Dynamic Programming Problem Solving

- Priyansh Agarwal

Time and Space Complexity in DP

Time Complexity:

Average

Estimate: Number of States * Transition time for each state

Exact: Total transition time for all states

Space Complexity:

fib → O(n)

grid → O(n·m)

Number of States * Space required for each state

$dp[i]$ = ith fibonacci program

$$\begin{cases} dp(i) = \underbrace{dp(i-1)}_{x} + \underbrace{dp(i-2)}_{y} \end{cases}$$

find $dp(n)$ $\quad O(1)$

States $= O(n)$, $T \cdot T = O(1)$

Grid Problem

$dp[i][j]$ = min path sum from $(0,0)$ to (i,j)

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

$\#states = o(n.m)$ $\quad T.T = \underline{O(1)}$

Fibonacci Problem → O(n)

Grid Problem → O(n·m)

---

dp(i) = some state

dp(i) = dp(i-1) + dp(i-2) + dp(i-3)
- - - - dp(0)

dp(n) ??

dp(0) = k

dp(1) = dp(0)

dp(2) = dp(1) + dp(0)

states = O(n)

0 → 1 --- 5 → 5

1 → 1 k → k

2 → 2 n → n

3 → 3

n/2

states $= O(n)$

Average T.T. per state $= n/2$

$= O(n)$

Total Time complexity $= O(n \times n) = O(n^2)$

dp[0] → 0

dp[1] → 1

dp[2] → 2

⋮

dp[n] → n

$0+1+2+\cdots+n$

$= \frac{n\cdot(n+1)}{2}$

$= O(n^2)$

$dp(1)$, $dp(2)$ ---- $dp(n)$

$dp(1) = n/1$ steps

$dp(2) = n/2$ steps

$\vdots$

$dp(n) = n/n$ steps

$\frac{n}{1}$ $n/2$ $n/3$ ---- n/n

States × worst case

$T \cdot T$

$= n \cdot n = n^2$

$\sum_x$ transition time of state x

$n + n/2 + n/3 + n/4 \quad \cdots \quad n/n$

$= n\left[1 + 1/2 + 1/3 + 1/4 \quad \cdots \quad 1/n\right]$

$\leq \log n$

$= \leq n \log n \qquad O(n \log n)$

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

Recursive

```
int f(int n) {
    if (n==1 || n==2)
        return 1;
    if (dp[n] != -1)
        return dp[n]
    dp[n] = f(n-1) + f(n-2)
    return dp[n]
}
```

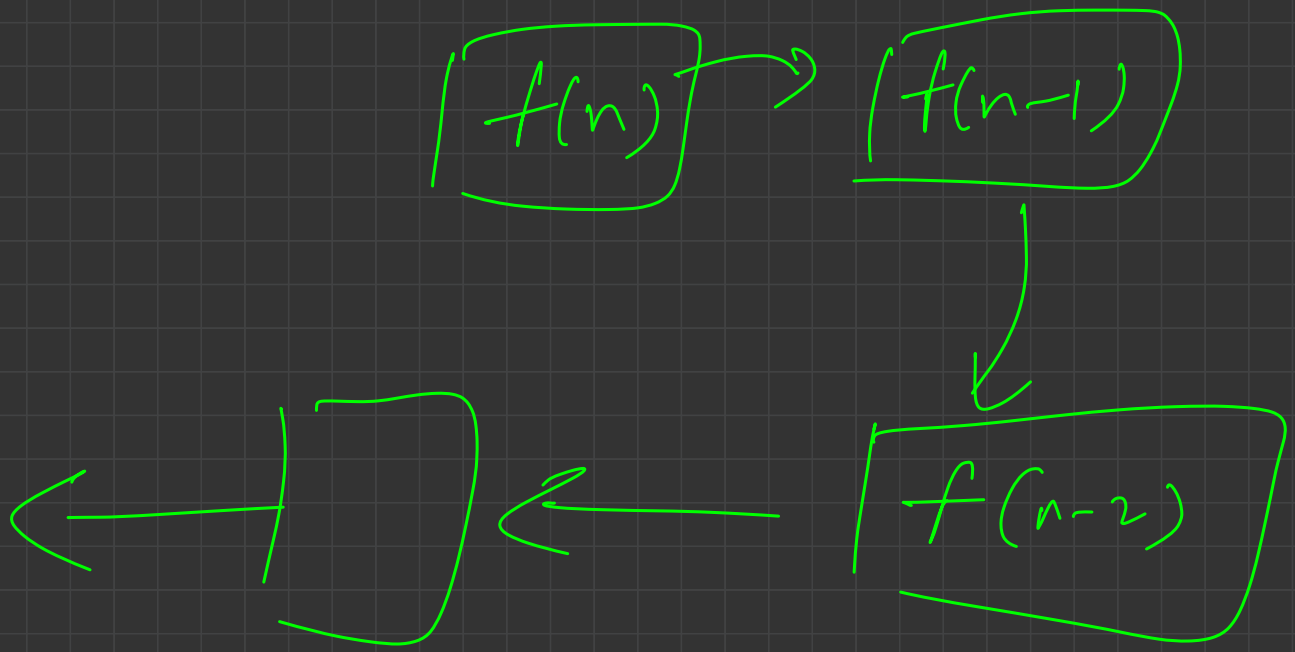
Iterative

```
int f(int n) {
    vector<int> dp(n+1);
    dp[1] = dp[2] = 1;
    for (int i=3; i<=n; i++)
        dp[i] = dp[i-1] + dp[i-2]
    return dp[n]
}
```

$$dp(n) = f(n-1) + f(n-2)$$

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

recursive



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.

s_1

$s_1 \longrightarrow (s_2, s_3, s_4)$

linear equation

$s_1 = s_2 + s_3 - s_4$

$s_1 = a s_2 + b s_3 + c s_4$

answer for state

x = answer for state y

+ answer for state z

$f(y)$ → returns answer for state y

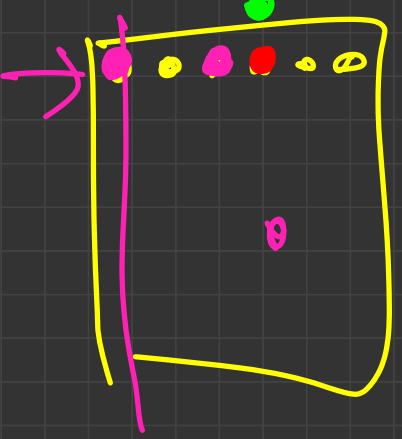
$f(z)$ → returns answer for state z

$A_x = f(y) + f(z)$ Recursive

Don't know answer

$A_x = A_y + A_z$

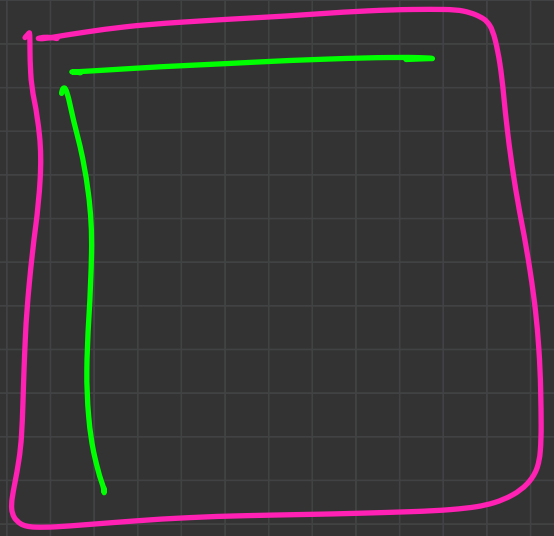
$$dp[i][j] = grid[i][j] + \min \begin{cases} dp[i-1][j] \\ dp[i][j-1] \end{cases}$$



```
vector<vector<int>> dp(n,
                       vector<int>(m));
for (int i=0; i<n; i++)
    for (int j=0; j<m; j++)
        if (i==0 and j==0)
            dp[i][j] = grid[i][j]
        else if (i==0
            dp[i][j] = grid[i][j]
```

```
                + dp[i][j-1]
else if (j==0)
    dp[i][j] = grid[i][j]
             + dp[i-1][j]
else
    dp[i][j] = grid[i][j]
             + min{ dp[i-1][j]
                    dp[i][j-1] }

cout << dp[n-1][m-1]
```



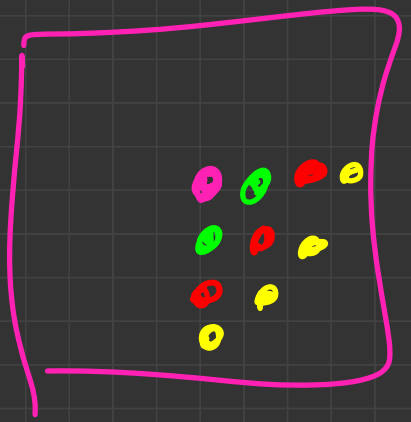
```
vector<vector<int>> dp(n,
                       vector<int>(m));
dp[0][0] = grid[0][0]
for (int i = 1 ; i < m ; i++)
    dp[0][i] = grid[0][i] + dp[0][i-1]
for (int i = 1 ; i < n ; i++)
    dp[i][0] = grid[i][0] + dp[i-1][0]
```

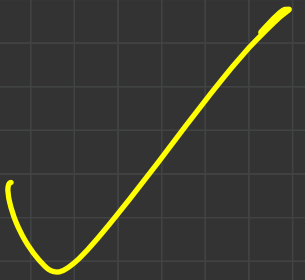
```
for (int i=1 ; i<n ; i++)
    for (int j =1 ; j<m ; j++)
```

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

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

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





S → T.E → flow of state

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?

General Technique to solve any DP problem

dp[i] / f(i) = ith fibonacci number

1. State

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

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

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.

dp[1] , dp[2]

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?

problem statement / definition of state } dp(n)

General Technique to solve any DP problem

1. State dp[i][j] = min sum path from (0,0) to (i,j)

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

2. Transition: dp[i][j] = grid[i][j] + min { dp[i-1][j], dp[i][j-1] }

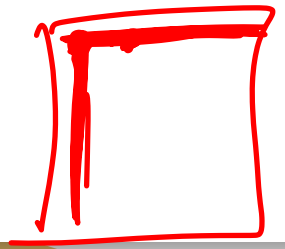
   Define a relation b/w states. Assume that states on the right side of the equation have been calculated. Don't worry about them.

3. Base Case first row & first column

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

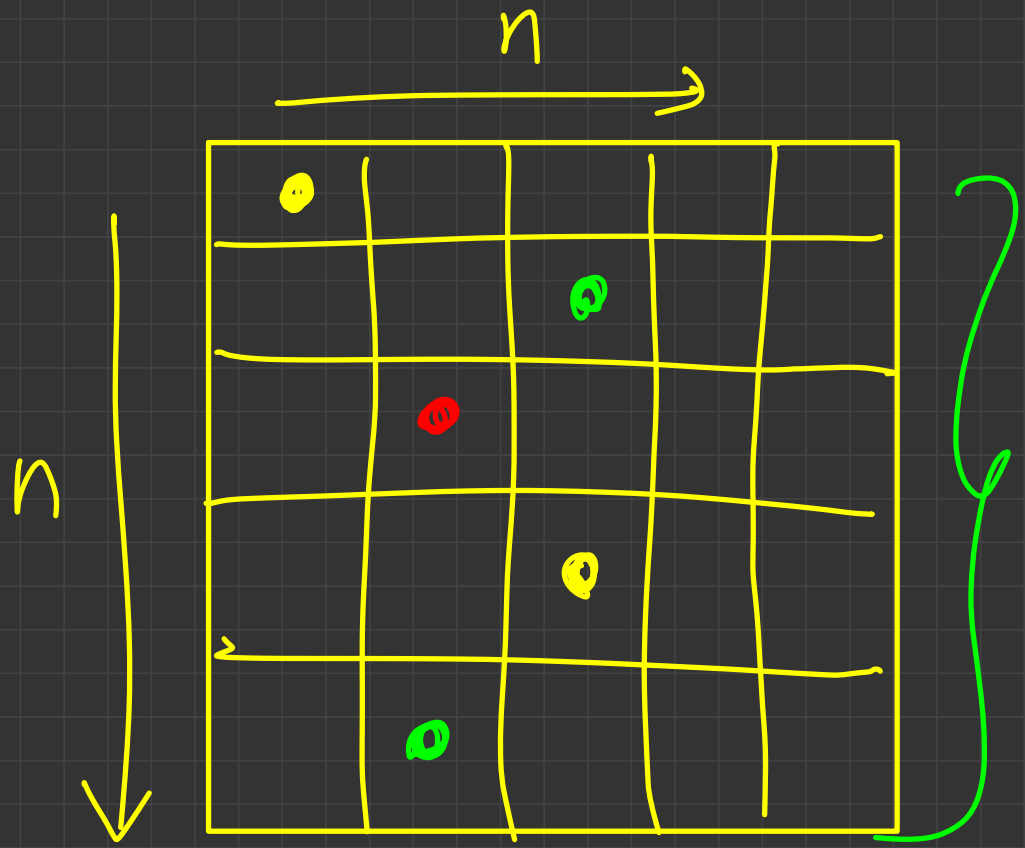
4. Final Subproblem dp[n-1][m-1]

   What is the problem demanding you to find?



Problem 1: XOR Problem using DP

Problem 2: Game Theory using DP



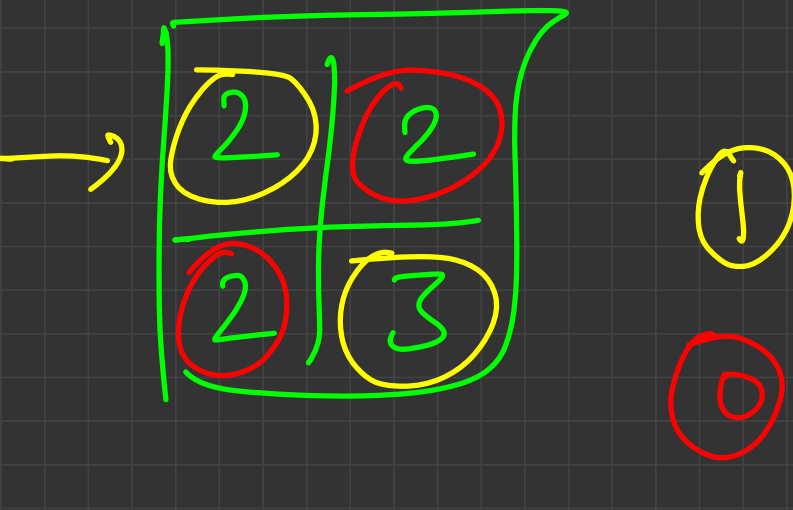
$n \leq 100$

$grid[i][j] \leq 127$

xor > 0

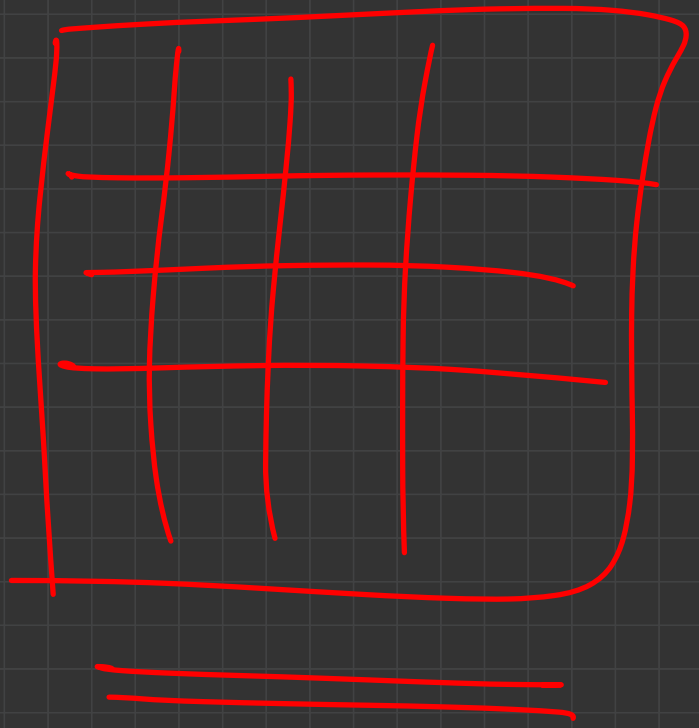
check for possibility

$(n \cdot n \cdot n \cdots n) = n^n$



1

0



all value ≤ 127

(1111111)

(0001111)

(1001001)

max value in grid $(1111111)_{bin} 127$

$a \oplus b$

8th

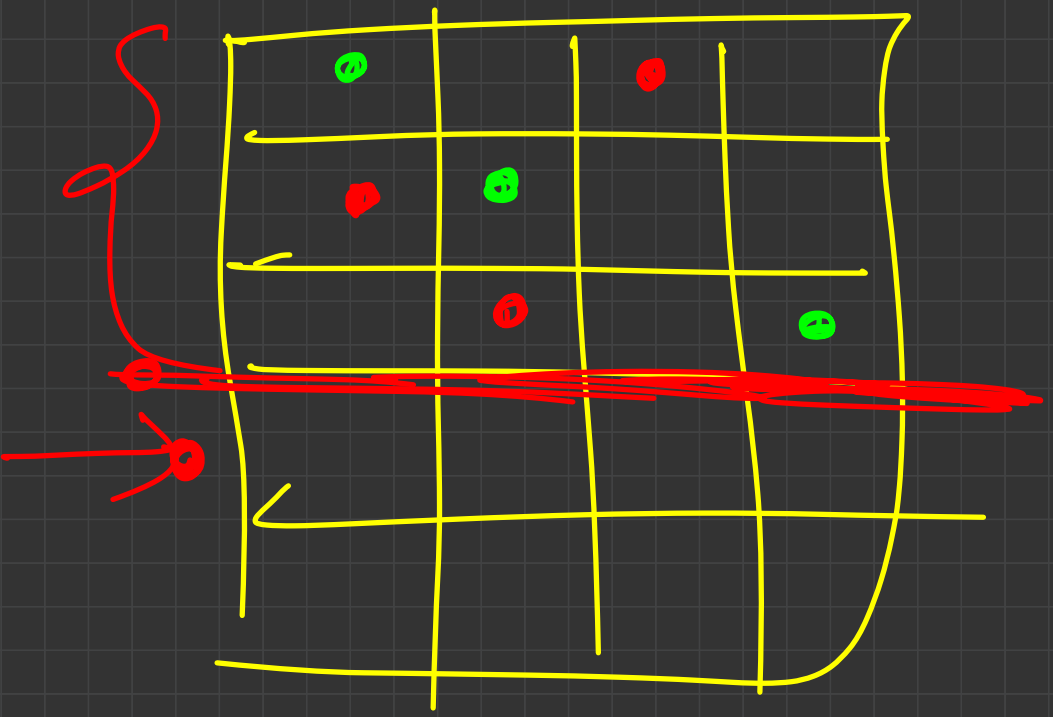
0 1 1 0 1 0 1 0

0 0 0 1 0 1 0 1

1

$a \oplus b \leq 127$

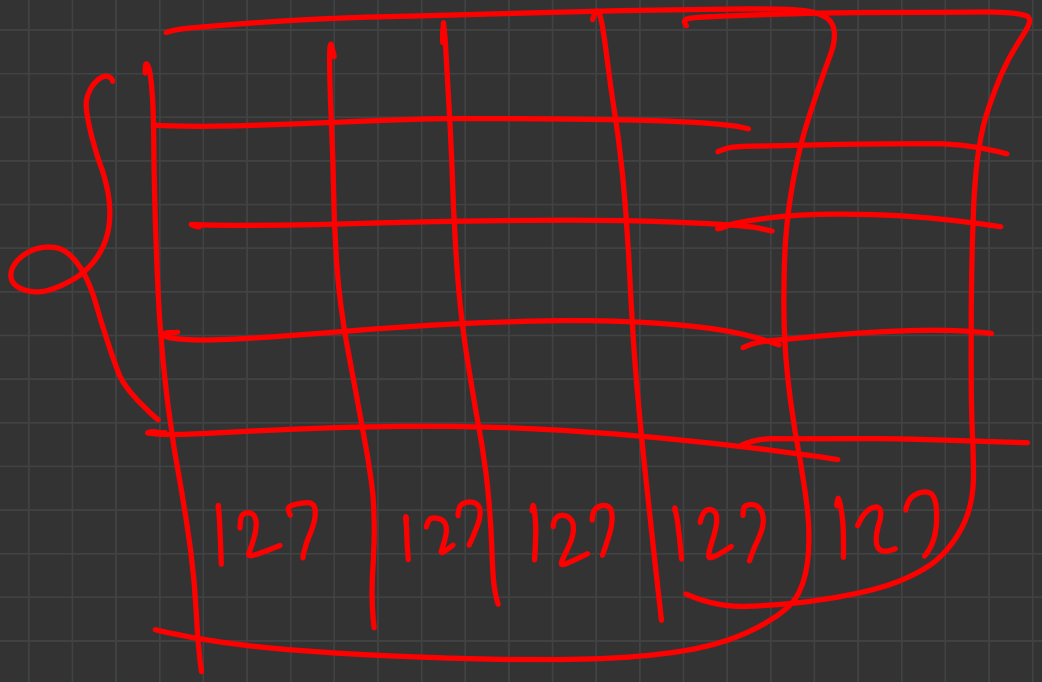
$a \oplus b \oplus c \oplus d \leq 127$



Xor = 19

Xor = 19

ith row, xors for



More Xor

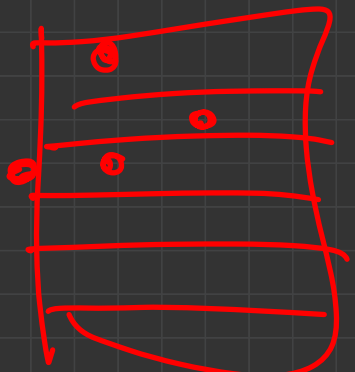
1001000

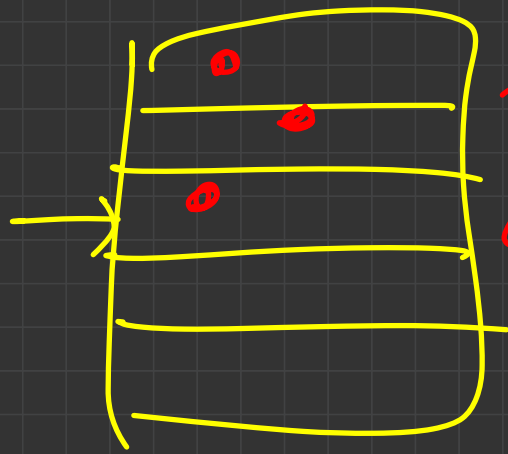
100100

dp(i)(x) = is there a possibility of choosing values from 0th row to ith row s.t. overall Xor = x

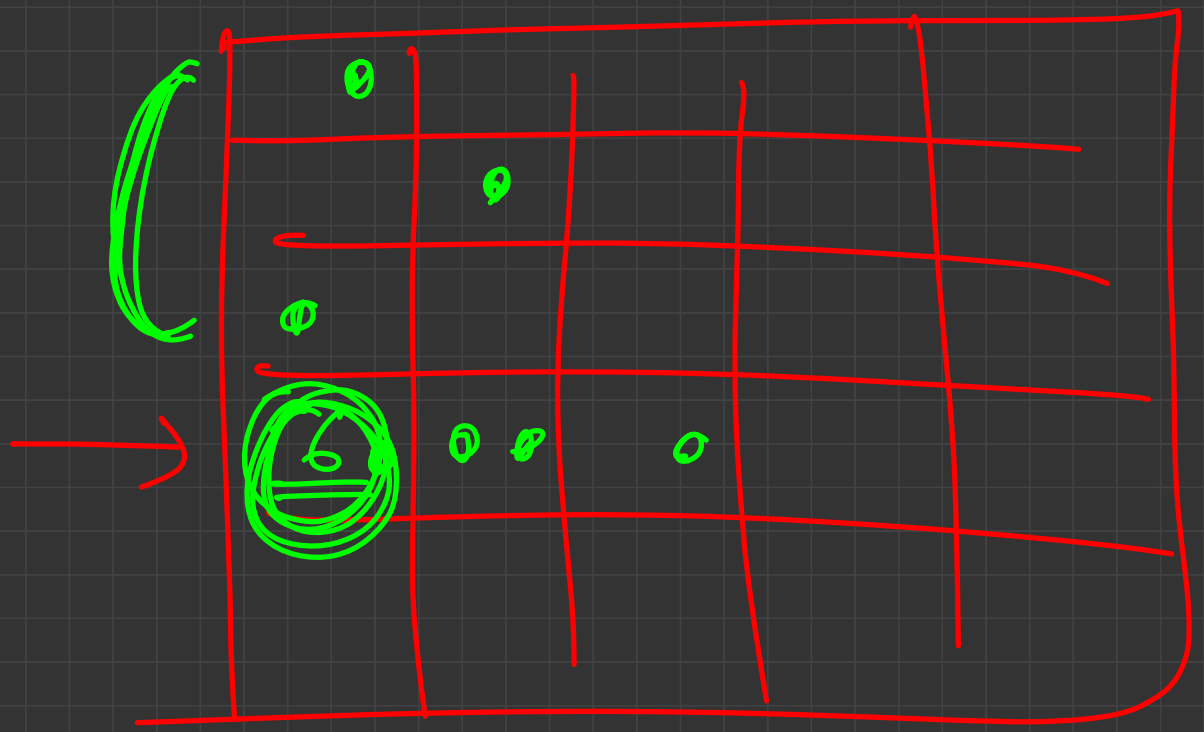
State

Ans





y = x



3rd row

$x = 12$

$dp[i][12] \rightarrow dp[i-1][12 \wedge 6]$

$(12 \wedge 6) \wedge 6 = 12$

$dp[i][x] \longrightarrow dp[i-1][x \wedge grid[i][0]]$

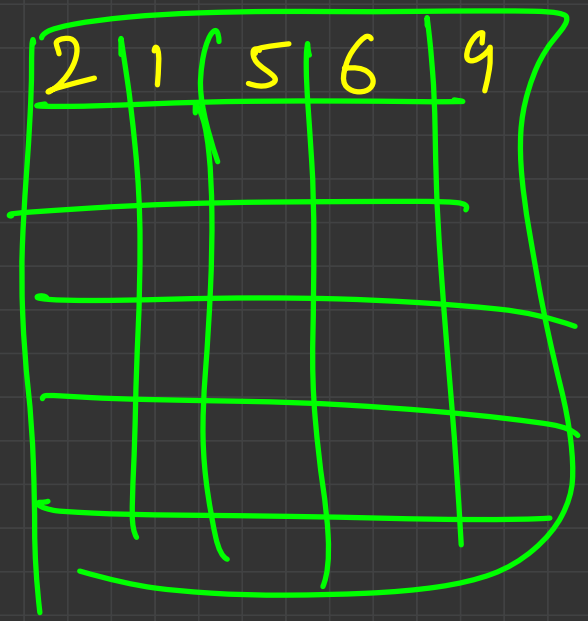
$dp[i-1][x \wedge grid[i][1]]$ ✓

$dp[i-1][x \wedge grid[i][2]]$ ✓

TE

dp[i][x] -> true if any of the following are true

- dp[i][x ^ grid[i][0]], dp[i-1][x ^ grid[i][1]]
- ... ... - dp[i-1][x ^ grid[i][n-1]]



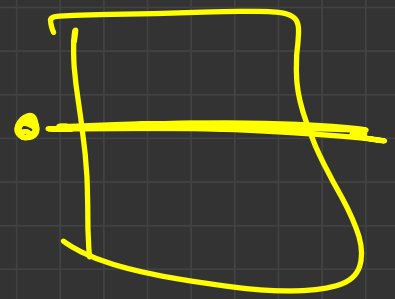
rest all will
so false

$O(C$

dp[0][2] → true
dp[0][5] → true
dp[0][1] → true
dp[0][6] → true
dp[1][9] → true

dp[n-1][1] ✓ dp[n-1][2] ✓ F.S

. . . dp[n-1][127] ✓

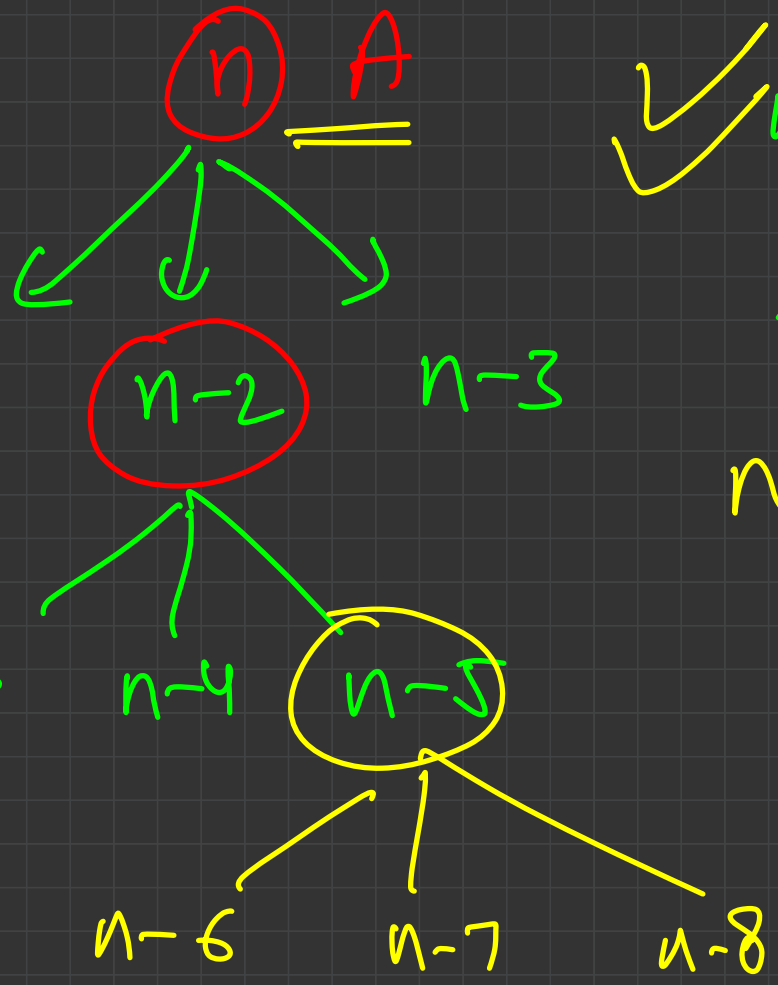


Time Complexity

⟶

#states ⟶ (n × 127) T.T ≈ O(n)

$n \geq 5$

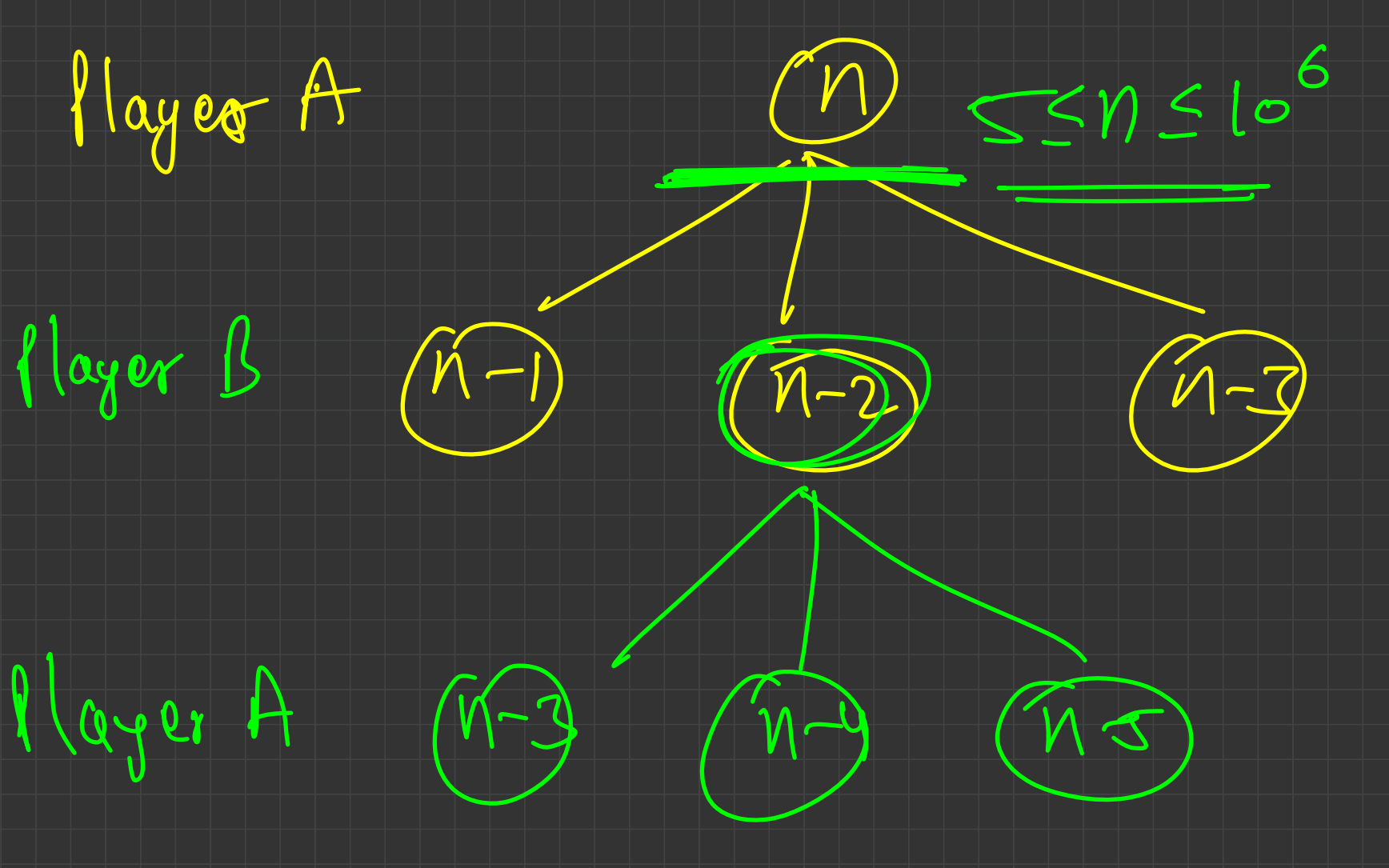


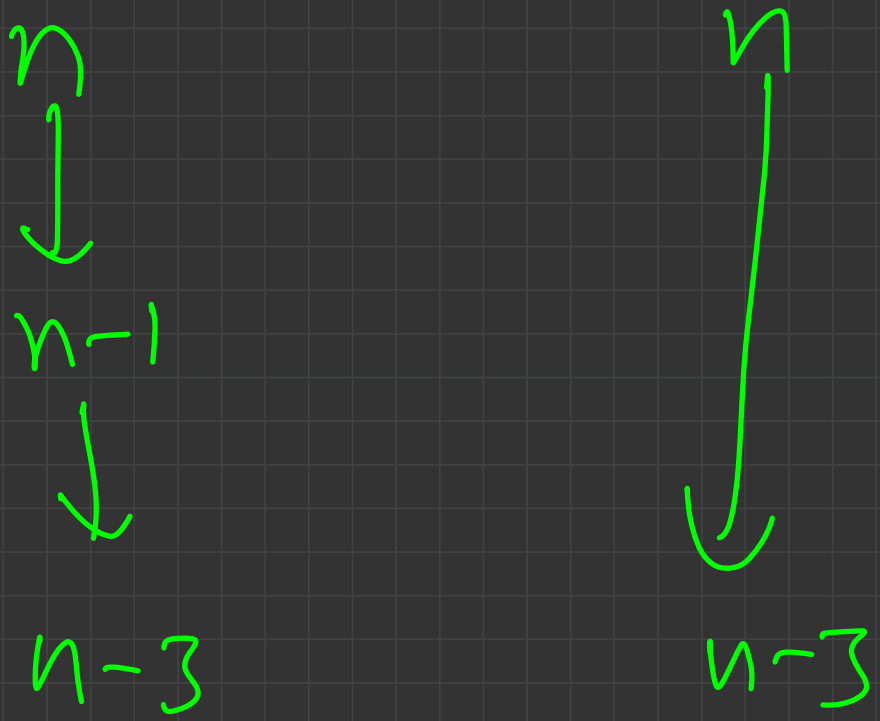
A Player

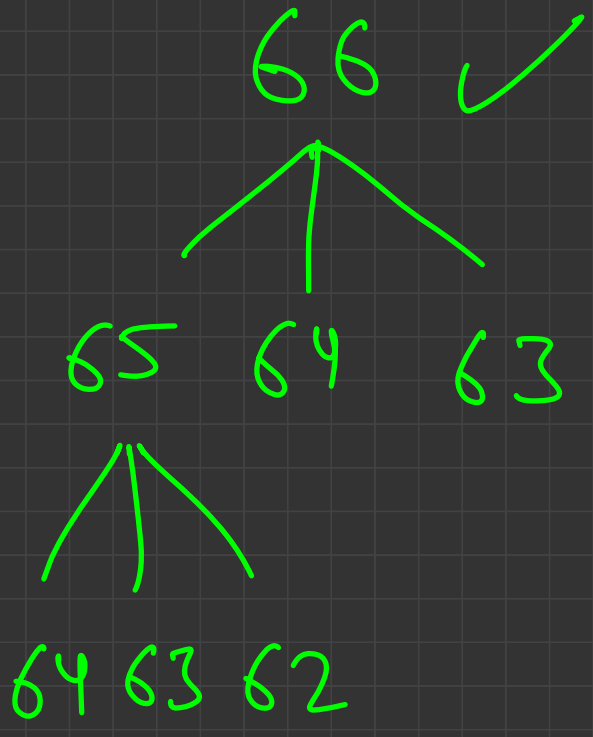
B Player

$n = 9$

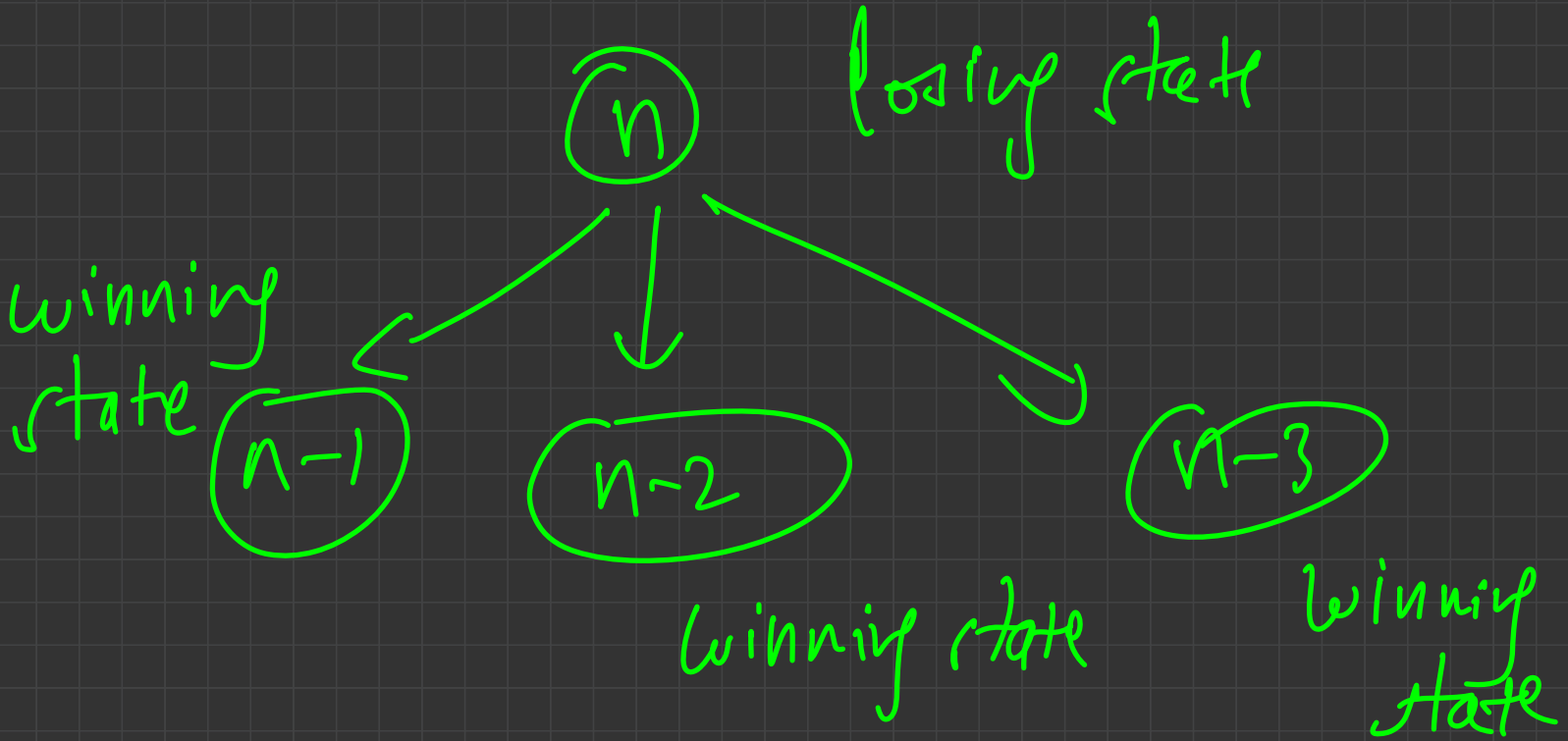
7


Player A
n
$1 \le n \le 10^6$
Player B
n-1
n-2
n-3
Player A
n-3
n-4
n-5


n
n-1
n-3
n
n-3



n = losing state if either n is a prime number or all of n-1, n-2, n-3 are winning states


n
losing state
winning state
n-1
n-2
n-3
winning state
winning state

n -> losing state if n

= prime

or all the states

n-1, n-2, n-3

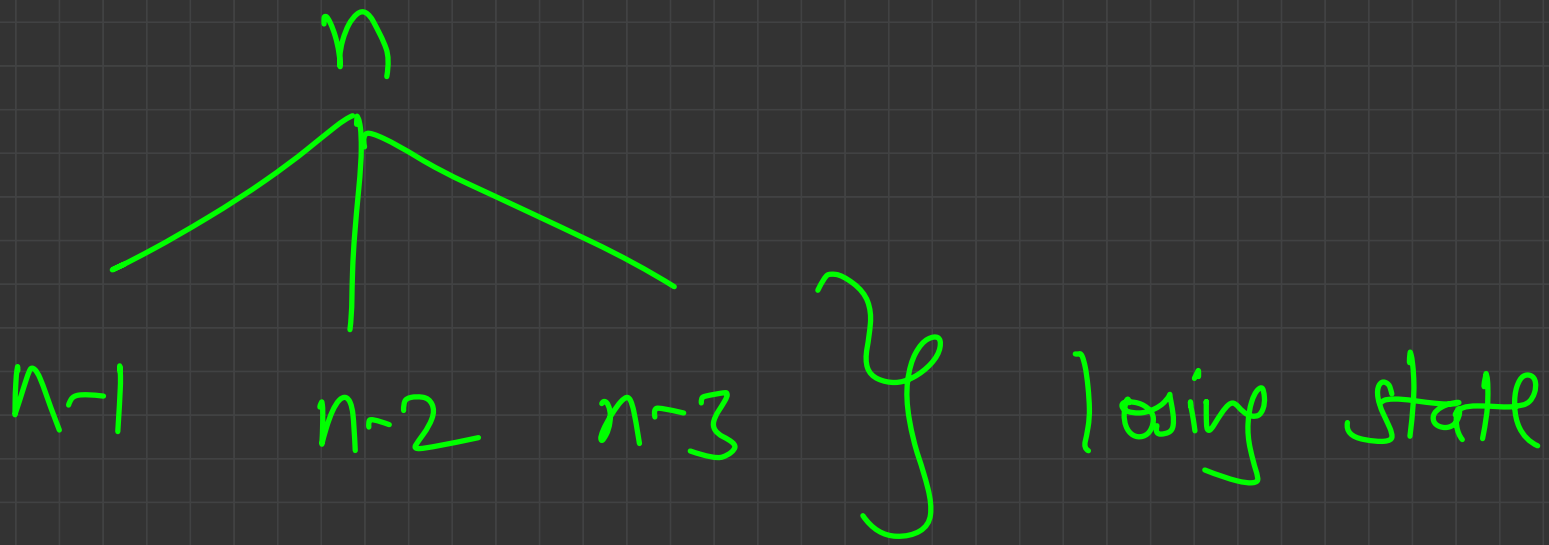
are winning

n -> winning state

if n is not a prime and

either of n-1, n-2

or n-3 is losing


n
n-1
n-2
n-3
losing state

winning

dp(n) →  → (1)

→ losing → (0)

dp(n) = ! prime(n) && e

! (dp(n-1) && dp(n-2) && dp(n-3))