30/03/2024

Day 23 of DSA

Tasks.

Check Box	Dynamic Programming :-
[x]	DP .- Top down Approach
[x]	DP - Bottom-Up Approach
[x]	DP - Space Optimization
[x]	Minimum cost Climbing stairs. 5 Approaches.

Dynamic Programming :-

2 Approaches :-

* Top - Down - Recursion + Memorization

* Bottom - Up - Tabulation.

- Space optimization.

* Fibonacci Series :-

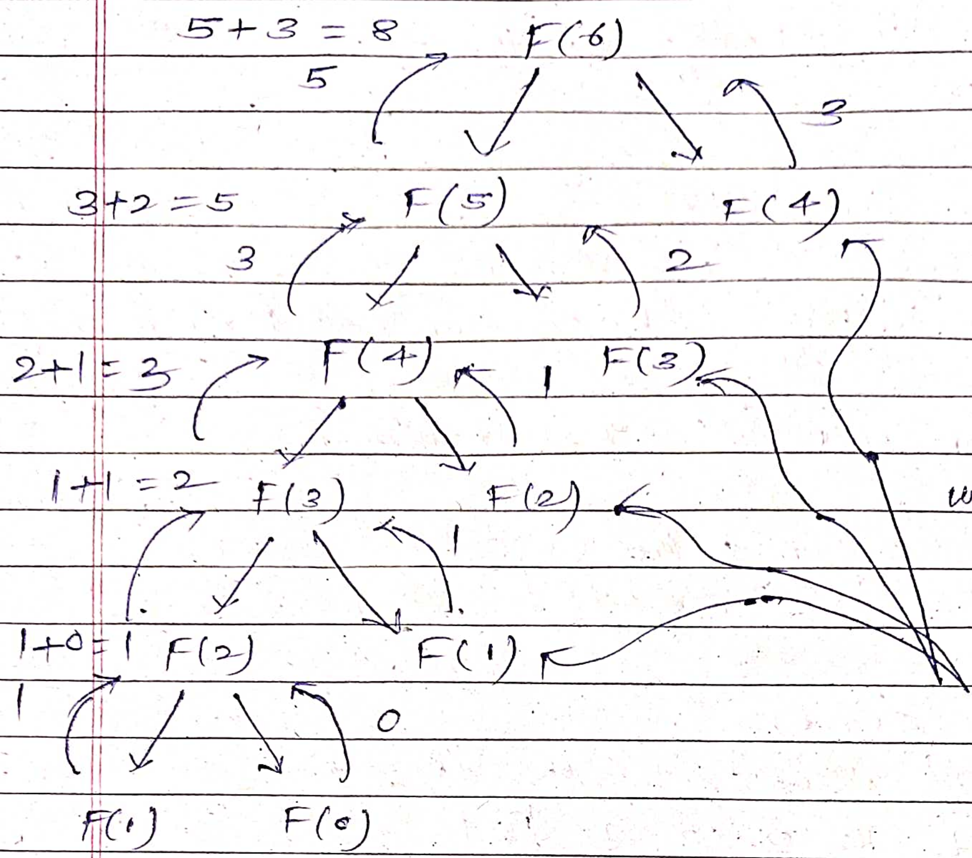
0th	1st	2nd	3rd	4th	5th	6th	7th
0	1	1	2	3	5	$\boxed{8}$	13

$(n-2)^{th}$ $(n-1)^{th}$

$$\boxed{F(n) = F(n-1) + F(n-2)}$$

Normal code :-

```
Funct^n (int n) {
    if (n == 1 || n == 0)
        return n;

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



Overlapping subproblem means we have calculate same thing again & again like this

① Top- Down Approach :

Recursion + Memorization

Store the value of subproblem in map / table

1 D Array (n+1) size

	1	1	2	3	5	
-1	~~-1~~	~~-1~~	~~-1~~	~~-1~~	~~-1~~	-1
0	1	2	3	4	5	6

Pseudocode:-

```
int Fib( int n, vector<int>& dp) {

Base Case        if( n<=1)
                 { return n;
                 }

Checking         if (dp[n] != -1)
in 1DArray       {
is their any         return dp[n];
value present.   }

recursive call - dp[n] = Fib(n-1, dp) + Fib(n-2, dp);
return           return dp[n];
}
```

Time Complexity: O(N)
Space complexity: O(N) + O(N) = O(N)

② Bottom Approach :- Tabulation:

Pseudocode

~~Fib(n)~~ int main()

```
{
    dp[n+1];

    dp[0] = 0;
    dp[1] = 1;

    For( int i = 2; i<=n; i++)
    {
        dp[i] = dp[i-1] + dp[i-2];
    }

    return dp[n];
}
```

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

$\boxed{\begin{array}{l}\text{Time Complexity} = O(N) \\ \text{Space Complexity} = O(N)\end{array}}$

3) Space optimization:-

0 1 1 $\boxed{2}$ 3 5

prev2 prev1 curr

pseudocode:-

0 1 2 → n

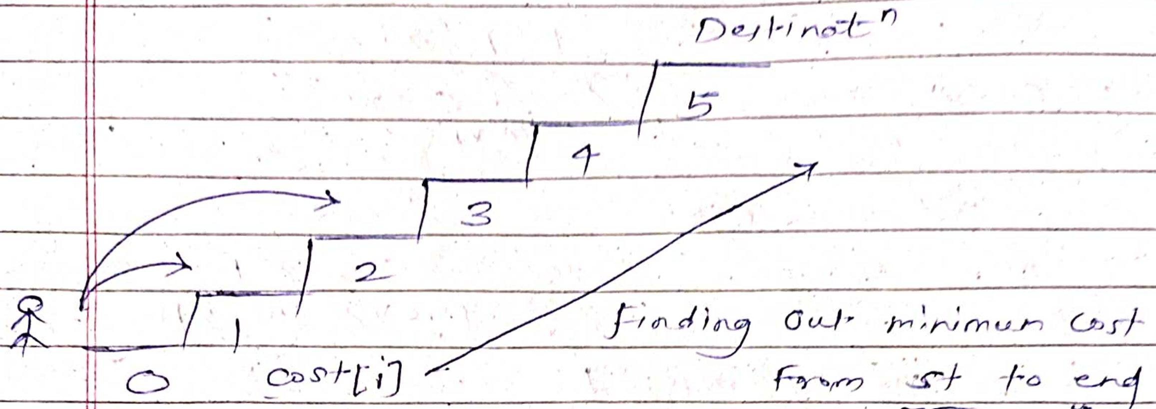
prev2 prev1 curr

```
int main()
{
    int prev1 = 1;
    int prev2 = 0;

    For(int i=2; i<=n; i++)
    {
        int curr = prev1 + prev2;

        prev2 = prev1;
        prev1 = curr;
    }

    cout << prev1 << endl;
}
```

Time Complexity = O(N)

Space complexity = O(1)

* Minimum Cost climbing pairs.



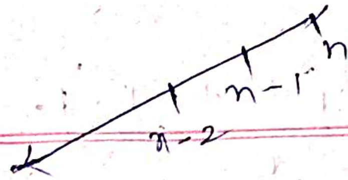
① Recursion Approach :-

$F(n) = (n+1) + (n+2)$

Pseudocode:-

```
int solve (long long nstairs, int i) {

Matlab ki waha pahuch gaya      if( i == nstairs)
                                    return 1;

Matlab ki des. ke aage          if( i > nstairs)
                                    return 0;

Top Tak keliye   return (solve(nstairs, i+1) + solve(nstairs, i+2)) % MOD;
                                                                   ↑
                                                              1000000007
}
```

Limitatⁿ : overlapping sub problem
Time limit Exceeded



(2) Recursion + ~~Memo~~

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

Pseudocode:-

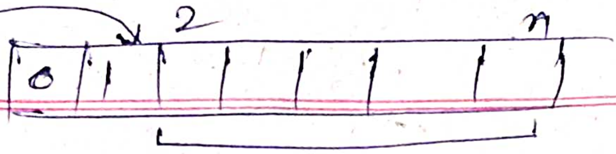
```
int solve (vector<int> & cost, int n)
{
    if( n==0)
    {   return cost[0];
    }
    if( n==1)
        return cost[1];

    int ans = cost[n] + min(solve(cost, n-1),
                            solve (cost, n-2));
    return ans;
}
```

(3) Recursion + Memoization:
Using dp.

$TC = O(n)$
$SC = O(n) + O(n) = O(n)$

pseudocode:-

```
int solve( vector<int> &cost , int n,
           vector<int> &dp) {
  Base Case {
    if( n == 0)
        return cost[0];
    if (n == 1)
        return cost[1];
  }
  DS solving overlapping subproblem {
    if (dp[n] != -1)
        return dp[n];
  }
  Recursion call <- dp [n] = cost [n] + min( solve( cost, n-1,
                                          dp )
                            solve( cost, n-2, dp));
    return dp[n];
}
```



④ Tabulation (Bottom - Up)

Pseudocode :-

```
int solve( vector<int> &cost, int n)
{
    vector<int> dp(n+1);

    dp[0] = cost[0];
    dp[1] = cost[1]

    for( int i = 2; i<n; i++) {

        dp[i] = cost[i] + min( dp[i-1], dp[i-2]);
    }

    return min(dp[n-1], dp[n-2]);
}
```

Time Complexity : O(n)
Space Complexity : O(n)

(5) Space optimization:

Pseudocode:

```
int solve (vector<int>& cost, int n)
{
    int prev2 = cost[0];
    int prev1 = cost[1];

    for( int i=2; i<n; i++){
        int curr = cost[i] + min(prev1,
                                 prev2);
        prev2 = prev1;
        prev1 = curr;
    }

    return min(prev1, prev2);
}
```

Time Complexity :	O(n)
Space Complexity :	O(1)