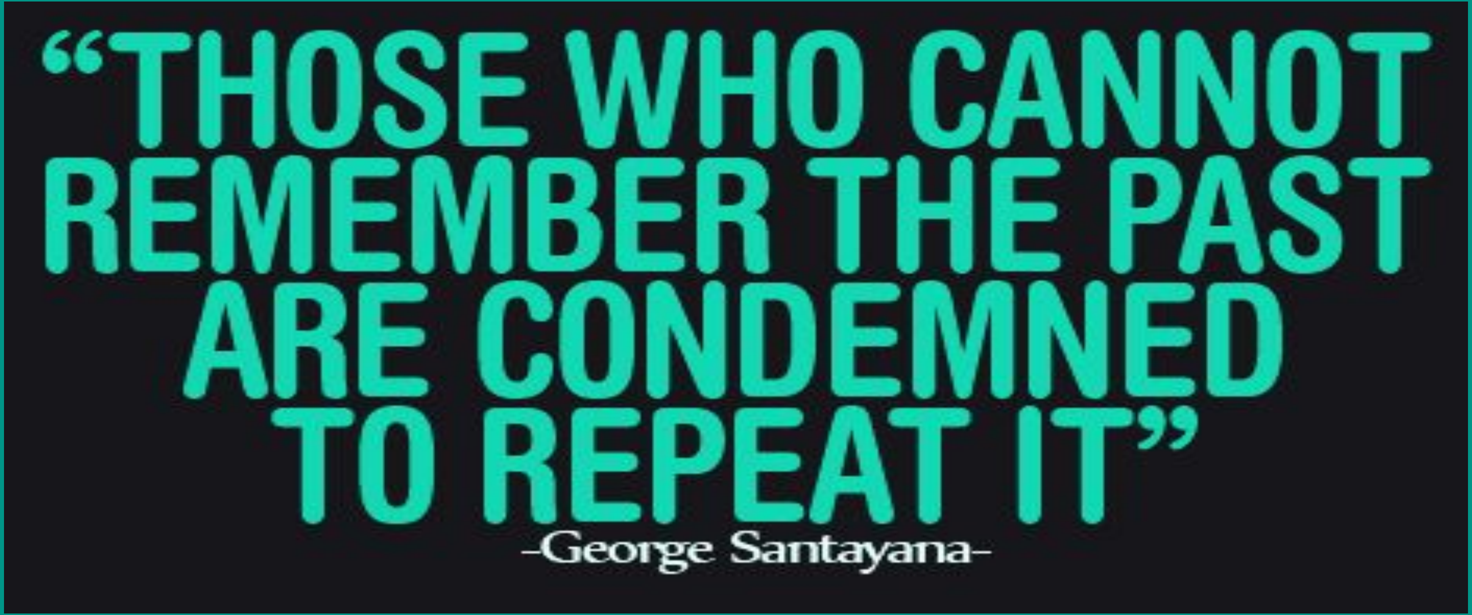
Dynamic Programming 1

-Priyansh Agarwal

Why Dynamic Programming?



“THOSE WHO CANNOT REMEMBER THE PAST ARE CONDEMNED TO REPEAT IT”

-George Santayana-

Why Dynamic Programming?

- Overlapping Subproblems
- Why calculate the answers for the subproblems again and again
- DP helps cover all possible cases (Greedy vs DP)

Disclaimer: You can never memorize all the DP techniques and tricks. It is more about developing the mindset to identify DP problems and tackle them

Problem 1:

Find the sum of first 5 natural numbers
Find the sum of first 6 natural numbers
Find the sum of first 7 natural numbers
…

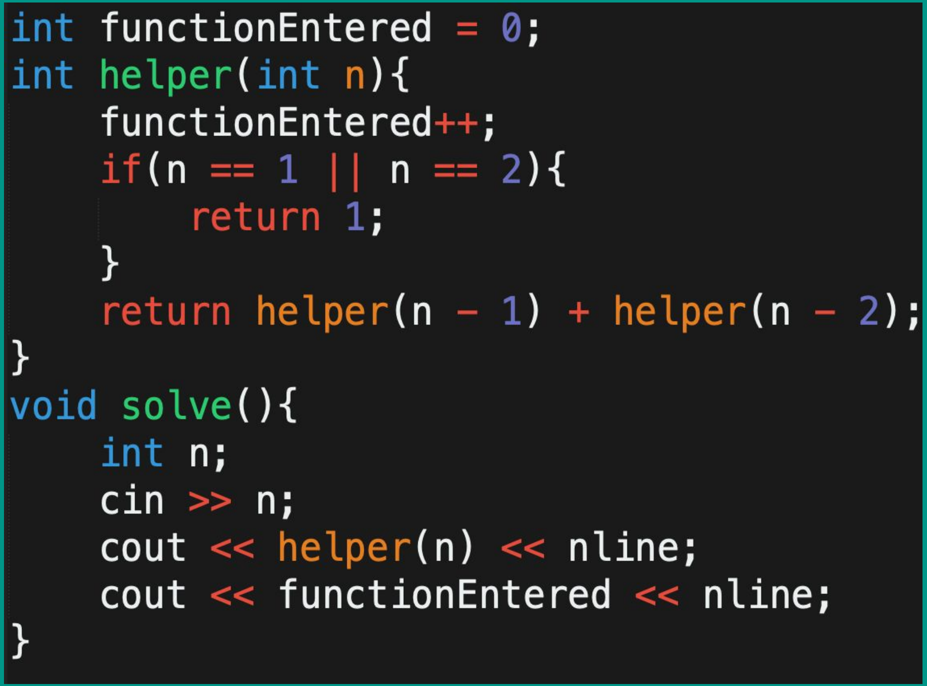
Note: Can’t use the formula (N * (N + 1)) / 2

Problem 2:

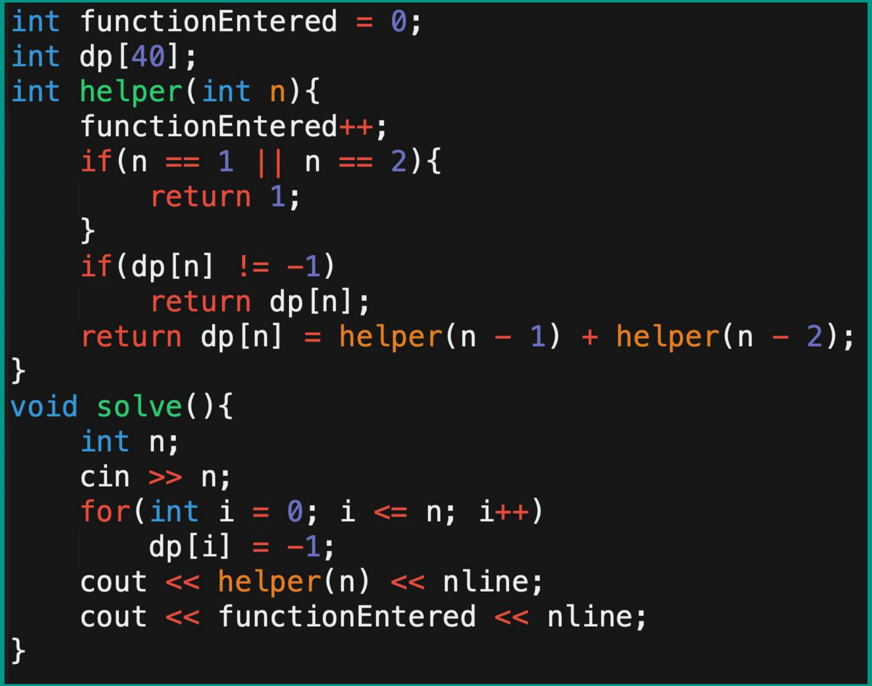
Find the Nth Fibonacci Number where $F(n) = F(n - 1) + F(n - 2)$

$F(1) = 1$
$F(2) = 1$
$F(3) = F(2) + F(1) = 2$
$F(4) = F(3) + F(2) = 3$
$F(5) = F(4) + F(3) = 5$

Problem 2: (Comparing with and without DP solutions)



```cpp
int functionEntered = 0;
int helper(int n){
    functionEntered++;
    if(n == 1 || n == 2){
        return 1;
    }
    return helper(n - 1) + helper(n - 2);
}
void solve(){
    int n;
    cin >> n;
    cout << helper(n) << nline;
    cout << functionEntered << nline;
}
```



```cpp
int functionEntered = 0;
int dp[40];
int helper(int n){
    functionEntered++;
    if(n == 1 || n == 2){
        return 1;
    }
    if(dp[n] != -1)
        return dp[n];
    return dp[n] = helper(n - 1) + helper(n - 2);
}
void solve(){
    int n;
    cin >> n;
    for(int i = 0; i <= n; i++)
        dp[i] = -1;
    cout << helper(n) << nline;
    cout << functionEntered << nline;
}
```

functionEntered = 1664079 with n = 30

functionEntered = 57 with n = 30

States and Transitions

State: A subproblem that we want to solve. The subproblem may be complex or easy to solve but the final aim is to solve the final problem which may be defined by a relation between the smaller subproblems. In the previous problem a state is simply (i) and dp[i] tells us the i^{th} fibonacci number

Transition: Calculating the answer for a state (subproblem) by using the answers of other smaller states (subproblems). In the previous problem dp[i] = dp[i - 1] + dp[i - 2] is a transition

Problem 3:

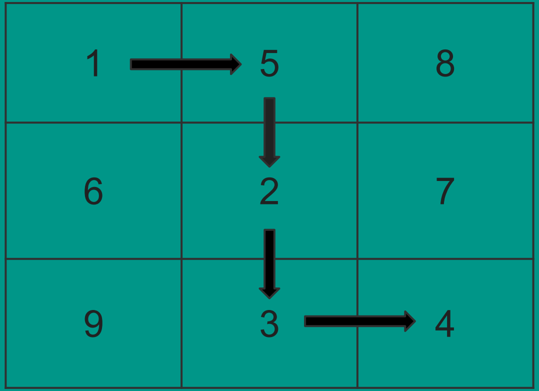
Given an array of integers (both positive and negative). Pick a subsequence of elements from it such that no 2 adjacent elements are picked and the sum of picked elements is maximized.

A = [1, 2, 3, 4, 5]
Picked subsequence = [1, 3, 5]

A = [1, 4, 2, -10, 10]
Picked Subsequence = [4, 10]

Problem 4:

Given a 2D grid (N X M) with numbers written in each cell, find the path from (0, 0) to (N - 1, M - 1) with minimum possible sum of values of the cells on the path You can only move down or right.



1	5	8
6	2	7
9	3	4

Problem 5

The game starts with a number ‘n’ and Alice and Bob make moves alternatively with Alice going first. If the current number is ‘k’, the player, whose turn it is, can convert k to {k-1, k-2, k-3}. If the current number is a Prime, the player cannot make a move and thus loses. Determine who will win if they both play optimally.

$$2 <= n <= 10^6$$

Examples:

N = 6 - Alice wins as she can subtract 1 from 6 and convert it 5 (a Prime)

N = 27 - No, matter what Alice converts 27 to {26, 25, 24}, Bob can convert the next number to 23 (a Prime)and win.

How to identify a DP problem?

Repeating subtasks: If I have the answer of state, then why should I calculate it again and waste time

Pro Tips for contests:

- Look for small constraints in the problem. (Most probably it would be dp and not greedy)
- Identify states and transition time for each state.
- Calculate time complexity as (number of states * transition time for each state).
- If this number fits into your Time limit (Great), if not, try to see if you can skip some states and still get the right answer.
- Try to reduce the transition time by using some Data Structure if transition time is the bottleneck
- Never try to over optimize. If your current states and transition time fit into your Time Limit, just code it and do not optimize it further.

Bonus Problem Link

Given an array ‘a’ of ‘n’ integers, find the maximum length subsequence from this array such that for every 2 elements in the chosen subsequence, the following condition holds:

Either a_i divides a_j

Or a_j divides a_i

$$1 <= n <= 10^5$$

$$1 <= A[i] <= 10^5$$

Example:

7, 9, 3, 14, 63 - Answer = 3 [9, 3, 63]

1, 2, 3, 5, 7 - Answer = 2 [1, 2] ([1, 3], [1, 5], [1, 7] are also valid)