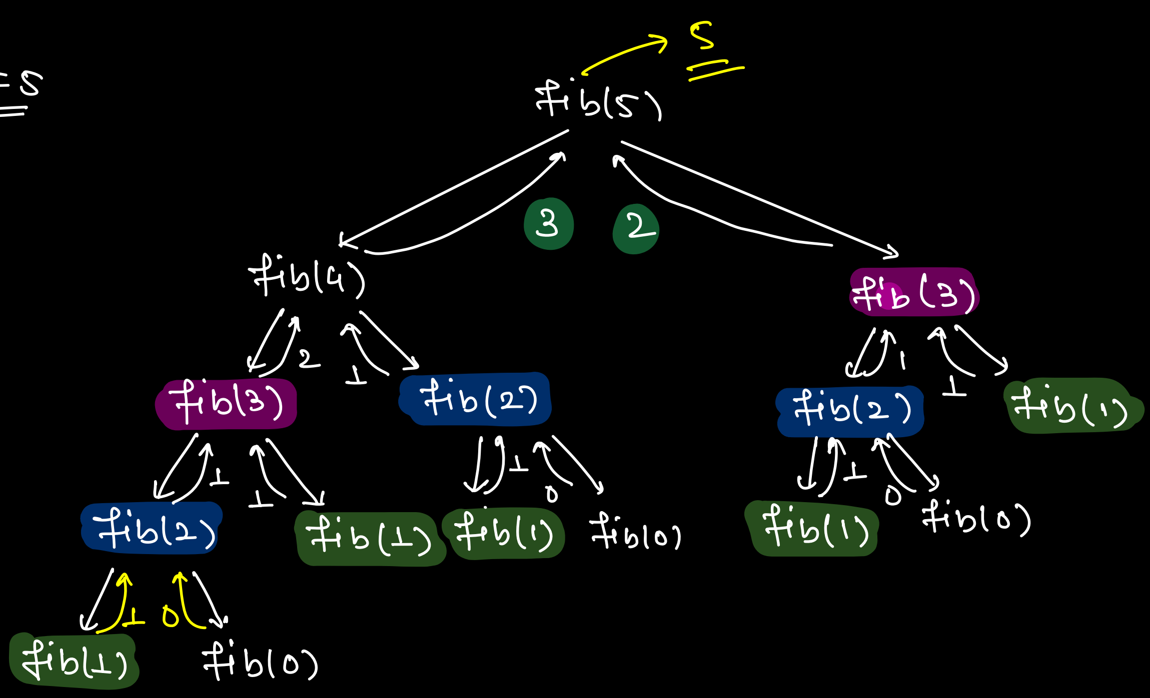
Dynamic Programming

fibonacci Series

N →	0	1	2	3	4	5	6	7	...
	0	1	1	2	3	5	8	13	...

```
int fib(int N){
    if(N<=1) return N;
    return fib(N-1) + fib(N-2);
}
```

N=5



TC : $O(2^N)$

$$T(N) = T(N-1) + \underbrace{T(N-2)}_{\simeq T(N-1)} + 1$$

$$= 2T(N-1) + 1 \Rightarrow 2^N$$

TC of recursive fun^n =

$\underbrace{\text{No. of fun}^n\text{ calls}}_{\simeq 2^N} * \underline{\text{TC of each fun}^n\text{ call}}_{1}$.

$\simeq 2^N * 1$

$\Rightarrow \underline{\underline{O(2^N)}}$

$\Rightarrow$

1) Solve a problem using subproblems. ⇒ Optimal Substructure
2) Solving subproblems more than once.
   ↳ Overlapping <u>Subproblems</u>.

<u>D.P</u>

↳ Solving a subproblem exactly <u>once</u>.

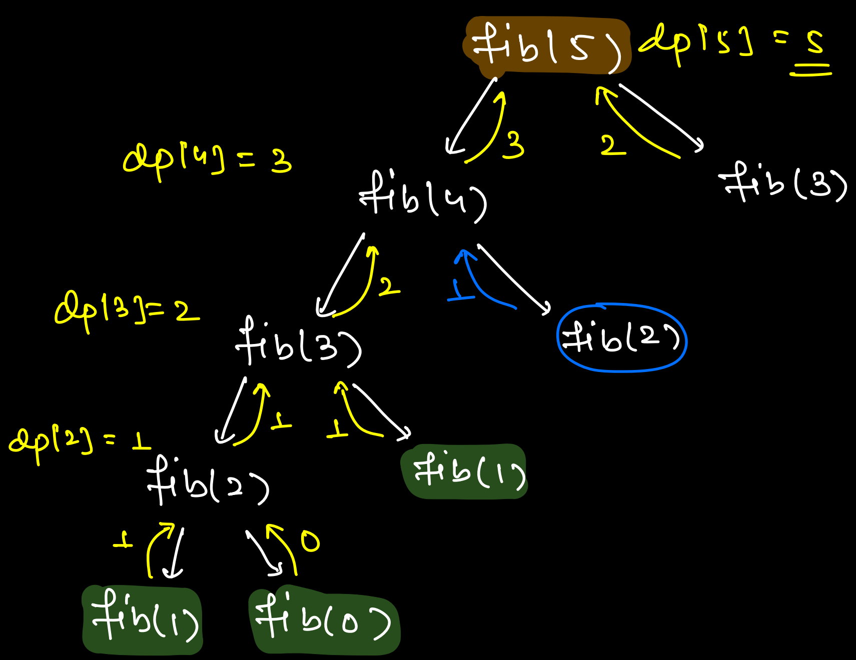
```
int dp[N+1] = {-1};
int fib( int N) {
    if (N<=1) dp[N]=N, return N;
    if( dp[N] == -1) { // Not calculated yet
        dp[N] = fib(N-1) + fib(N-2);
    }
    return dp[N];
}
```

TC: $N \times 1 \Rightarrow O(N)$

SC: $O(N + N)$

dp[]   <u>Stack</u>

0	1	2	3	4	5
~~-1~~	~~-1~~	~~-1~~	~~-1~~	~~-1~~	~~-1~~
0	1	1	2	3	5

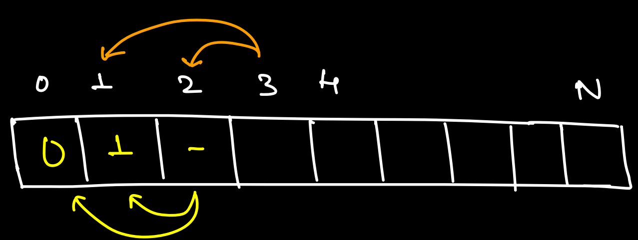


⇒ Top Down DP. fib(5) → fib(4) → fib(3) → ...

⇒ Recursion + Memory : Memoization DP

#

dp[N+1] :



dp[N] = dp[N-1] + dp[N-2] ⇒ Dp Expression.

```
dp[0] = 0, dp[1] = 1;
for(i = 2; i <= N; i++) {
    dp[i] = dp[i-1] + dp[i-2]
}
return dp[N];
```

TC: O(N)

SC: O(N)

dp[]

⇒ Bottom Up DP

⇒ Iterative + Table {dp[] / Memory} ⇒ Tabulation DP

Steps.

1. Optimal Substructure.
2. Overlapping subproblems.
   → Same subproblem is repeating lot of times.
3. dp State : What dp table should contain.
4. dp Expression : How to calculate dp state using smaller subproblems.
5. Base Cases : Values for which dp expression won't work.
6. Code.
7. TC & SC analysis.
8. Optimization → TC, SC

```
int fib(int N) {
    int a = 0, b = 1;
    int c;
    fib( i=2; i<=N; i++) {
        c = a+b;
        a = b;
        b = c;
    }
    return c;
}
```

a	b	c
0	1	1
1	1	2
1	2	3
2	3	5
	:	:

$$\boxed{\begin{array}{l} TC : O(N) \\ SC : O(1) \end{array}}$$

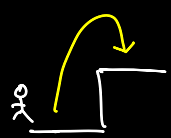
Q. ★ N stairs.

Amazon / SAP

Given N stairs, in how many ways we can go from $0^{th} \longrightarrow N^{th}$

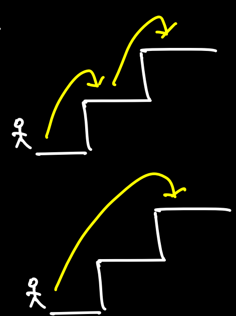
Note : from i^{th} floor $\longrightarrow$ $i+1$ OR $i+2$ floors.

N=1



$\Rightarrow$ 1

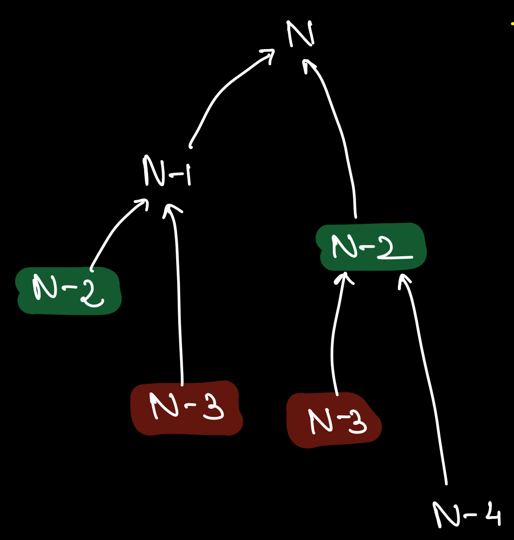
N=2



$\Rightarrow$ 2 ways.

N=3 $\Rightarrow$ {1 1 1}
{1 2}
{2, 1}
$\Rightarrow$ 3 ways.

N=4 $\Rightarrow$ {1 1 1 1}
{1 1 2}
{1 2 1}
{2 1 1}
{2 2}
$\Rightarrow$ 5 ways.



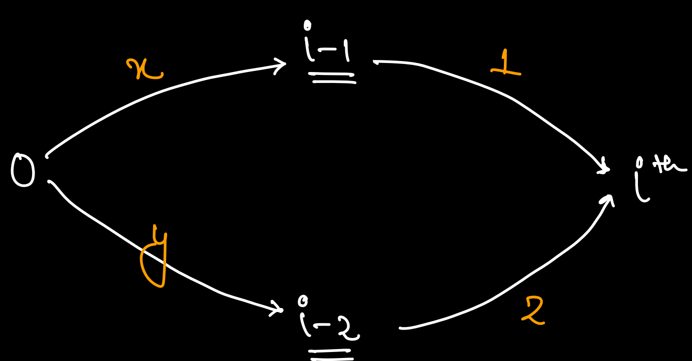
→ Optimal substructure
→ Overlapping subproblems.

DP

⇒ DP State

dp[i]: # of ways to reach i^{th} floor.

⇒ DP Expression



$$\boxed{dp[i] = x + 2y}$$

$$dp[i] = dp[i-1] + 2\,dp[i-2]$$

$$dp[1] = 1, \quad dp[2] = 2$$

$$dp[3] = 2 + 2 \times 1 = 4$$

X

$N=4$

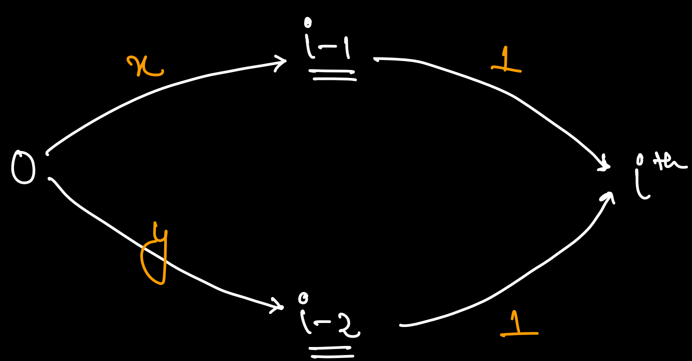
$\{1\ 1\ 1\ 1\} \Rightarrow 0 \to 1 \to 2 \to 3 \to 4$

$\{1\ 1\ 2\} \Rightarrow 0 \to 1 \to 2 \to 4$

$\{1\ 2\ 1\} \Rightarrow 0 \to 1 \to 3 \to 4$

$\{2\ 1\ 1\} \Rightarrow 0 \to 2 \to 3 \to 4$

$\{2\ 2\} \Rightarrow 0 \to 2 \to 4$



$$dp[i] = x * 1 + y * 1$$

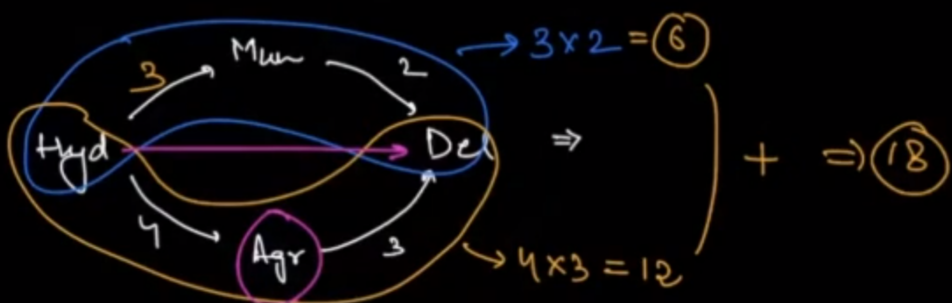
$$\boxed{dp[i] = dp[i-1] + dp[i-2]}$$

$dp[1] = 1,\ dp[2] = 2$

$dp[3] = dp[1] + dp[2]$

$= 3$

Base Case : $i=1$, $i=0$



$$dp[2] = dp[1] + \underline{dp[0]}$$

$$2 = 1 + dp[0]$$

$$\underbrace{dp[0]}_{\text{\# of ways to reach ground floor.}} = 1 \checkmark$$

$$\boxed{\begin{aligned} &TC: O(N) \\ &SC: O(N) \longrightarrow O(1) \end{aligned}}$$

Q. 2 faced dice $\rightarrow$ 1, 2

We can roll this dice as many times as we want.

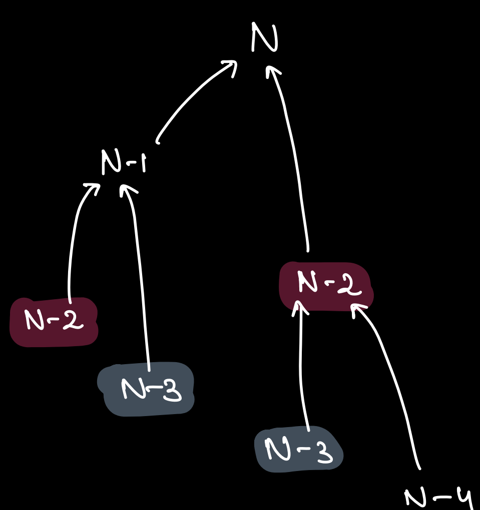
# of ways to make sum = N.

N = 1 $\Rightarrow$ 1 way

N = 2 $\Rightarrow$ {1 1} {2} $\Rightarrow$ 2 ways.

N = 3 $\Rightarrow$ {1 1 1} {1 2} {2 1} $\Rightarrow$ 3 ways.

N = 4 $\Rightarrow$ {1 1 1 1} {1 1 2} {1 2 1} {2 1 1} {2 2} $\Rightarrow$ 5 ways.



- → Optimal substructure
- → Overlapping subproblems.

$\Rightarrow$ DP

dp[i]: # of ways to make sum = i.

$$dp[i] = dp[i-1] + dp[i-2].$$

Q. 6 faced dice $\to$ 1, 2, 3, 4, 5, 6

We can roll this dice as many times as we want

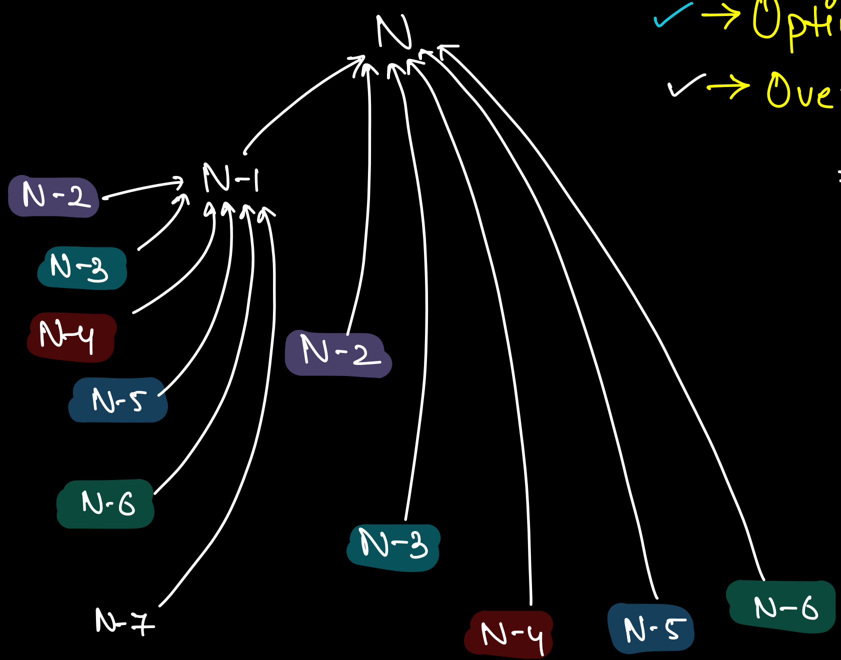
# of ways to make sum = N.

N = 1 ⇒ 1

N = 2 ⇒ {1 1} ⇒ 2 ways.
{2}

N = 3 ⇒ {1 1 1}
{1 2} ⇒ 4 ways.
{2 1}
{3}

N = 4 ⇒ {1 1 1 1}
{1 1 2}
{1 2 1} ⇒ 8 ways.
{2 1 1}
{2 2}
{1 3}
{3 1}
{4}



✓ → Optimal substructure
✓ → Overlapping subproblems.

⇒ DP

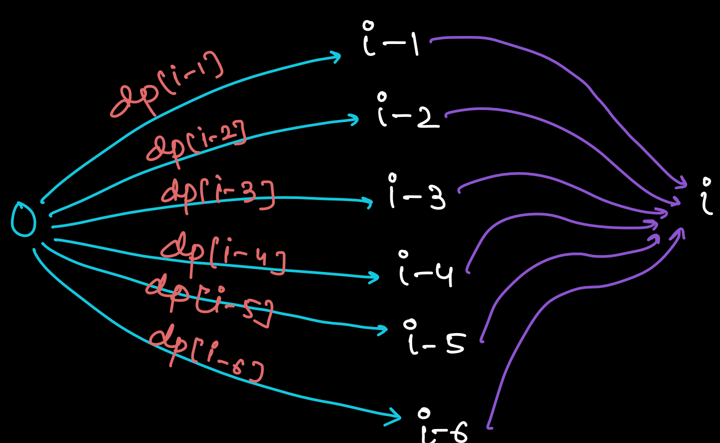
dp State.

dp[i] : # of ways to make sum = i.

dp table.

dp[N+1]

dp Expression.



$$dp[i] = dp[i-1] + dp[i-2] + dp[i-3] + dp[i-4] + dp[i-5] + dp[i-6]$$

Base Case

$i = 0, 1, 2, 3, 4, 5$

$dp[0] = 1$ $\qquad$ $dp[3] = 4$

$dp[1] = 1$ $\qquad$ $dp[4] = 8$

$dp[2] = 2$ $\qquad$ $dp[5] = 16$

$\Rightarrow$

$$dp[i] = \sum_{\substack{j=1 \\ i>=j}} dp[i-j]$$

$$dp[1] = dp[1-1]$$

$$\boxed{1 = dp[0]}$$

$$dp[2] = dp[2-1] + dp[2-2]$$

$$= dp[1] + dp[0]$$

$$= 1 + 1 = 2$$

$$dp[3] = dp[3-1] + dp[3-2] + dp[3-3]$$

$$= 2 + 1 + 1$$

$$= 4$$

Code :

```
int dp[N+1];
dp[0] = 1;
for ( i = 1 ; i <= N; i++) {
    sum = 0
    for ( j = 1; j <= i && j <= 6; j++ ) {
        sum += dp[i-j];
    }
    dp[i] = sum;
}
return dp[N];
```

$i-j$ >= 0

6

TC : # of dp states * No. of Iterations of 1 dp state

: N * 6

: O(N)

SC : O(N)

Can we optimise SC ?

Keep only 7 Variables] TODO

→ Bottom Up DP

$ways(N) = ways(N-1) + ways(N-2) + ways(N-3) +$
$ways(N-4) + ways(N-5) + ways(N-6)$

{ Smaller Subproblems. } Optimal Substructure. ✓

TODO Solve this using Memoization.

——— * ———