DP1 : One Dimentional

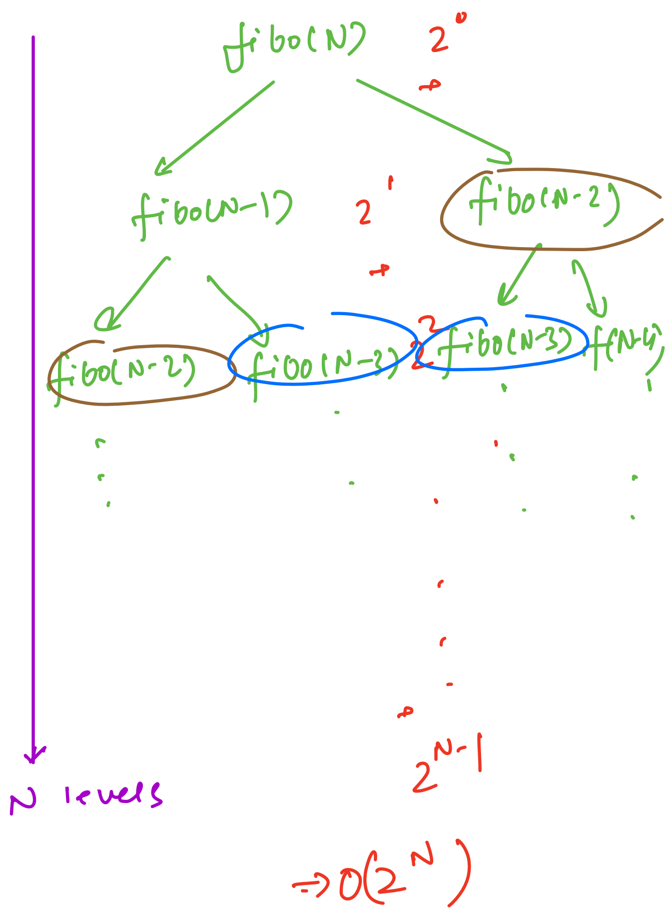
fibonacci Series

0 1 1 2 3 5 8 13 21 . . . . .

N → 0 1 2 3 4 5 6 7 8 . . . .

fibo(N)

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



TC = $O(2^N)$ SC = $O(N)$

Properties of Dynamic Programming (DP)

Optimal Substructure → Can the problem be divided into smaller subproblems.

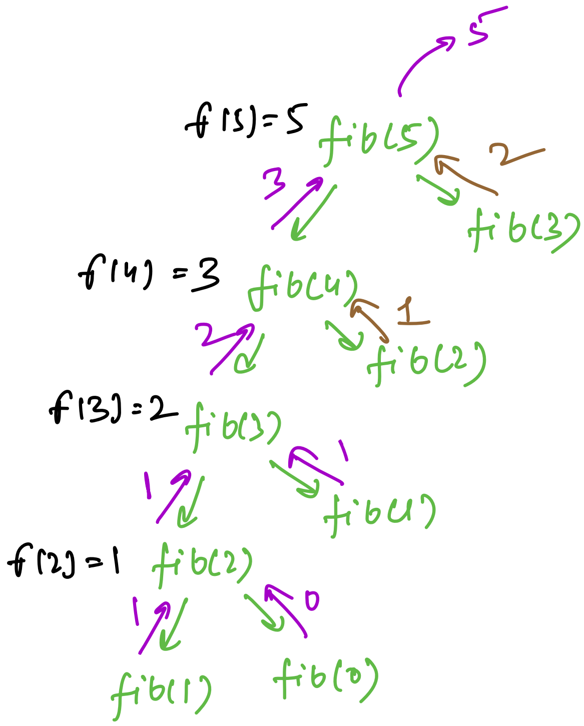
Overlapping Subproblems → Repeating subproblems.

↳ apply DP

Store the subproblem result & re-use it.

```
int f[N+1]   // initialize with -1
int fibo( int N) {
    if ( N <= 1)  return N
    if ( f[N] != -1)   return f[N];
    f[N] = fibo(N-1) + fibo(N-2);
    return f[N];
}
```

$\underbrace{fibo(N-2)}_{\downarrow \; f[N-2]}$



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

$$\Downarrow$$

$$f(N) = f(N-1) + 1$$ $\Rightarrow$ $TC = O(N)$ $\quad SC = O(N)$

$N = 20 \qquad 2^N = 2^{20} \approx 10^6$

2 types of DP solution

1. **Top down / Recursive DP** → start with main problem, divide it into smaller subproblem, solve & store subproblem result & use it to solve main problem.

memoization

2. **Bottom up / iterative DP** → start with smallest subproblem & use it to calculate bigger subproblem till we get the final result.

```
int f[N+1]
f[0] = 0 ,  f[1] = 1
for( i=2 to N) {
    f[i] = f[i-1] + f[i-2];
}
return f[N]
```

TC = O(N)

SC = O(N)

Recursive

Simple to write & understand code

Iterative ← start

no recursion space ⇒ there is possibility of reducing SC.

```
int f[N+1]
f[0] = 0 , f[1] = 1
for( i=2 to N) {
    f[i] = f[i-1] + f[i-2];
}
return f[N]
```

$f(2) = f(1) + f(0)$

$f(3) = f(2) + f(1)$

$f(4) = f(3) + f(2)$

⋮

optimize SC

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

N=5

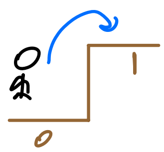
TC = O(N)

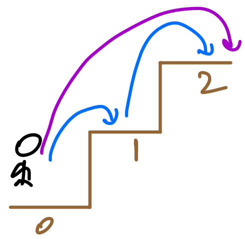
SC = O(1)

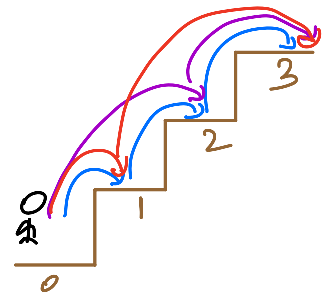
	a=0	b=1	
i=2	c=1	a=1	b=1
i=3	c=2	a=1	b=2
i=4	c=3	a=2	b=3
i=5	c=5	a=3	b=5

⋮

Question → find the number of ways to climb N stairs s.t. in 1 step we can climb either 1 or 2 stairs.







N=1 ans=1

[1]

N=2 ans=2

[1,1] , [2]

N=3 ans=3

[1,1,1] , [2,1] , [1,2]

N=4 ans=5

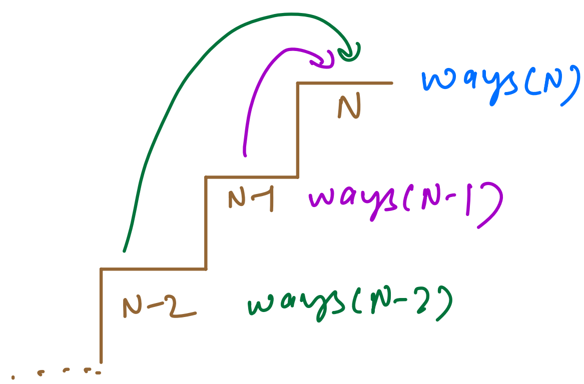
[1,1,1,1]

[1,1,2]

[1,2,1]

[2,1,1]

[2,2]



$$\boxed{\begin{aligned} ways(N) &= ways(N-1) + ways(N-2) \\ ways(1) &= 1 \\ ways(2) &= 2 \end{aligned}}$$

Similar to fibonacci series

3^{rd} step

2^{nd} step $\qquad$ 1^{st} step

$\begin{pmatrix} 1,1 \\ 2 \end{pmatrix}$ 1 $\qquad$ (1) 2 $\qquad \Rightarrow \begin{pmatrix} 1,1,1 \\ 2,1 \\ 1,2 \end{pmatrix}$

4^{th} step

3^{rd} step $\qquad$ 2^{nd} step

$\begin{pmatrix} 1,1,1 \\ 2,1 \\ 1,2 \end{pmatrix}$ 1 $\qquad$ $\begin{pmatrix} 1,1 \\ 2 \end{pmatrix}$ 2 $\qquad \Rightarrow \begin{pmatrix} 1,1,1,1 \\ 2,1,1 \\ 1,2,1 \\ 1,1,2 \\ 2,2 \end{pmatrix}$

n^{th} step

$(n-1)^{th}$ step 1 $\qquad$ $(n-2)^{th}$ step 2

Question

find the minimum number of perfect square required to get sum = N.

$1^2, 2^2, 3^2, \ldots\ldots$

1 4 9

N = 6 1 + 1 + 1 + 1 + 1 + 1

4 + 1 + 1 ans = 3

N = 10 9 + 1 ans = 2

N = 9 9 ans = 1

N = 5 $2^2 + 1^2$ ans = 2

(4 + 1)

Sol 1 → Greedy → select largest perfect sq <= N

N = 10 10 − 9 = 1

1 − 1 = 0 ans = 2

$N = 12$

$12 - 9 = 3$

$3 - 1 = 2$

$2 - 1 = 1$

$1 - 1 = 0$

$\cancel{ans} = 4$

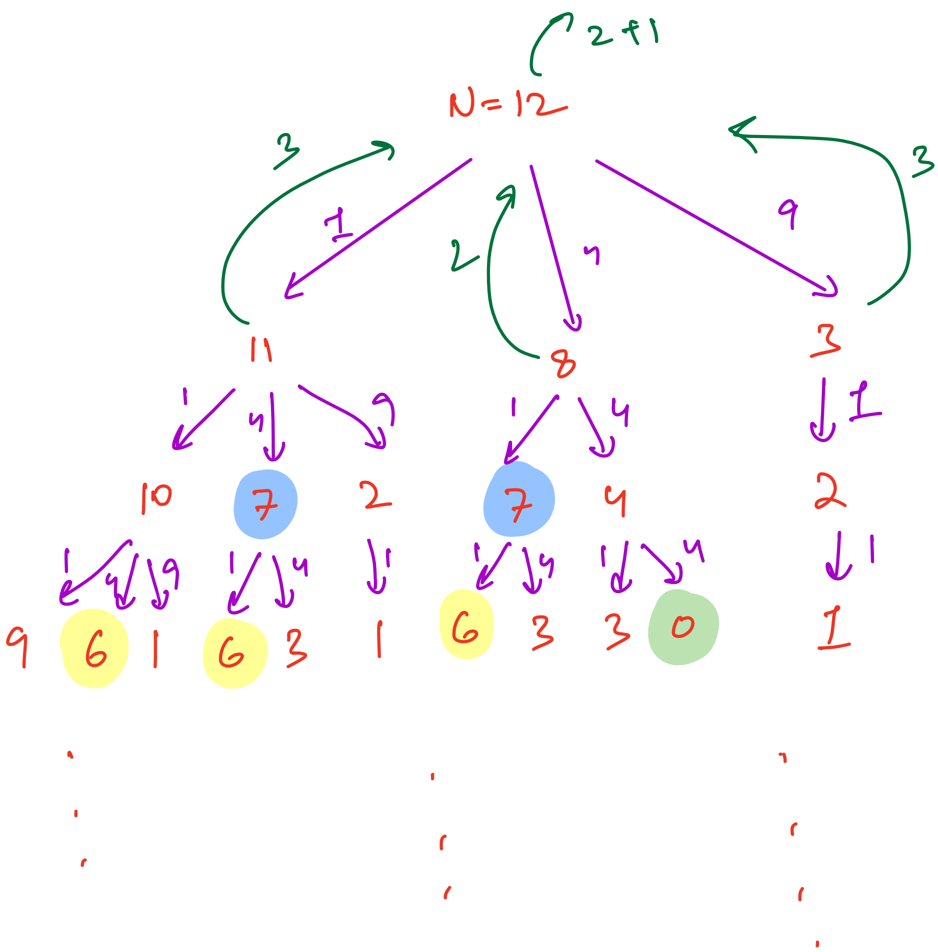
$12 - 4 = 8$

$8 - 4 = 4$

$4 - 4 = 0$

$ans = 3$

Bruteforce



optimal substructure?

overlapping subproblem?

⇒ use DP

$$\boxed{ans(N) = 1 + \min\{\, ans(N - x^2) \quad \forall\, x^2 <= N \,\}}$$

$$N \xrightarrow{1^2} (N-1^2) \qquad N \xrightarrow{2^2} (N-2^2) \qquad N \xrightarrow{3^2} (N-3^2) \quad \ldots$$

$(N-1^2) \quad (N-2^2) \quad (N-3^2) \quad \ldots$

code

```
int solve(N){
    if(N==0) return 0;
    ans=N
    for(x=1; x*x <=N; x++){        → √N times
        ans= min( ans, 1+ solve(N-x²));
    }
    return ans;
}
```

$TC = O\left((\sqrt{N})^N \right)$

$SC = O(N)$

```
int dp[N+1];   // initialize with -1

int solve(N) {
    if(N==0)   return 0;
    if( dp[N] != -1)   return dp[N]
    ans=N
    for( x=1 ; x*x <=N;  x++ ){        → √N times
        ans= min( ans, 1+ solve(N-x²));
    }
    dp[N] = ans;
    return ans;
}
```

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

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

$SC = O(N)$

Protip: TC of dp code = size of dp array * TC in single f^n call

* Disclaimer: not 100% correct, only works 99% of time.

Iterative code

```
int dp[N+1]
fi, dp[i] = i

for(i=1 to N) {
    for(x=1; x*x <= i; x++){
        dp[i] = min(dp[i], 1 + dp[i-x^2])
    }
}

return dp[N]
```

$TC = O(N\sqrt{N})$

$SC = O(N)$

not possible to reduce