Welcome ☺

Agenda: DP - dynamic programming

2 approaches.

2 problems.

---

chocolates → [3 1 2 3 2] ⇒ Total = 11

1 2 3 4 5

1 new student → 2 chocolates.

Total → 3 + 1 + 2 + 3 + 2 + 2 = 13

Total → ✓ $\boxed{11}$ + 2 = 13 ⇒ save time & effort

→ using already calculated value ⇒ Dynamic Prog.

eg: Prefix Sum

$P[i] = A[i] + P[i-1]$

↳ already calc. value.

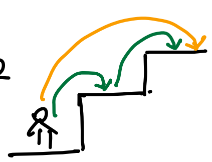
Q In how many ways can we climb N stairs, s.t in one step we can climb only 1 or 2 stairs.

N = 1

Ans = 1

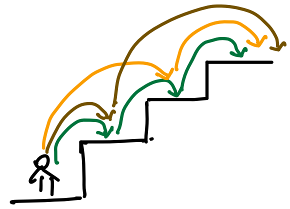
N = 2

Ans = 2



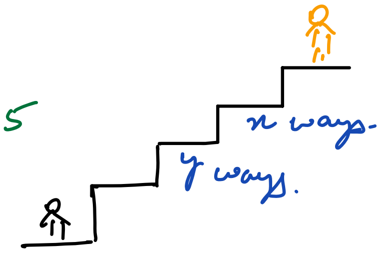
N = 3

Ans = 3



N=4

Ans = 5



Last Step

Total = x + y ways.

$$\text{ways}(N) = \text{ways}(N-1) + \text{ways}(N-2) = \text{Fib}(N+1)$$

N=0

Ans = 1 (no move)

Fibonacci Series

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

Recursion Code

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

T.C → $O(2^N)$

S.C → $O(N)$

Optimal substructure. ⇒ Ans of a big problem can be calculated using ans of its sub problems.

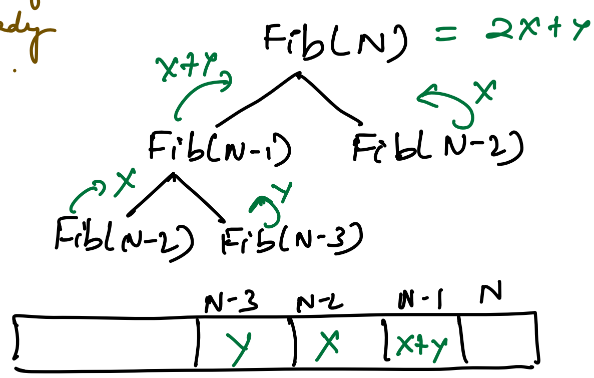
Overlapping subproblems ⇒ Same problem is calculated multiple times.

store the ans of subproblems to avoid recomputation.

```
int fib(N)
{
    if(N<=1) return N
    if(F[N]>0) return F(N) // checking if we have already calculated.
    // storing value.
    F[N] = fib(N-1) + fib(N-2)
    return F[N]
}
```

T.C ⇒ O(N)

S.C ⇒ O(N + N) — recursion, storage.



Types of DP

1) Top Down / Recursive approach

⇒ start with big problem, go down till the smallest subproblem for which you already know the answer. & use that to compute ans for bigger/original problem.

2) Bottom Up / Iterative approach

⇒ start with smallest subproblem for which you already know the answer & use it to iteratively get the ans for current problem.

```
F[0] = 0     F[1] = 1
for( i-> 2 to N )
{
    F[i] = F[i-1] + F[i-2]
}
return F[N]
```

T.C → O(N)

S.C → O(N)

Which approach to choose ??

Recursive DP $\Rightarrow$ easy to write code

Iterative DP $\Rightarrow$ No recursive space was used $\Rightarrow$ There are chances to optimize space.

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

T.C $\Rightarrow$ $O(N)$

S.C $\Rightarrow$ $\cancel{O(N)} \rightarrow O(1)$

Q Find the min. no. of perfect square to add to get sum = N   1 4 9 16 25 . . . .

N		Ans.
1	1	1
2	1+1	2
3	1+1+1	3
4	4	1
⋮		
10	3^2+1^2	2

x – (largest perfect square $\leq x$)

Greedy solⁿ. ✗

N = 80

$80 - 8^2 = 80 - 64 = 16$

$16 - 4^2 = 16 - 16 = 0$

Ans = 2

N = 12

$12 - 3^2 = 12 - 9 = 3$

$3 - 1^2 = 2$

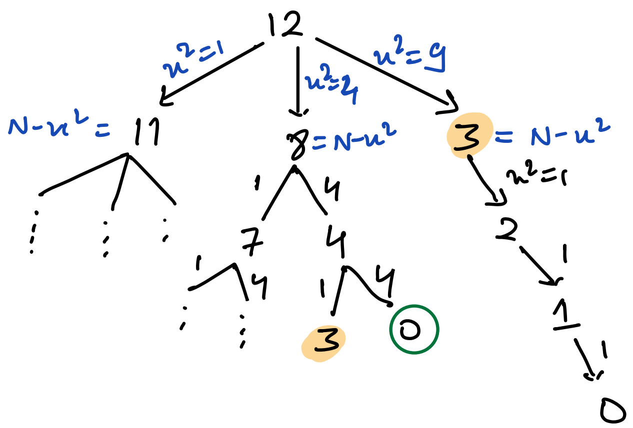
$2 - 1^2 = 1$

$1 - 1^2 = 0$

Ans = 4 ✗

$12 \Rightarrow \underbrace{4 + 4 + 4}$

Ans = 3 ✓



Overlapping subproblem ✓

Optimal substructure ✓

$$\boxed{\begin{array}{c} count[N] = \min(1 + count[N - x^2]) \\ \forall x \ \text{s.t} \ x^2 \leq N \end{array}}$$

Code

```
ans[0] = 0
for( i -> 1 to N)
{
    ans[i] = i      // 1^2 + 1^2 + 1^2 + 1^2 . . . . . i times.
    for( x=1 ; x*x <= i ; x++ )
    {
        ans[i] = min( ans[i] , 1+ ans[ i-(x*x)] )
    }
}
return ans[N]
```

T.C -> $O(N\sqrt{N})$

S.C -> $O(N)$

ans =

0 ✓	1 ✓	2 ✓	3 ✓	~~4~~ 1
0	1	2	3	4

i = ~~1~~ ~~2~~ ~~3~~ 4