Week 9

Monday, August 18, 2025 12:46 PM

Dynamic Programming → The goal is to achieve the solution to a problem via means of solving smaller sub-problems.

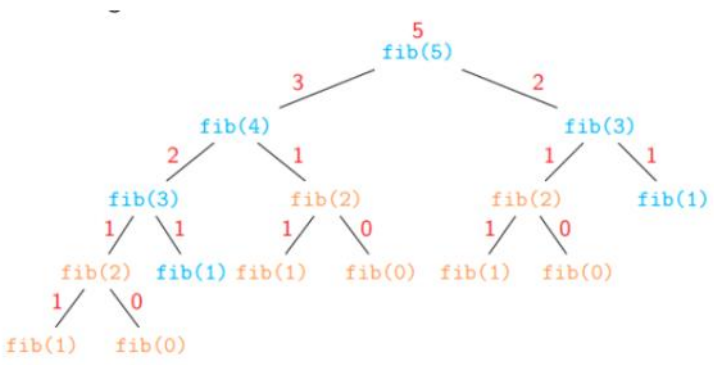
→ Fibonacci series, Factorial calculation and insertion sort are popular examples.

Memorization is a characteristic feature.

→ That a bigger problem is a combination of smaller easy problems. These are also repetitive in nature.

Simple recursive — The basic implementation of Fibonacci.

```
def fibrec(n):
    if n <= 1:
        return n
    return fibrec(n - 1) + fibrec(n - 2)
```



We can see that there are repetitive steps in between the process of finding the fibonacci of any number.

If these values of intermediate steps can be somehow memorised, the time to get the answers will get reduced!

Memorization →

Memoization — A memory dict.

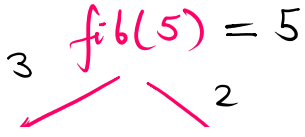
```
memo ={}
def fib(n):
    if n <= 1:
        memo[n] = n
    if n not in memo:
        memo[n] = fib(n-1) + fib(n-2)
    return memo[n]
```

Each intermediate step value can be taken back from the memory dict.

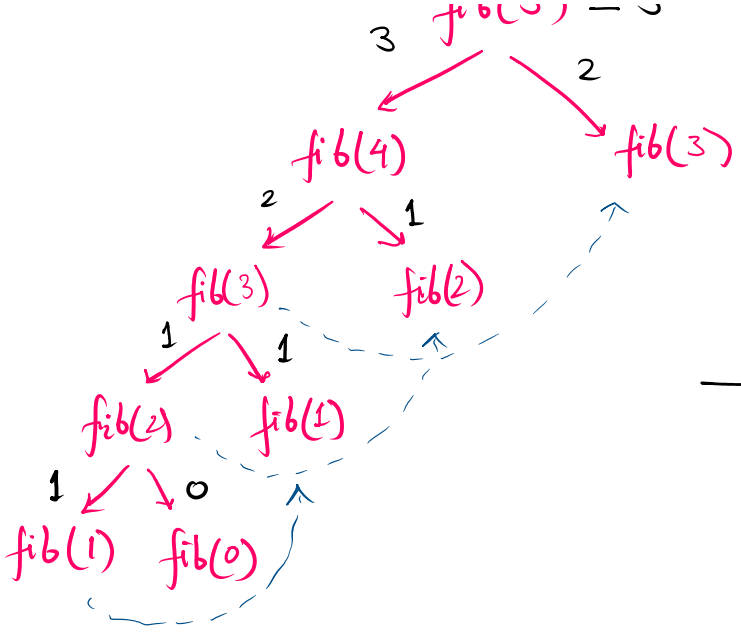
The memory dict as table →

k	0	1	2	3	4	5
fib(k)	0	1	1	2	3	5

And after memorization, the code will reduce to even simpler form →



The values can be easily



→ The values can be easily back traced from the memory dictionary.

→ Reduces the compution time.

The dynamic programming approach →

Dynamic programming

```
def fib(n):
    T = [0] * (n + 1)
    T[1] = 1
    for i in range(2, n + 1):
        T[i] = T[i - 1] + T[i - 2]
    return T[n]
```

A list of (n+1) values.

element at index 1 = 1

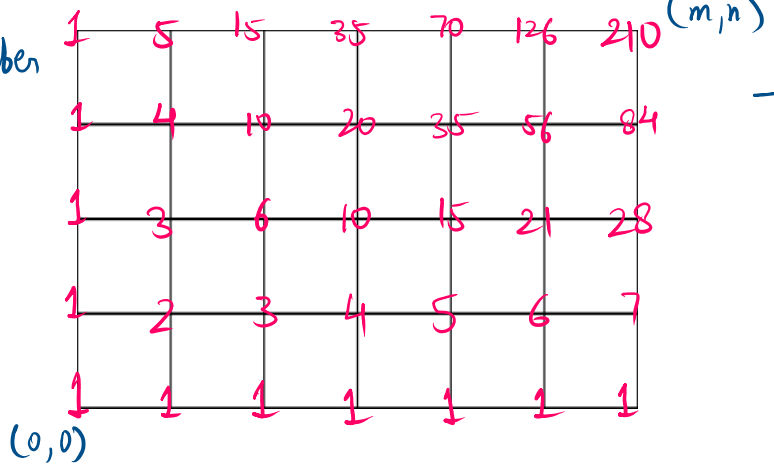
Every other element is the sum of previous 2.

Grid path problem →

- Rectangular grid of one-way roads
- Can only go up and right
- How many paths from (0, 0) to (m, n)? → $^{m+n}C_m$ or $^{m+n}C_n$
- Every path has $(m + n)$ segments
- What if an intersection is blocked?
- Need to discard paths passing through blocked intersection

Remove those all paths which pass through that blocked intersection.

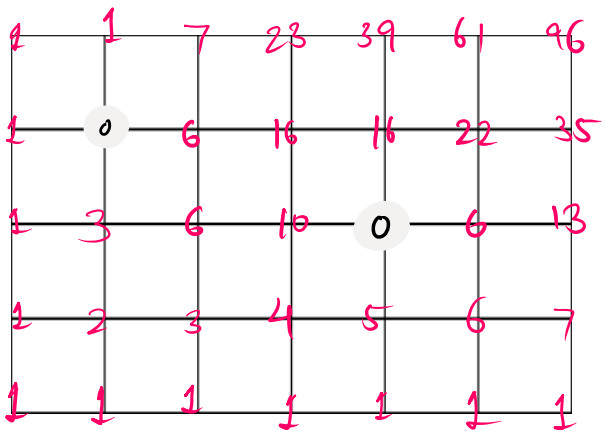
→ This is the number of paths for the given vertex (m,n) from (0,0).



→ No intermediate blockage!

When there are blockages present in between.

→ Blockage means zero.

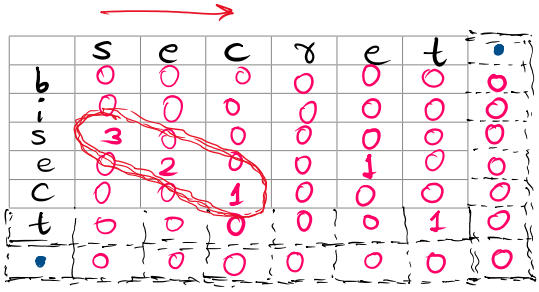


→ For cases of blockages, ignore the pathways passing through the blockages!

→ The complexity is $O(mn) \approx O(n^2)$ with Dynamic Programming and $O(m+n) \approx O(n)$ using memorizing.

Longest Common subword (LCS) →

Common subwords mean; a substring that are common to two given strings, and are consecutive.



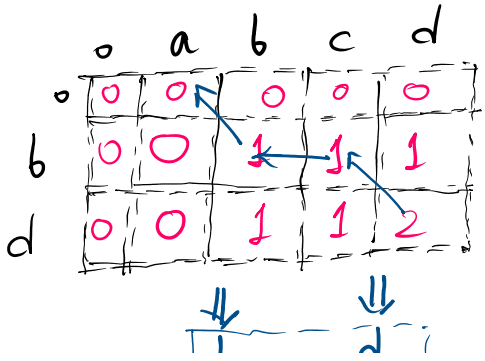
→ sec is the longest common subword.

- Table of $m + 1$ $n + 1$ values
- Inductive structure

$$LCW[i,j] = \begin{cases} 1 + LCW[i+1, j+1], & if\ a_i = b_j \\ 0, & if\ a_i \neq b_j \end{cases}$$

→ Complexity of the process is $O(mn) \approx O(n^2)$.

Longest Common Subsequence → Very much similar to the longest common subwords, where we can omit any word in between, but sequence must stay intact!

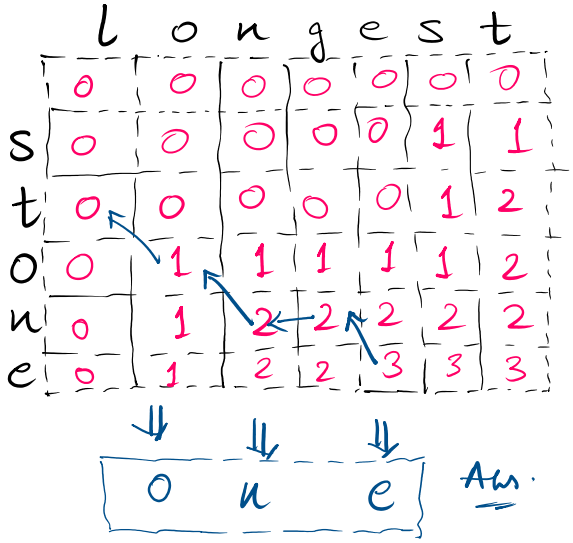


→ Get the elements at the diagonal arrows.

→ Those will denote the common(longest) subsequence.

b d Answer sub-sequence.

Ques: Find longest common subsequence for the strings → stone & longest.



Ans.

→ Complexity is $O(m \times n) \approx O(n^2)$.
(m: len of str1, n: len of str2)

Edit Distance → Aim is to identify min number of editing operations to be done, for transforming a document to another.

→ The complexity is $O(n \times m) \approx O(n^2)$.

Matrix Multiplication → (Chain multiplication)

⇒ For a case of 'n' matrices, we will have $\frac{2^n C_n}{n+1}$ binary trees.

$A1 \times A2 \times A3$
$2 \times 3 \quad 3 \times 4 \quad 4 \times 2$
$d_0 \; d_1 \quad d_1 \; d_2 \quad d_2 \; d_3$

→ $(A_1 \times A_2) \times A_3$
$2 \times 3 \times 4 \quad 4 \times 2$
$+ \; 2 \times 4 \times 2$
⇒ 24 + 16
⇒ 40 multiplications needed!

→ $A_1 \times (A_2 \times A_3)$
$3 \times 4 \times 2$
⇒ 24 + 12

$3 \times 4 \times 2$ + $2 \times 3 \times 2$ $\Rightarrow 24 + 12$ $\Rightarrow 36$ multiplications needed!

Aim is to derive an approach, so as to perform these multiplications with least number of computation/calculations.

Formulation → $C[i,j] = \min_{i \le k < j} \{ C[i,k] + C[k+1,j] + d_{i-1} \times d_k \times d_j \}$

Qus. $A_1 \times A_2 \times A_3 \times A_4$

$3 \times 2 \quad 2 \times 4 \quad 4 \times 2 \quad 2 \times 5$

$d_0 \quad d_1 \quad d_2 \quad d_3 \quad d_4$

$C[1,2] = 0 + 0 + 3 \times 2 \times 4 = 24$

$C[2,3] = 0 + 0 + 2 \times 4 \times 2 = 16$

$C[3,4] = 0 + 0 + 4 \times 2 \times 5 = 40$

$C[1,3] = \min \begin{cases} k=1 & 0 + 16 + 3 \times 2 \times 2 = 28 \\ k=2 & 24 + 0 + 3 \times 4 \times 2 = 48 \end{cases}$

$C[2,4] = \min \begin{cases} k=2 \\ k=3 \end{cases}$

$C[1,4] = \min \begin{cases} k=1 \\ k=2 \\ k=3 \end{cases}$

C

	1	2	3	4
1	0	24	28	58
2	24	0	16	36
3	28	16	0	40
4	58	36	40	0

K

	1	2	3	4
1		1	1	3
2			2	3
3				3
4				

$\Rightarrow$ Worst case complexity is $O(n^3)$.