Today's Content :

→ Dynamic Programming Intro : {Fib}

→ { When to use Dp }

→ Steps for Dp

→ 1 hour

→ # N Stairs

→ Party Pairs

→ Dice Sum

3 prob

7 Sessions → Sat | Tu | Thu | Sat | [—]

Intro

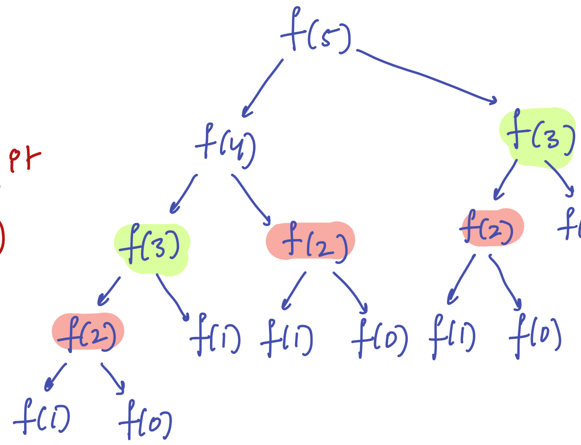
Tu | Thu | Sat

0	1	2	3	4	5	6	7
0	1	1	2	3	5	8	13

Fib: (values above)

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

trace it → f(8)



TC : 2^N

Idea:

1) f(N) = f(N-1) + f(N-2) } optimal Substructure ← → { necessary }

2) Calculating same Sub problem over & over } Overlapping Subproblems

3) Dp : { Calculating all unique subproblem only once } → Dynamic Programming

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

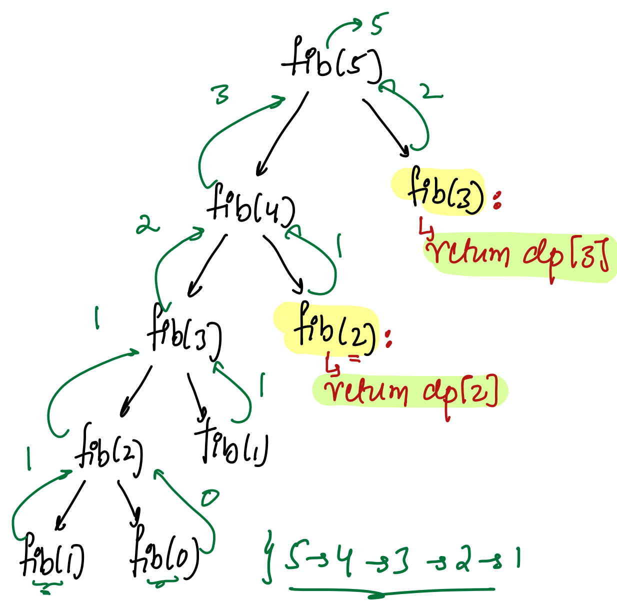
→ { fib value can never be negative }

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

T(N) = N * 1

dp[6] =

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



{ 5 → 4 → 3 → 2 → 1

Topdown approach

Top down Recursion + Extra space

: Memoization

```
int fibite(int N){  TC: O(N)
    int dp[N+1] = {0}
    dp[0] = 0, dp[1] = 1
    i=2; i<=N; i++){
        dp[i] = dp[i-1] + dp[i-2]
    }
    return dp[N]
}
```

fibite(5)

: dp[6] =

0	1	2	3	4	5
0	1	1	2	3	5

: return dp[5]

iteration :

0 → 1 → 2 → 3 → -- N

Bottom up approach : Tabulation

// dp[i] = i^{th} fibnacci Number

// dp[i] = dp[i-1] + dp[i-2] : dp Expression

```
int fibite(int N){  TC: O(N)
    if(N <= 1) return N      SC: O(1)

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

fibite(6) :

0	1	2	3	4	5
a	b	c			
~~0~~	~~1~~	1			
~~1~~	~~1~~	2			
~~1~~	~~2~~	3			
2	3	5			

// { Memoization vs Tabulation } → { In coming Sessions }

Steps: (Dynamic Programming)

1 → Optimal Substructure
2 → Overlapping Subproblems

} → Exactly same as Recursion
→ Only then dp

→ dp[i] → { dp state } : Assumption

→ dp expression : Main logic → { Calculate dp state }

→ dp table → { We store all states }

→ dp base conditions

→ Code, return ans

→ TC: → { # How many dp state * { Time for each dp state } }

SC: → { Dp table Size }

→ Optimization → SC : { Iterative code }

→ { 10:30 break }

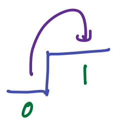
Stairs:

Q) Given N Stairs, how many ways we can go from $0^{th} \rightarrow N^{th}$ Step

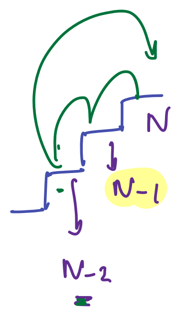
Note: from i^{th} Step we can directly go to $\underset{+1}{(i+1)^{th}}$ or $\underset{+2}{(i+2)^{nd}}$ Step

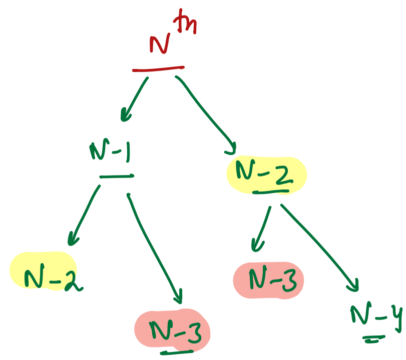
& $\underset{0}{—}$: 1 way → # ways to reach N^{th} Step

Ex: N=1

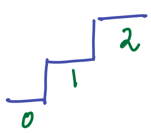


: 1 way :





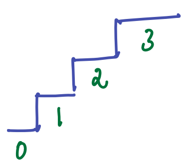
N=2 ways :



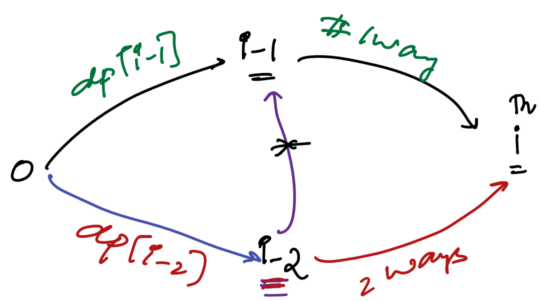
1 1
2

N=3 ways :

→ dp[i] = # ways to reach i^{th} Step

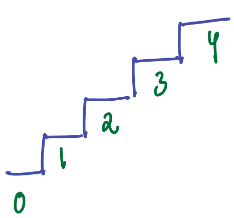


1 1 1
1 2
2 1



N=4

Total ways:



ways :

1 1 1 1 → {0 → 1 → 2

1 1 2

1 2 1

2 1 1

2 2

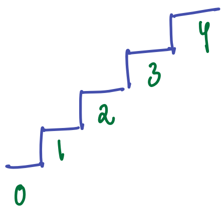
dp[i] = dp[i-1] + 2 dp[i-2] {wrong Express}

dp[0] = 1
dp[1] = 1

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

dp[3] = dp[2] + 2 dp[1] = 5

N = 4



ways :

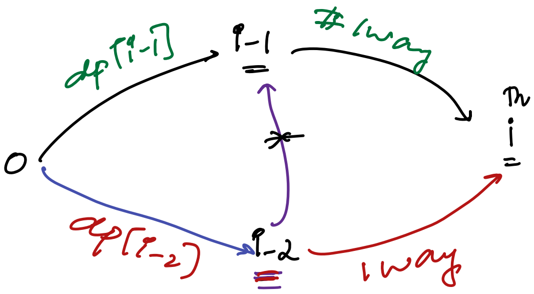
1 1 1 1 → {0 → 1 → 2 → 3 → 4}

1 1 2 → {0 → 1 → 2 → 4} → {0 → 1 → 2 → 3 → 4}

1 2 1 → {0 → 1 → 3 → 4}

2 1 1 → {0 → 2 → 3 → 4}

2 2 → {0 → 2 → 4}



Total ways { dp Expression

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

{Exactly Same as fib}

38) Given 6-phase dice, Number of ways we can get required Sum: {N}

Note: We can roll dice as many time as we want

\# ways to get N Sum

N=0 : 1

N=1 : 1 : 1

N=2: ways : 2

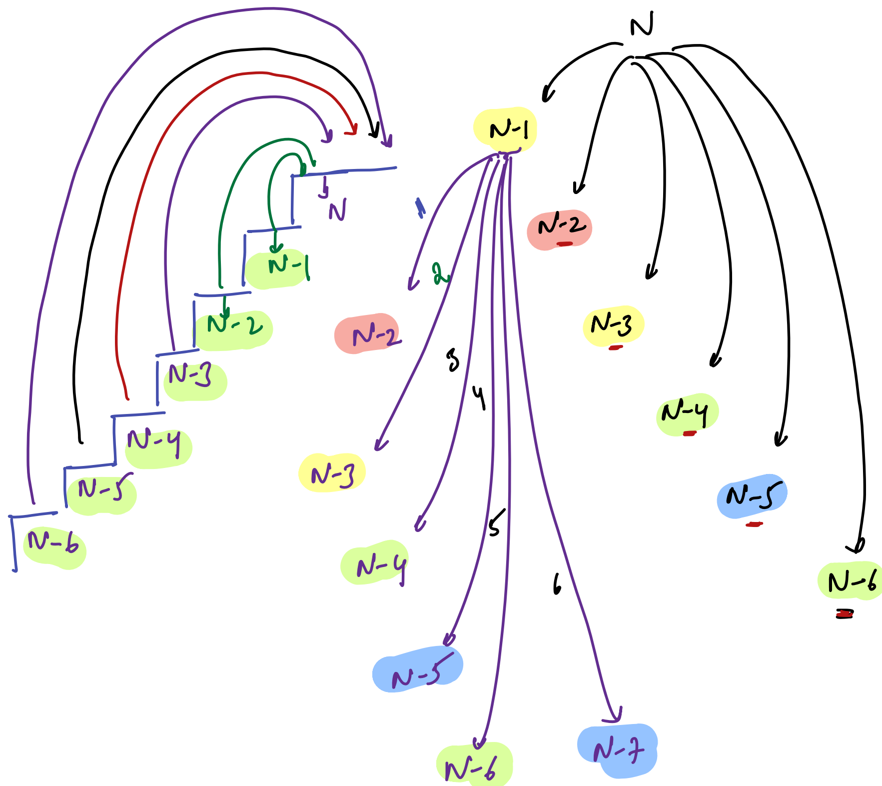
{ 1 1
2 }

N=3: ways : 4

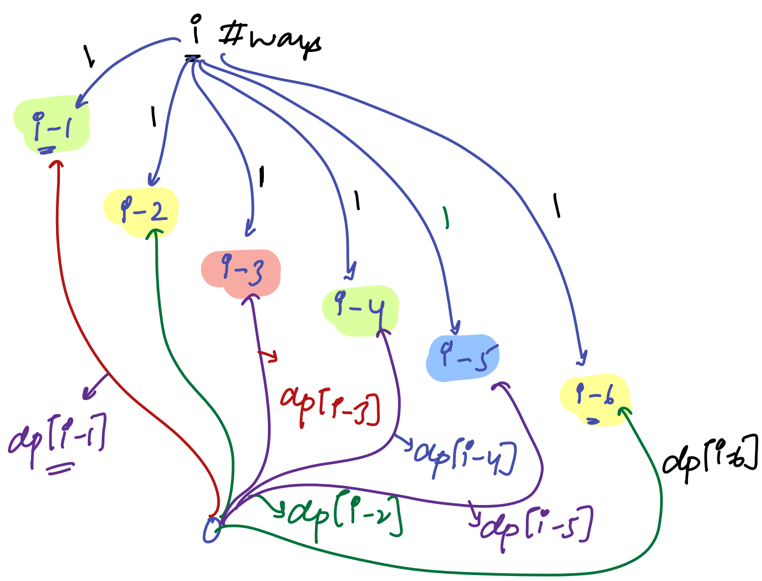
{ 1 1 1
1 2
2 1
3 }

N=4: ways : 8

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



// dp[i] = # no:of ways to get i



// dp Expression

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

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

// dp[N+1] :

Base Case:

: For all inputs for which code will fail

$dp[0] = 1$

$dp[1] = 1$

$dp[2] = 2$

$dp[3] = 4$

$dp[4] = 8$

$dp[5] = 16$

TC: # States × TC for each State

States → N, TC for each State → 6

TC: O(N)

SC: O(N)

// Space optimization:

→ Can do in 7 vars

→ a[7] | it out → (TODO)

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

i>=j → { Extra Condition }

dp[0] = { Edge Case }

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

$dp[1] = dp[0]$

Base Con:

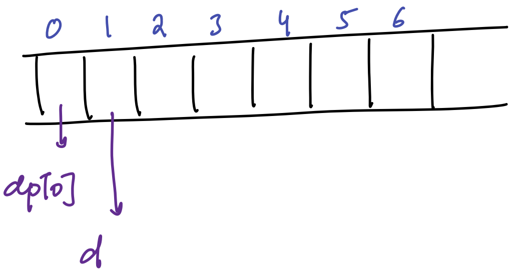
dp[0] = 1 =

Code:

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

// Space Optimization Pseudocode

dp[7] :



dp[i]		pos
0	→	0
1	→	1
2	→	2
3	→	3
4	→	4
5	→	5
6	→	6
7	→	0
8	→	1
9	→	2
10	→	3
11	→	4
12	→	5
13	→	6
14	→	0
15	→	1

$\boxed{dp[i] \rightarrow \underbrace{i \% 7}_{pos}}$

dp[7] = {0}

T.C : O(N)

S.C : O(1)

Base Con :

dp[0] = 1

Code :

```
i = 1 ; i <= N; i++) {
    s = 0;
    j = 1 ; j <= 6 && j <= i ; j++) {
        s = s + dp[ (i - j) % 7]
    }
    dp[i % 7] = s
}
return dp[N % 7]
```

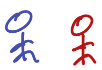
4Q) Given N persons, How many ways we can pair all people

Note: A person either wants to stay alone or get paired → (Ass)

(TODO: on Tuesday)

$N=1$:

$N=2$:



$N=3$:

4Q) Find min no of perfect squares needed to get sum = N

(Discuss on Tuesday)

N=6 :

N=10 :

N=9 :

N=12 :

// $\underset{+k}{\underset{\downarrow}{a_0}}\ \underbrace{a_1\ a_2\ a_3\ a_4\ a_5\ a_6}_{-k}$

$+k \qquad -k$

$+k \qquad -k$

$+k \qquad \frac{-k}{2}$