Dynamic Programming 2

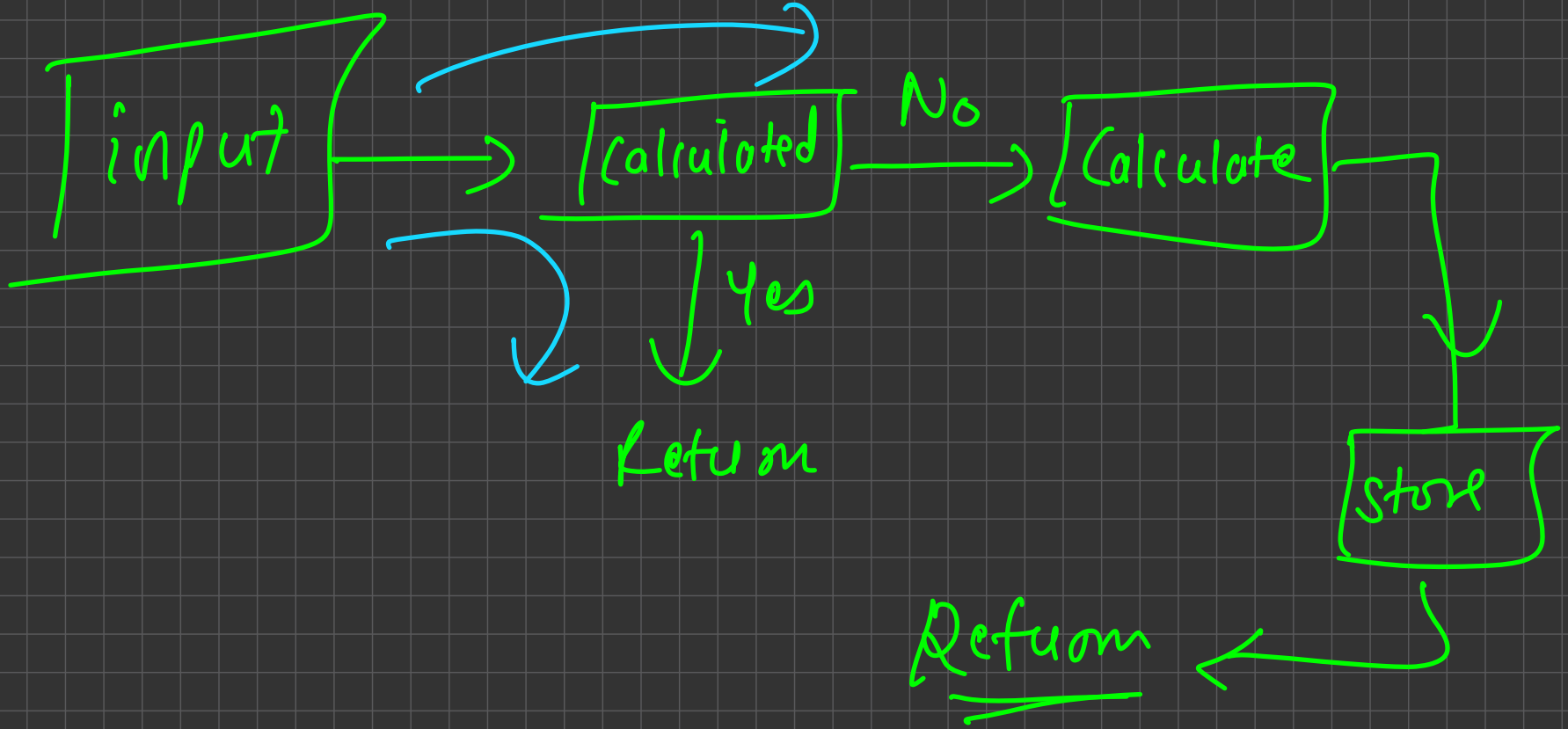
- Priyansh Agarwal

Recursive vs Iterative Solutions

Generic way to solve any dp problem

2 dp problems using technique

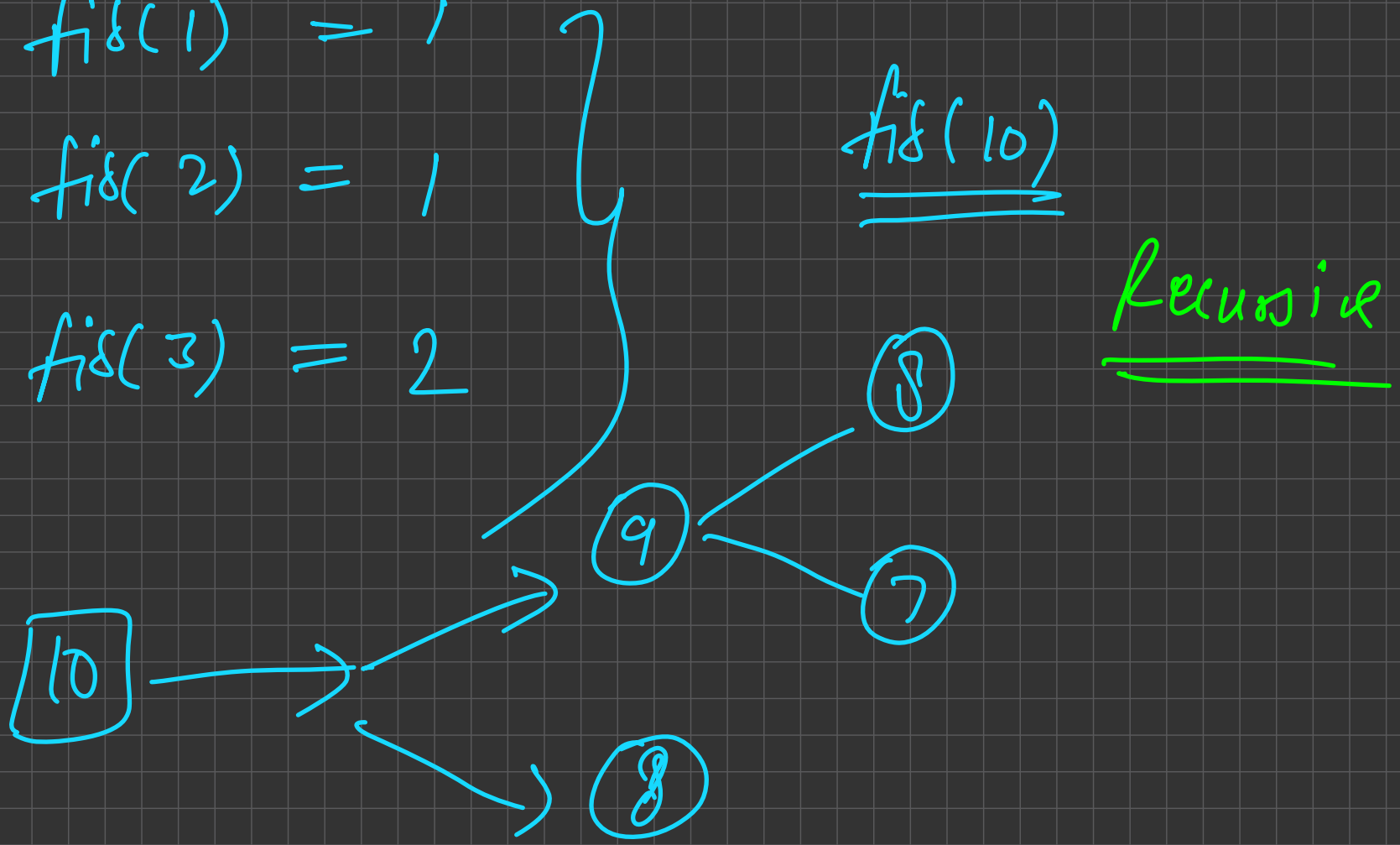
memoization



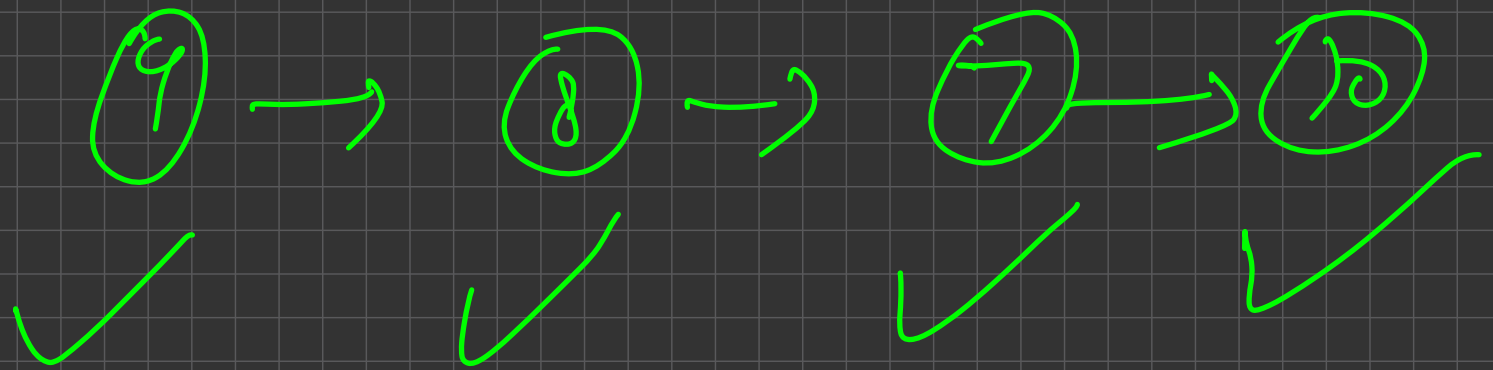
$fib(1) = 1$

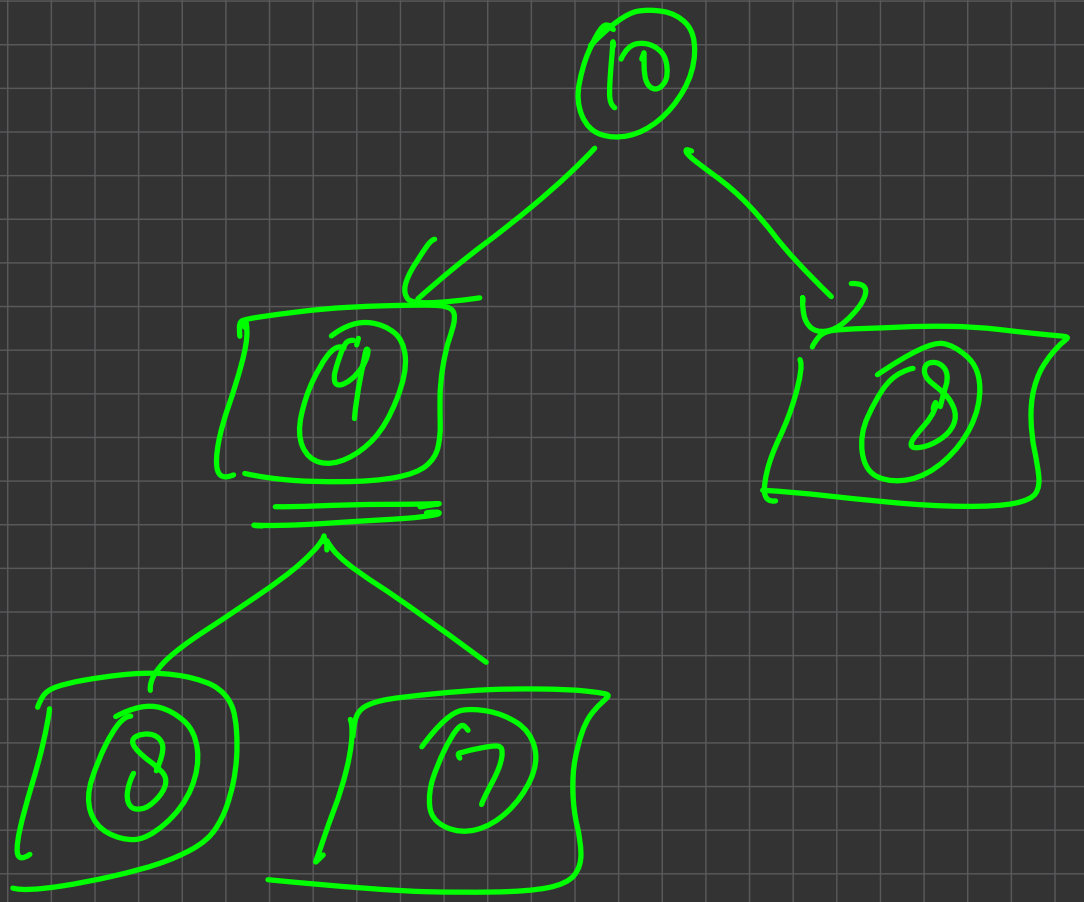
$fib(2) = 1$

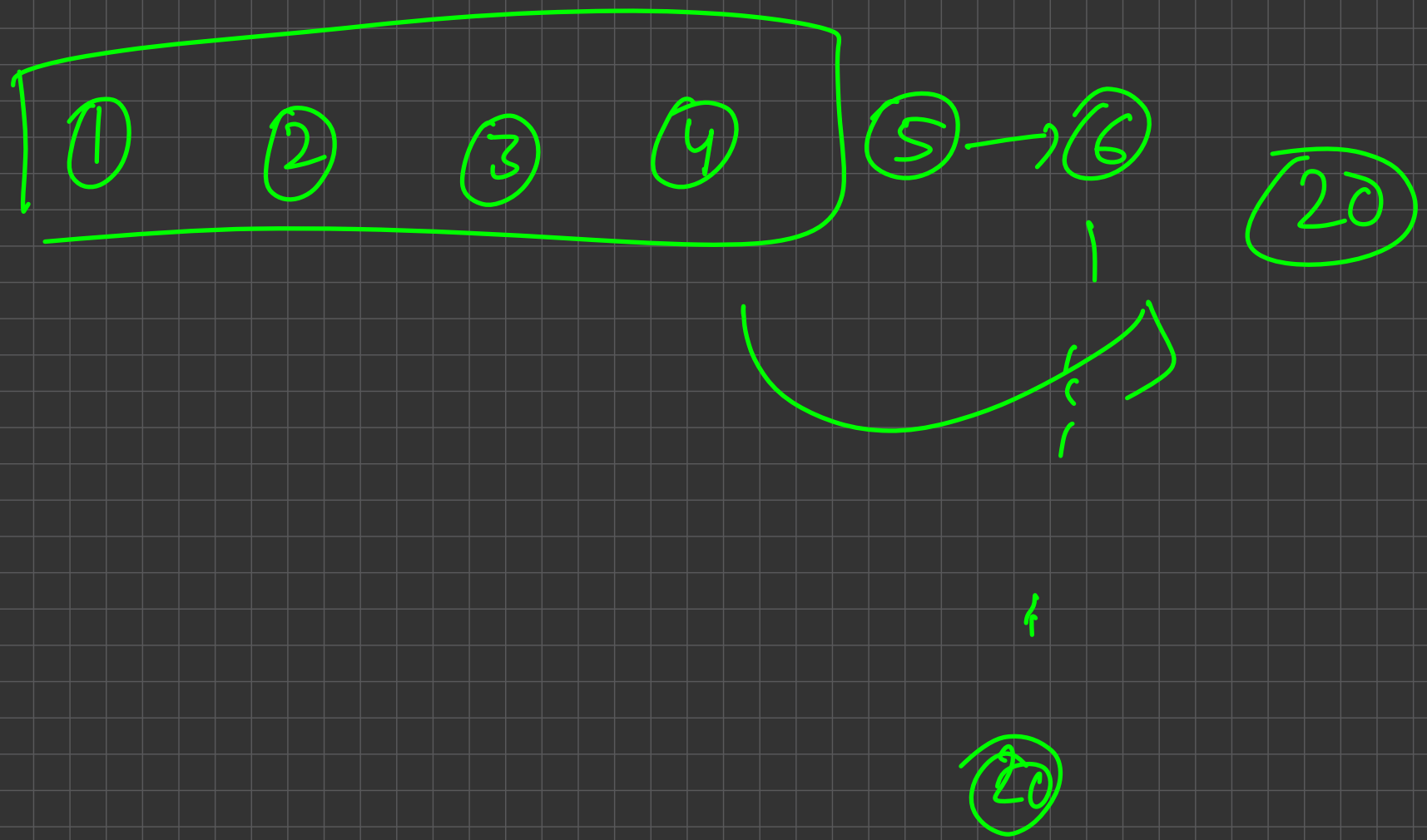
$fib(3) = 2$



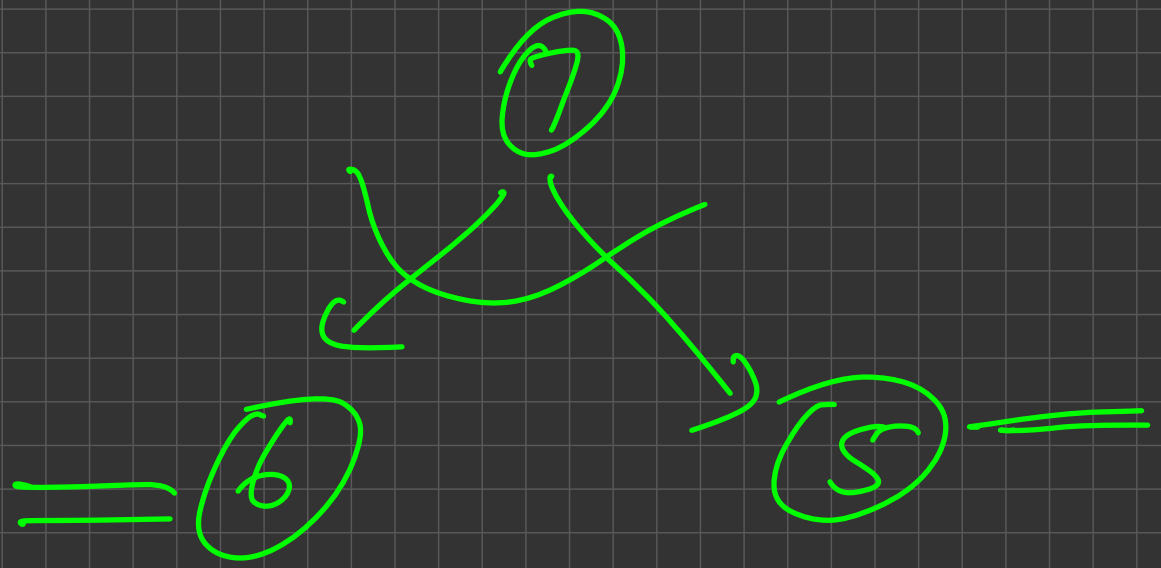
Flow of states



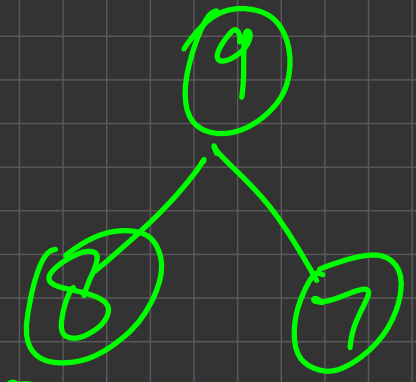

10
9
3
8
7


1
2
3
4
5
6
20
1
20

```
for (int i = 3 ; i <= n ; i++) {
    fib(i) = fib(i-1) + fib(i-2)
}
cout << fib(n) << endl;
```

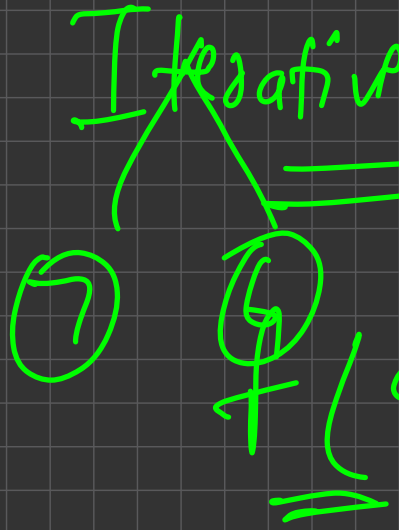

7
6
5

Recursive Solution



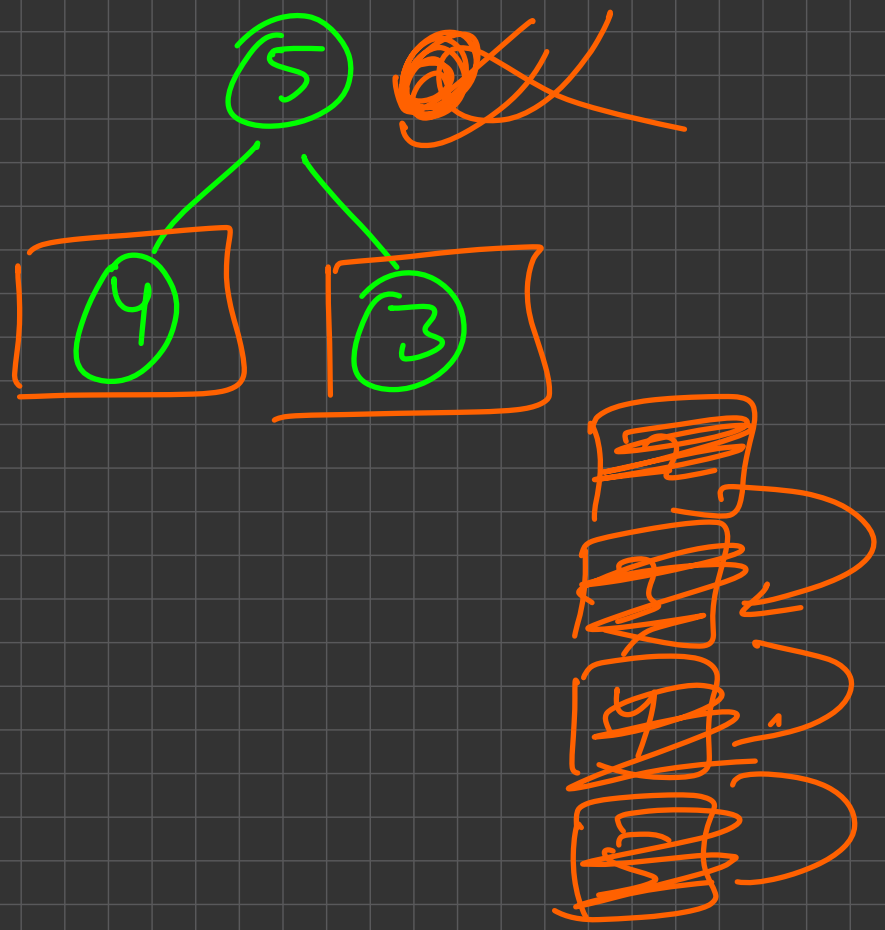
f(1) f(5) f(6)

Iterative Solution



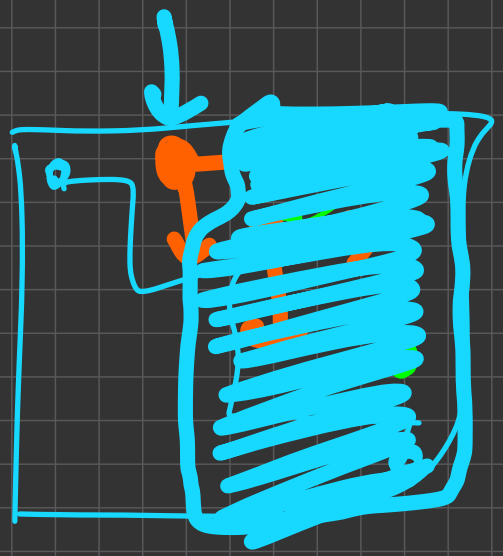
f(9)

f(1) , f(5) , f(6)



Recursive vs Iterative DP

Recursive	Iterative
Slower (runtime)	Faster (runtime)
No need to care about the flow	Important to calculate states in a way that current state can be derived from previously calculated states
Does not evaluate unnecessary states	All states are evaluated
Cannot apply many optimizations	Can apply optimizations

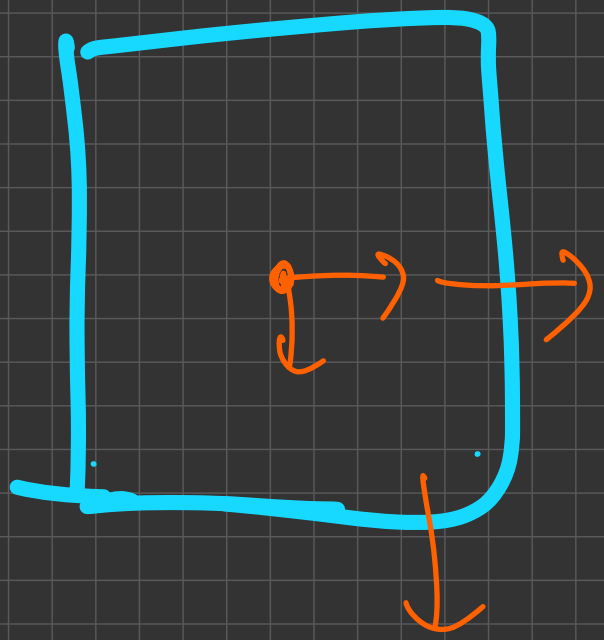


$dp[i][j]$ = min sum path from (i,j) to $(n-1, m-1)$

$dp[i][j] = grid[i][j] + \min \begin{cases} dp[i+1][j] \\ dp[i][j+1] \end{cases}$

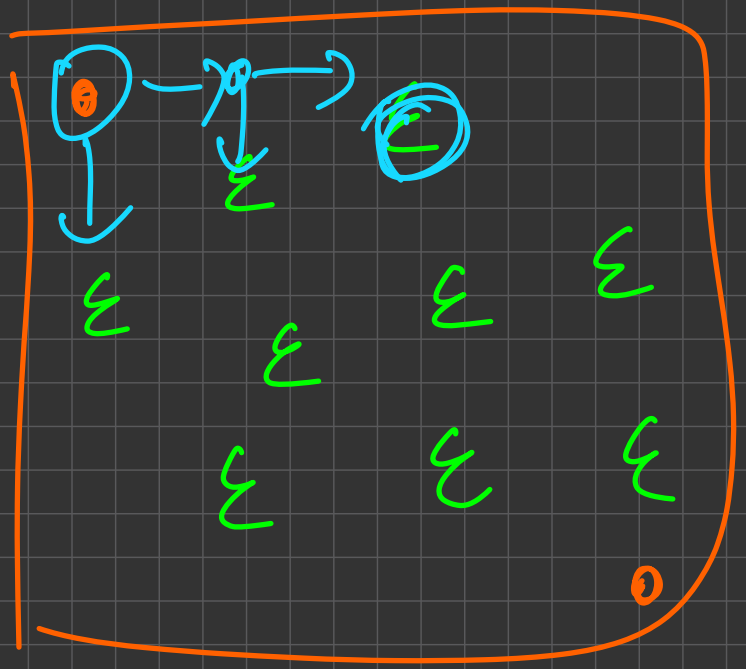
L.H.S.

R.H.S



```
dp[n-1][m-1] = grid[n-1][m-1]

for (int i = n-1; i >= 0; i--)
    for (int j = m-1; j >= 0; j--)
        if (i == n-1 && j == m-1)
            continue
        dp[i][j] = grid[i][j] +
            min { 2 possibilities
```



only 5000 cells are enclosed

1000 × 1000 grid

n x m

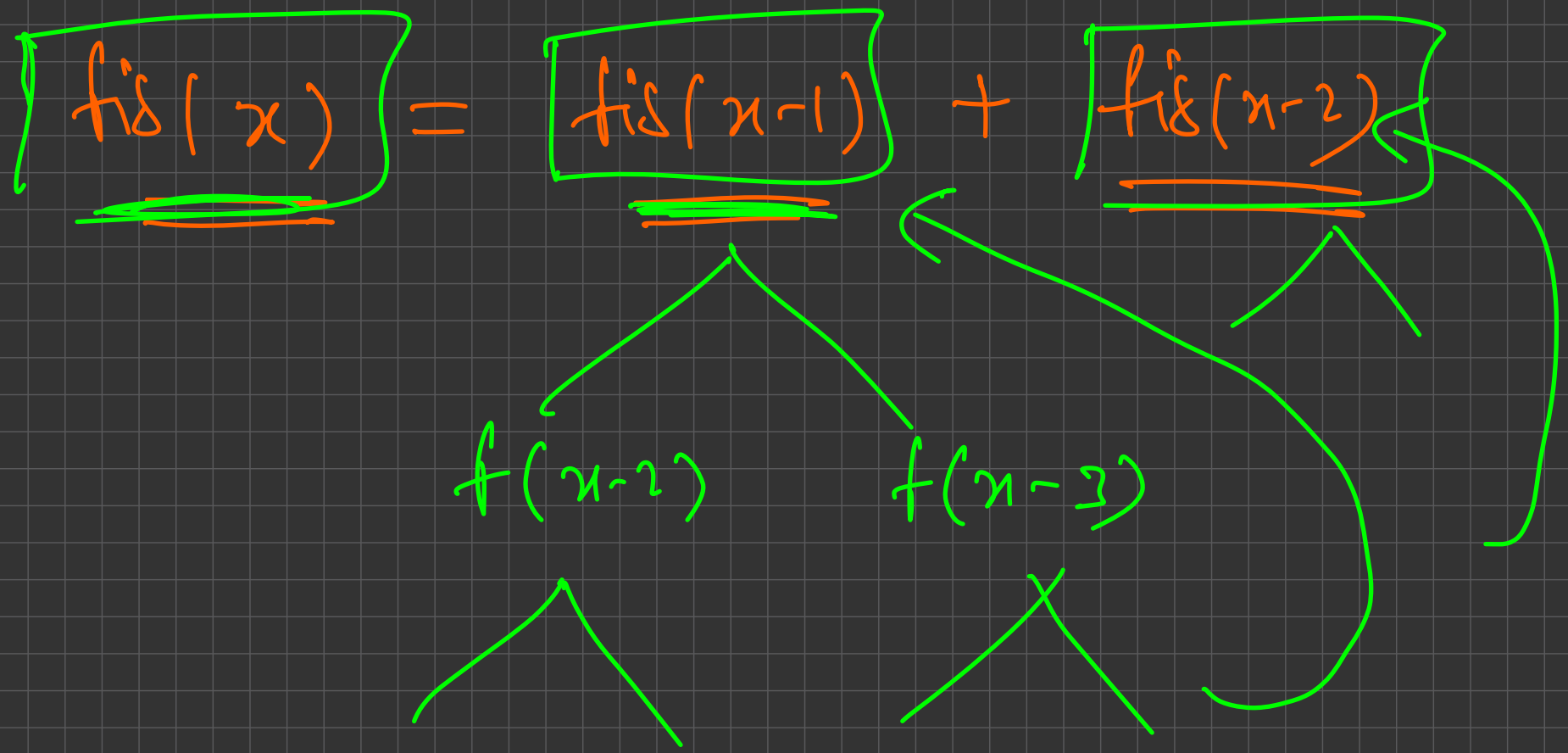
n x m 10^6

O(5000)

dp[i][j] O(n.m)

Recursive DP algo with map for memoization

O(relevant states). log( )


fib(x) = fib(x-1) + fib(x-2)
f(x-2)
f(x-3)

```
map < pair<int, int>, int > dp

int f ( int i, int j )
    { base cases here }
    if grid[i][j] == E
        return Int
    if ( dp.find(i,j) != dp.end() )
        return dp{i,j}

calculate
dp[i,j] = ans }
```

Converting Recursive to Iterative

Rule 1:

All the states that a particular state depends on must be evaluated before that state

Note:

You don't have to convert Recursive to Iterative if it is not intuitive at this point.

LHS

RHS

General Technique to solve any DP problem

1. State

   Clearly define the subproblem. Clearly understand when you are saying dp[i][j][k], what does it represent exactly

2. Transition:

   Define a relation b/w states. Assume that states on the right side of the equation have been calculated. Don't worry about them.

3. Base Case

   When does your transition fail? Call them base cases answer before hand. Basically handle them separately.

4. Final Subproblem

   What is the problem demanding you to find?

$df(0)(v)$

$df(n-1)(m-1)$

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

$fib(1) \rightarrow$ B.C

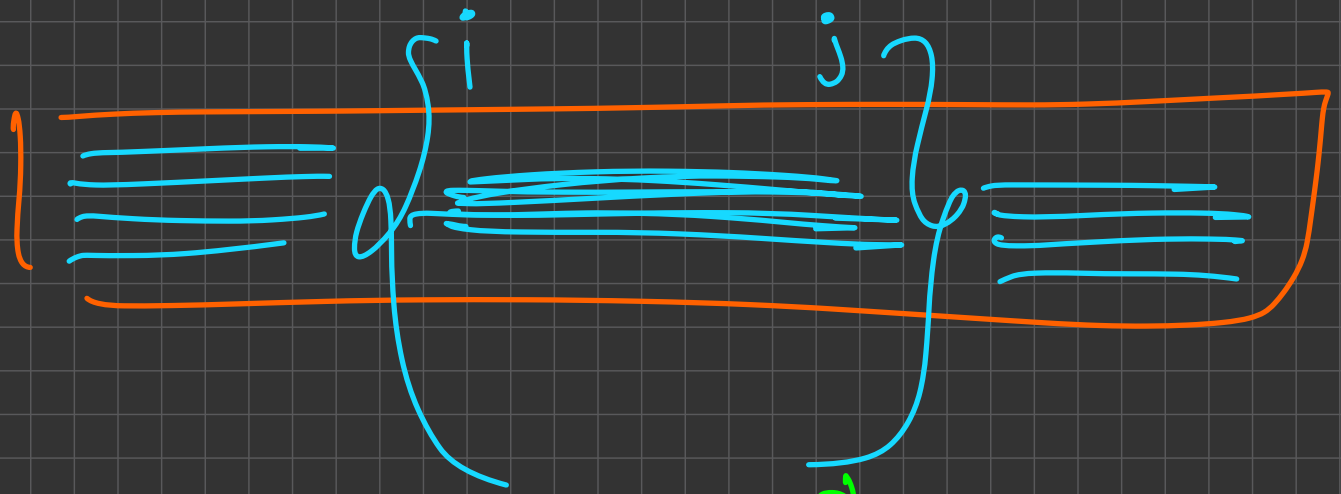
$fib(2) \Rightarrow fib(1) + fib(0)$ B.C

$fib(3)$

$$dp[i][j] = grid[i][j] + \min \begin{cases} dp[i+1][j] \\ dp[i][j+1] \end{cases}$$

Problem 1: Link

Problem 2: Link

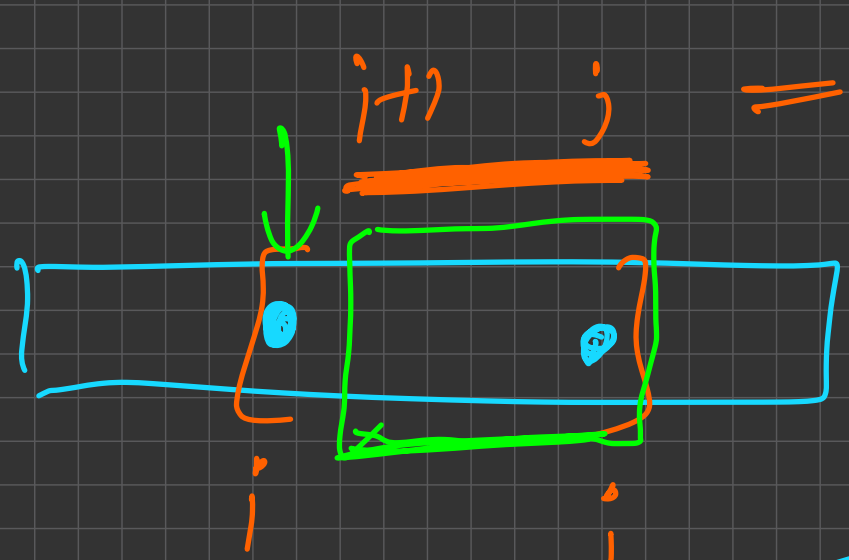


dp[i][j] = max score a player can get from the subarray (i to j) if it is their turn

dp[i+1][j]

dp[i][j] ① ②

dp[i+1][j] ② ①



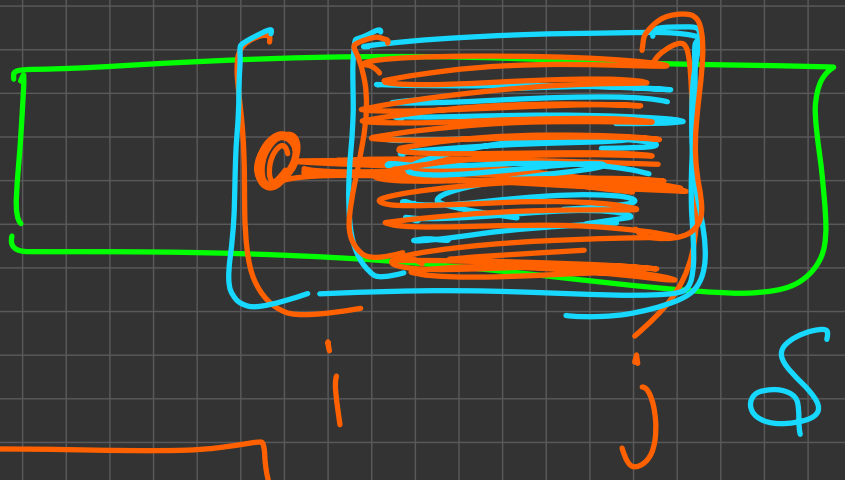
$= S_1$

$(i+1, j)$

$(i \text{ th } j)$

$(i, j-1)$

$dp(i)(j)$



$$\max(i) + s[i+1][j] - dp[i+1][j]$$

$dp[i+1][s_j]$

$s - dp[i+1][j]$

$$dp[i][j] = \max \begin{cases} a[i] + s[i+1][j] - dp[i+1][j] \\ a[j] + s[i][j-1] - dp[i][j-1] \end{cases}$$

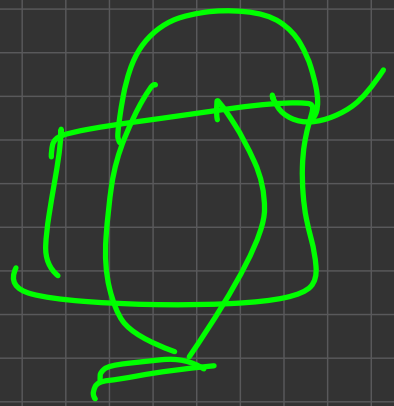
$dp[i][i]$

$dp[i+1][i]$

State —) Transition

Base Cue

$dp(i)(i) = a[i]$



$i > j$

$dp(i)(j)$

$dp[i][j][0]$

0 → 1st player
1 → 2nd player

= max sum list players can get from (i to j) if it is 1st player's turn

$$dp(i)[j](0) = \begin{cases} a[i] + s[i+1][j] - dp[i+1][j][1] \\ a[j] \end{cases}$$

$dp[0][n-1]$

$S[0][n-1] - dp[0][n-2]$

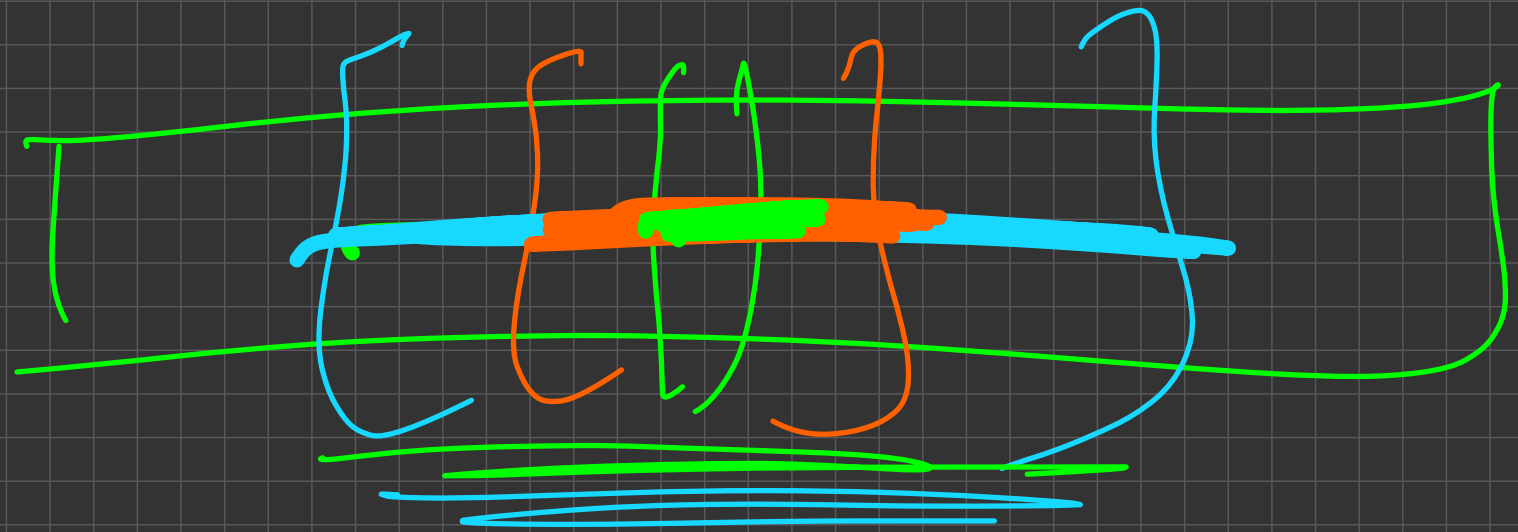
Problem 1: Link

Problem 2: Link

$dp[i][j] = 10^{18}$

Sum[n][n] → O(n²)

```
int f(int i, int j)
    if (i == j)
        return arr[i]
    if (dp[i][j] != 1e18)
        return dp[i][j]
dp[i][j] = max { arr[i] + sum[i+1][j] - f(i+1, j)
               { arr[j] + sum[i][j-1] - f(i, j-1)
return dp[i][j]
```



Problem 1: Link

Problem 2: Link