DP 1 : One Dimentional

* Points where people fail
  - → Attendance
  - → Solving
  - → Revision

⇒ DSA → mock Interview → Content

Question :- Fibonacci Series

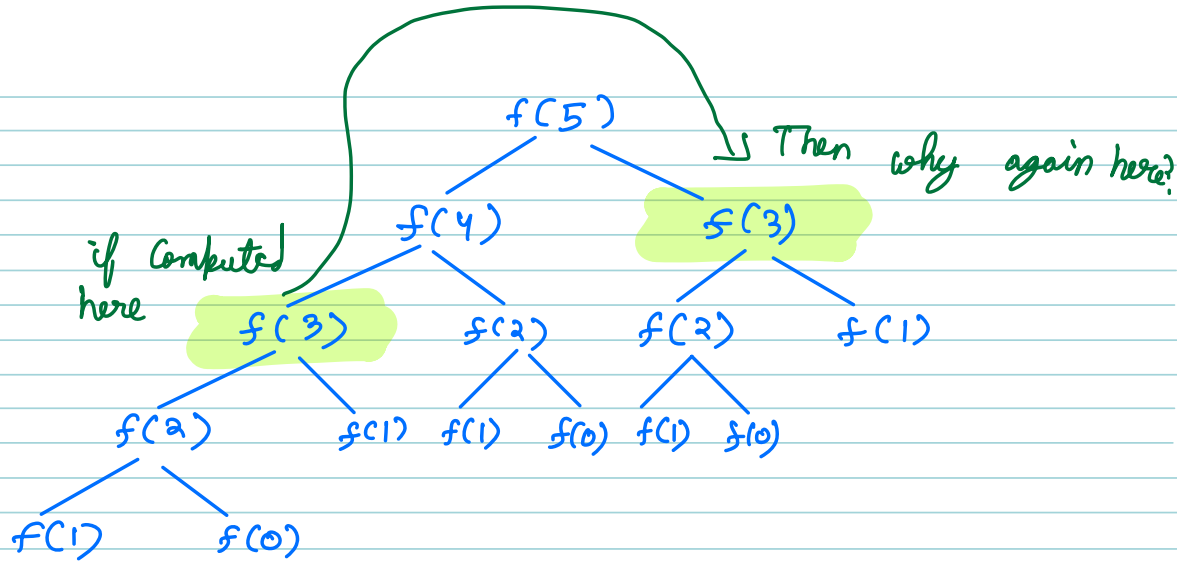
Ex:- 0 1 1 2 3 5 8 13 21 .....

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

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

TC - $O(2^n)$

SC - $O(n)$



⇒ Impact Analysis.

↳ $f(10) \approx 2^{10} = 1024$

↳ $f(20) \approx 2^{20} \approx 10^6$

Problem :- Too many Recursion calls

CONDITION FOR DP

① Optimal Sub Structures :- Solve a problem with smaller problems.

② Overlapping Sub problems :- The Sub Structure are repeating

DP → Solve a problem only once.

Updated PSEUDO CODE

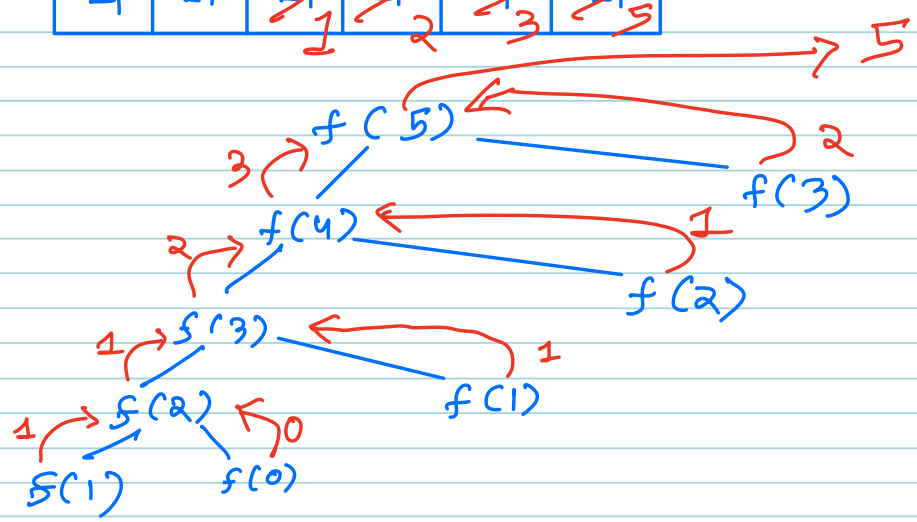
```
int Solve (n) {
    int dP[n+1] = {-1};
    return fib(n, dP);
}
int fib (n, dP) {
    if ( n <= 1) { return n; }
    if ( dP[n] != -1) { return dP[n]; }
    Ans = fib(n-1) + fib(n-2);
    dP[n] = Ans;
    return dP[n];
}
```

Caching

DRY RUN

dP=

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



★ Way for computing TC

TC → No. of States * Time taken by each state Recursion

↓

length of dp array.

≈ O(n * 1) ≈ O(N)

SC → O(N)

↳ N + N

↙ DP array

↘ Recursion Stack

★ Any DP Problem can be Solved in 2 ways:-

Types of DP

Recursion (Memoization) / TOP-DOWN

Iterations (Tabulation) / BOTTOM-UP

TABUCATION PSEUDO CODE

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

dp=

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

TC — O(N)
SC — O(N)

Q what is Similar in Memoization & Tabulation

↳ dp[i] = dp[i-1] + dp[i-2];

↳ This is DP Expression.

Q At one time how many DP states are being used.

↳ 3 (i^{th} , $(i-1)^{th}$, $(i-2)^{th}$)

PSEUDO CODE [SPACE optimization]

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

DRY RUN

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

TC — O(N)
SC — O(1)

* We can optimize Space after tabulation if your current state dependent upon some fixed Previous states

Prefix Sum

PS[5] = PS[4] + Arr[5]

↑ This was also a DP.

Quiz 1 :- What is the purpose of memoization in dynamic programming?

To store & reuse solutions to subproblems

Quiz 2 :- which approach is considered as an iterative process?

↳ BOTTOM — UP.

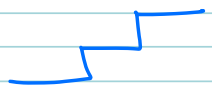
10:23 — 10:30

Question :- Calculate the no. of ways to reach N^{th} Stair. You can take 1 step at a time or 2 steps at a time.

Eg1 :- $n = 1$

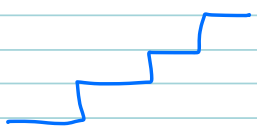
$\Rightarrow \{1\} \rightarrow 1$ Way

Eg2 :- $n = 2$



$\Rightarrow \{1\,1, 2\} \rightarrow 2$ Ways

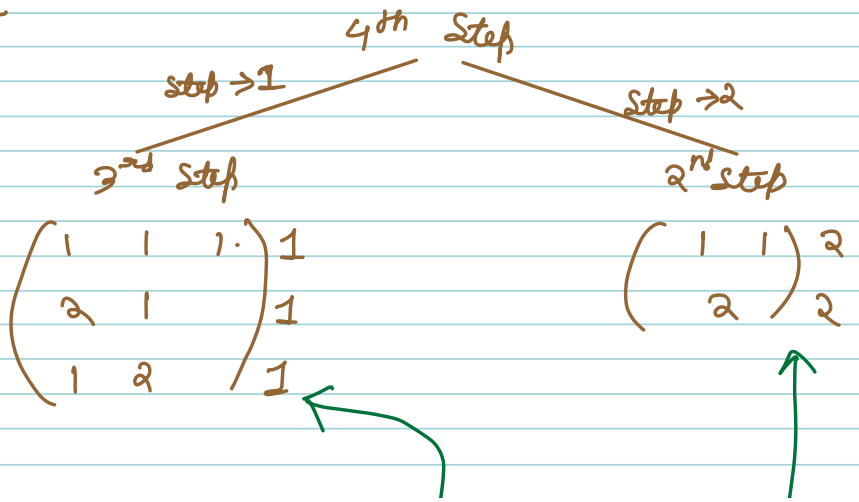
Eg3 $n = 3$

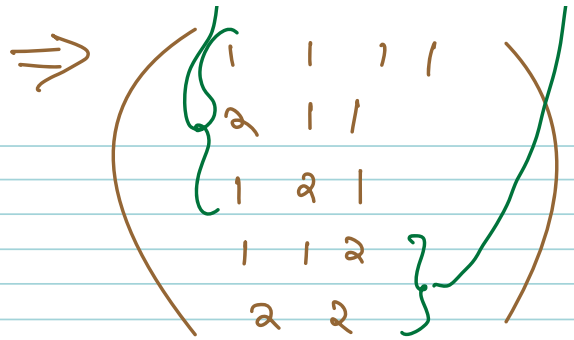


$\Rightarrow \{1\,1\,1, 2\,1, 1\,2\} \rightarrow 3$ Ways

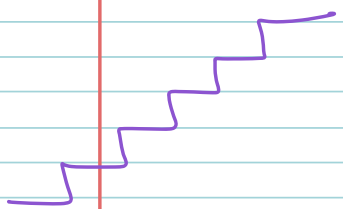
Quiz 3 :- In Stair problems, the result for N=4

N=4





Q No. of ways to reach N = 5?



N = 5

Step of length = 1 | Step of length = 2

N = 4 | N = 3

1 1 1 1	1
2 1 1	1
1 2 1	1
1 1 2	1
2 2	1

1 1 1	2
2 1	2
1 2	2

No. of Way (5) = No. of Ways (4) + No. of Way (3)

= 5 + 3

= 8

Generalize

N^{th} Step

$(N-1)^{th}$ Step | $(N-2)^{th}$ Step

No. of Ways (N^{th}) = No. of Way $(N-1)^{th}$ + No. of Way $(N-2)^{th}$

DRY RUN using DP table

0	1	2	3	4	5
1	1	2	3	5	8
	1	2 1 1	2 1 1 1 1 1 2	2 1 1 1 1 1 1 1 2 1 2 2 1 1 2	2 1 1 1 1 1 1 1 1 1 2 1 1 2 2 1 1 1 2 1 2 1 2 1 1 1 2 1 2 2

Q How many ways for reaching 0th step?

↳ 1 [Do Nothing, like 5 Star AD]

Q what I will be Storing in Each DP state?

↳ DP[i] = No. of ways to reach ith step

⇒ DP Expression

step of 1 / step of 2

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

CONCLUSION :- It is just a fibonacci Series.

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

return dp[n];

↳ Space optimisation

Question :- Find minimum no. of perfect squares (PS) required to get Sum = N

↳ 1, 4, 9, 16, 25.....

Note :- Duplicate Squares are allowed

Eg :- N=2 $\longrightarrow$ $1^2 + 1^2$ = 2 PS

N = 3 $\longrightarrow$ $1^2 + 1^2 + 1^2$ = 3 PS

N = 4 $\longrightarrow$ 2^2 = 1 PS

Quiz 4 :- What is minimum no. of perfect Squares required to get Sum = 5 [Duplicates are allowed]

↳ Sum = 5 $\longrightarrow$ $2^2 + 1^2$ $\longrightarrow$ 2 PS

Eg N = 6 $\longrightarrow$ $2^2 + 1^2 + 1^2$ $\longrightarrow$ 3 PS

N = 50 $\longrightarrow$ $7^2 + 1^2$ OR $5^2 + 5^2$ $\longrightarrow$ 2 PS

IDEA :- GREEDY

↳ Try to subtract highest Perfect Square

Eg :- N=50 $\longrightarrow$ $50 - 7^2 = 1 - 1^2 = 0$

↳ 2 PS

Eg :- N = 70 $\longrightarrow$ $70 - 8^2 = 6 - 2^2 = 2 - 1^2 = 1 - 1^2 = 0$

↳ 4 PS

↓ then what is this?

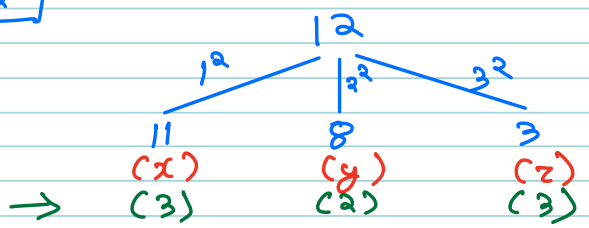
N = 70 $\longrightarrow$ $70 - 6^2 = 34 - 5^2 = 9 - 3^2 = 0$

↳ 3 PS

Hence, Greedy Not a feasible Solution.

IDEA :- Explore all possible ways

Eg :- N = 12

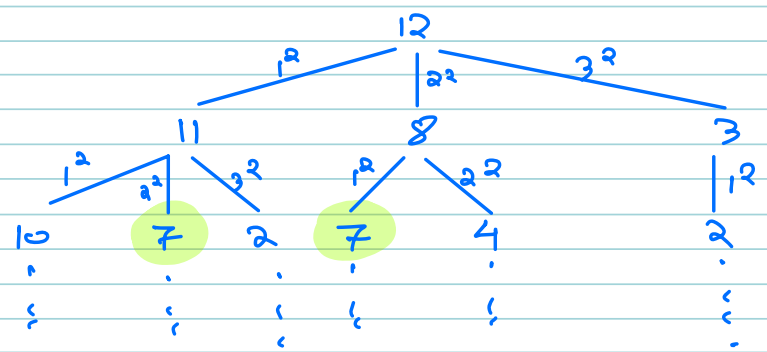


Q If you already know the ans for 11, 8, 3 then ans for 12 will be ?

$$Ans(12) = Min[Ans(11), Ans(8), Ans(3)] + 1$$

Because you need to add one PS to get 12.

Lets Expand a tree



PSEUDO CODE

```
int PerfectSquares(N) {
    if (N == 0) { return 0; }
    ans = ∞;
    for(i = 1; i*i <= N; i++){
        ans = Math.min(ans, PerfectSquares(N - i^2));
    }
    return ans + 1;
}
```

Memoization PSEUDO CODE

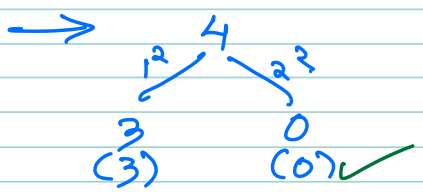
```
int dp[n+1] = {-1};
int PerfectSquares(N) {
    if (N == 0) { return 0; }
    ans = ∞;
    if (dp[N] != -1) { return dp[n]; }
    for(i = 1; i*i <= N; i++){
        ans = Math.min(ans, PerfectSquares(N - i^2));
    }
    dp[N] = ans + 1;
    return dp[N];
}
```

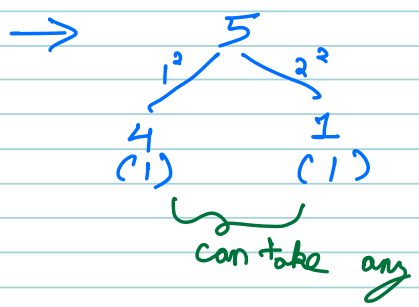
TC — $O(N * \sqrt{N})$

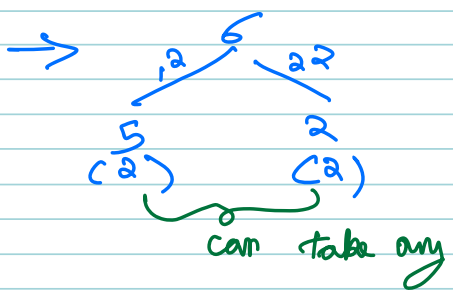
Q what you will be storing in dp array?

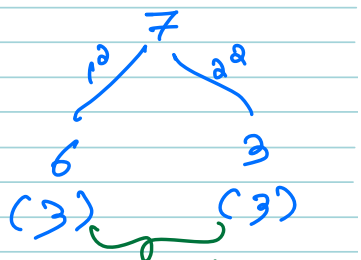
↳ dp[i] = Min PS to form i

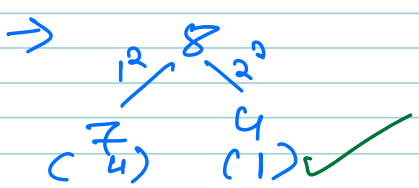
0	1	2	3	4	5	6	7	8	9
0	1	2	3	1	2	3	4	2	1
	1^2	1^2+1^2	$1^2+1^2+1^2$	2^2	1^2+2^2	$1^2+2^2+1^2$	$1^2+2^2+1^2+1^2$	2^2+2^2	3^2

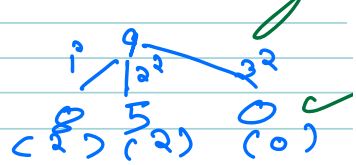












PSEUDO CODE (TABULATION)

```
dp[N+1] = {∞}
dp[0] = 0;
for (i = 1 ; i <= N; i++){
    for (x = 1; x * x <= i ; x++){
        dp[i] = min (dp[i], dp[i-x*x])
    }
    dp[i] += 1;
}
return dp[N];
```

TC ⟶ $O(N\sqrt{N})$

SC ⟶ $O(N)$

Q. Can we optimize over Space ?

↳ Space optimization is not possible since we can't determine a constant no. of previous data for every index.

question

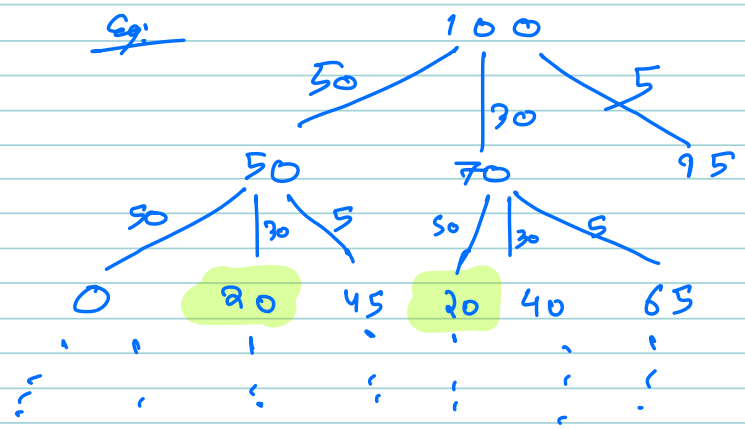
Problem :

RBI wants to reduce **paper usage** for money. Imagine you need to withdraw a specific amount of money from an ATM. The ATMs should be programmed to give you the least number of notes possible.

The available notes in the ATM are **₹50, ₹30, and ₹5**. Your task is to figure out the **minimum number of notes** the ATM should give you for any amount of money you request, ensuring the ATM dispenses the exact amount you asked for.

IDEA :- Explore all possible ways

Eg:



DP Expression

$$dp[i] = \min(dp[i-50], dp[i-30], dp[i-5]) + 1$$

PSEUDO CODE

↳ H.W