Welcome ☺

Agenda: DP
Types
3-4 problems.

Dynamic Programming → Save & Reuse.

choclates → [3 1 2 3 2]

3 + 1 + 2 + 3 + 2 = 11

1 new student ⇒ 2

Total → 11 + 2 → efficient saves time & effort

→ 3 + 1 + 2 + 3 + 2 + 2 = 11

[3 1 2 3 2 2]

Q Fibonacci series

0 1 1 2 3 5 8 . . . . . .

fib(N) = fib(N-1) + fib(N-2)

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

T.C ⇒ $O(2^N)$

S.C ⇒ recursive space.

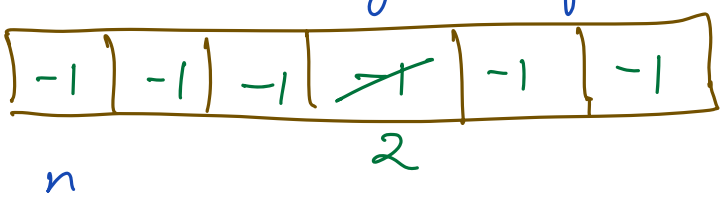
2 properties

1. Optimal substructure ⇒ Ans of a big problem can be calculated using ans of its subproblem.

2. Overlapping subproblems ⇒ Same problems is calculated multiple times.

Memoization

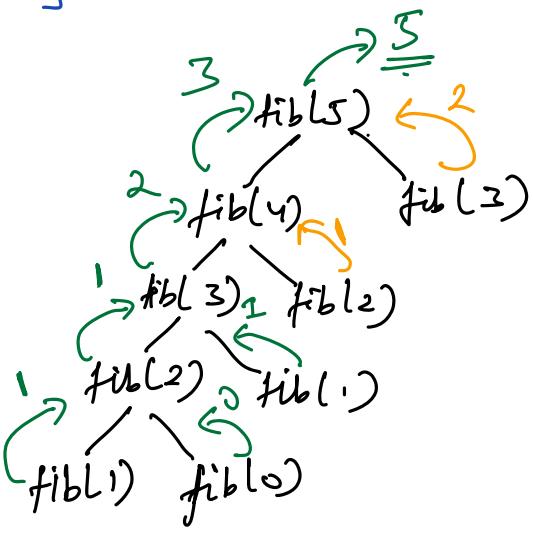
array / map

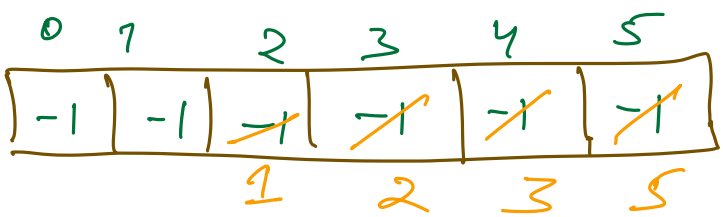



```
int fib(n)
{
    if(n <= 1) return n
    if(dp[n] != -1) return dp[n]  // Reuse
    dp[n] = fib(N-1) + fib(N-2)   // Save.
    return dp[n]
}
```

T.C ⇒ O(N)

S.C ⇒ O(N) + recursive stack.





Types of D.P

1. Top Down → Memoization. → recursive app.

→ 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. and use it to iteratively get the answer for current problem.

Iterative code for fibonacci

```
F[0] = 0        F[1] = 1

for( i=2 ; i<=N ; i++)
{
    F[i] = F[i-1] + F[i-2]
}
return F[n]
```

T.C ⇒ O(N)

S.C ⇒ O(N) no recursive space.

Which approach to choose ?

Recursive DP ⇒ Easy to code.

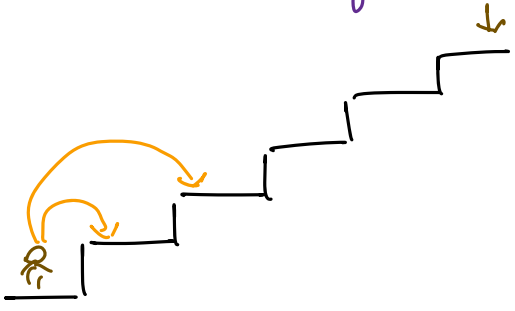
Iterative DP ⇒ No recursive space is required ⇒ There are chances to optimize space.

```
a = 0      b = 1
for ( i=2 ; i<=n ; i++)
{
    c = a + b
    a = b
    b = c
}
return c
```

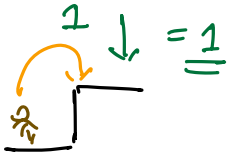
T.C ⇒ O(N)

S.C ⇒ ~~O(N)~~ → O(1)

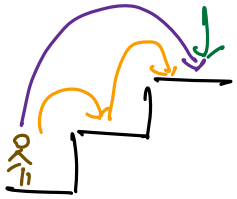
Q Calculate # ways to reach N^{th} stair. You can take 1 step or 2 steps at a time.



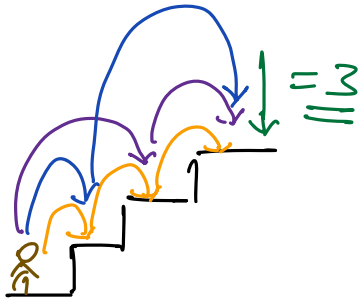
N=1



N=2



N=3



Step(3) = # way step(2) + # way step(1)

ways (N) = # ways (N-1) + # ways (N-2)

Same as fibonacci

Q Find min # of perfect squares required to get sum = N (duplicates are allowed)

eg: sum = 6

$\Rightarrow 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 \Rightarrow 6$

$\Rightarrow 1^2 + 1^2 + 2^2 \Rightarrow 3$ perfect square.

Quiz sum = 5

$\Rightarrow 1^2 + 1^2 + 1^2 + 1^2 + 1^2 \Rightarrow 5$

$1^2 + 2^2 \Rightarrow 2$ ✓

eg: 12 $\Rightarrow$

$12 - 3^2 = 3$

$3 - 1^2 = 2$

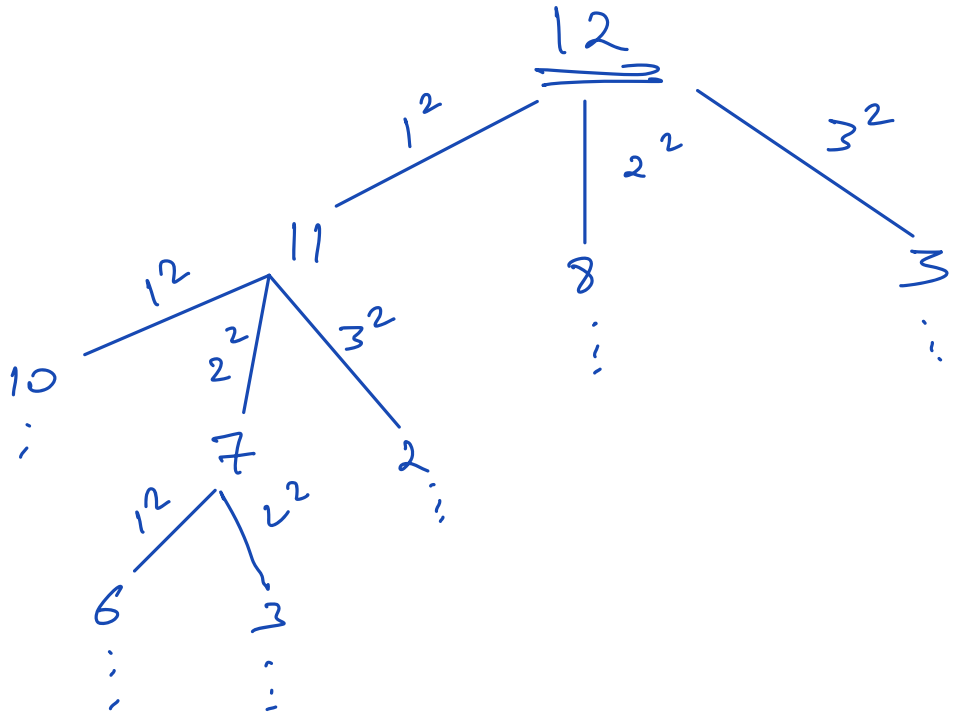
$2 - 1^2 = 1$

$1 - 1^2 = 0$

} 4 perfect square.

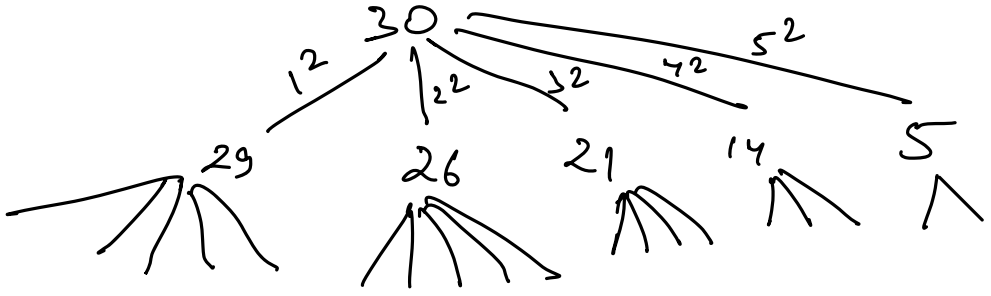
12 $\Rightarrow$ $12 - \underline{2^2} - \underline{2^2} - \underline{2^2} = 0$

✓ 3 perfect squares



squares(i) $\Rightarrow$ $1 + \min\{ \text{squares}(i - n^2) \}$

$\forall\ n^2 \leq i$



Code

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

int psquares(N, dp[])
{
    if(N==0) return 0
    if(dp[N] != -1) return dp[N]
    ans = INT_MAX
    for(i=1; i^2 <= N; i++)
    {
        ans = min(ans, psquare(N-i^2))
    }
    dp[N] = 1 + ans
    return dp[N]
}
```

T.C ⇒ O(N)

S.C ⇒ O(N) + rec.

Q # notes required for the amount

Notes are → 50, 30, 5

greedy not always solves.

eg: 65 ⇒

65 − 50 = 15

15 − 5 = 10

10 − 5 = 5

5 − 5 = 0

} 4 notes

$65 \Rightarrow 30 + 30 + 5 \Rightarrow 3$ notes

$\Rightarrow$ minNotes(N) = 1 + min(minN(N-50), minN(N-30), minN(N-5))

```
int minNotes( N, dp[])
{
    if( N==0) return 0
    if( N< 0) return INT_MAX
    if( dp[N] != -1) return dp[N]

    ans = INT_MAX
    ans = min( ans , minNotes( N-50 ))
    ans = min( ans , minNotes( N-30 ))
    ans = min( ans , minNotes( N-5 ))

    dp[N] = 1 + ans ;
    return dp[N]
}
```

T.C = O(N)

S.C = O(N)

M.U → Iterative code

Doubts

map < int , pair < int , vector > >

i

min # psquare combination