Thursday, March 3, 2022 10:47 AM

Dynamic Programming

- → Define subproblems.
- → Define recursive formula to solve subproblems
- → (Remember & Reuse) Memoization ↔ Table

Evaluate formula by memoization

OR Bottom-up using the table.

Ex 0: Evaluate, given n, $0 \le K \le n$

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

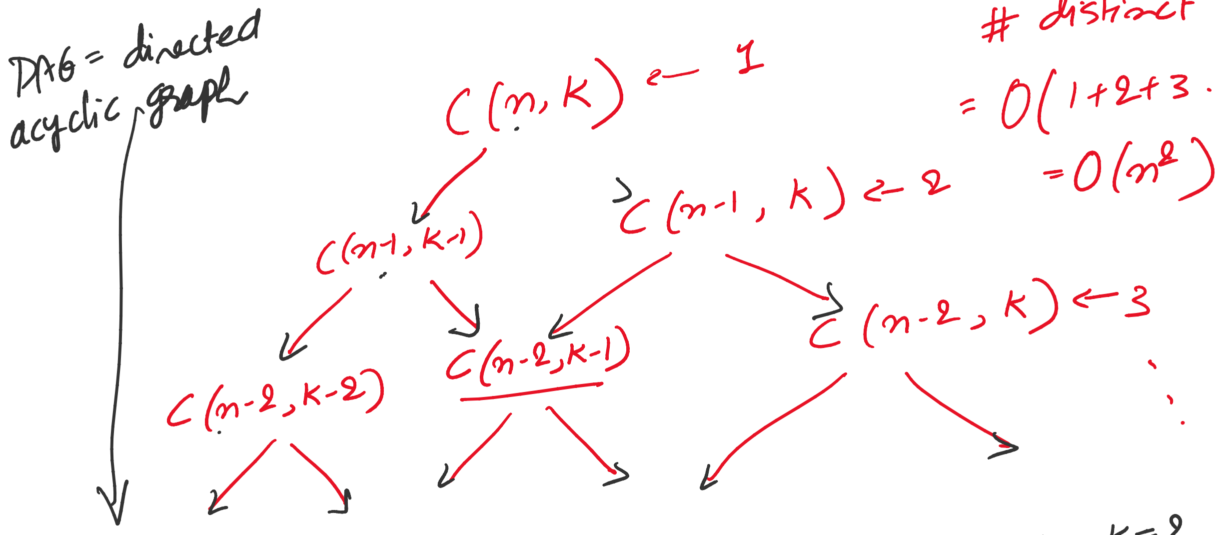
$$= 1 \quad \text{if } \underline{K=0} \text{ or } \underline{K=n}$$

Binomial coefficient.

Naive Implementation:

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

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



\# distinct subprob.

$$= O(1+2+3\ldots+n-1)$$

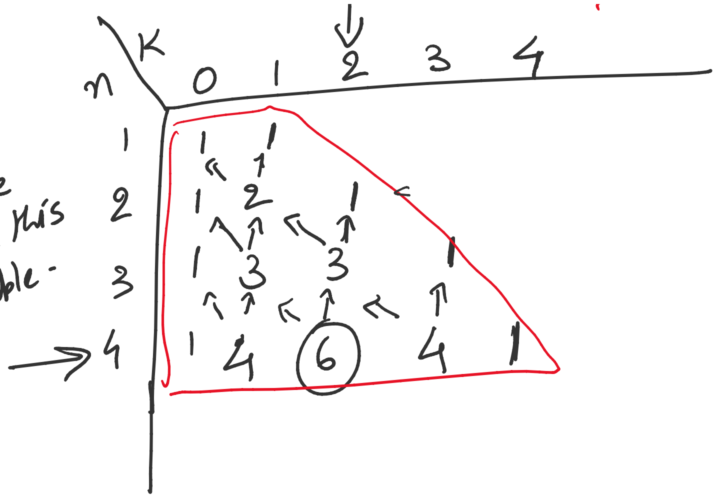
$$= O(n^2)$$

$n=4$, $K=2$

$C(4,2)$

n \ K	0	1	2	3	4

$C(4,2)$



space = size this table.

Pascal's Triangle.

Evaluation order smallest to largest value of n

for each n " " " " K.

Psuedo code:

```
For i = 1 to n
    c[i, 0] = 1,   c[i, i] = 1
    For j = 1 to i-1
        O(1) <- C[i, j] = C[i-1, j-1] + C[i-1, j]
Return C[n, K].
```

$O(n)$ ← (inner loop)

$O(n^2)$ ← (outer loop)

⇒ $O(n^2)$ time.

$O(n^2)$ space

↳ can be improved $O(n)$ by only storing $(i-1)^{th}$ & i^{th} row.

"Real" Ex 3: Longest Common Subsequence (LCS).

Given two seq. $A = a_1 a_2 \dots a_n$

$B = b_1 b_2 \dots b_m$

Find longest subseq. of A that is also a subseq of B.

eg. ALGORITHM

LOGARITHM

LORITHM of size 7

LGRITHM " "

Appl: check similarity of two string (# deletes/inserts)

UNIX diff

Intuition: suppose $a_m = b_m$.

Then $LCS(a_1 \ldots a_{m-1}, b_1 \ldots, b_{m-1})$

$+1$.

→ Define subproblem.

O(mn) subproblems

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

$i = 0 \ldots n$, $j = 0, \ldots, m$

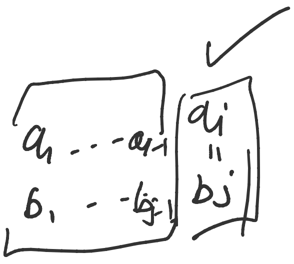
Answer: $C(n,m)$.

Base case: $i=0$ or $j=0$

$C(0,j) = 0$, $c(i,0) = 0$

$\overset{\uparrow B=\phi}{}$ $\underset{A=\phi}{\Downarrow}$

→ Recursive Formula. $c(i,j)$



3 cases:

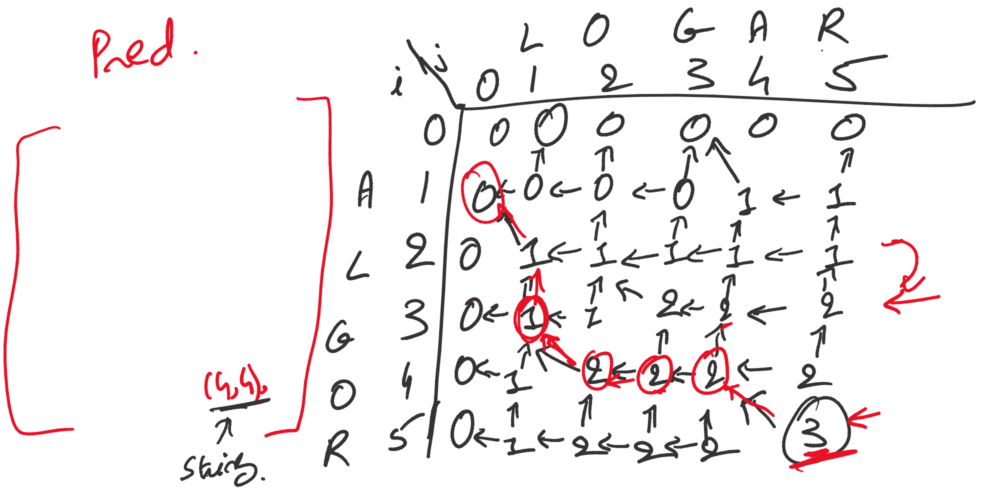
1. not take a_i → $c(i-1,j)$
2. not take b_j → $c(i,j-1)$
3. Take a_i & b_j when $\underline{a_i = b_j}$

→ $c(i-1,j-1) + 1$

$\{c(i-1,j), c(i,j-1)\}$ if $a_i \neq b_j$

$$c(i,j) = \begin{cases} \max\{c(i-1,j),\ c(i,j-1)\} & \text{if } a_i \neq b_j \\ \max\left\{ \begin{array}{l} \cancel{c(i-1,j)},\ \cancel{c(i,j-1)}, \\ c(i-1,j-1)+1 \end{array} \right\} & \text{if } a_i = b_j \leftarrow \end{cases}$$

eg. ALGOR
LOGAR



		0	1	2	3	4	5
	0	0	0	0	0	0	0
A	1	0	0	0	0	1	1
L	2	0	1	1	1	1	1
G	3	0	1	1	2	2	2
O	4	0	1	2	2	2	2
R	5	0	1	2	2	2	3

LOR

Evaluate in increasing order of i (row)
for each i, in increasing order of j

Psuedo code: Exe.

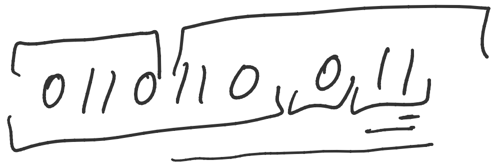
Analysis: mn subproblems.
each O(1) time

$\Rightarrow$ O(mn) time
O(mn) space.
↳ can be improved to O(n)
by storing two rows at a time.
when only want length & NOT the actual LCS.

Ex2: Given string $X = a_1 a_2 \ldots a_n$
split X into min # of Palindromes.

Split ^

eg.



→ Define subproblem :

$\underbrace{a_1 \dots a_{j-1}}_{\uparrow \text{ solve on this}} \underbrace{a_j \dots a_n}_{\text{last palindrome.}}$

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

Answer : $c(n)$

→ Base case : $c(0) = 0$

→ Recursive Formula.

Cases for opt sol'n

if last palindrome . $a_j \dots a_i$

↳ $c(j-1) + 1$

$$C(i) = \min_{\substack{j=1 \text{ to } i \\ \text{s.t. } a_j \dots a_i \text{ is} \\ \text{a palindrome.}}} C(j-1) + 1.$$

→ Evaluate in increasing order of i.

→ Psuedocode .

```
C[0] = 0
for i=1 to n
    C[i] = ∞
    for j = 1 to i                                          } O(n^2) time.
        O(n) ← if aj.. ai is a palindrome
               & C[j-1]+1 < C[i]
        then C[i] = C[j-1]+1 ;
```

```
O(n) ←
time.
        then    c[i] = c[j-1]+1;
                Pred[i] = j;
Return c[n].
```

→ Analysis:

$O(n)$ subproblems. ⇒ Total $O(n^3)$ time

each takes $O(n^2)$ time

$O(n)$ space.

→ Psuedocode to output the answer

```
OutputAns(i)
    if i=0 then return;
    j = pred[i]
    OutputAns(j-1)
    Print a_j .. a_i // palindrome
```