Today's content

1) Dynamic Programming Intro
2) When to use DP
3) Steps for DP
4) # N stairs
5) Sqrt()
6) max^m Subseq sum without adj elements (idea)

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

	0	1	2	3	4	5	6	7	8
Fib:	0	1	1	2	3	5	8	13	21

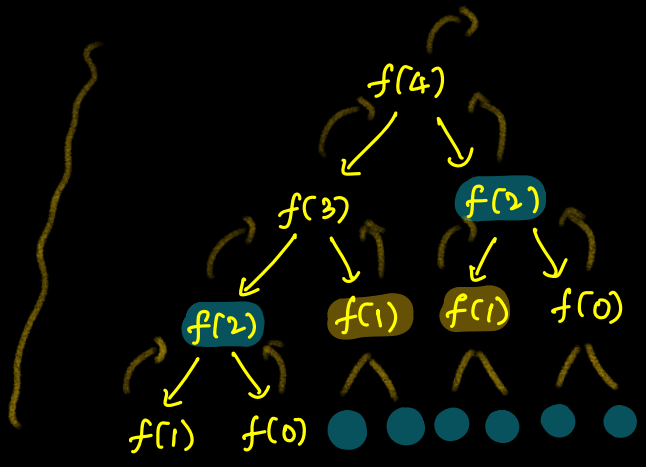
$fib(3) = fib(2) + fib(1)$

$fib(8) = fib(7) + fib(6)$

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

```
int fibo(int N)
{
    if(N <= 1)
        return N;

    return fibo(N-1) + fibo(N-2);
}
```



Dynamic Programming

1. Solve a subproblem only once, and re-use it whenever required

$N = 4$, 15 fun^n calls $= 2^4 - 1$

$N = 3$, 7 fun^n calls $= 2^3 - 1$

N, $2^N - 1$ fun^n calls.

$\Rightarrow$ TC: $O(2^N)$

(Why? Same fun^n calls called again & again).

To calculate 5^{th} fibo.

$dp[6]$:

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

N^{th} fibo $\Rightarrow dp[N+1]$.

dp[6] :

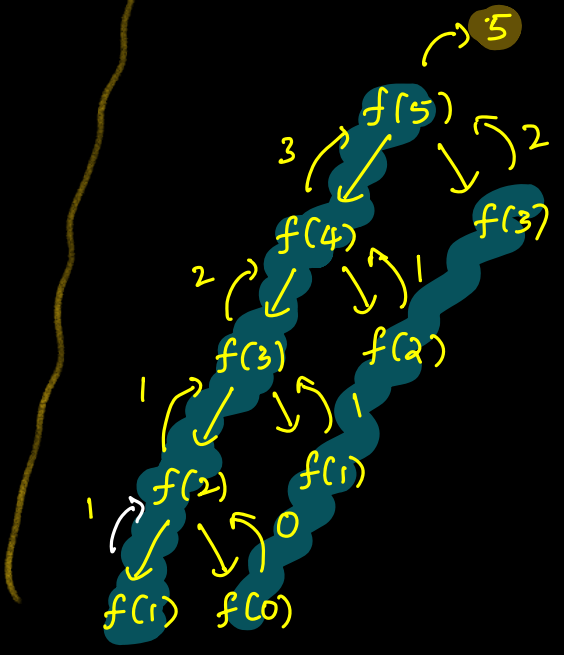
0	1	1	2	3	5
0	1	2	3	4	5

```
int dp[N+1] = {-1} // everything is "-1".

int fibo(int N)
    if(N <= 1)
        dp[N] = N
        return dp[N]

    if(dp[N] == -1)  // not calculated.
        dp[N] = fibo(N-1) + fibo(N-2)

    return dp[N]
```



TC : O(n).

DP.

1. Top-down (Memoization) [Recursive code]
2. Bottom-up. [Tabulation]. [Iterative code].

} TC is same.
SC is same (except for recursion).

```
int dp[N+1] = {-1} // everything is "-1".

int fibo(int N)

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

    for(i=2; i<=N; i++)
        dp[i] = dp[i-1] + dp[i-2]

    return dp[N]
```

```
a, b, c

a = 0, b = 1

for(          )
    c = a+b
    a = b
    b = c

return c
```

When to use DP?

(i) Can the problem be solved using subproblems (Recursion)

(ii) Overlapping should be present in subproblems

Steps

1. dp state (dp[i], dp[i][j] ..) : What are you storing in dp (name of recursive fun^n).

   dp[i] = i^th fibo number.

2. dp expression : Solve dp state using subproblems. (recurrence rel^n)

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

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

3. dp table : Where we store all the o/p.

   Size of dp table, ex: dp[N+1].

4. TC / SC :

   TC: [No. of dp states] * [TC for each dp state]

   [Size of dp table] * (TC of dp exp^n).

   SC: (DP table size) + (recursion)

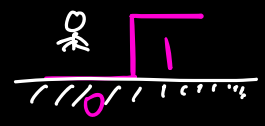
N Stairs.

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:

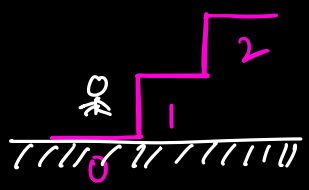
N=1



ways: 1

0 → 1

N=2

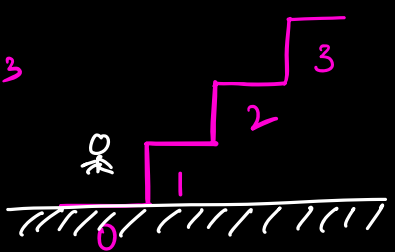


ways: 2

0 → 1 → 2

0 → 2

N=3



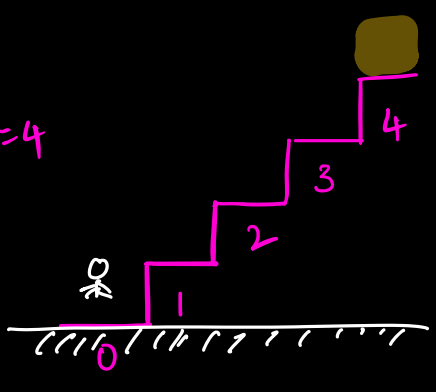
ways: 3.

0 → 1 → 2 → 3

0 → 1 → 3

0 → 2 → 3

N=4



ways : 5.

0 → 1 → 2 → 3 → 4

0 → 1 → 2 → 4

0 → 1 → 3 → 4

0 → 2 → 3 → 4

0 → 2 → 4

From what steps I can reach 4^{th} step?

(i) From 3^{rd} : 3

What are the ways to reach 3^{rd} step

0→1→2→3→4
0→1→3→4
0→2→3→4.

(ii) From 2^{nd} : 2

What are the ways to reach 2^{nd} step

0→1→2→4
0→2→4

OR

0→1→2→3→4
0→2→3→4

4^{th} step = 3^{rd} step + 2^{nd} step.

= 3 + 2 = 5.

Ways to reach 5^{th} step ⟹ 4^{th} step + 3^{rd} step.

5 + 3 = 8. ⟹

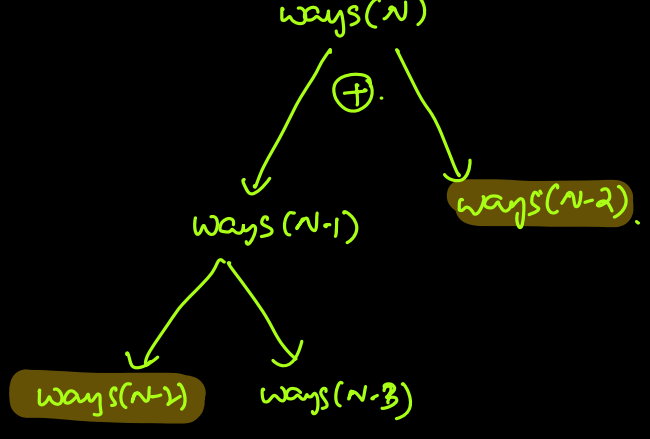
0→1→2→3→4→5
0→1→2→4→5
0→1→3→4→5
0→2→3→4→5
0→2→4→5

+

0→1→2→3→5
0→1→3→5
0→2→3→5

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

↳ Same as previous q^n.



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

N = 6

$1^2+1^2+1^2+1^2+1^2+1^2 = 6$ (6 no's are used)

$2^2+1^2+1^2 = 6$ (3 no's are used)

} ans = 3

N = 10

$1^2+1^2+1^2+$ ___ $+1^2$ (10 no's)

$2^2+2^2+1^2+1^2$ (4 no's)

3^2+1 (2 no's)

} ans = 2

N = 9

$3^2 = 9$ => ans = 1

N = 12.

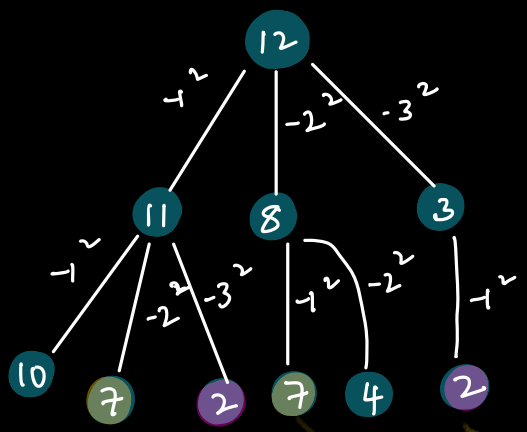
$3^2+1^2+1^2+1^2$ (4 no's are used)

$2^2+2^2+2^2$ (3 no's are used)

} ans = 3

Greedy fails.

Try all possibilities



$f(12) = 1 + \min \begin{cases} f(11) & (i=1) \\ f(8) & (i=2) \\ f(3) & (i=3) \\ f(4) & (i=4)\ X \end{cases}$

+

overlapping

DP can be used.

$$mps(n) = 1 + \min_{i=1,\ i^2 \le n} [mps(n-i^2)]$$

Recurrence rel^n.

Steps

1. dp state (dp[i], dp[i][j] ..) : What are you storing in dp

   dp[i] = min. no. of perf squares required to get 'i'.

2. dp expression : Solve dp state using subproblems.
   (recurrence rel^n)

$$dp[n] = 1 + \min_{i^2 \le n,\ i=1} \forall \left(dp[n-i^2] \right)$$

3. dp table : Where we store all the o/p. (final ans).

   Size of dp table; dp[n+1] (∵ final ans @ dp[n])

4. TC / SC : TC: [No. of dp states] * [TC for each dp state]

   [Size of dp table] * (TC of dp exp^n).

   [N+1] * $\sqrt{N}$ ⇒ TC: $N\sqrt{N}$

   SC: (DP table size) + (recursion)

   N+1 ⇒ SC: O(N).

```
int dp[N+1] = {-1}  // cannot be the ans.

int mps(int N)
    if (N==0)
        return 0

    if (dp[N] == -1)  // not calculated
        minValue = INT_MAX, i=1
        while (i*i <= n)
            minValue = min (mps(N-i^2), minValue)

        dp[n] = 1+minValue
```

TOP-DOWN.

```
    return dp[n]
```

Bottom-up.

```
int mps(int N)
    int dp[N+1] = {-1}
    if (N==0)
        return 0
    for (i=1 ; i<=n ; i++)
        minValue = INT_MAX, x=1
        while (x*x<=i)
            minValue = min (minValue, dp(i-x^2))
        dp[i] = minValue+1
    return dp[n]
```

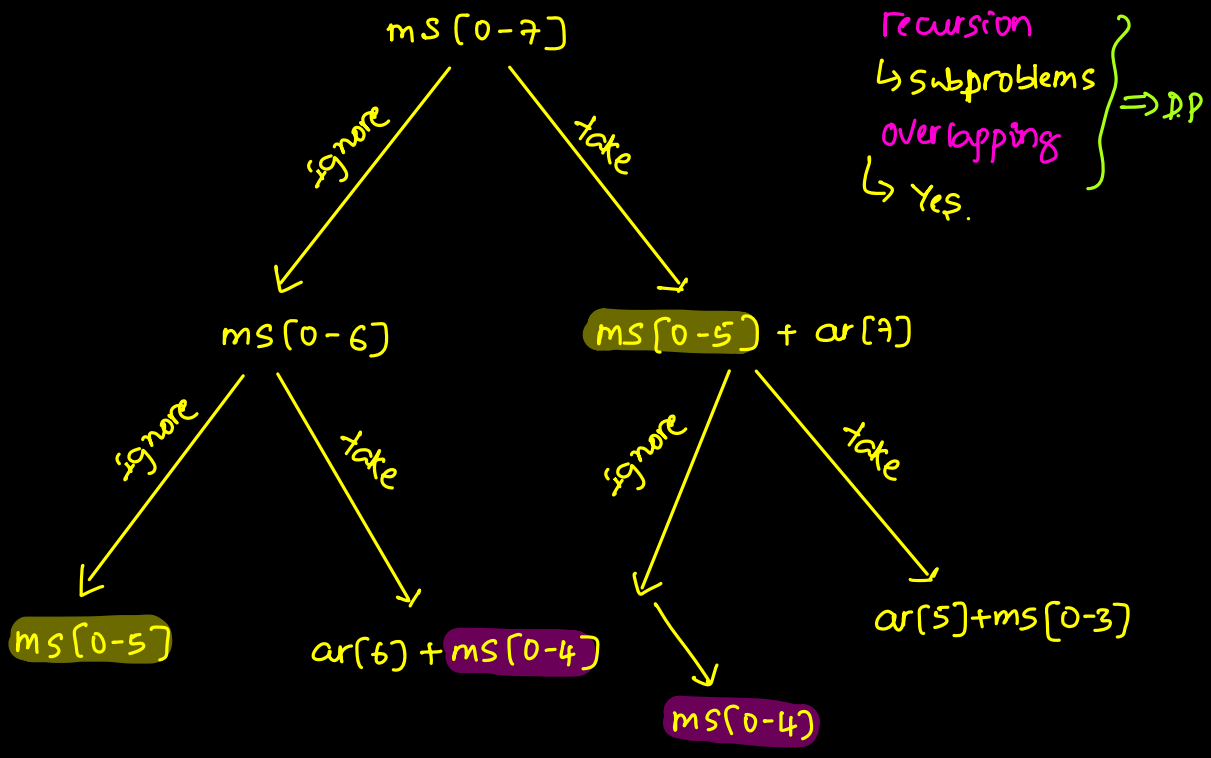
Bottom-up.

Q. Given ar[N] elements, calculate max subseq sum without adjacent elements. Empty subsequence is also a valid subsequence.

		ans
ar[3] :	9 14 3	; 14
ar[4] :	9 4 13 24	; 33
ar[3] :	13 14 2	; 15
ar[4] :	-4 -3 -2 -3	; 0

	0	1	2	3	4	5	6	7
ar[8] : [	2	-1	-4	5	3	-1	4	2]

max subseq sum from [0-7].



DP steps

dp state: what is dp[i]?
Max subseq sum from [0..i] } ms[0-i]

dp exp^n: dp[i] in terms of [0, 1, 2 - i-1]

ms[0-i] = max(ms[0-(i-1)], ms[0-(i-2)] + ar[i])

dp[i] = max(dp[i-1], dp[i-2] + ar[i])

dp table:

dp[0] => Max subseq sum from [0..0]

dp[2] => Max subseq sum from [0..2]

dp[N] => Max subseq sum from [0..N]

not part of ar[]

dp[N-1] => Max subseq sum from [0..[N-1]]

Size of dp[] => N.

TC:

(# of dp states) * (TC for each state) : TC: N*1 => O(N).

N 1

SC

(dp table size) + (recursion stack) : SC: O(N+N) : O(N).

N N

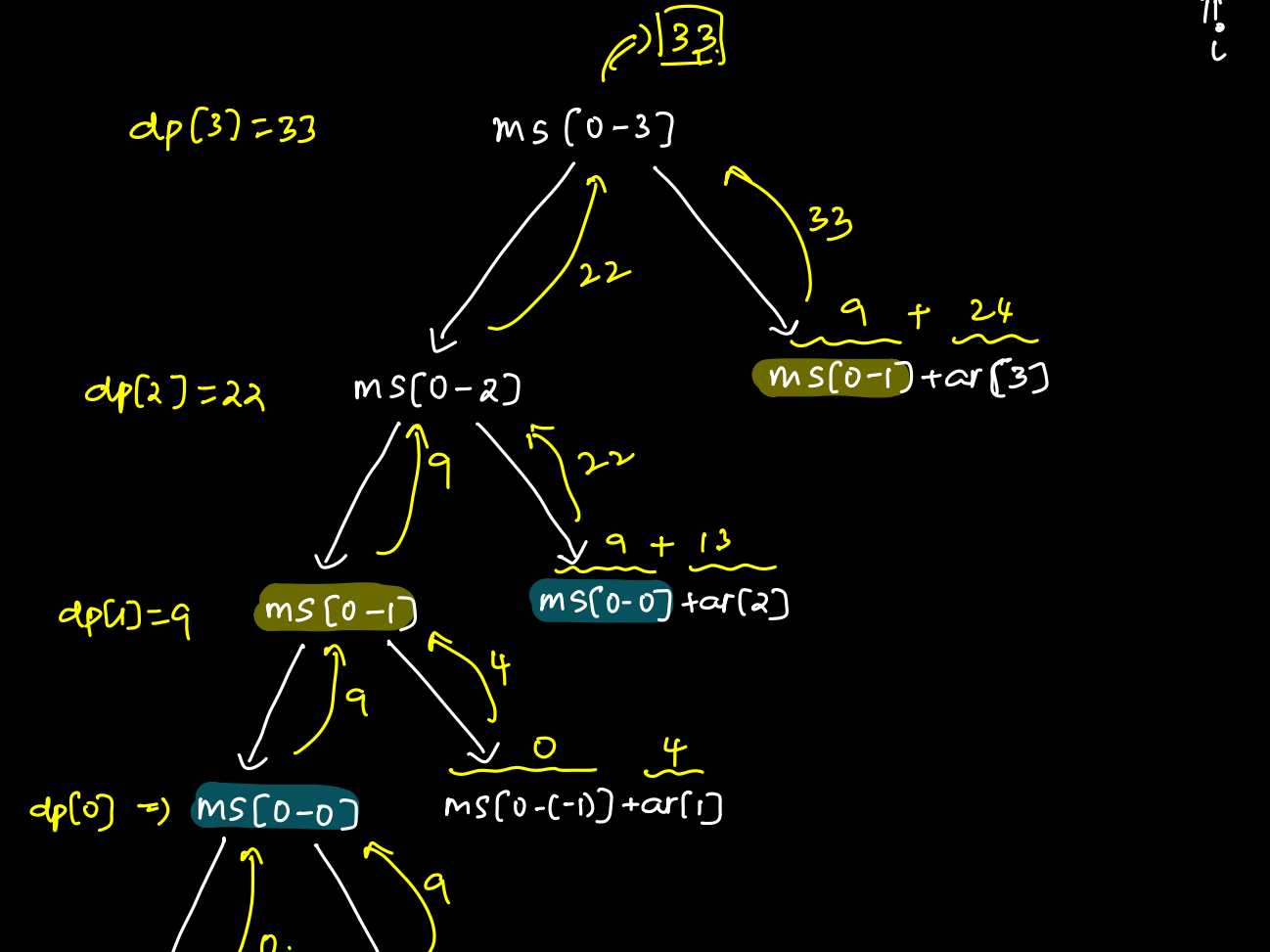
```
int[] dp = new dp[N] // initialize each dp[i] with "-1".

int maxSubSeqSum(int ar, int N, int i)
    if(i<0)
      return 0.
    if(dp[i] == -1) //not calculated
        dp[i] = max( maxSubSeqSum(ar, N, i-1),
                     maxSubSeqSum(ar, N, i-2) + ar[i] )
    return dp[i]
```

DRY-RUN

dp: [9 9 22 33]
0 1 2 3

ar[4] : 9 4 13 24
0 1 2 3
i



TO-DO: Try space optimisation

Remember!

Most important thing to learn something new

PATIENCE