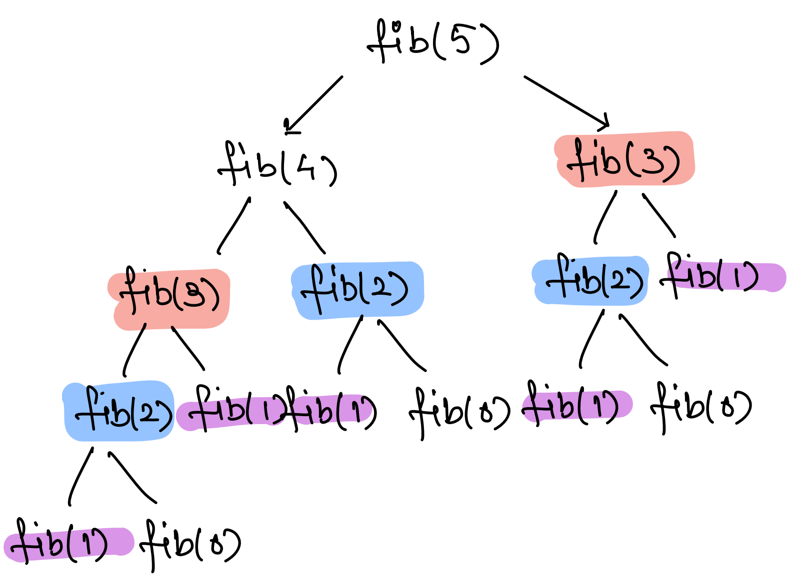
N:	0	1	2	3	4	5	6	7	8	....
fib:	0	1	1	2	3	5	8	13	21	....

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



$\boxed{TC: O(2^N)}$

$T(N) = 2T(N-1)+1$

$\hookrightarrow 2^N$

1) Solving a problem, using smaller subproblems.

↳ Optimal substructure.

2) Solving same problem more than once.

↳ Overlapping subproblems.

Calling each unique subproblem exactly once.

⇒ Dynamic Programming.

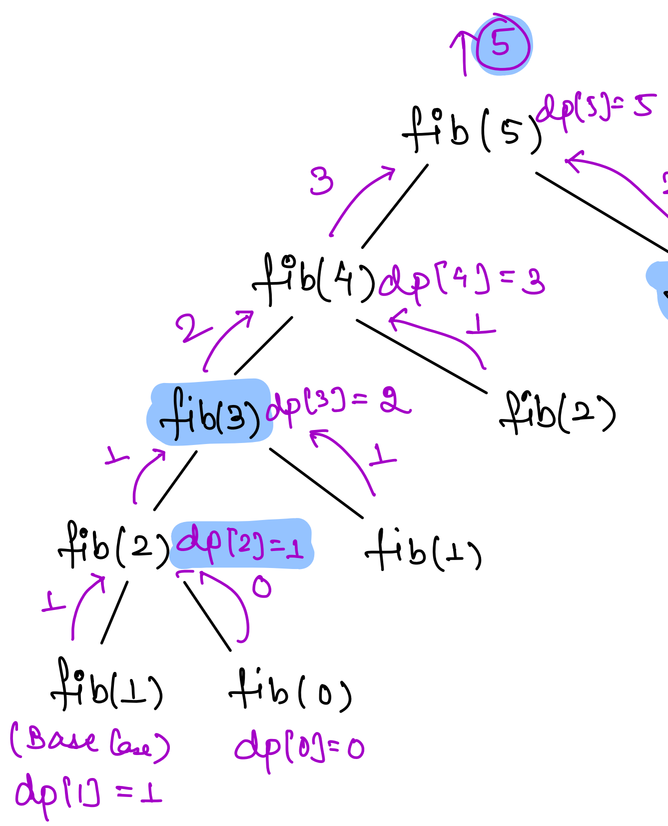
```
# int dp[N+1] = -1;
```

↳ -1 represents that it is NOT known yet.

```
int fib(N) {          // Recursive funⁿ
    if (N <= 1) {
        dp[N] = N;
        return dp[N];
    }
    if (dp[N] == -1) {
        // fib(N) is getting called for the 1st time.
        dp[N] = fib(N-1) + fib(N-2);
    }
    return dp[N];
}
```

dp:

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



Recursion + Memory } Memoization

Top Down DP

TC: O(N) , SC: $\underset{\text{dp Array}}{O(N)} + \underset{\text{Recursion}}{O(N)} \Rightarrow \underline{\underline{O(N)}}$

```
#   int fib(N) {    // Iterative Approach.
        int dp[N+1];
        dp[0] = 0 , dp[1] = 1;
        for(i = 2; i <= N; i++) {
            dp[i] = dp[i-1] + dp[i-2];
        }
        return dp[N];
    }
```

0	1	2	3	4	5	6
0	1	1	2	3	5	8

dp[0] → dp[1] → dp[2] → dp[3] → dp[4] → dp[5] → dp[6]

Bottom Up DP

Iterative + Memory }	Tabulation DP

dp[i] ⇒ i^{th} fibonacci no.

⇒ DP State.

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

⇒ DP Expression.

TC: O(N)

SC: O(N)

↑ DP array.

Steps to solve a problem using DP :-

1) Solve a problem using subproblems : Optimal Substructure

2) Overlapping subproblems.
   ↳ Solve unique subproblem exactly once

Notes :-

1) DP State
2) DP Expression.
3) Base case
4) DP table.
5) TC : # of DP States * TC for each state.
   SC : DP table

#

```
int fib(N) {
    a = 0   // 0th fib no
    b = 1   // 1st fib no.
    c;
    for(i=2; i<=N; i++) {
        c = a+b;
        a=b
        b=c
    }
    return c;
}
```

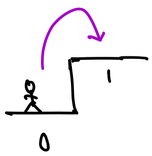
a	b	c	
0	1	1	2nd
1	1	2	3rd
1	2	3	4th
2	3	5	5th
3	5	8	6th

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

Q: N stairs

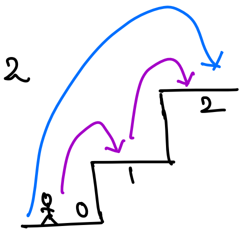
Given N stairs, How many ways we can go from 0th to Nth step.
From ith step, we can go to (i+1)th or (i+2)th step.

N=1



⇒ 1 way

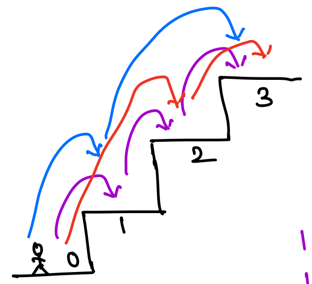
N=2



11
2

⇒ 2 ways

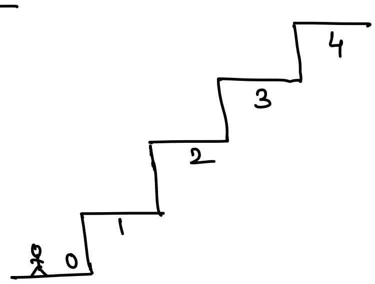
N=3



111
12
21

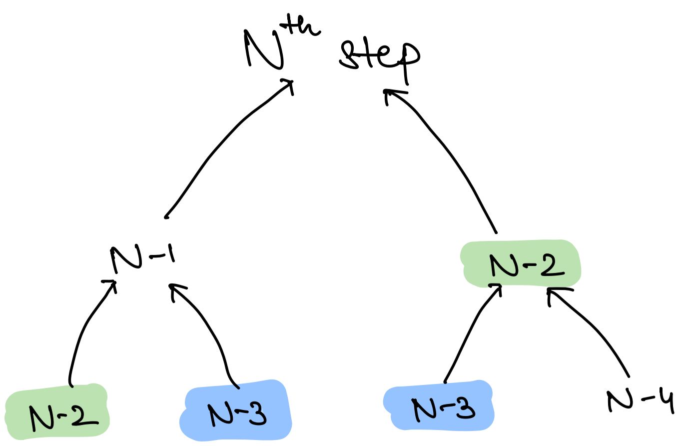
⇒ 3 ways.

N=4



1111
112
121
211
22

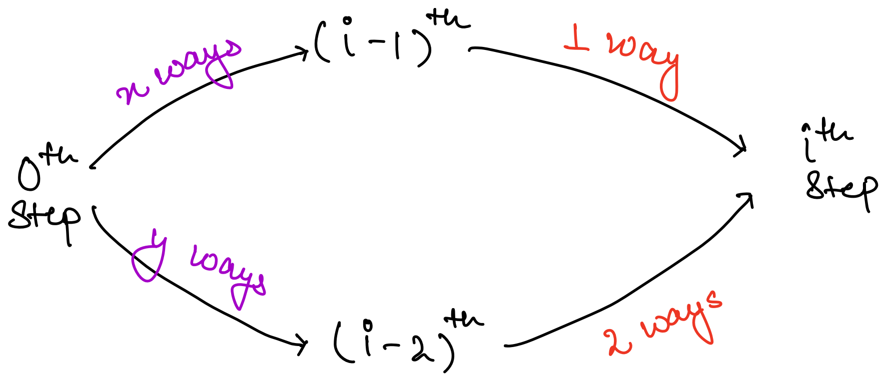
} 5 ways.



- Optimal substructure ✓
- Overlapping Subproblems ✓

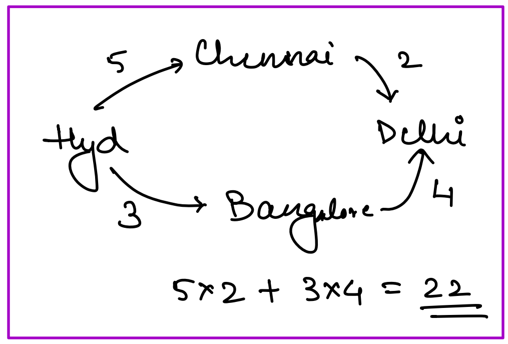
dp[i]: No. of ways to reach ith step.
↳ state

DP expression



# of ways for i^{th} step = $x + 2y$

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



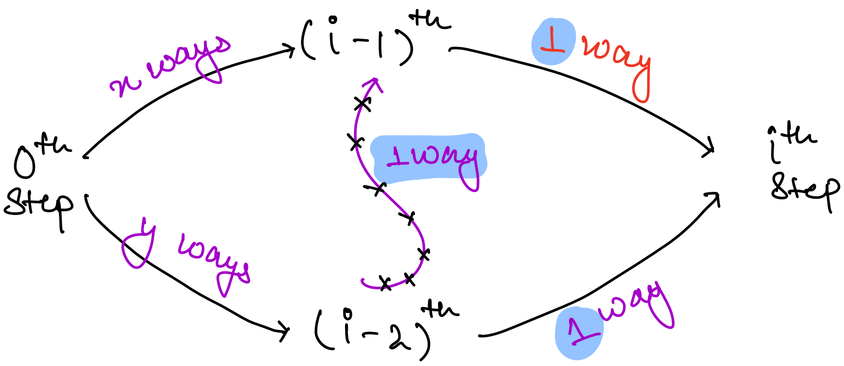
$dp[1] = 1$

$dp[2] = 2$

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

$= 2 + 2*1$

$= 4$ ✗



Only consider the direct paths from (i-1) to i and i-2 to i.

$dp[i] = x + y = dp[i-1] + dp[i-2]$ ✓

Base Cases

$dp[1] = 1$

$dp[2] = 2$

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

$2 = 1 + dp[0]$

$dp[0] = 1$ ✓

There's a way to reach to Ground floor.

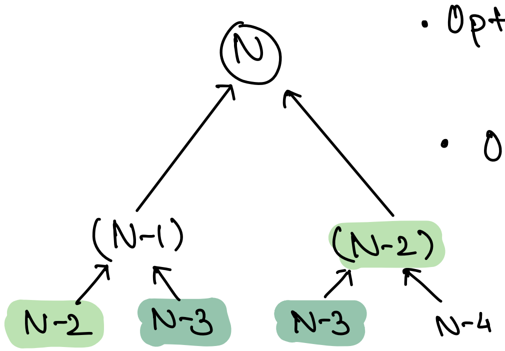
Q: 2 face dice → 1, 2

⇒ We can roll the dice as many times as we want

⇒ Count the No. of ways to get the sum = N.

N=1	N=2	N=3	N=4
→ 1 way	1 1	1 1 1	1 1 1 1
	2	1 2	1 1 2
	→ 2 ways	2 1	1 2 1
		→ 3 ways	2 1 1
			2 2

→ 5 ways.



- Optimal substructure ✓
- Overlapping Subproblems ✓

} DP

* DP State

$dp[i]$: No. of ways to get a sum of i

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

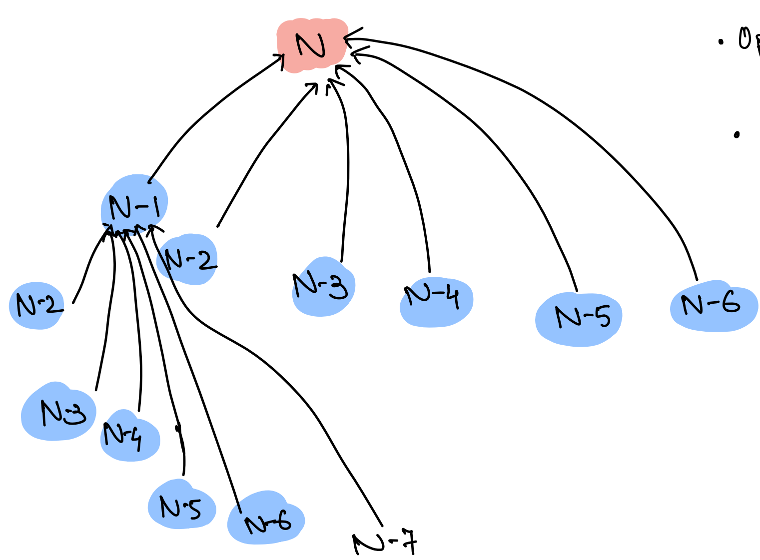
$$\left.\begin{array}{l} dp[0] = 1 \\ dp[1] = 1 \end{array}\right\} ✓$$

Q 6 faces dice

1 2 3 4 5 6

⇒ We can roll the dice as many times as we want

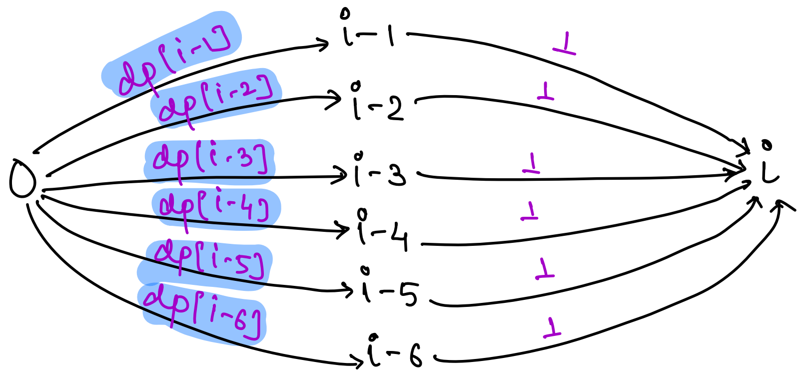
⇒ Count the No. of ways to get the sum = N.



- Optimal substructure ✓
- Overlapping subproblems ✓

} DP

$dp[i]$: No. of ways to get the sum $= i$



$$\boxed{dp[i] = \sum_{j=1}^{6} dp[i-j]}$$

DP Expression

$dp[0] = 1$ | $dp[3] = 4$
$dp[1] = 1$ | $dp[4] = 8$
$dp[2] = 2$ | $dp[5] = 16$

$\left.\right\}$ Base Case.

\# $dp[1] = dp[1-1]$

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

$$\boxed{dp[i] = \sum_{j=1 \,\&\&\, i >= j}^{6} dp[i-j]}$$

$dp[3] = dp[2] + dp[1] + dp[0]$

Base Case

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

#

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

TC: $O(N \times 6) \rightarrow O(N)$

SC: $O(N)$

--- * ---