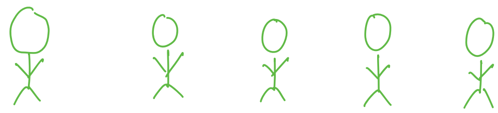
Lecture : Dynamic programming - 1

Agenda

- Introduction
- fibonacci
- Stairs
- Min no. of squares

Analogy

Students: 

Marks: 5 3 4 7 9

Total marks = 28

New student: 

5

Total marks

- 5 + 3 + 4 + 7 + 9 + 5 = 33
- $\underbrace{\text{prev\_total\_marks}}_{28} + 5 = 33$

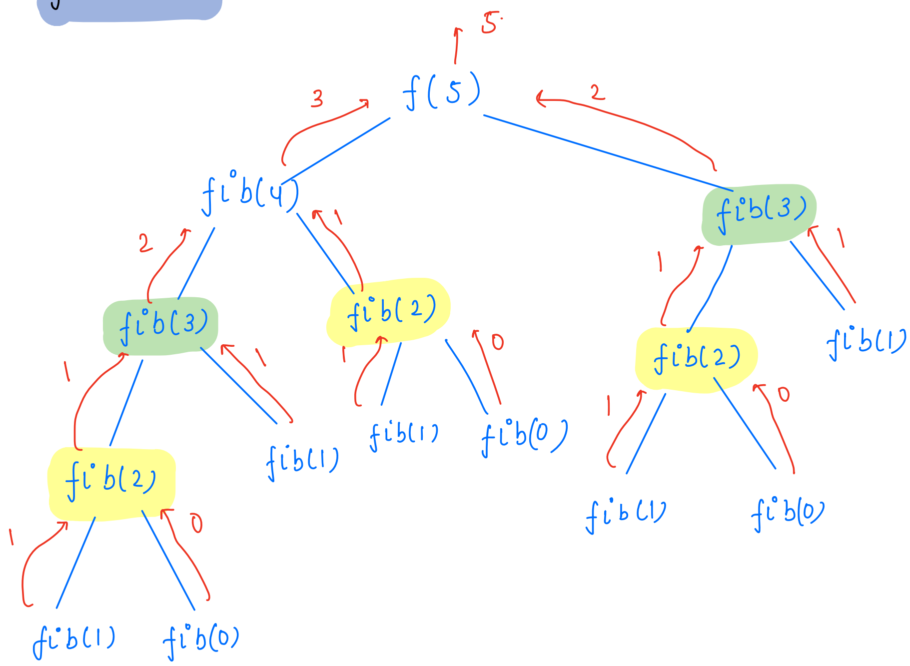
DP: Use the previous results to calculate current results.

Example: Prefix array

$pf[i] = pf[i-1] + arr[i]$

fibonacci

0	1	2	3	4	5	6	7	8	9	10	
0	1	1	2	3	5	8	13	21	34	55	----




```
int fib(n) {
    if(n==0||n==1) {
        return n;
    }
    fir = fib(n-1)
    sec = fib(n-2);
    return fir + sec;
}
```

TC: $O(2^n)$

SC: $O(n)$

TC: $2^n \longrightarrow$ $n=10$ $\quad 2^{10} = 1024$ unit

$n=20$ $\quad 2^{20} = 10^6$ unit

$n=50$ --- TLE. $> 10^8$ unit

Conditions for DP

- Overlapping subproblems ✓
- Optimal substructure [Later in DP]

Dynamic programming

Calculate the unique results only once.

DP code for fibonacci : Memoised

```
int fib(n, dp[]) {
    if(n==0 || n==1) {
        dp[n] = n;
        return n;
    }
    if( dp[n] != -1) {
        return dp[n];
    }
    int fir = fib(n-1);
    int sec = fib(n-2);
    dp[n] = fir + sec;
    return fir + sec;   (or)   return dp[n];
}
```

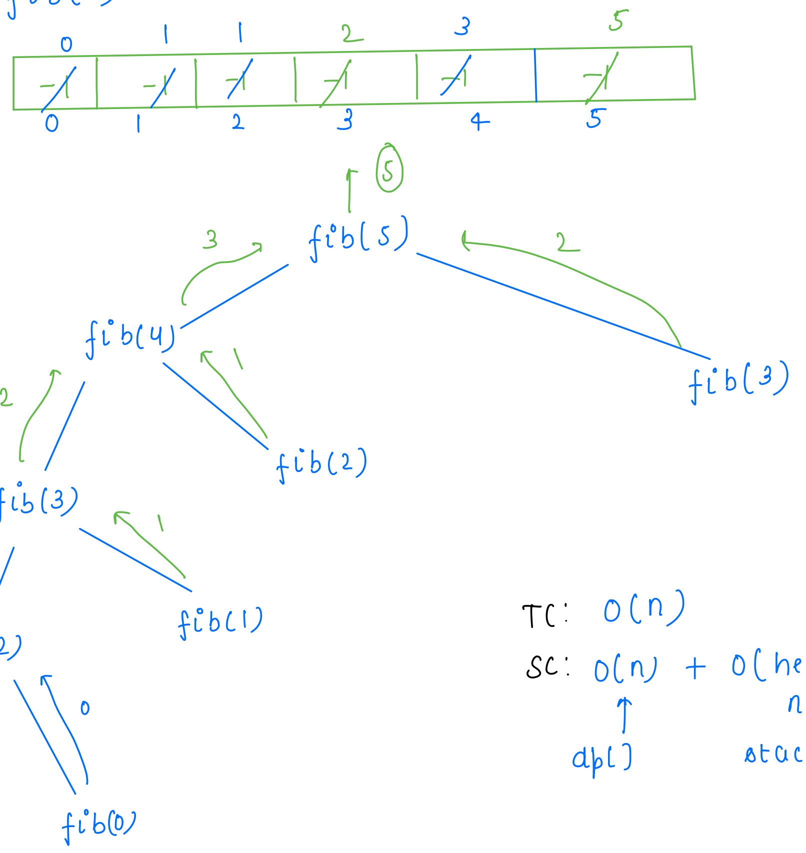
Dry run:

fib(5)

dp[6]

↑

n+1



TC: $O(n)$

SC: $O(n) + O(\text{height})$

$O(n)$ ← dp[]

height = n ← stack size.

DP

Memoisation
[Recursive]
Top down

Tabulation
[Iterative]
Bottom up

Tabulative approach of DP fibonacci:

fib(5): dp[6] =

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

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

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

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

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

return dp[5];

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

TC: O(n)

SC: O(n)
dp[]

Qu: Solve it using O(1) space

0	1	2	3	4	5	6	7	8	9	10	
0	1	1	2	3	5	8	13	21	34	55	----
a	b	c									
	a	b									

fib(5)

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

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

a	b	c	
0	1		
1	1	1	i=2
1	2	2	i=3
2	3	3	i=4
3	5	5	i=5

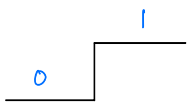
Ans

TC: O(n)

SC: O(1)

<u>Qu</u> Given n stairs, how many ways can we go from 0th step ⟶ nth step. [can take only 1 | 2 steps]

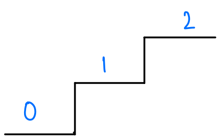
$n = 1$



{ 1 }

1 way

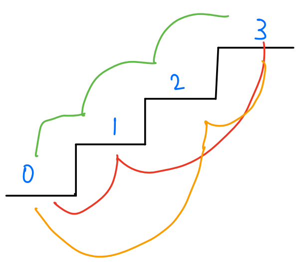
$n = 2$



{ 1 1
2 }

2 ways.

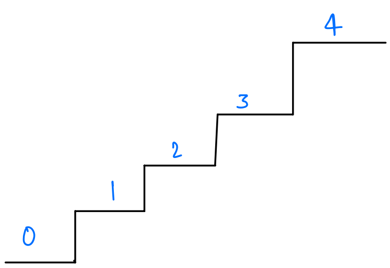
$n = 3$



{ 1 1 1
1 2
2 1 }

3 ways

$n = 4$



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

5 ways

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

Idea foundation.

$n = 4$ $\quad$ ans = ways(3) + ways(2)

↳ $n = 3$

1 1 1 1

1 2 1

2 1 1

$n = 2$

1 1 2

2 2

Dry run:

ways(5)

ways(4) $\quad$ ways(3)

Same as fibonacci

Recursive code

```
int ways(n) {
    if(n==0||n==1){
        return n;
    }
    int w = ways(n-1);
    int w2 = ways(n-2);
    return w1 + w2;
}
```

Memoised code

Same as fibonacci

Tabulation code

same as fibonacci

Break: 8:14 – 8:27 AM

Qu find min count of perfect squares to add to get sum = n.

		count
n = 2	$1^2 + 1^2$	2
n = 3	$1^2 + 1^2 + 1^2$	3
n = 4	2^2	1
n = 5	$2^2 + 1^2$	2
n = 6	$2^2 + 1^2 + 1^2$	3
n = 7	$2^2 + 1^2 + 1^2 + 1^2$	4
n = 50	$7^2 + 1^2$ (or) $5^2 + 5^2$	2

Greedy idea: Subtract greatest perfect square $<= n$ from n.

$n = 50 \Rightarrow 50 - 7^2 = 1 - 1^2 = 0$ 2.

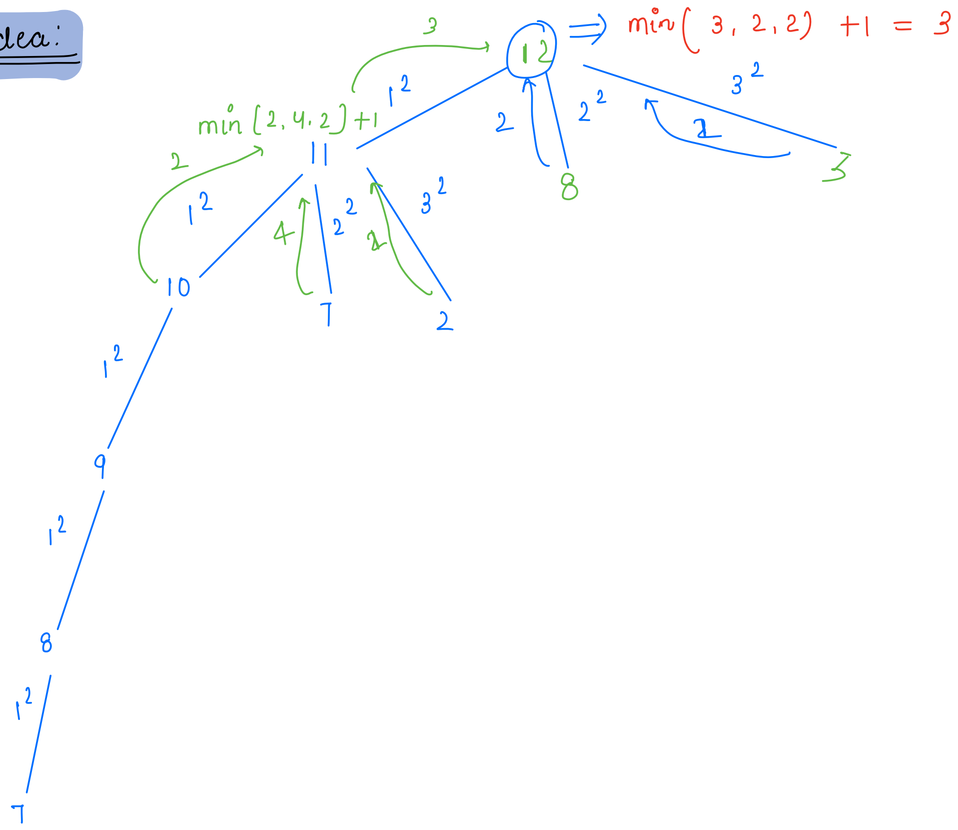
$n = 70 \Rightarrow 70 - 8^2 = 6 - 2^2 = 2 - 1^2 = 1 - 1^2 = 0$ 4

$n = 12 \Rightarrow 12 - 3^2 = 3 - 1^2 = 2 - 1^2 = 1 - 1^2 = 0$ 4

$12 = 2^2 + 2^2 + 2^2$ 3

Greedy idea wont work.

Idea:



Code:

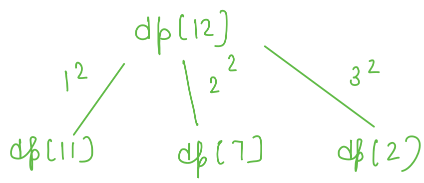
Brute force:

```
int count(n) {
    if(n==0 || n==1){
        return n;
    }
    if(n<0){
        return 0;
    }
    ans = ∞;
    for(i=1; i*i<=n; i++){
        ans = min(ans, count(n-i²));
    }
    return ans+1;
}
```

DP code:

n = 12

0	1	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
0	1	2	3	4	5	6	7	8	9	10	11	12




```
int count(n){
    dp[] = new int[n+1];
    dp[0] = 0;
    dp[1] = 1
    for(i=2; i<=n; i++){
        for(x=1; x*x<=i; x++){
            dp[i] = min(dp[i], dp[i-x*x]) +1;
        }
    }
    return dp[n];
}
```

TC: $O(n * \sqrt{n})$

SC: $O(n)$

Dry run:

0	1	2	3	4	5	6	7	8	9	10	11	12
0	1	~~∞~~ 2	~~∞~~ 3	~~∞~~ ~~4~~ 1	∞	∞	∞	∞	∞	∞	∞	∞

$x = 1$

$dp[2] =$

$\min(dp[2], dp[2-1^2])$

$\min(dp[2], dp[1])$

$1 + 1 = 2$

$dp = 3$

$x = 1$

$\min(dp[3], dp[2])$

$2 + 1 = 3$

$dp[4]$

$x = 1$

$\min(dp[4], dp[3])$

$3 + 1 = 4$

$x = 2$

$\min(dp[4], dp[0])$

$0 + 1 = 1$

```
int count(n) {
    dp[] = new int [n+1];
    dp[0] = 0;
    dp[1] = 1
    for (i=2; i<=n; i++) {
        for (x=1; x*x<=i; x++) {
            dp[i] = min(dp[i], dp[i-x*x]) +1;
        }
    }
    return dp[n];
}
```

Thanks