Dynamic Programming
Class 2/2

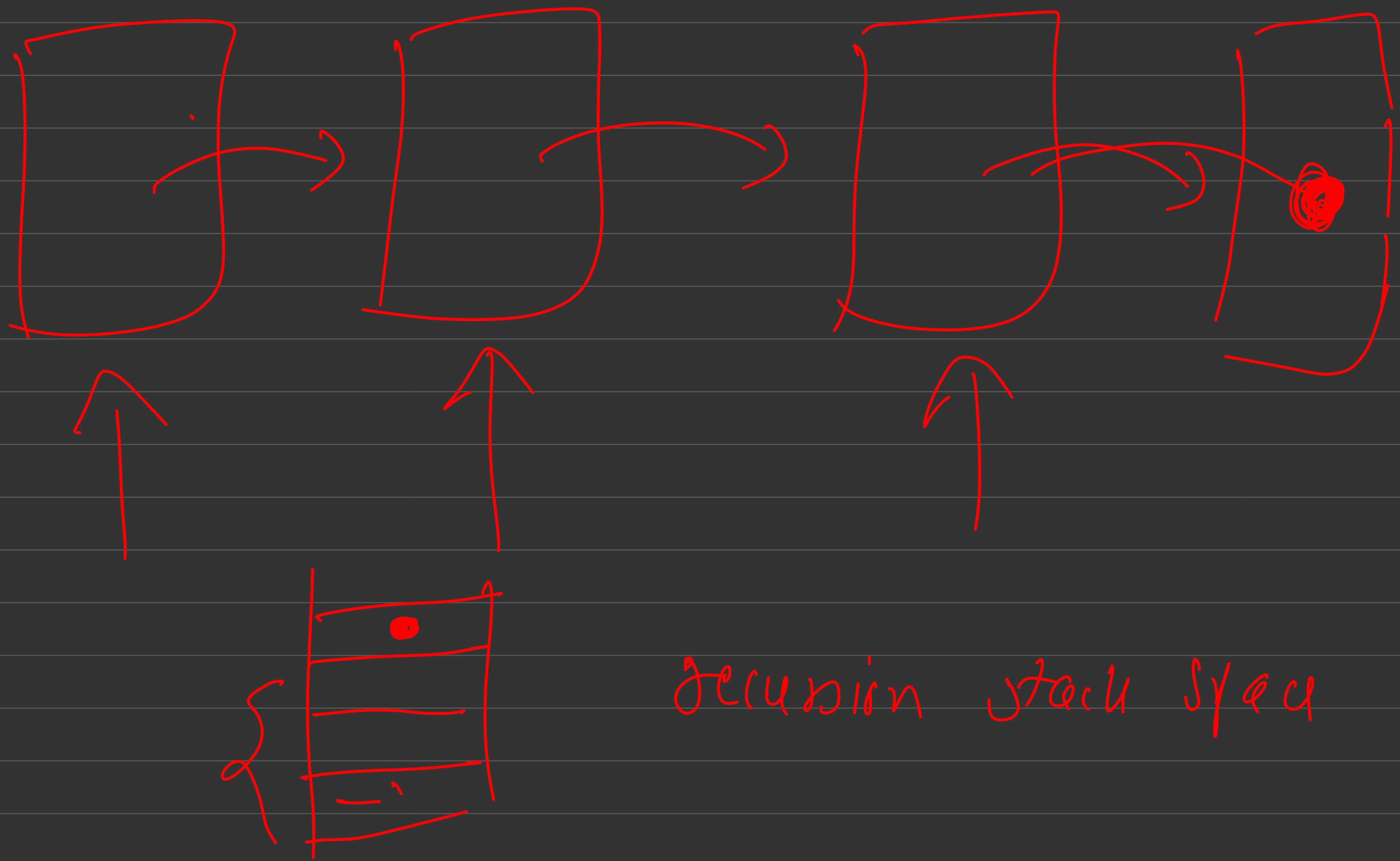
- Priyansh Agarwal

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

Default value req.

Not required

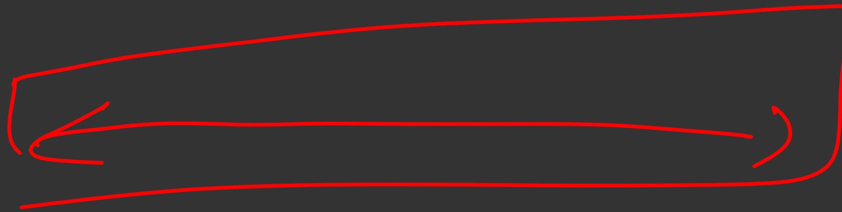


recursion stack space

```
for ( i -> 0 — N)
{ for (j -> 0 — N)
  { int a = 10;
  }
}
```

vectorization

np.sum( )   O(N)



np.sum

$O(N)$

28

Transition

L.H.S = R.H.S

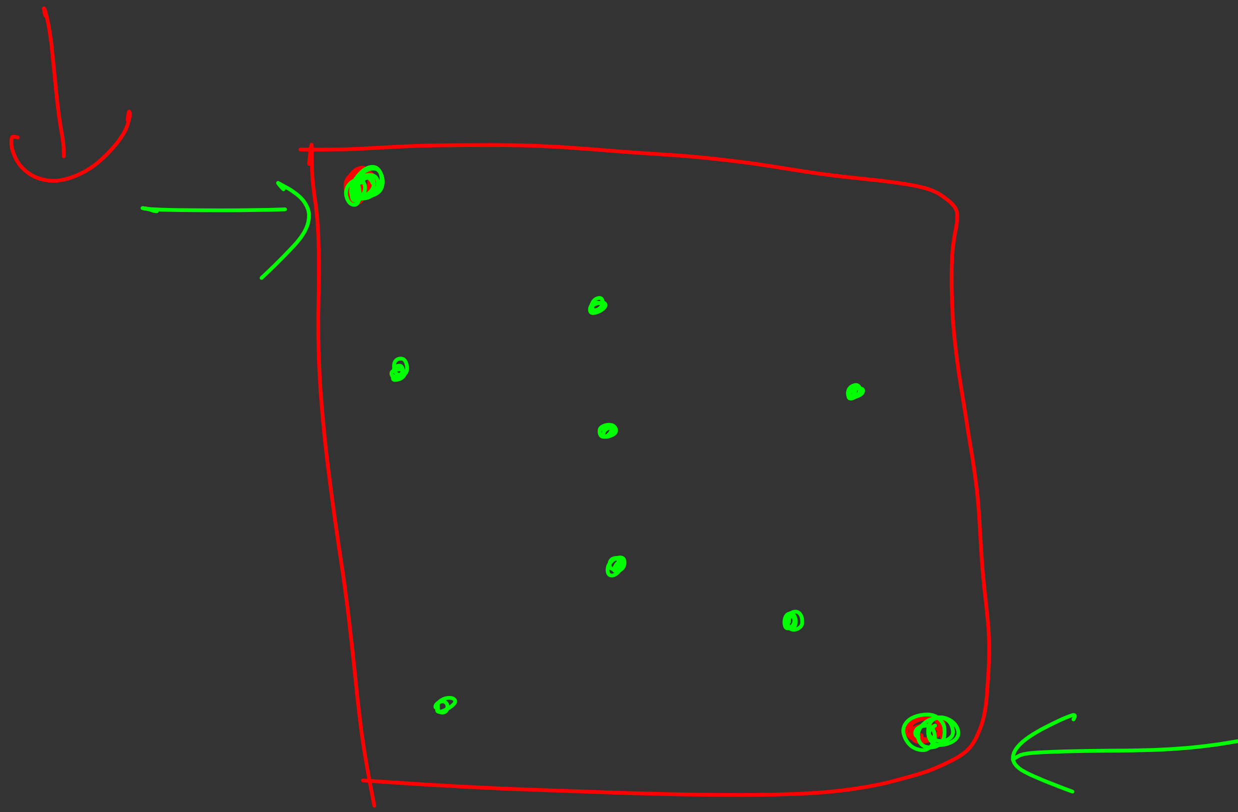
$a = \boxed{b + c}$

$$dp[i] = m + f(i+1) \cdot x$$

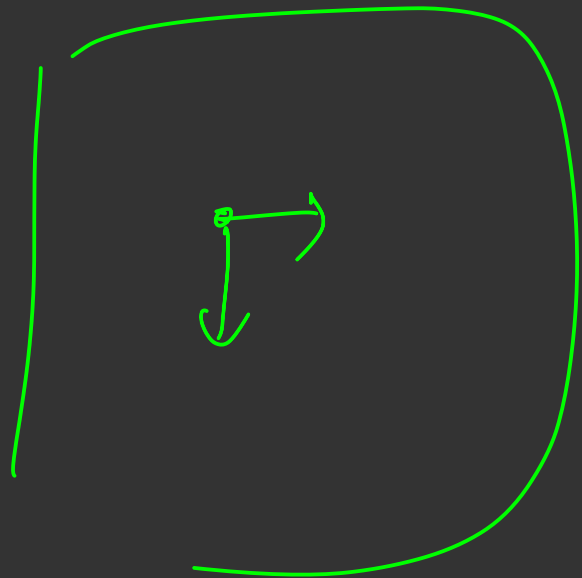
$$dp[i] = - + dp[i+1] \cdot x$$

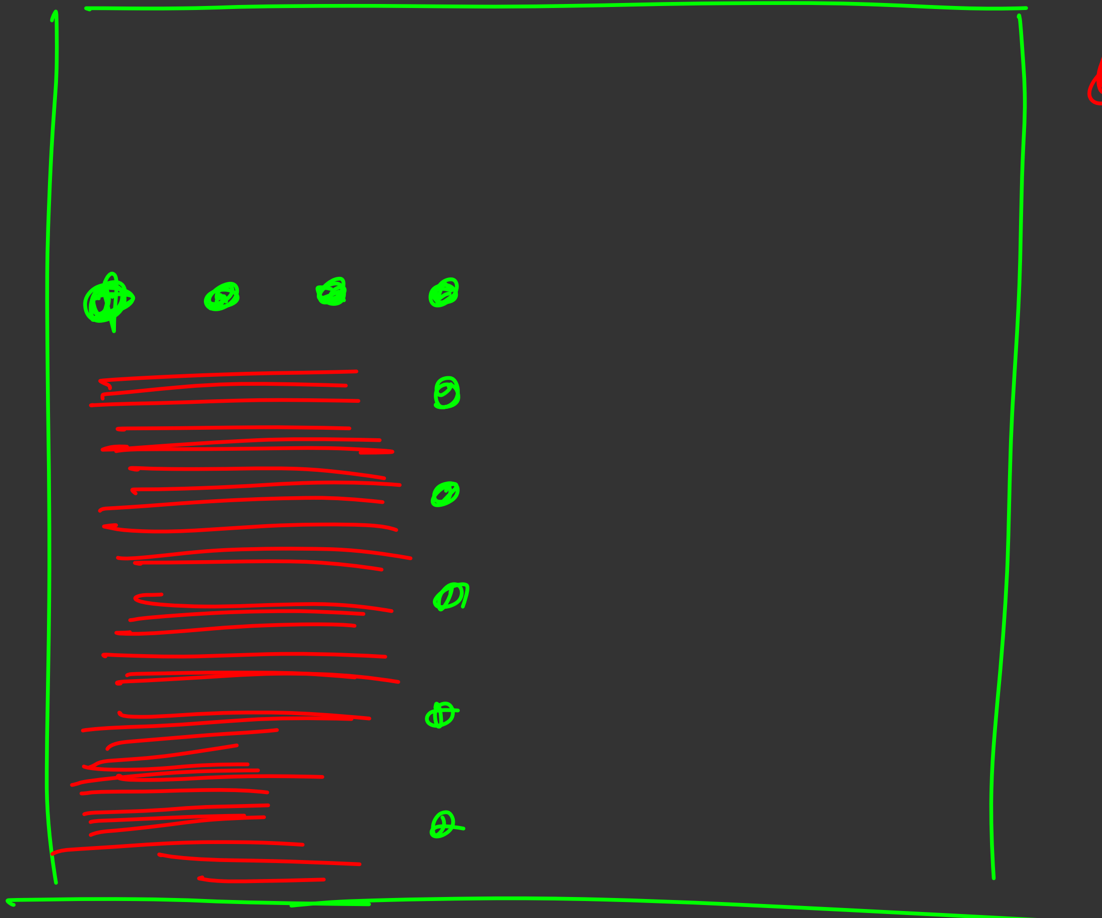
Recursive

Iterative



$$\{ dp[i][j] = dp[i+1][j] + dp[i][j+1] \}$$

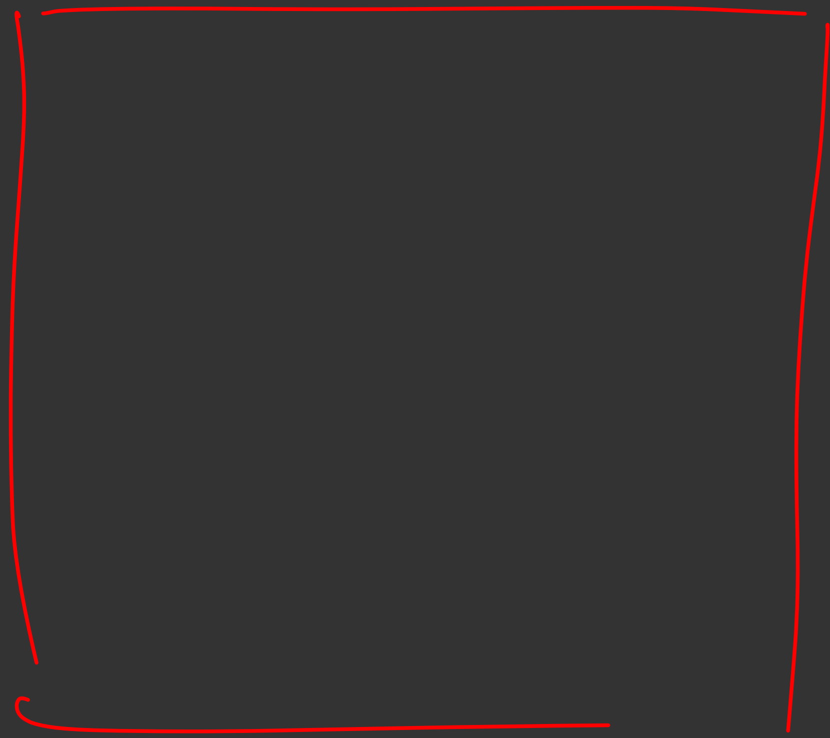




if grid[i][j]

== ods

return 0;

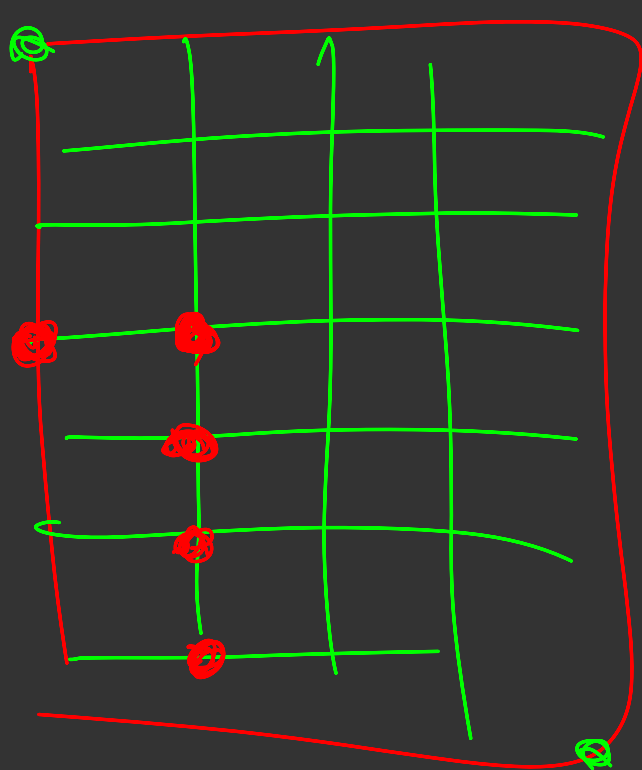


$10^9 \times 10^4$

$1/100$ of cells

do not

have an obstacle



$n \times n$

$n \to 10^5$ ①

$\frac{1}{1000}$th cells which do not have an obstacle

$dp[i][j]$

$dp[i][j] = dp[i+1][j] + dp[i][j+1]$

$\boxed{10^5 \times 10^5}$

```
if grid[i][j] == 'O'
    return 0;
```

n.n

map

~~a~~ int dp[n][n]

map(pair, int)

1

1000th

O(n²)

①

O(non obstacle states)

②

✓ $\frac{10^5 \times 10^5}{10^3}$

for (i → 0 to N-1)

{

for (j → 0 to N-1)

N^2

Recursion Iteration

dp[i][j]

dp[i-1][j]

dp[j][i-1]

dp[5][4]

dp[4][4]

L-H-S

dp[5][3]

R-H-S

R → I flow of states

$\boxed{dp(i) \to dp(i-1)}$

for (int i=0 , i<N ; i++) ① 0, 1, 2, 3

dp(i) → dp(0)

for (int i=n-1 , i>=0 , i--) ② 9, 8, 7, 6 dp[i] → dp[i+1]

dp[i][j] → dp[i+1][j]

dp[i][j+1]

① for (0 — N-1)

for (0 — N-1)

dp[i][j]

② for (N-1 — 0)

for (N-1 — 0)

dp[i][j]

$$dp[i][j] \rightarrow \begin{cases} dp[i][j-1] \\ dp[i][j-2] \\ \vdots \\ dp[i][i+1] \end{cases}$$

$i < j$

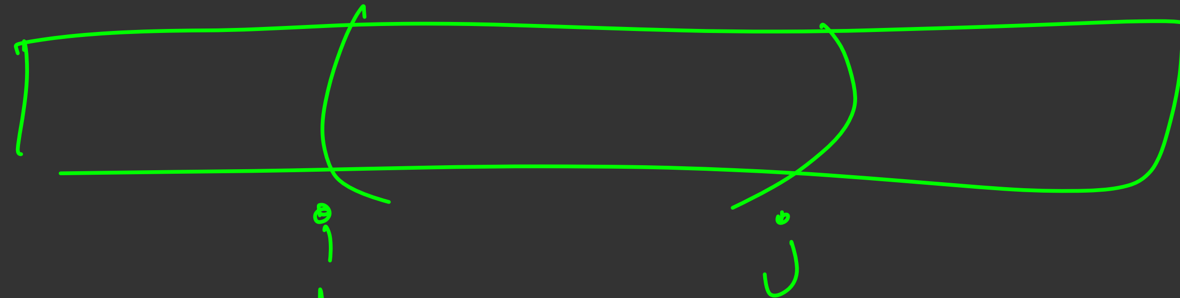
$(j - i + 1)$

$dp[i][j]$

$i \leq j$

0 —— k-1

i —— j



$dp[i][j] \longrightarrow dp[i][j-1]$

seg of ~~a~~ len x

$\longrightarrow x-1$

$\longrightarrow x-2 \cdots 1$

for (len → 1 → N −)

for (i → 0 — N − length)

N = 5 $\quad$ i = i

5 $\quad$ 3 $\quad$ j = i + len − 1

i = 4 $\quad$ dp[i][j]

i , $\quad$ j + len − 1

dp[i][j]

$j - i + 1 = x$

seq of len $x \longrightarrow x-1$

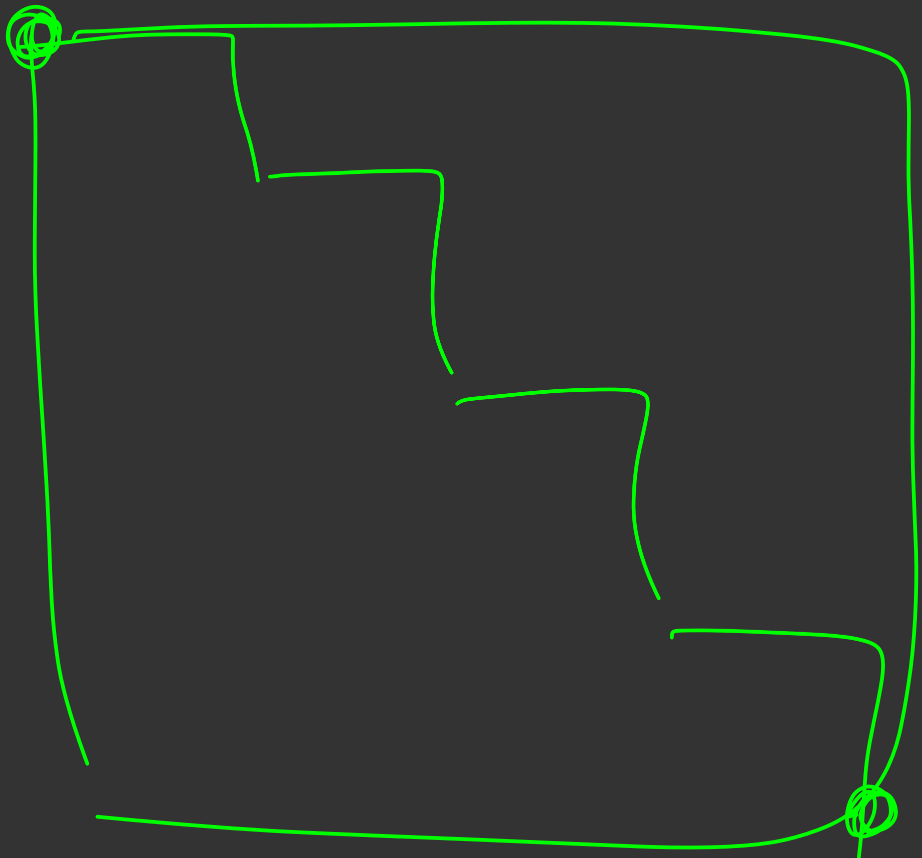
$x-2$

$\underline{\underline{5}} \longrightarrow 4, 3, 2 - 1$

1

1

1



min sum

path

```
f(i, j)      if(i == N-1 && j == N-1) return
                                       ar
    if (i > N-1 || j > N-1)
        return Inf
    if (dp[i][j] != -1)
        return dp[i][j]
    dp[i][j] = grid[i][j] + min(dp[i+1][j],
return dp[i][j]                    dp[i][j+1])
```

```
for (int i = n-1, i >= 0, i--)
  for (int j = n-1, j >= 0, j--)
    dp[i][j] = grid[i][j]
      + min { dp[i+1][j] : INF (i < N-1)
              dp[i][j+1] : INF (j < N-1)
```

$N \to 10^3$

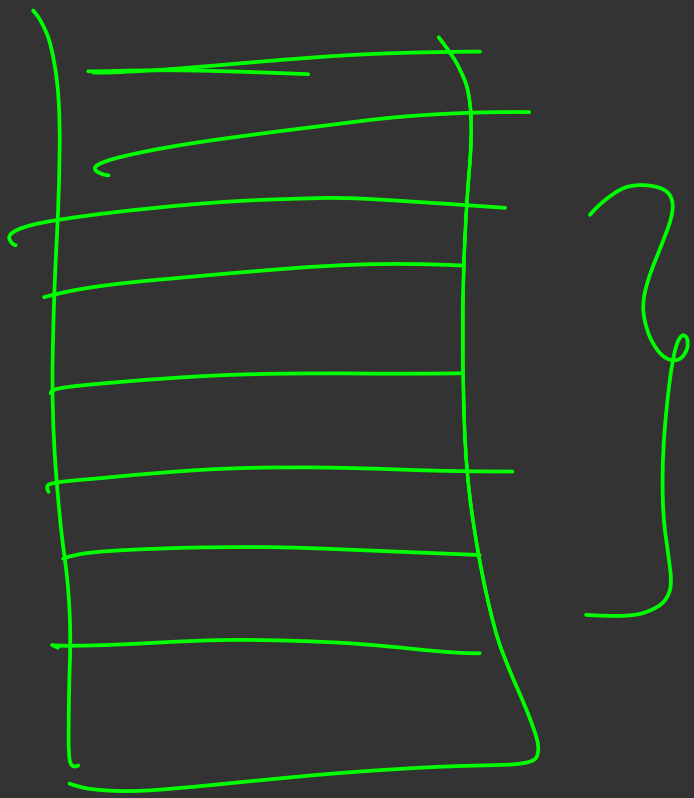
$Q(N,N) \to 10^9$

$[10^9 \times 10^3]$

10^{12}

$> 10^{10}$

stackov

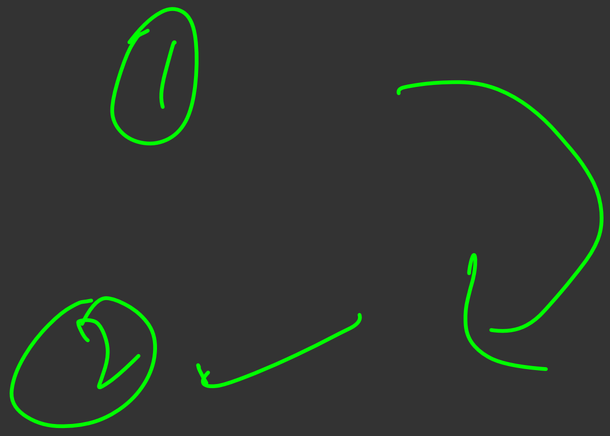


Stack overflow

Codefor

Start

State



$$dp[0] = 1$$

$$dp[1] = dp[0]$$

$$\left\{ \begin{aligned} dp[2] &= dp[0] + dp[1] \\ dp[3] &= dp[0] + dp[1] + dp[2] \\ &\;\vdots \\ dp[i] &= dp[0] + \underline{\qquad\qquad} \; dp[i-1] \end{aligned} \right.$$

$n!$

n — states — $\frac{n!}{2}$ $\left\{ \frac{1}{2}, \ldots, n \right\}$ $\frac{n!}{2}$

for (i → 1 — N−1)
for (j → 0 — i−1)
dp[i] += dp[j]

$O(n^2)$

$dp[i] = dp[0] + dp[1] + \dots dp[i-1]$

$O(n)$

$dp[i] = 2 \times dp[i-1]$

$$dp[i] = \boxed{dp[0] + dp[1] - - - - dp[i-2]} + dp[i-1]$$

$$dp[i-1] = \underline{dp[0] + dp[1] - - - - dp[i-2]}$$

$$dp[i] = dp[i-1] + dp[i-1]$$

$$= \boxed{2 \cdot dp[i-1]}$$

$\underline{\underline{O(1)}}$

$dp[0] = 1$, $dp[1] = 1$

2^n

$dp[1] = dp[0]$

$$\boxed{dp[n] = 2^{n-1}}$$

$dp[2] = 2 \cdot dp[1] = 2$

$dp[3] = 2 \cdot dp[2] = 4$

$dp[4] = 2 \cdot dp[3] = 8$

Matrix exponentiation

Linear Algebra

↳ fibonacci nu in o(logn)

without using any formula's

dp[a] [b] [c]

c = a+b

a = a, b = b,

c = c,

State → parameter } $c = a + b$

$c_1 = a_1 + b_1$

diff b/w 2 state

$d_f(a)(b)(c)$, $d_f(a_1)(b_1)(c_1)$

$c = a + b$

$a \neq a_1$ ||

~~a + b ⟩⟩ ⟩~~ $\neq$ $b \neq b_1$

State optimization

$c = a + b$

表 $(a, b) \longrightarrow a + b = c$

dp [1e9]

0 1e9 1

2^{1e9-1} 2

log n

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.

R v I

state opt

O(logn)

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?

Default Value

if dp[i][j] == -1

$B \rightarrow T \rightarrow S$

Final subproblem $\rightarrow$ S

Problem start

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

1

$dp[i] =$ ith fibn
number

2

dp[i]   dp[i][j] ...

dp[i][j]

$i, j+1$

$i+1, j$

$i-1, j$

$i, j-1$

fibonacci

state → $dp[i]$ = ith fibon numbers

tran → $dp[i] = dp[i-1] + dp[i-2]$

Base case → $dp[1] = 0$, $dp[0] = 0$

$dp[-1]$ defined ?

$dp[1] \rightarrow dp[0]$ & $dp[-1]$

nth fibonacci n ——

final sub —— dp(n)

0 1 —— n

$$\left(\sum_{i \geq 0}^{n} dp[i] \right)$$

Problem 1: Link

Problem 2: Link

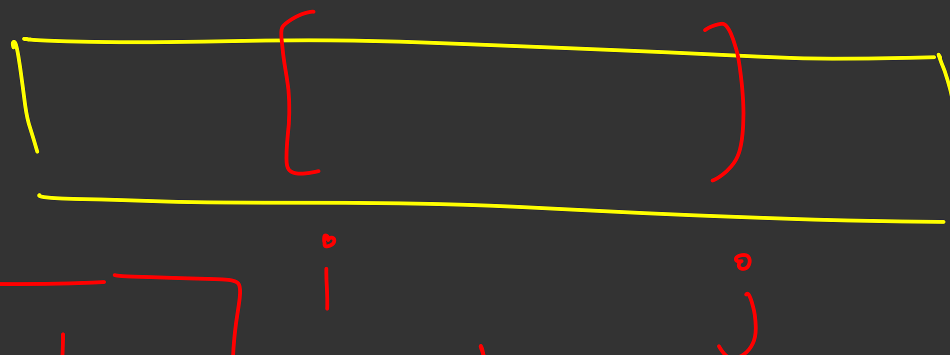
Problem 3: Link

$\{ dp[i][j]$ = max. sum a player can get from subarray (i to j) provided it is ~~our turn currently~~ that player's turn $\}$

$dp[i][j]$ = max. sum a player can get from subarray $[i \ldots j]$ provided it is that player's turn

$\downarrow$

$\boxed{\text{State}}$



$\boxed{\text{Player 1's turn}} \longrightarrow \underline{dp[i][j]} \longrightarrow P1$

$sum[i][j] - dp[i][j] \longrightarrow P2$

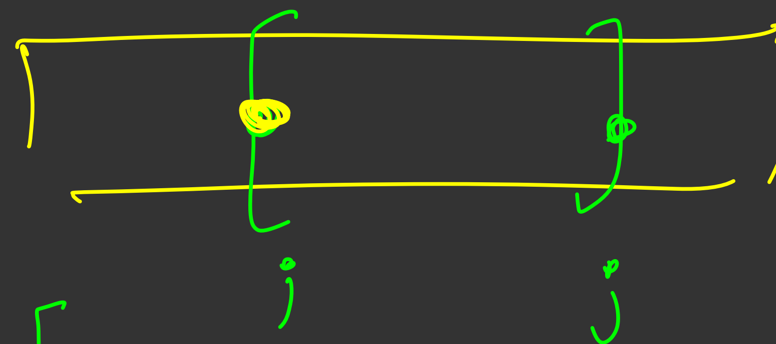
Player 2's turn → dp[i][j] → P2

sum[i][j] − dp[i][j] → P1

$dp[i][j]$ = max. sum a player can get from subarray $[i \ldots j]$ provided it is that player's turn

$\downarrow$

State



Transition

take $i \longrightarrow arr[i] + [\underline{sum[i+1][j]} - \underline{dp[i+1][j]}]$

take $j \longrightarrow arr[j] + [sum[i][j-1] - dp[i][j-1]]$

$$dp[i][j] = \max \begin{cases} arr[i] + \left[\underline{sum[i+1][j]} - \underline{dp[i+1][j]}\right] & (1) \\ arr[j] + \left[sum[i][j-1] - dp[i][j-1]\right] & (2) \end{cases}$$

State, Transition,

Base Case

$i < j$

$dp[i][i] = arr[i]$

Final Subproblem → State, Problem State

$dp[0][n-1]$,

$sum[0][n-1] - dp[0][n-1]$

Problem 1: Link

Problem 2: Link

Problem 3: Link