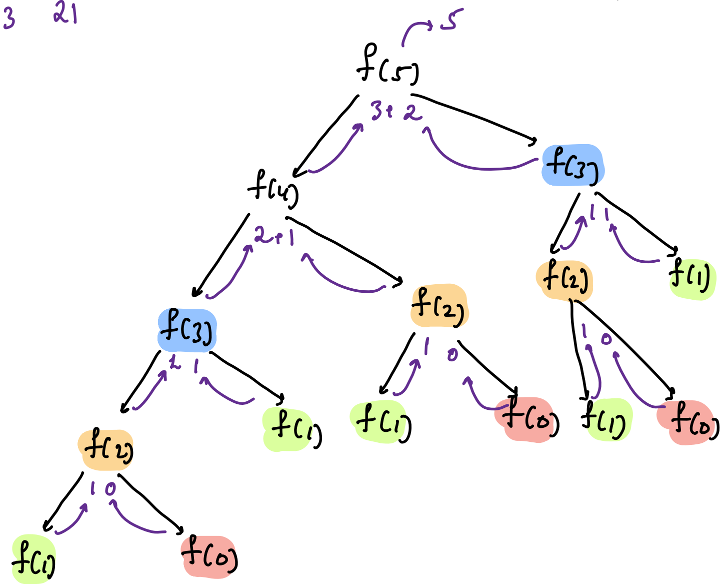
Todays Content :

- → Dynamic Programming Intro :
- → When to use Dp
- → Steps for Dp
- → # N Stairs
- → Sqrt( )

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| fib: | 0 | 1 | 1 | 2 | 3 | 5 | 8 | 13 | 21 |

```
int fib(int N){ TC ≈ O(2^n) ?
    if ( N<=1) {return N}
    return fib(N-1)+ fib(N-2)
}
```




Dynamic Programming:

: If same subproblems are called again & again,
Store subproblem & re-use them

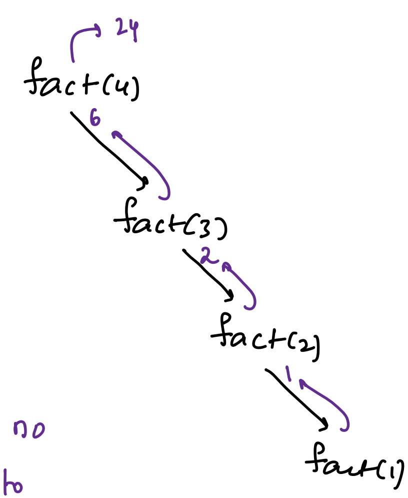
When to apply Dp?

a) Solve Problem with Sub Problems : Recursion / Optimal Substructure

b) Same Sub Problems Called again & again : Overlapping SubProblems

If above 2 occur, we optimize

```
int fact(n){
    if(n== 1) {return 1}
    return n * fact(n-1)
}
```




→ Sub Problems ✓

→ No Overlapping

: Nothing to store, no point in trying to optime.

## Memoization: Recursion + Memory

Dp table.

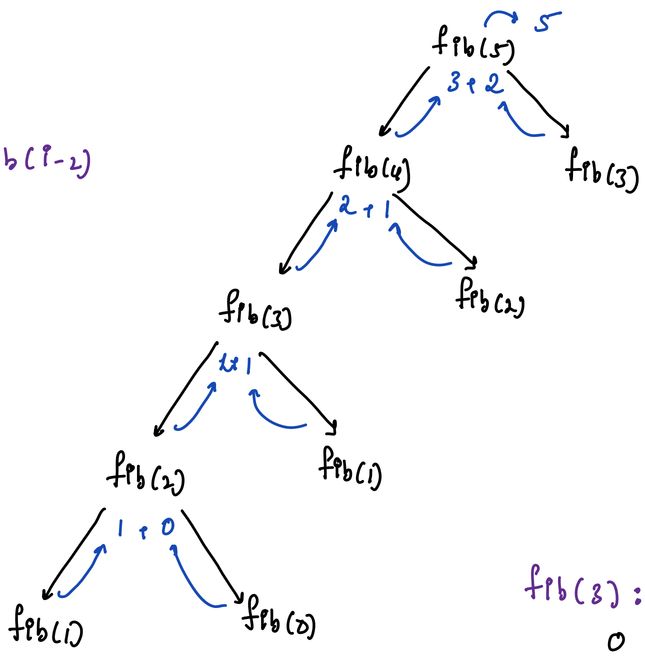
fib cannot be -ve, hence we are initialize it with -ve indicating subproblem called $1^{st}$ time

```
int dp[n+1] = -1 / INVALID        // here i^th fib num is dp[i]

int fib(i){      Note: store base won't effect logic
    if(i <= 1) { dp[i] = i; return i }
    if(dp[i] == -1){
        //Called 1st time
        dp[i] = fib(i-1) + fib(i-2)
    }
    return dp[i]
}
```

fib(5): dp[6] = -1

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| -1 | -1 | 1 | 2 | 3 | 5 |




fib(3): dp[4] = -1

| 0 | 1 | 2 | 3 |
|---|---|---|---|
| 0 | 1 | 1 | 2 |

## Tabulization: Iterative + Memory

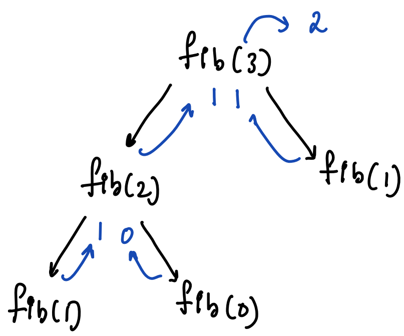
```
int fib(n){
    int dp[n+1] = {-1}
    dp[0] = 0  dp[1] = 1
    for( i = 2; i <= n; i++){
        dp[i] = dp[i-1] + dp[i-2]
    }
    return dp[n]
}
```

fib(5): dp[6] = -1

| 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 0 | 1 | 1 | 2 | 3 | 5 |

i = 2 .. 3 .. 4 .. 5

return dp[5] = 5



→ For a Problem When to apply Dp?

a) Solve Problems with SubProblems

b) Overalapping Subproblems

Dp Steps:

1. dpState : What is significance of value stored in table

$dp(i) =$ $dp[i] = i^{th}$ fib number

2. dp Expression : Solving state using subproblems

$dp(i) = dp(i-1) + dp(i-2)$

3. Final ans : In dp table at which we have ans

$dp[n]$

4. dp Table : Memory where we store subproblems & re-use it

5. TC : # no: of States * TC for each State

# $N * O(1)$

6. SC → : Memoization : Recursion + memory

: StackSize + TableSize

Tabulation : Iteration + memory

: TableSize.

7. Code.

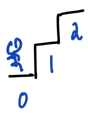
# N Stairs:

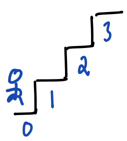
Q) Given N Steps, how many ways we can go from $0^{th} \to N^{th}$ Step

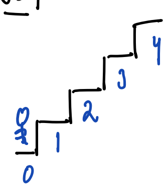
Note: from $i^{th}$ step we can directly go to $(i+1)^{th}$ or $(i+2)^{nd}$ Step:

Ex:

N=1  : 1, way: 1

N=2  : 0→1→2 :
0→2 :
ways: 2

N=3  : 0→1→2→3
0→2→3
0→1→3
ways: 3

N=4 
: 0→1→2→3→4
0→1→3→4
0→2→3→4 ] 3
0→1→2→4
0→2→4 ] 2

ways: 5

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

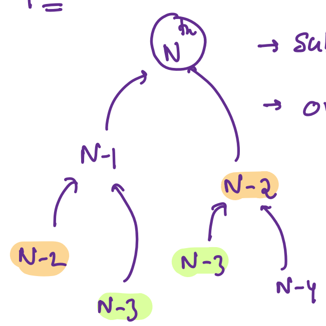
$dp[1] = 1, dp[2] = 2$

$dp[3] = dp[2] + 2 * dp[1] = 4$

$dp[4] = dp[3] + 2 * dp[2] = 8$

Note: Above value are wrong?

Step N:



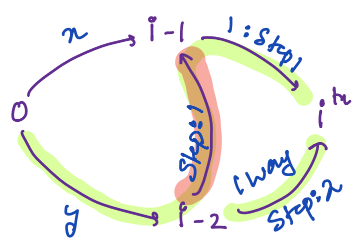

→ Subproblems

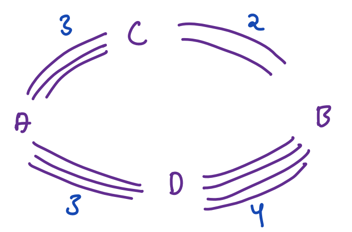
→ overlapping

Dp Steps:

Dp State: dp(i) = # no: of ways to reach from 0-i

Dp Exp: Solving state with Subproblems?




A → B via C = 6

A → B via D = 12

A → B via C a D = 18

# Say ways to reach 0, i-1 = x

# Say ways to reach 0, i-2 = y

# ways to reach 0 to i = x + y

$dp[i] = dp[i-1] + dp[i-2]$ ✓

Code: TODO

3Q) Find minimum number of perfect squares sum required to sum = N

N = 6 ans = 3

= $1^2 + 1^2 + 2^2 = 6$ : 3 pe

= $1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2$ : 6 pe

N = 10 : ans = 2

= $2^2 + 2^2 + 1^2 + 1^2$ = 4 pe

= $3^2 + 1^2$ = 2 pe

N = 9 : ans = 1

= $3^2$ = 1 pe

= $1^2 + 2^2 + 2^2$ = 3 pe

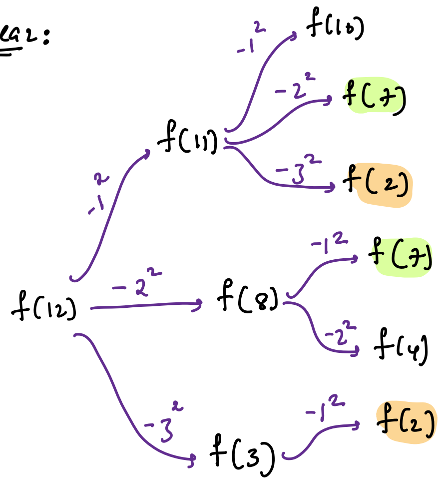
N = 12 ans = 3

= $3^2 + 1^2 + 1^2 + 1^2$ = 4 pe

= $2^2 + 2^2 + 2^2$ = 3 pe

Idea: Remove as big square possible: Greedy Fail

Idea2:




1) Subproblems

2) Overlapping

Dp Steps:

dpState: dp(i) = min no: of perfect square sums to get i

dpexpress: dp(i) =

Take min of all

$\xrightarrow{-1^2}$ 1 + dp(i-1) # min ps to get i-1

$\xrightarrow{-2^2}$ 1 + dp($i-2^2$) # min ps to get $i-2^2$

dp(i) $\xrightarrow{-3^2}$ 1 + dp($i-3^2$) # min ps to get $i-3^2$

$\xrightarrow{-4^2}$ 1 + dp($i-4^2$) # min ps to get $i-4^2$

⋮

$\xrightarrow{-j^2}$ 1 + dp($i-j^2$) # min ps to get $i-j^2$

if $i >= j^2$

Final ans: dp(n) : min pf square to get n

Dp table : int dp[n+1]

TC: # States * TC for each State SC: O(N + N) ≈ O(N)

$O(n) * \sqrt{n} = O(N\sqrt{N})$

**Code:**

```
int dp[n+1] = -1/INVALID

int minsq(int i){
    if(i==0){return 0}
    if(dp[i]==-1){
        // Called first time
        int ans = INT_MAX
        j=1; j*j <= i ; j++){
            // We subtract j^2 from i  rem val = i-j^2
            // rem val we need to get min no:of pf
            ans = min(ans, minsq(i-j2))
        }
        dp[i] = ans+1
    }
    return dp[i]
}
```