DP Lec-1 (Till Fibonacci Memoization)

26 March 2022 9:27

Dynamic Programming

"Those who do not remember their past are condemned to repeat it"

Print N^{th} fibonacci Number

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

① Expectation: fib(N) → N^{th} fib Number

fib(6) → 8

② faith: fib(N-1) → $(N-1)^{th}$ fib No

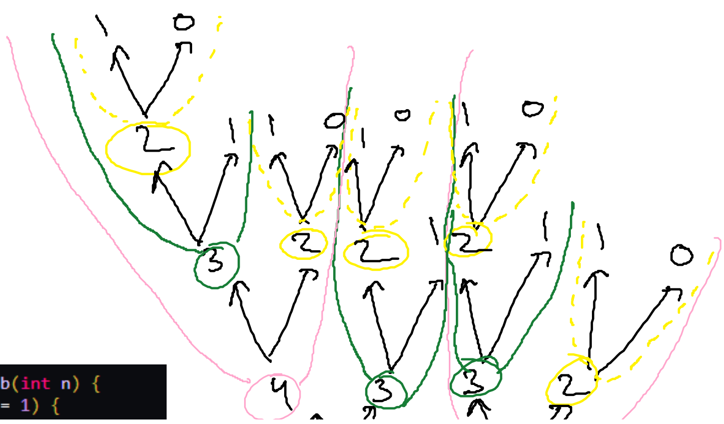
fib(N-2) → $(N-2)^{th}$ fib No

fib(5) → 5

fib(4) → 3

③ Meeting Expectation from faith

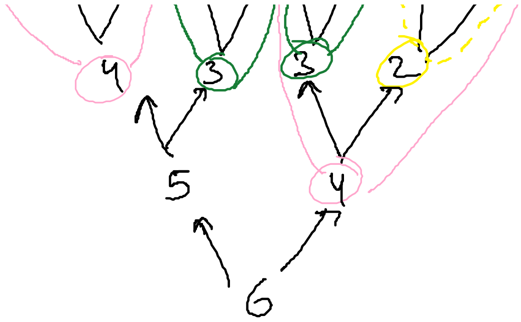
fib(N) = fib(N-1) + fib(N-2)



So many repetitive calls and the entire recursion sub-trees are repeating

```java
public static int fib(int n) {
    if(n == 0 || n == 1) {
```

```java
public static int fib(int n) {
    if(n == 0 || n == 1) {
        return n;
    }

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



are repeating

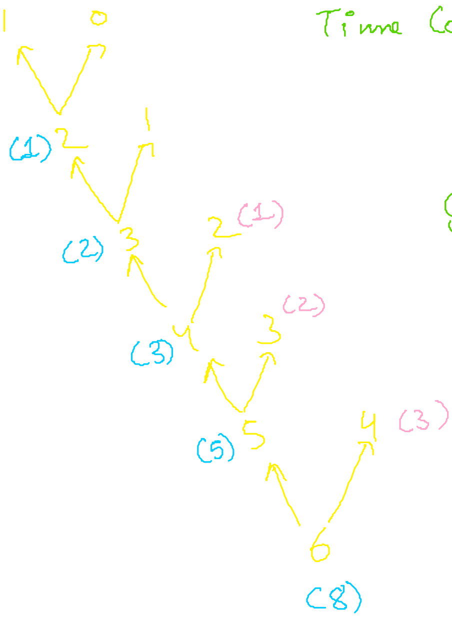
Time Complexity = $(\text{No of calls})^{\text{height}} + (\text{pre} + \text{post}) * \text{height}$

$= (2)^N + O(N) \approx O(2^N)$

Space Complex = O(1)

Memoization

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



Time Complexity = $(1)^N + (K)^N$

$\approx O(N)$

Space Complexity = O(N)

```java
public static int fibMemoized(int n, int[] dp) {
    if(n == 0 || n == 1) {
        return n;
    }

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

    int fibn = fibMemoized(n-1,dp) + fibMemoized(n-2,dp);
    dp[n] = fibn;
    return fibn;
}
```

DP Lec-1

26 March 2022 9:05

Tabulation

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

① Storage and Meaning (N^{th} cell will have N^{th} fib No)

2 Direction of Solving (0^{th} index smallest Problem
5^{th} index largest Problem)

3 Traverse and Solve (Now simply travel and solve)

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

```java
public static int fibTabulation(int n) {
    int[] dp = new int[n+1];
    dp[0] = 0;
    dp[1] = 1;

    for(int i=2;i<dp.length;i++) {
        dp[i] = dp[i-1] + dp[i-2];
    }

    return dp[n];
}
```

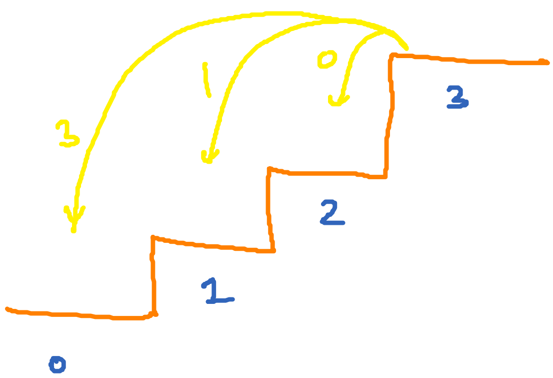
Time Complexity = O(N)

Space Complexity = O(N)

** Java Call Stack can have max of ≈ 20K functions

So, memoization can give Stack Overflow.

Ques 2) Climb Stairs [(Climb Stairs Link)]



3
2 1
1 2
1 1 1 } 4 paths

① Expectation: paths(N) → Total No of paths

↳ paths(3) → 4

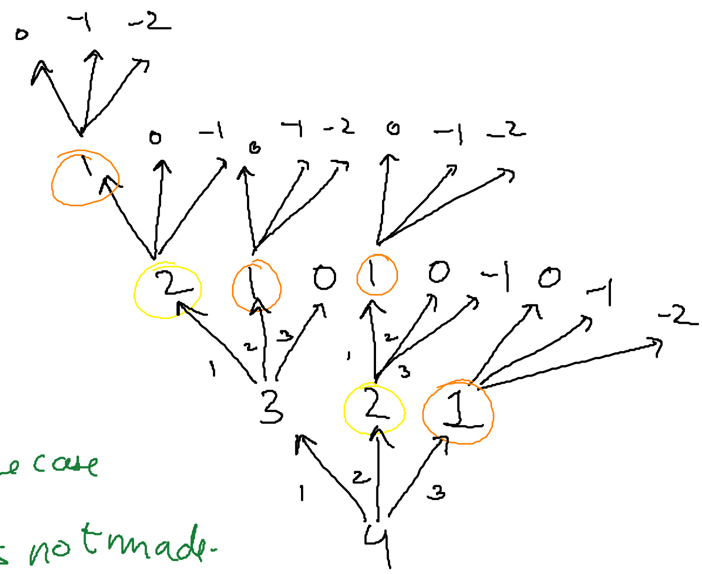
↳ paths(3) → 4 'n'

② faith : paths(N-1) → Total no of paths from N-1
paths(N-2) → Total no of paths from N-2
paths(N-3) → Total no of paths from N-3

③ Meeting Expectation : Add all of them

paths(N-1) + path(N-2) + path(N-3)

```java
public static int paths(int n) {
    //negative base case
    if(n < 0) {
        return 0;
    }

    //positive base case
    if(n == 0) {
        return 1;
    }

    int pathsnm1 = paths(n-1);
    int pathsnm2 = paths(n-2);
    int pathsnm3 = paths(n-3);

    return pathsnm1 + pathsnm2 + pathsnm3;
}
```

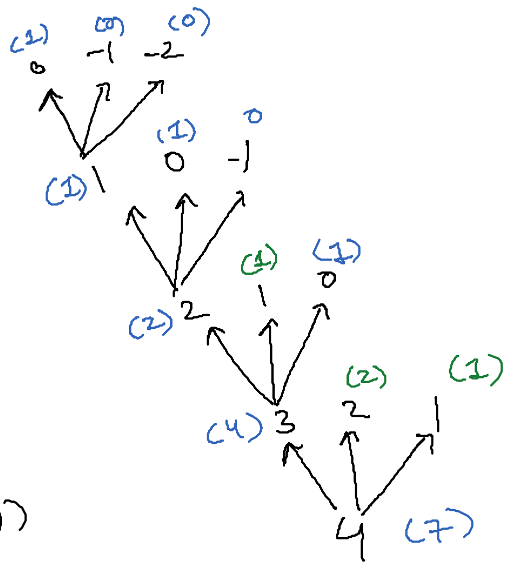


Obviously, we can prune the tree by handling negative base case in such a way that its call is not made.

$TC = O(3^n)$ $SC = O(1)$

Memoization

0	1	2	3	4
1	1 ~~-1~~	2 ~~-1~~	4 ~~-1~~	7 ~~-1~~



$TC = O(N)$

$SC = O(N)$

```java
public static int pathsMem(int n, int[] dp) {

    if(n == 0) return 1;

    if(dp[n] != -1) return dp[n];

    int pathsnm1 = 0;
    int pathsnm2 = 0;
    int pathsnm3 = 0;

    //pruning
    if(n-1 >= 0) {
        pathsnm1 = pathsMem(n-1,dp);
    }

    //pruning
    if(n-2 >= 0) {
        pathsnm2 = pathsMem(n-2,dp);
    }

    //pruning
    if(n-3 >= 0) {
        pathsnm3 = pathsMem(n-3,dp);
    }
```

SC = O(N)

4 (7)

```
    //pruning
    if(n-3 >= 0) {
        pathsnm3 = pathsMem(n-3,dp);
    }

    int pathsn = pathsnm1 + pathsnm2 + pathsnm3;
    dp[n] = pathsn; //memoization

    return pathsn;
}
```

Tabulation

0	1	2	3	4
1	1	2	4	7

for 4 stairs, no of paths = 7.

① Storage and meaning :

At n^{th} cell, no of paths from N^{th} stair should be given

0	1	2	3	4

↳ No of paths from 2^{nd} stair to 0^{th} stair

② Direction of Solving

0	1	2	3	4
1				

↑ smallest Problem { Path from 0^{th} stair to 0^{th} stair }

⟶ (Direction of solving)

③ Traverse & solve

0	1	2	3	4
1	1	2	4	7

for $n \geq 3$ $dp[n] = dp[n-1] + dp[n-2] + dp[n-3]$

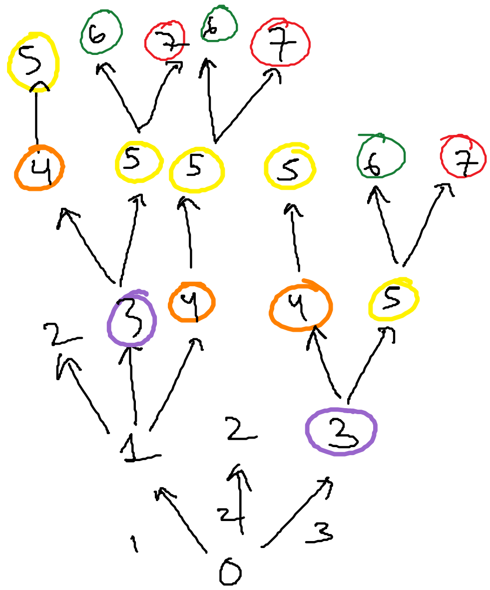
```java
public static int tabulation(int n) {
    int[] dp = new int[n+1];

    dp[0] = 1;


    for(int i=1;i<dp.length;i++) {
        if(i==1) {
            dp[i] = dp[i-1];
        } else if(i==2) {
            dp[i] = dp[i-1] + dp[i-2];
        } else {
            dp[i] = dp[i-1] + dp[i-2] + dp[i-3];
        }
    }

    return dp[n];
}
```

TC = O(N)

SC = O(N)

Ques 3 Climb Stairs with variable jumps [(Climb Stairs With Variable Jumps)]

0	1	2	3	4	5	6	7	8	9
3	3	0	2	1	2	4	2	0	0



Tree is only till 6 stairs