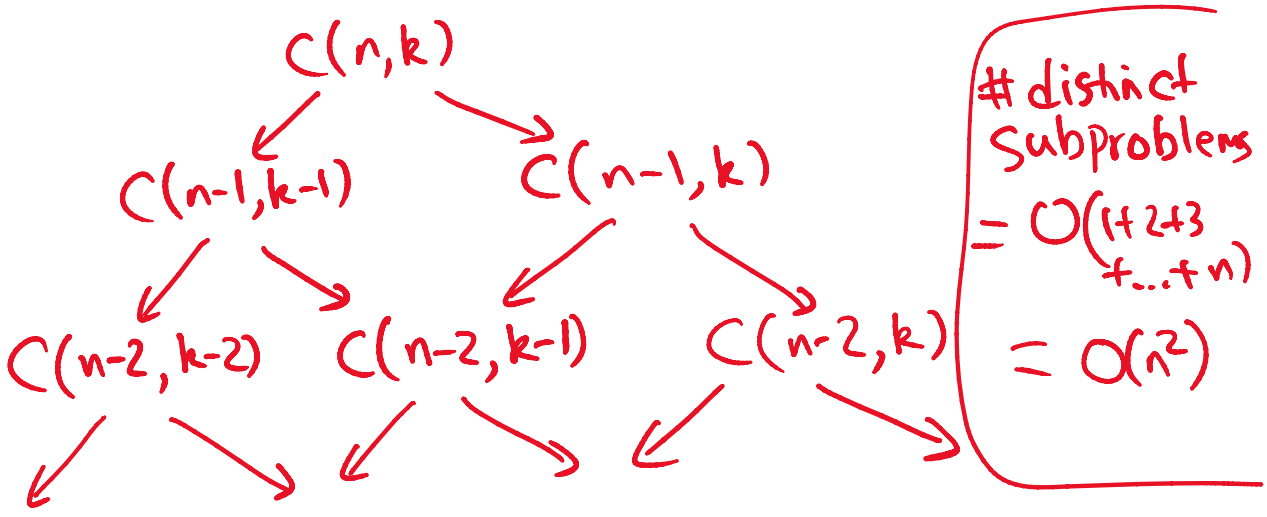
Ex 0 evaluate (binomial coefficients)

$$C(n,k) = \begin{cases} C(n-1,k-1) + C(n-1,k) & \text{if } 0 < k < n \\ 1 & \text{if } k = 0 \text{ or } k = n \end{cases}$$

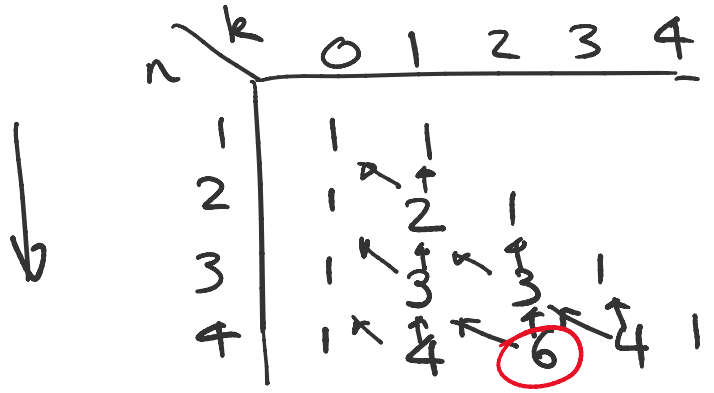
Naive recursion:

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

$$\Rightarrow O(2^n) \text{ time}$$



Memoization: (bottom-up vers.)



C(4,2)

(Pascal's triangle)

evaluate in increas. order of i

Pseudocode:

```
for i = 1 to n
    C[i,0] = 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]
```

⇒ $O(n^2)$ time

$O(n^2)$ space → can be reduced to $O(n)$ (keep track of only last 2 rows)

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

Given 2 sequences $A = a_1 \cdots a_m$ & $B = b_1 \cdots b_n$.

Given 2 sequences $A = a_1 \cdots a_m$
& $B = b_1 \cdots b_n$,

find longest subsequence of A that is also a subsequence of B.

e.g. ALGORITHM
LOGARITHM

LGRITHM
length 7
LORITHM

(appl. similarity of strings
* # deletes/inserts
UNIX diff)

Define subproblems: $(i=0,..,m,\ j=0,..,n)$

$C(i,j)$ = length of LCS of $a_1 \cdots a_i$ & $b_1 \cdots b_j$

Answer: $C(m,n)$.

Base cases: $C(i,0) = 0 \quad i=0,..,m$
$C(0,j) = 0 \quad j=0,..,n$

Recursive formula:

3 cases for opt sol'n for $C(i,j)$:

1. not use a_i $\longrightarrow$ $C(i-1, j)$
2. not use b_j $\longrightarrow$ $C(i, j-1)$
3. use a_i & b_j $(a_i = b_j)$

$\longrightarrow$ $C(\cdots) + 1$

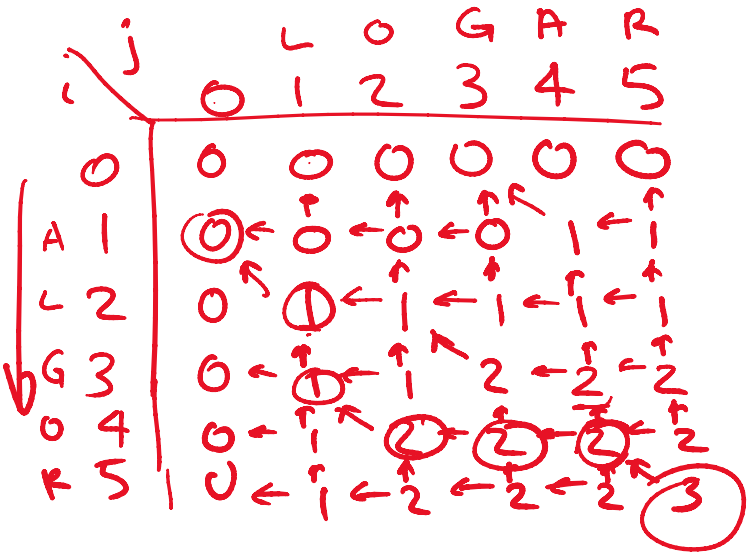
3. use a_i & b_j $(a_i = b_j)$

$\longrightarrow C(i-1, j-1) + 1$

$$C(i,j) = \begin{cases} \max\{C(i-1,j), C(i,j-1)\} & \text{if } a_i \neq b_j \\ \max\{C(i-1,j), C(i,j-1), C(i-1,j-1)+1\} & \text{if } a_i = b_j \end{cases}$$

e.g. ALGOR
LOGAR

ans = 3



C(3,4) = length of LCS of ALG & LOGA = 2

LOR ←

evaluate in increas order of i & for equal i, increas. order of j

Analysis: $O(mn)$ subproblems each $O(1)$ time

$\Rightarrow$ $\boxed{O(mn)}$ time

$O(\quad)$ space

$\Rightarrow$ [O(mn)] ...

O(mn) space

Rmk: edit distance (min # of insert/delete/change chars) is similar

---

Ex2: Given string $x = a_1 a_2 \cdots a_n$, split x into min # of palindromes

e.g. 0 1 1 0 1 1 0 0 1 1

Define subproblems: $(i = 0, \ldots, n)$

$C(i)$ = min # of palindromes for splitting $a_1 a_2 \cdots a_i$

Answer: $C(n)$.

Base case: $C(0) = 0$.

Recursive formula:

Cases for opt sol'n:

last palindrome used is $a_j \cdots a_i$.

$\rightarrow C(j-1) + 1$

$$C(i) = \min_{\substack{j \in \{1,\ldots,i\} \\ \text{s.t. } a_j \cdots a_i \text{ is}}} \left(C(j-1) + 1 \right).$$

$j \in \{1, \ldots, i\}$
s.t. $a_j \ldots a_i$ is
a palindrome

evaluate in increas. order of i

Pseudocode:

```
C[0] = 0
for i = 1 to n do
    C[i] = ∞
    for j = 1 to i                    O(n) time
        if (aj ... ai is a palindrome)
            C[j-1]+1 < C[i]
        then C[i] = C[j-1]+1, pred[i]=j
return C[n].
```

Analysis

$O(n)$ subproblems
each requiring $O(n^2)$ time
$\Rightarrow$ $\boxed{O(n^3)}$ total time
$O(n)$ space

```
OutputAns(i):
    if i=0 return
    OutputAns(pred[i]-1)
    output a_pred[i] ... a_i.
```

$O(n)$ extra time

Call OutputAns(n).