Today's content

1) Dynamic Programming Into
2) When to use DP
3) Steps for DP
4) # N stairs
5) sqrt( )

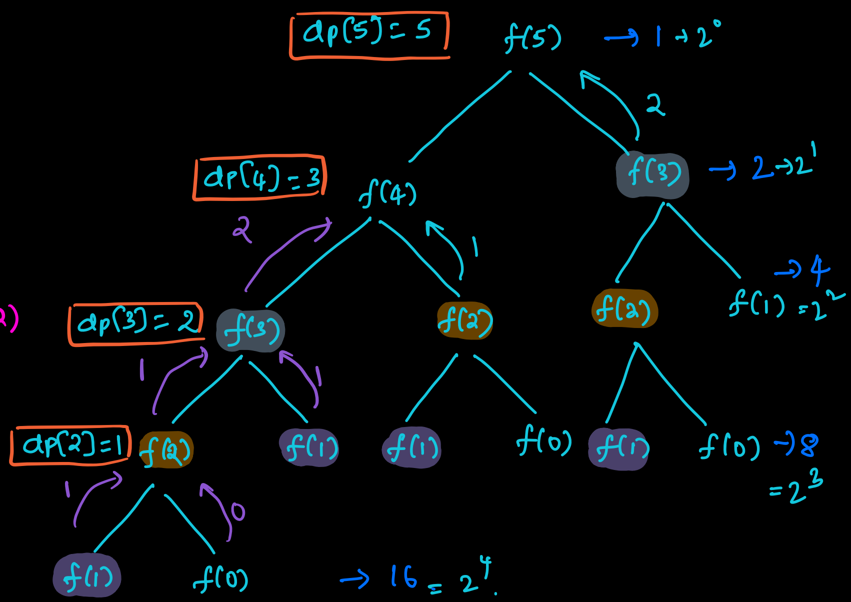
1. Write a recursive function to generate N^{th} fibonacci number.

	0	1	2	3	4	5	6	7	8	9 → fib(8) + fib(7)
Fib:	[0	1	1	2	3	5	8	13	21	34] ← dp.

```
int   fib(int N)
{
     if(N ≤ 1)
          return  N

     return  fib(N-1)+fib(N-2)
}
```

TC: $O(2^n)$



In each level, at the max, how many entries can be there?

```
int   fib(int N)
{
     if(N ≤ 1)
          return  N
     if (dp[N] == -1)
     |    dp[N] = fib(N-1)+fib(N-2)
     |    return dp[N]
     // already calculated
     return  dp[N]
}
```

index	0	1	2	3	4	5
dp[]	[-1	-1	1	2	3	5]

TC: $O(N)$

DP: Every unique subproblem should be solved only once.

Solving problems : The problem can be divided into subproblems. ; recursive

Overlapping : Same problems are called again and again.

Types of DP.

(i) Top - down (memoization)

(ii) bottom - up (no overhead of recursion stack

	0	1	2	3	4	5
dp[]	0	1	1	2	3	5

```
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]
}
```

i= 2 dp[2] = dp[1] +dp[0]

3 dp[3] = dp[2]+dp[1]

4 dp[4] = dp[3] +dp[2]

5 dp[5] = dp[4]+dp[3].

SC: O(N), TC: O(N).

Steps for DP.

i. solve it with subproblems
ii. Overlapping in subproblems

} When to use.

Optimization with DP.

(i) dp state : What are we storing , dp[i] → i^{th} fib no.

(ii) dp exp^n : Calculating dp state using subproblems.

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

(iii) dp table : Space to store all subproblems, so that we can re-use them.

dp[N+1] → size

(iv) code

(N+1) * (1) => TC : O(n).

TC : (# No. of dp state) * (TC for each state)

SC : dp table size + (stack size)
(memoization)

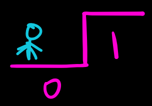
N Stairs.

Q. Given N steps, In how many ways we can go from $0^{th} \rightarrow N^{th}$ step.

Note: From i^{th} step, we can directly go to $(i+1)^{th}$ or $(i+2)^{th}$ step.

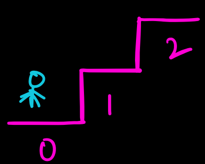
Ex:

N=1



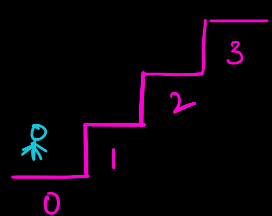
ways = 1

N= 2



Ways = 2

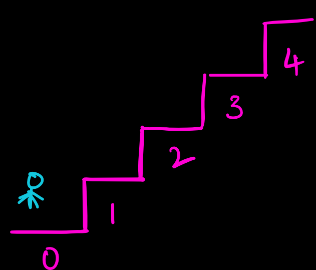
N=3



ways : 3

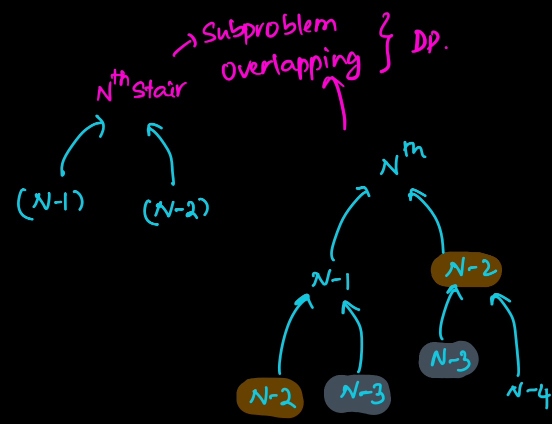
1 1 1
1 2
2 1

N=4



ways = 5

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



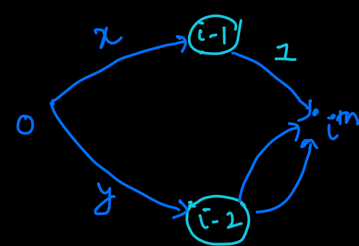
dp steps:

dp state:

dp[i] → No. of ways to reach i^{th} step.

dp expression:

solving dp state using subproblems.



ways to reach i^{th} step from 0^{th} step.

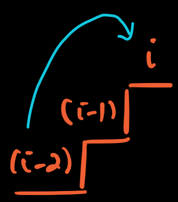
$= x*1 + y*2 = x + 2y$ (X)

(X) $\boxed{dp[i] = dp[i-1] + 2*dp[i-2]}$

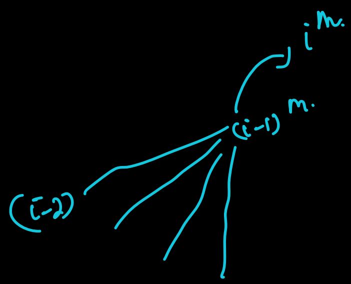
$dp[1] = 1, dp[2] = 2.$

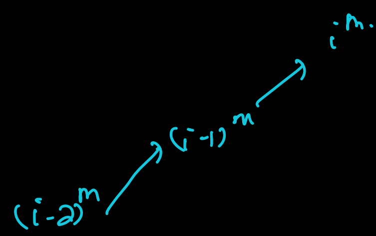
$dp[3] = dp[2] + 2*dp[1]$

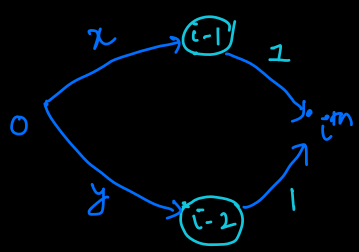
$= 2 + 2*1 = 2+2 = 4.$



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







$dp[i] = x + y$

$\boxed{dp[i] = dp[i-1] + dp[i-2]}$

Code: TO-DO.

Q. Find minimum no. of perfect squares required to make sum = N.

$N = 6$ $\quad 1^2+1^2+1^2+1^2+1^2+1^2 \rightarrow$ 6 no's are used

$2^2+1^2+1^2 \rightarrow$ 3 no's are used

ans = 3.

$N = 10$ $\quad 1^2+1^2+ \dots +1^2 \rightarrow$ 10 no's

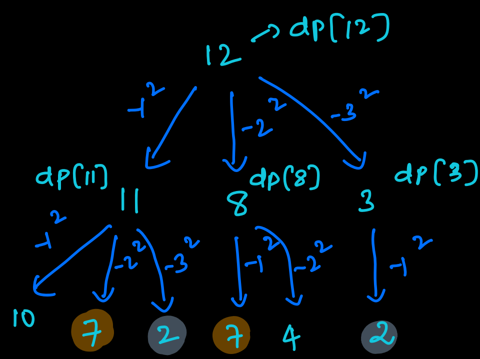
$2^2+2^2+1^2+1^2 \rightarrow$ 4 no's

$3^2+1^2 \rightarrow$ 2 no's

ans = 2.

$N = 9$

$\hookrightarrow 3^2$, ans = 1.



subproblems
overlapping } DP.

$N = 12.$

$3^2+1^2+1^2+1^2 \rightarrow$ 4 no's. $\quad$ ans = 3

$2^2+2^2+2^2 \Rightarrow$ 3 no's

dp States:

dp state : dp[i] → minimum no. of squares I need to get 'i'.

dp expⁿ : Solving dp state using subproblems.

$$dp[i] = \min \begin{cases} dp[i-1^2] + 1 \\ dp[i-2^2] + 1 \\ dp[i-3^2] + 1 \\ \vdots \\ dp[i-j^2] + 1 \end{cases} \qquad j = 1, 2, 3 \dots \sqrt{i}$$

$j^2 \le i \Rightarrow j \le \sqrt{i}$

dp table : final ans -> dp[n].

hence size is dp[n+1].

tc : (# No. of dp state) * (Tc for each state)

$(n+1) * (\sqrt{n})$

=> $\boxed{Tc: O(n\sqrt{n})}$

Code:

```
dp[N+1] = N+1

int minsquares( int i)
{
    if(i<=0)
        return 0.

    if (dp[i] == N+1)

        for (j=1; j<=√i; j++)
            dp[i] = min(dp[i], minSquares(i-j²)+1)

    return dp[i]
}

main ( )
{
    minSquares(12).
}
```

iterative

```
int dp[N+1] = N+1
dp[0] = 0 .
for (i=1; i<=n; i++)
    j=1
    while(j*j <= i)
        dp[i] = min(dp[i], dp[i-j²]+1)
        j=j+1
return dp[n]
```