Madhan Kumar M S
Abhishek Sharma
amit khandelwal
Ashish
Balaji S K
Bhaveshkumar
Burhan
Divya P
Gagan Kumar S
Gowtham
Hemant Kumar
Karan Dobriyal
KULDEEP PATIDAR
Nikhil Pandey
PREETHAM
Purusharth A
Rajat Sharma
Rajendra
Sanket Giri
Saurabh Ruikar
Shani Jaiswal
sharath r
shilpa mamillapalli
Shradha Srivastava
Shreya Gupta
Sneha L
Sridhar Hissaria
Subhashini
Sumit Adwani
Suyash Gupta
Vasanth
Vimal Kumar

DP 1 — Dynamic Programming

AGENDA:

→ Introduction
→ Fibonacci
→ Stairs
→ Min no. of perfect squares

Contest 5 →

Trees, Heaps, Greedy

26th April

Fibonacci Series

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

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

$$fib(0) = 0$$

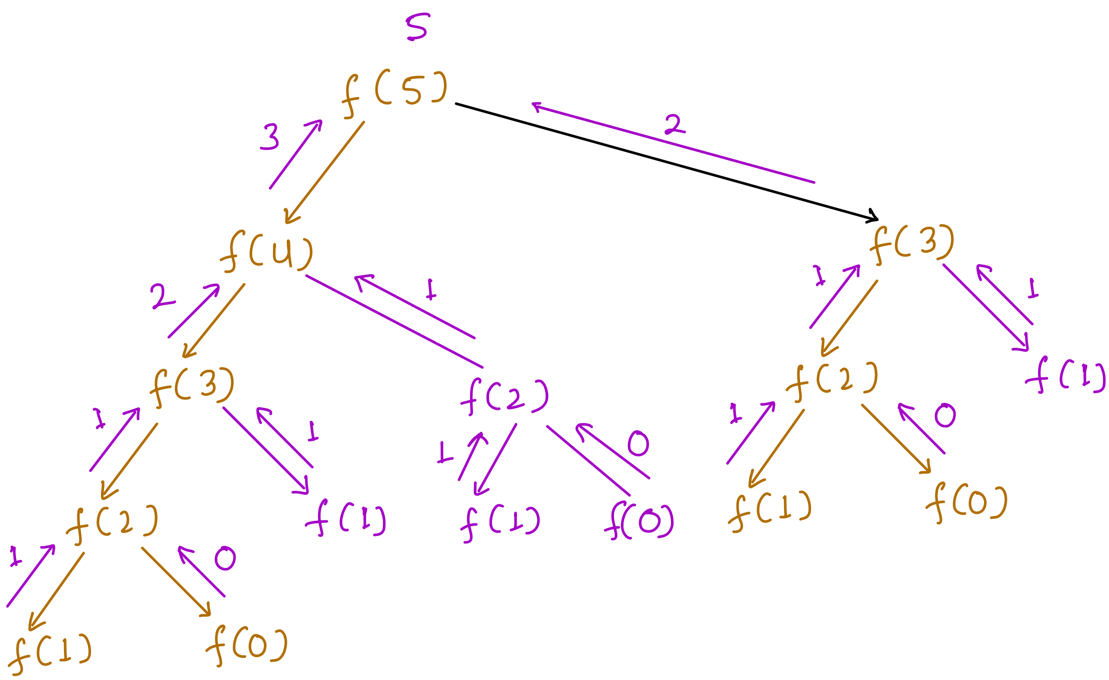
$$fib(1) = 1$$

```
int fib(n) {
    if (n <= 1) return n

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

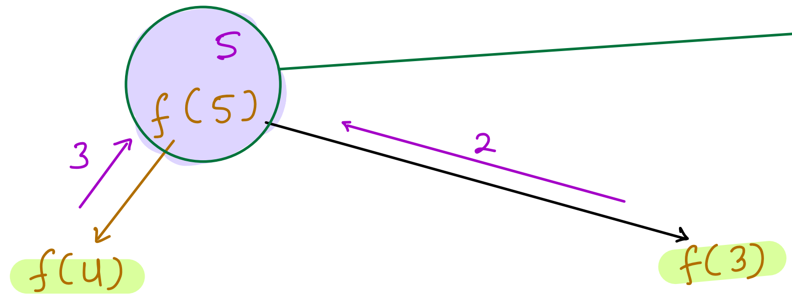
TC: $O(2^N)$

SC: $O(N)$



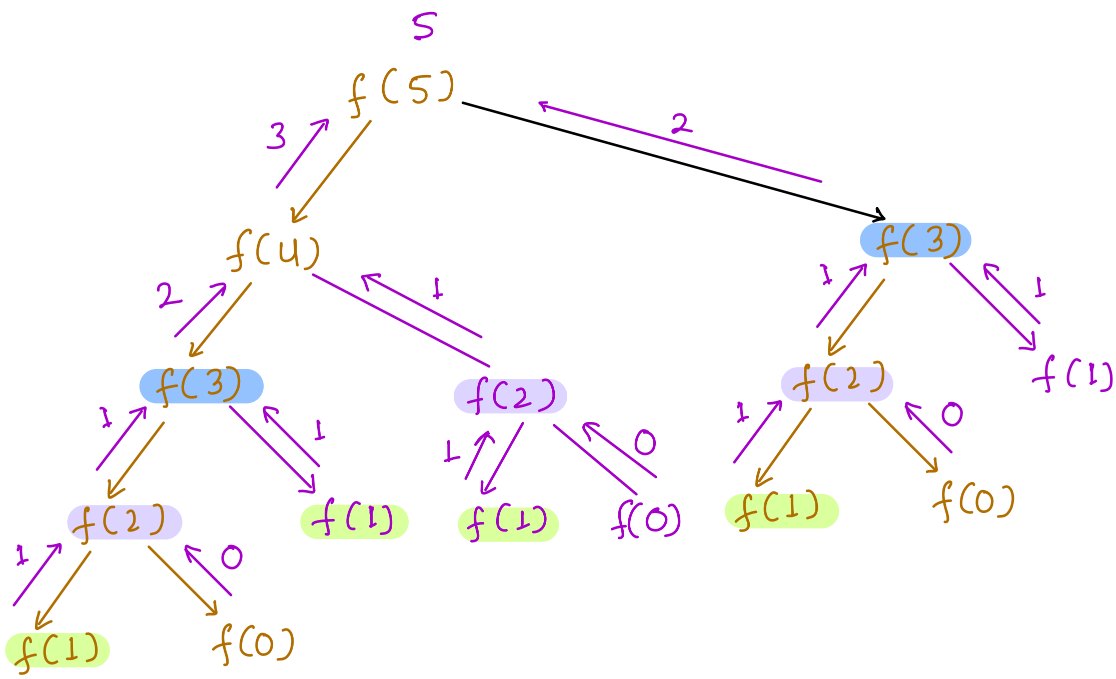
Properties

① → Optimal Substructure — Solve bigger problems using smaller sub problems.



AND

② → Overlapping sub problem — Repeated sub-prob...



To eliminate repeated subproblem

— store the result and reuse

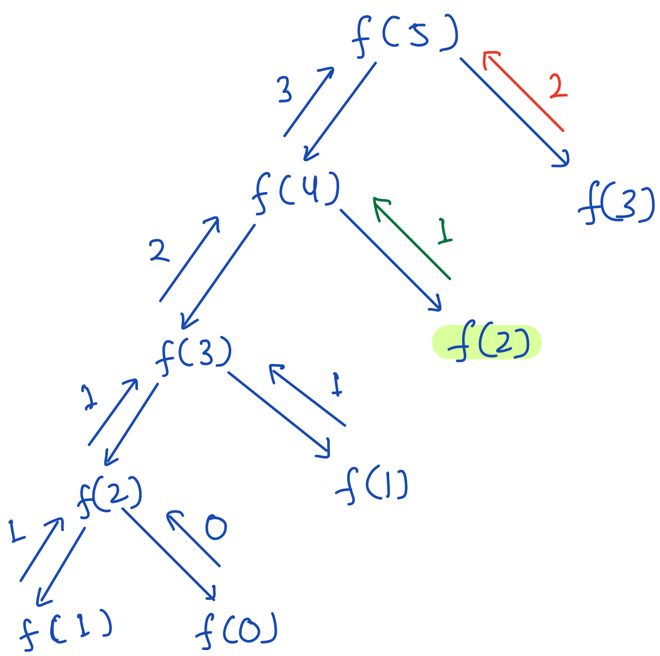
Hashmap   Array

dp[n+1]   // ∀ i dp[i] = −1 → memoization

```
int fib(n) {
    if (n <= 1) return n

    if (dp[n] != -1) { return dp[n] }   // reuse

    ans = fib(n-1) + fib(n-2)
    dp[n] = ans   // store
    return ans
}
```



		1	2	3	5
−1	−1	~~−1~~	~~−1~~	~~−1~~	~~−1~~
0	1	2	3	4	5

TC : O(N)

SC : O(N)

TC for any dp problem ⟶ # of unique possible inputs
* TC per input

Types of DP

① Top Down approach — recursive approach {memoization}

Bigger problem ⟶ smaller problem

② Bottom up approach {tabulation} — iterative approach

smaller problem ⟶ Bigger problem.

Top Down approach	Bottom up approach
Easy to implement	No recursive stack space ∴ There is a possibility of space optimization

Iterative approach

```
// dp[n+1]          → tabulation
dp[0] = 0
dp[1] = 1

for i → 2 to N {
    dp[i] = dp[i-1] + dp[i-2]
}

print (dp[n])
```

```
if( n <= 1) return n

a = 0
b = 1
c = -1
for i ⟶ 2 to N {
        c = a + b
        a = b
        b = c
}

print(c)
```

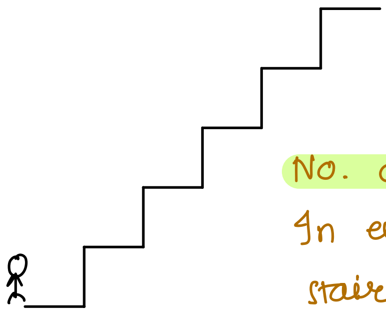
a b c

a b c

TC : $O(N)$

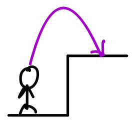
SC : $O(1)$

Stairs

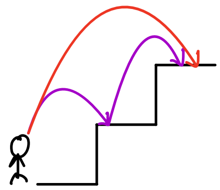


No. of ways to reach the N^{th} stair ?
In every step you can climb either 1 or 2 stairs

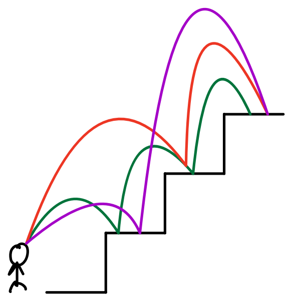
ans = 1



N = 1



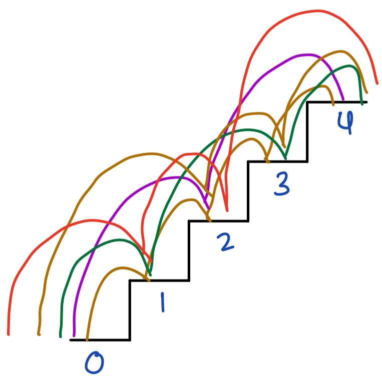
N = 2
ans = 2



N = 3

① ⟶ 1 1 1
② ⟶ 2 1
③ ⟶ 1 2

ans = 3

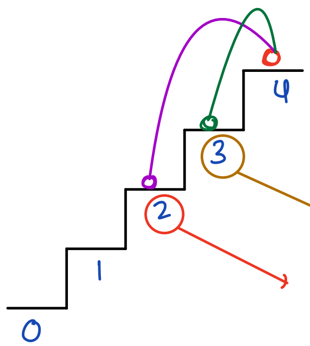


ans = 5

① ⟶ 1 1 1 1
② ⟶ 2 2
③ ⟶ 1 2 1
④ ⟶ 2 1 1
⑤ ⟶ 1 1 2

4th stair

From which stairs it can be reached in 1 step



x ways to reach 3rd stair

y no. of ways to reach 2nd stair

ways to reach 4th stair $\longrightarrow$ $x+y$

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

$$\text{ways}(1) = 1$$
$$\text{ways}(0) = 1$$

Break 22:35

$1^2 \quad 2^2 \quad 3^2$ ....

Find the minimum no. of perfect squares required to get sum = N. $N >= 1$

Note → Duplicate Squares are allowed

$N = 6$ → $1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 = \# 6$

$1^2 + 1^2 + 2^2 = \# 3$

ans = 3

$N = 10$ → $\left.1^2 + 1^2 \ldots\ldots 1^2\right\}$ 10 times # 10

$\left.2^2 + 2^2 + 1^2 + 1^2\right\}$ # 4

$\left.3^2 + 1^2\right\}$ # 2

ans = 2

$N = 9$ → $\left.1^2 + 1^2 + \ldots\ldots 1^2\right\}$ 9 times # 9

$\left.3^2\right\}$ # 1

$\left.2^2 + 2^2 + 1^2\right\}$ # 3

ans = 1

$N = 5$ → $1^2 + 1^2 + 1^2 + 1^2 + 1^2$ # 5

$1^2 + 2^2$ # 2

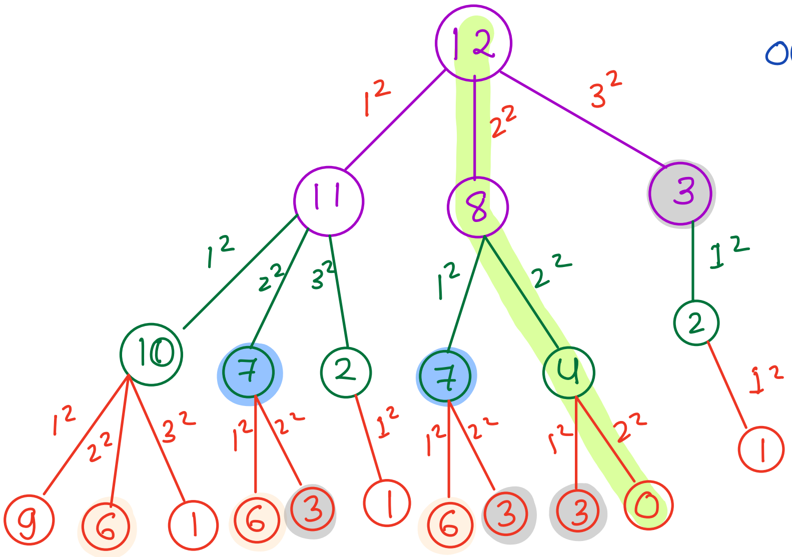
ans = 2

Idea 1 → Subtract the largest possible perfect square

$N = 12$ $3^2 + 1^2 + 1^2 + 1^2 = \# 4$

$2^2 + 2^2 + 2^2 = \# 3$

ans = 3



Overlapping subproblem

$f(0) = 0$

add nothing

Optimal substructure

$$f(12) = \left.\begin{matrix} 1 + f(8) \\ 1 + f(11) \\ 1 + f(3) \end{matrix}\right\} \min$$

$$f(N) = 1 + \min\left(f(N-1^2),\ f(N-2^2),\ f(N-3^2) \ldots\ldots\right)$$

$$\boxed{f(N) = 1 + \min\left(f(N-i^2)\right) \quad \forall i \quad i*i <= N}$$

perfect sq.

Pseudocode

```
dp[N+1]      ∀ i  dp[i] = -1

int  minSqCnt ( int N) {
     // Base cond"
     if ( N == 0 )  return 0

     if ( dp[N] != -1 )  return dp[N]      // reuse

     cnt = ∞
     for ( i = 1 ; i*i <= N ; i++ ) {
          cnt = min ( cnt , 1 + minSqCnt ( N - i² ) )
     }
     dp[N] = cnt      // store
     return cnt
}
```

TC: $O(N\sqrt{N})$

SC: $O(N)$

Bottom up

```
dp[A+1]    //  ∀i dp[i] = -1  or ∞
dp[0] = 0
                                  ⟶ smaller subproblem to
for n ⟶ 1 to A {                       larger subproblem
    cnt = ∞
    for (i = 1 ; i*i <= n ; i++) {
        cnt = min ( cnt , 1+ dp[n - i²] )
    }
    dp[n] = cnt
}

return dp[A]
```

TC : $O(N\sqrt{N})$

SC : $O(N)$

N = 6

dp =	0	1	2	3	~~4~~ 1	2	3
	0	1	2	3	4	5	6

Arc labels: 1^2, 1^2, 1^2, 2^2, 1^2, 1^2

n	i
1	[1]

Minimum notes for sum of money

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.

N = 100 ans = 2

N = 55 ans = 2

N = 65 $\underbrace{50+5+5+5}_{4 \text{ notes}} = 65$

$\underbrace{30+30+5}_{3 \text{ notes}} = 65$

Pseudocode

```
dp[N+1]      ∀ i  dp[i] = -1
notes = [5, 30, 50]
int minSqCnt ( int N) {
    // Base cond"
    if ( N==0) return 0

    if ( dp[N] != -1 ) return dp[N]     // reuse

    cnt = ∞
    for (i = 0 ; i < 3 ; i++) {
```

```
        if ( N >= notes[i] )
            cnt = min ( cnt , 1 + minSqCnt ( N - notes[i] ))
    }
    dp[N] = cnt      // store
    return cnt
}
```

TC : O(N)

SC : O(N)