Dynamic Programming 3

- Priyansh Agarwal

Removal game (CSES)

State

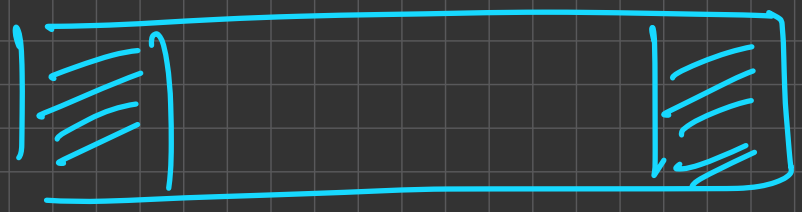
$dp[i][j]$ = max sum the current player can get from subarray (i to j)

Transition

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

B.C = $dp[i][i] = arr[i]$, F.S = $dp[0][n-1]$

$dp[i][j]$



↳ (max score that 1st player can get from (i to j))

$dp[0][n-1]$

$$\begin{cases} arr[0] + dp[1][n-1] \\ \max \\ arr[n-1] + dp[0][n-2] \end{cases}$$

$dp[1][n-1]$

$$\begin{cases} arr[1] + dp[2][n-1] \\ \min \\ arr[n-1] + dp[1][n-2] \end{cases}$$

(0,5) → 1st player

length of array = 6

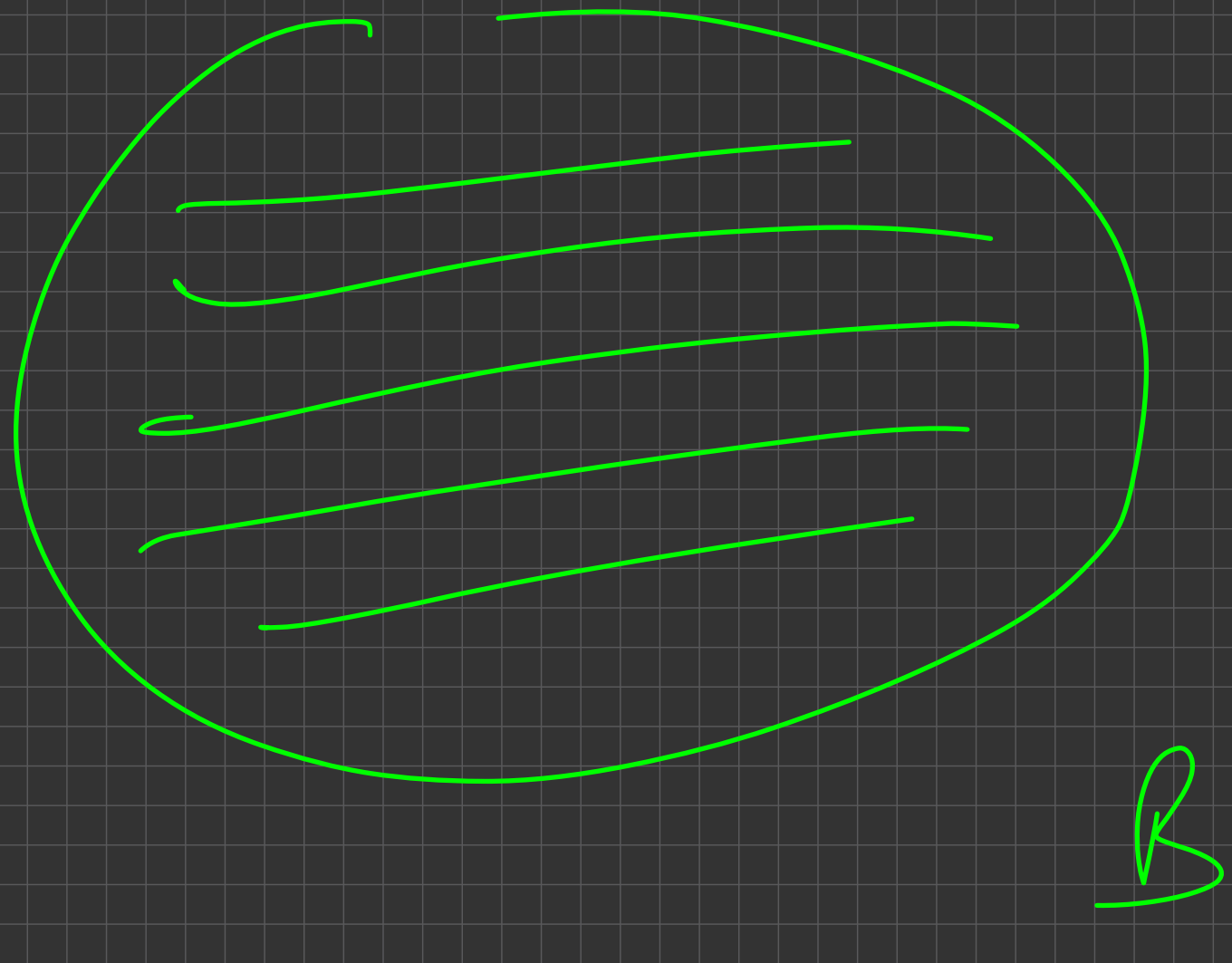
even — 1st player

odd — 2nd player

length of array = 9

odd — 1st player

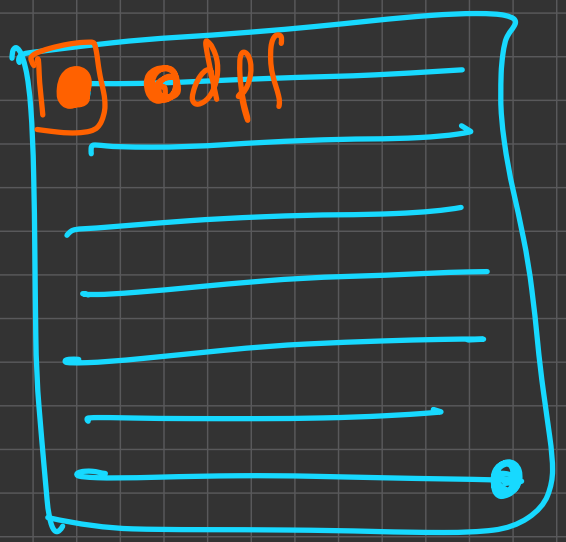
even — 2nd player


A
B

- Answer Construction ✓
- 1 hard DP problem ✓
- Space optimisation ✓
- State optimisation
- Transition optimisation ✓

Answer Construction

- Grid problem: Find the actual path with the minimum sum.
- [redacted]
- At every state we are making some optimal choice.
  - If we store this choice, we can be sure that if we are at any state we know what is the best choice.
  - Start from the state that contains your final subproblem and keep making the best choice (which was already stored) until you reach the end.



minimize sum of values on the path

find out the exact path

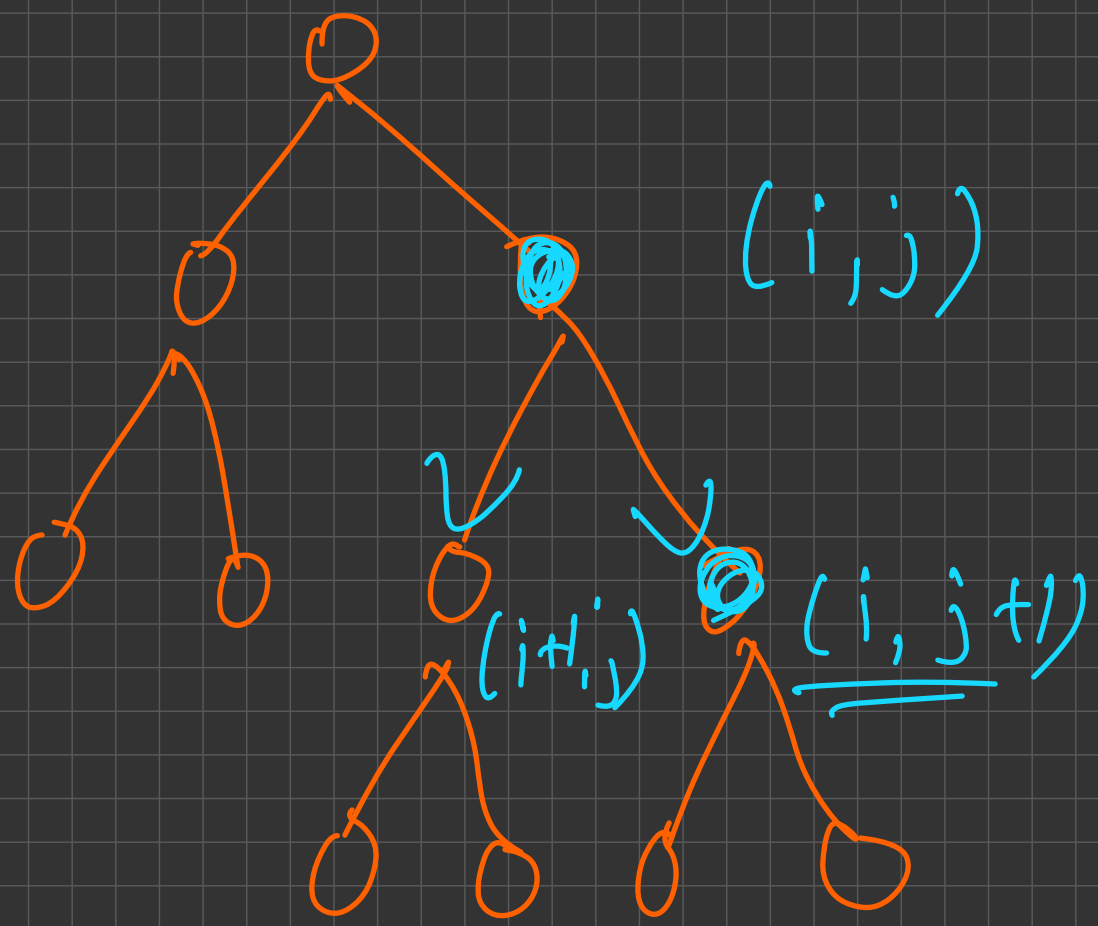
$dp[i][j]$ = min possible sum path from (i,j) to $(n-1, m-1)$

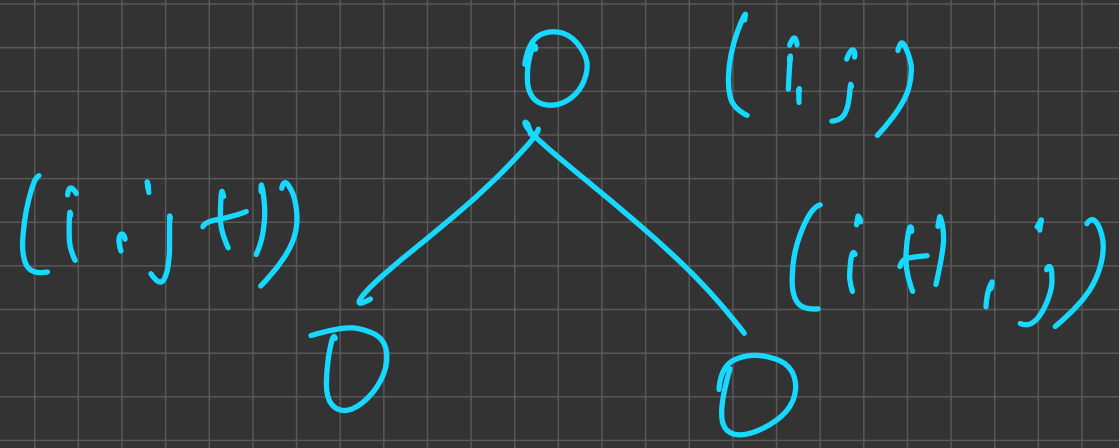
$dp[0][1]$
$dp[1][0]$

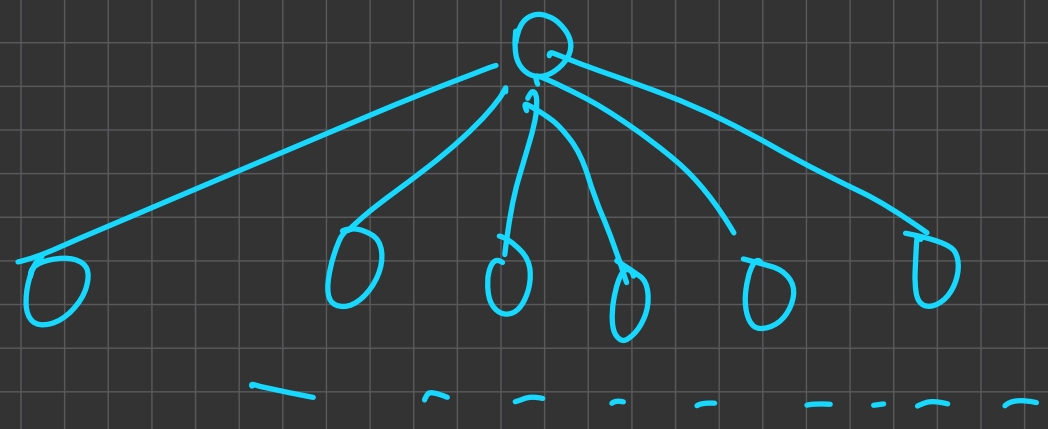
$(0,0)$

(i,j) — $dp[i+1][j]$

$dp[i][j+1]$


(i, j)
(i+1, j)
(i, j+1)

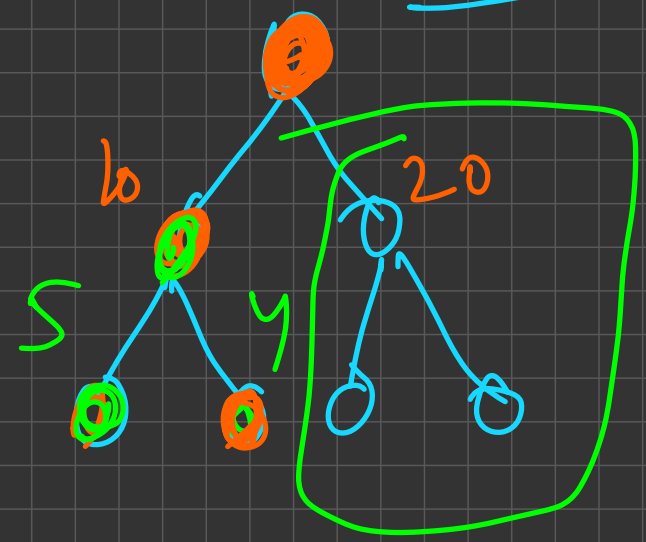

(i, j)
(i, j+1)
(i+1, j)

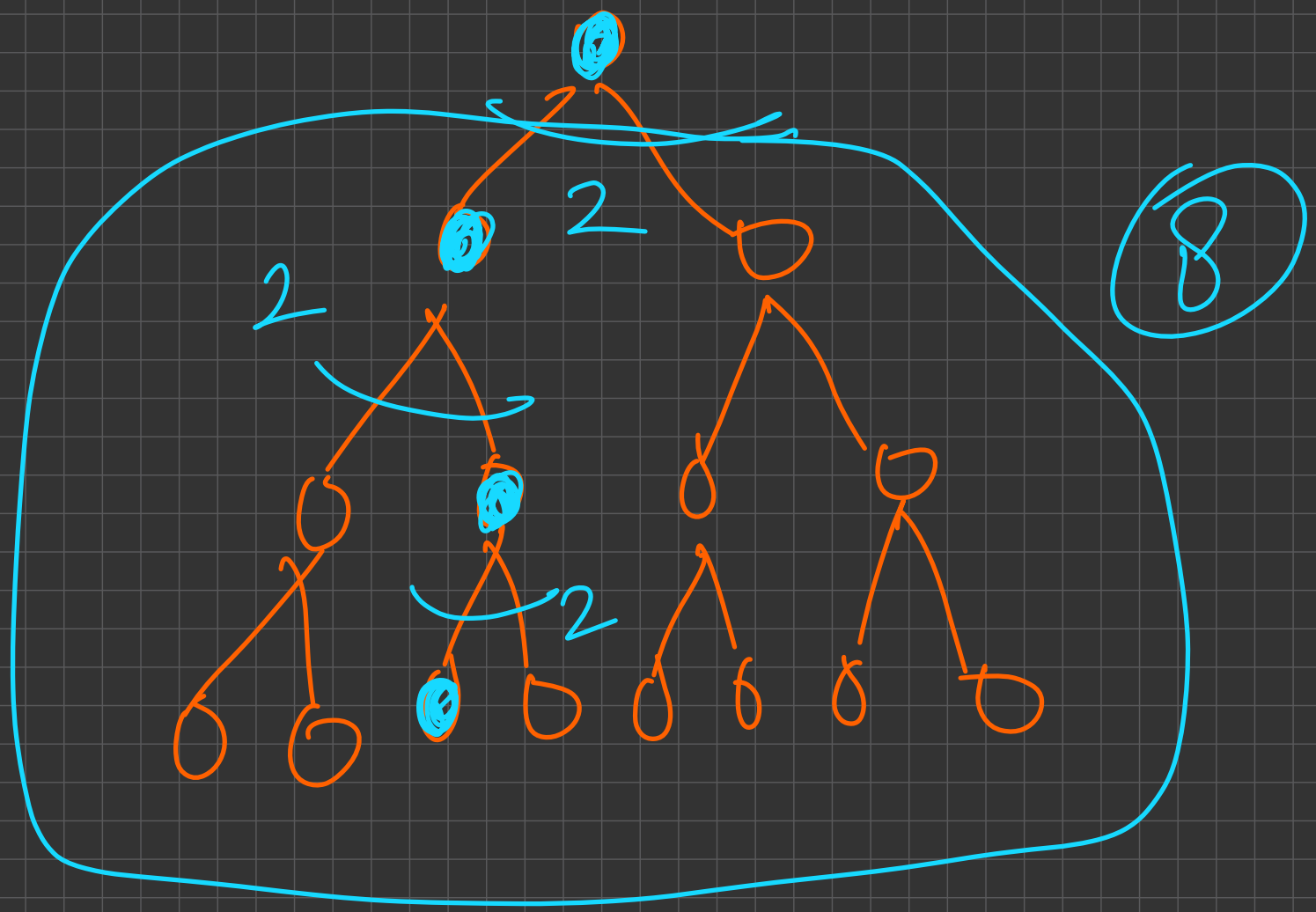


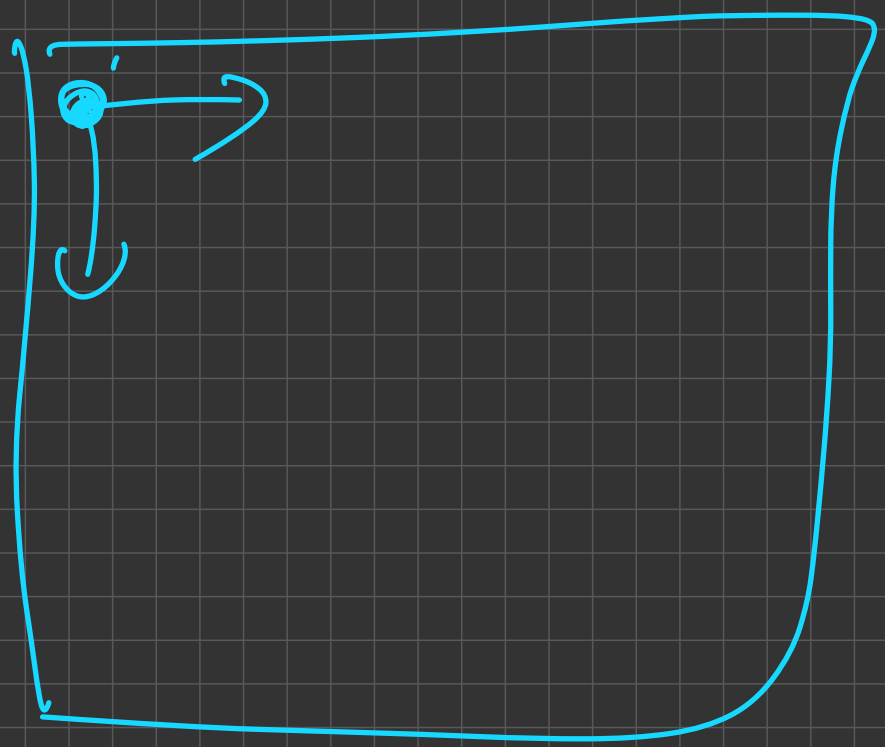
10 choices

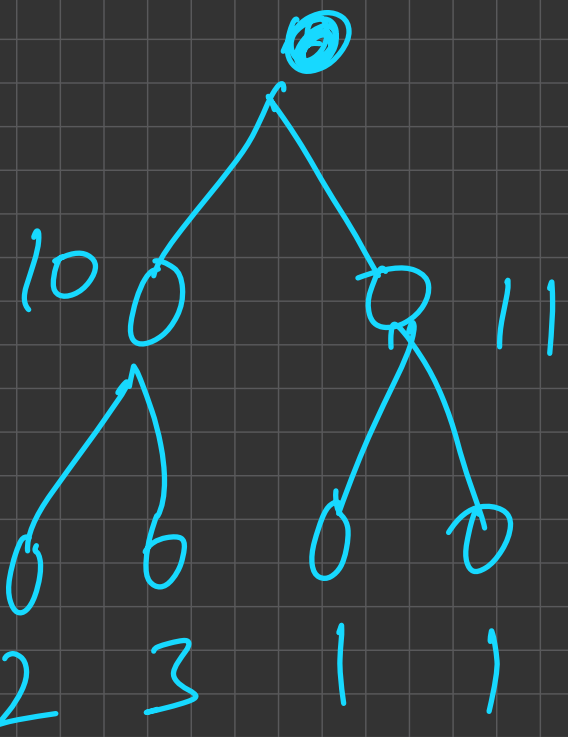
Grid problem

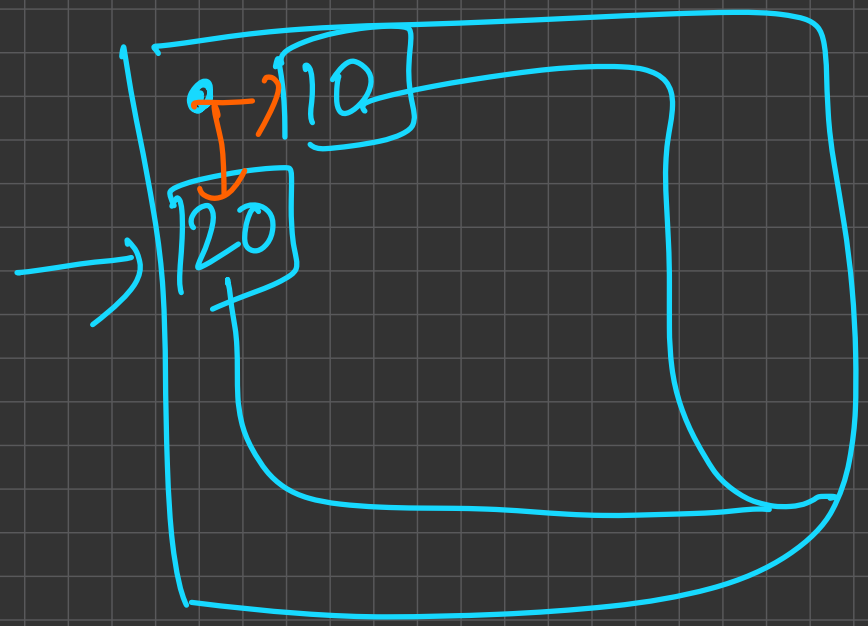
100 cells




2
2
2
8




10
11
2
3
1
1

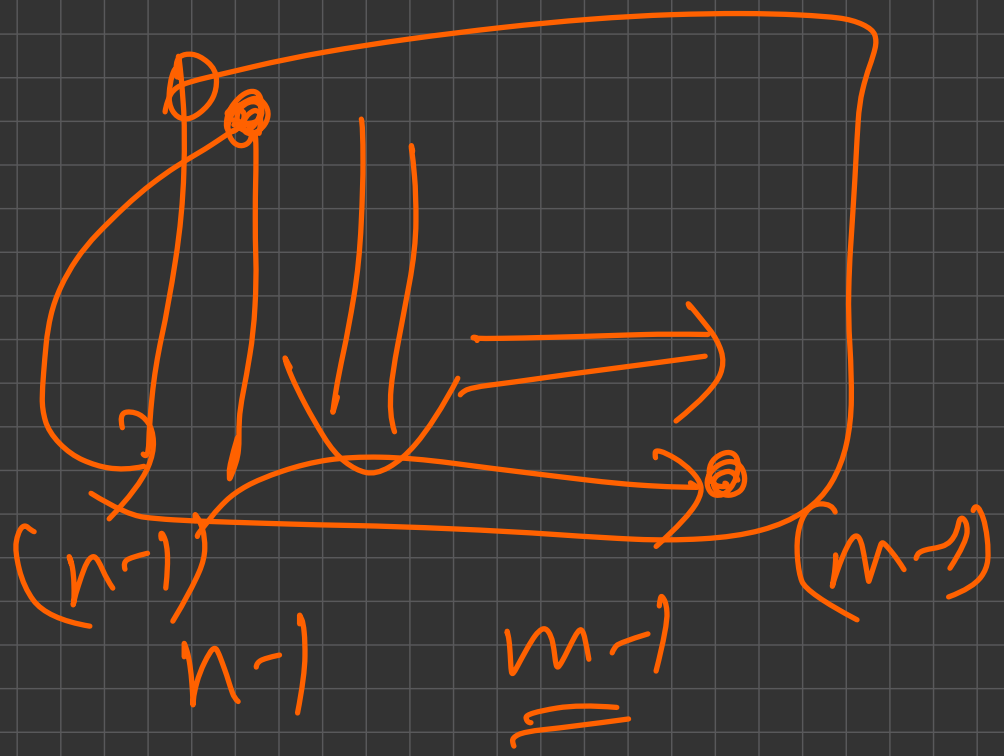


$\underline{\underline{10}}$

2^{10}

$\underline{\underline{2 \cdot 10}}$

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



$$(n+m-2)$$

$$(n+m-1) \cdot 2$$

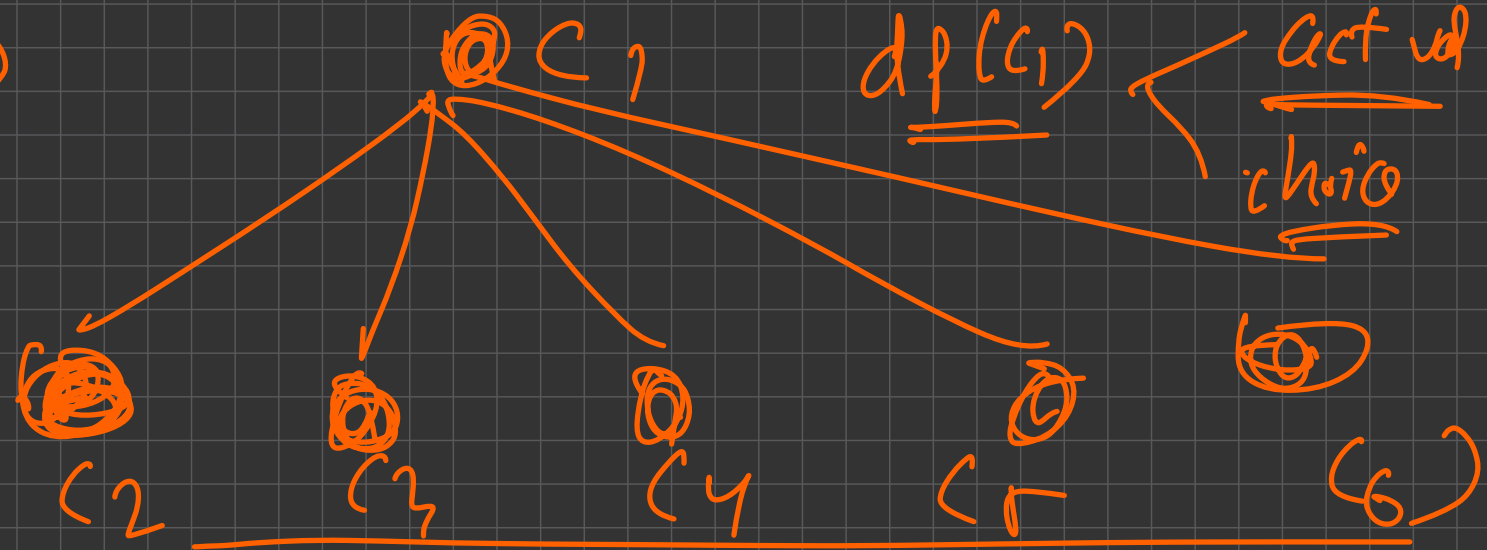
$$O(2 \cdot (n+m-1)) = O(n+m-1)$$

$dp(c_1)$

c_2

c_3

c_6

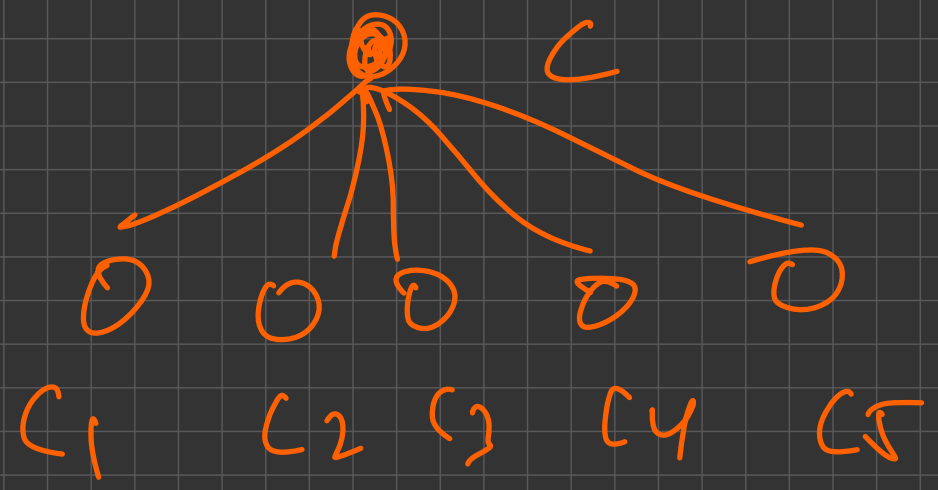


$dp(c_1)$ ← actual choice

k possibilities

p cells	k possibilities

$O(p \cdot k)$



$$dp(c) = v + \underbrace{\min}_{\text{actual}} \left\{ \begin{matrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ \cancel{c_5} \end{matrix} \right\}$$

$dp(c)$ — choice c_4

$\boxed{l}$

$l \cdot k$

$O(l)$

Answer Construction - Grid Problem

```
int n = 3, m = 3;
vector<vector<int>> grid(3, vector<int>(3));
vector<vector<pair<int, int>>> dp(n, vector<pair<int, int>>(m, {-1, 0}));
// 0 -> take a down direction
// 1 -> take a right direction
int f(int i, int j){
    if(i == n || j == m)
        return 1e9;
    if(i == n - 1 && j == m - 1)
        return grid[n - 1][m - 1];
    if(dp[i][j].first != -1)
        return dp[i][j].first;

    int ans1 = f(i + 1, j);
    int ans2 = f(i, j + 1);
    if(ans1 < ans2){
        dp[i][j].second = 0;
    }else{
        dp[i][j].second = 1;
    }
    return dp[i][j].first = grid[i][j] + min(ans1, ans2);
}
```

cost val , choice

dp[i][j].first -> val

dp[i][j].sec -> choice

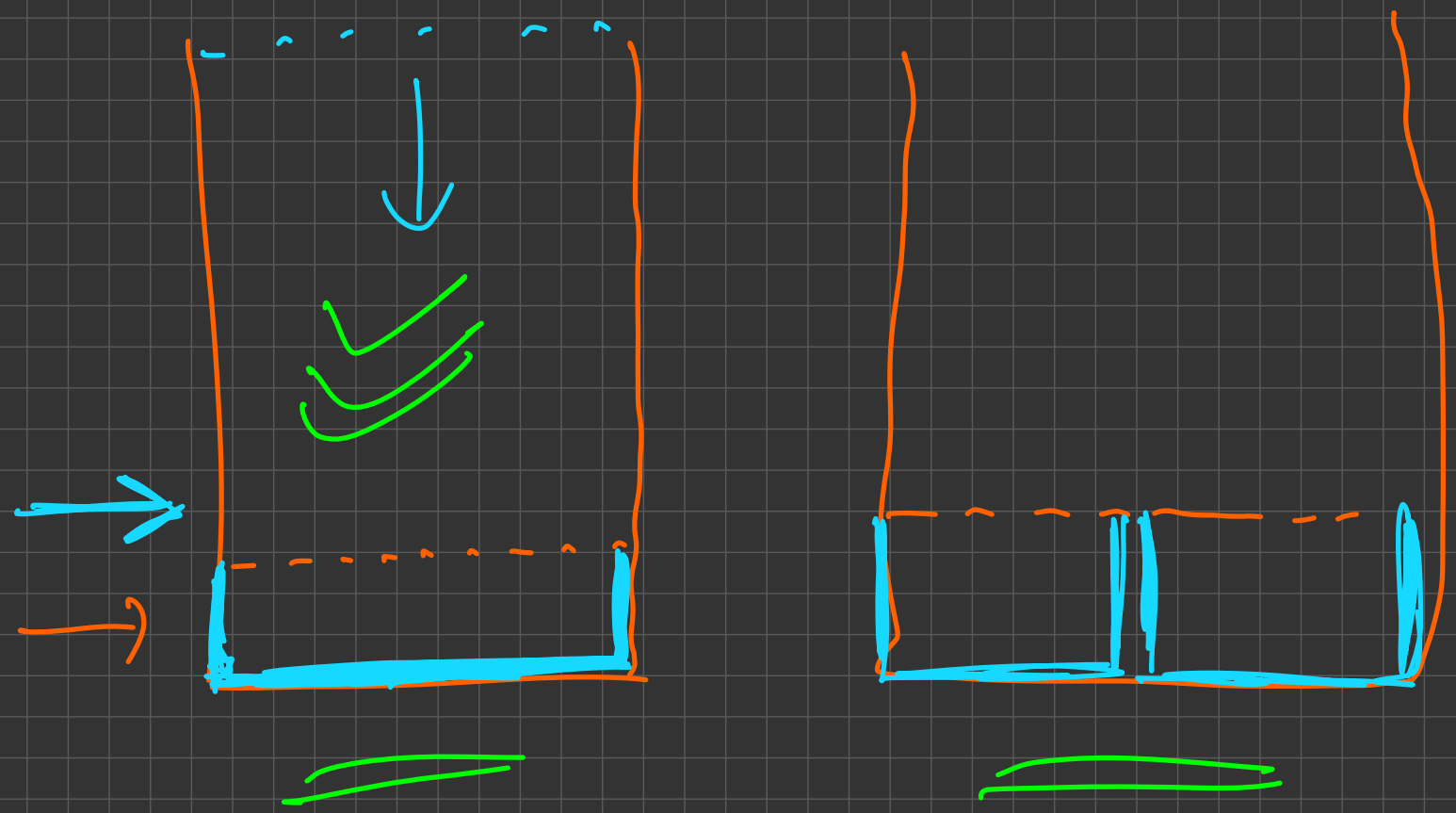
Answer Construction - Grid Problem

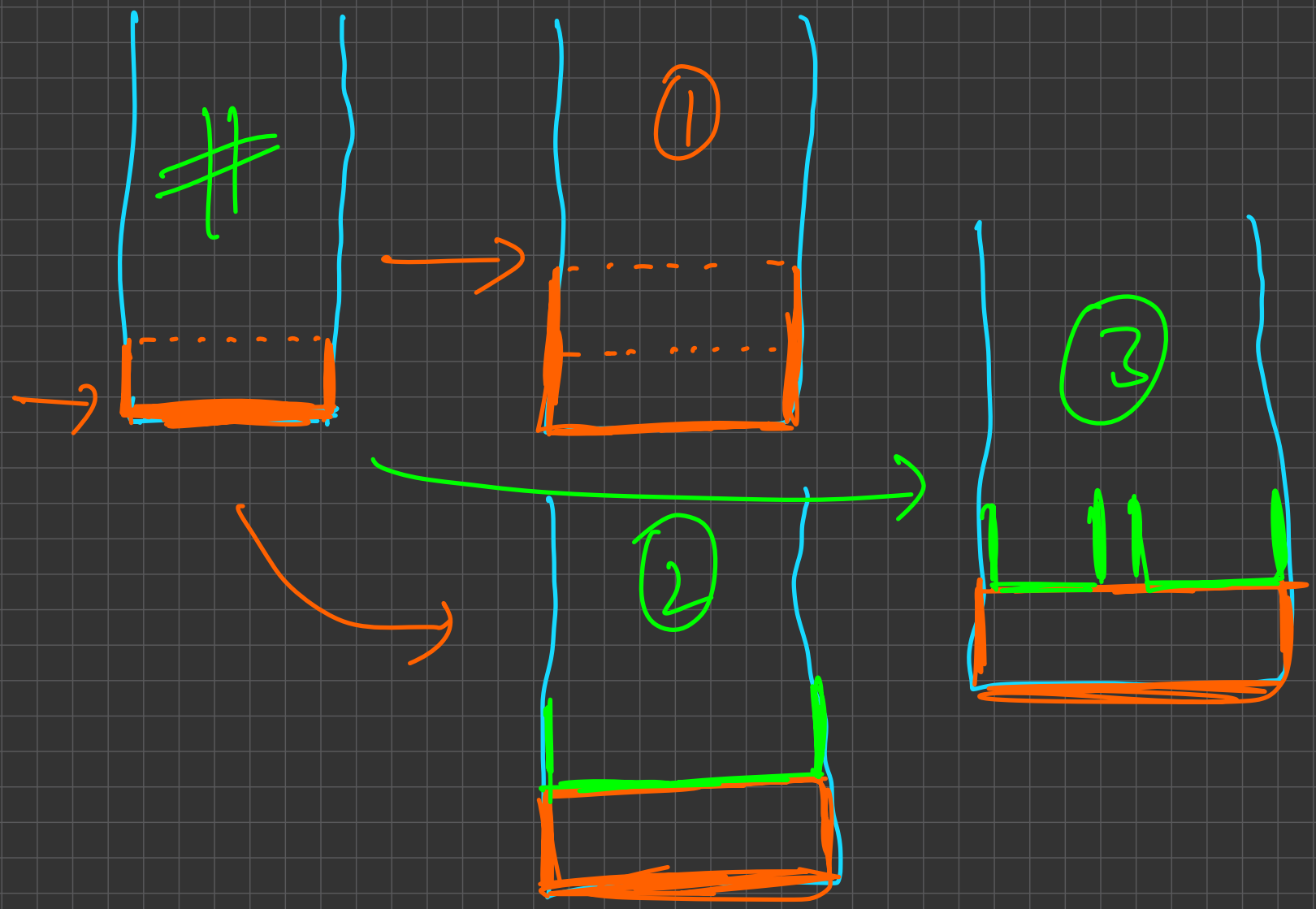
```
void solve(){
    for(int i = 0; i < 3; i++){
        for(int j = 0; j < 3; j++){
            cin >> grid[i][j];
        }
    }
    cout << f(0, 0) << nline;
    pair<int, int> current = {0, 0};
    while(current != mp(n - 1, m - 1)){
        cout << current.first << " " << current.second << nline;
        if(dp[current.first][current.second].second == 0)
            current.first++;
        else
            current.second++;
    }
    cout << current.first << " " << current.second << nline;
}
```

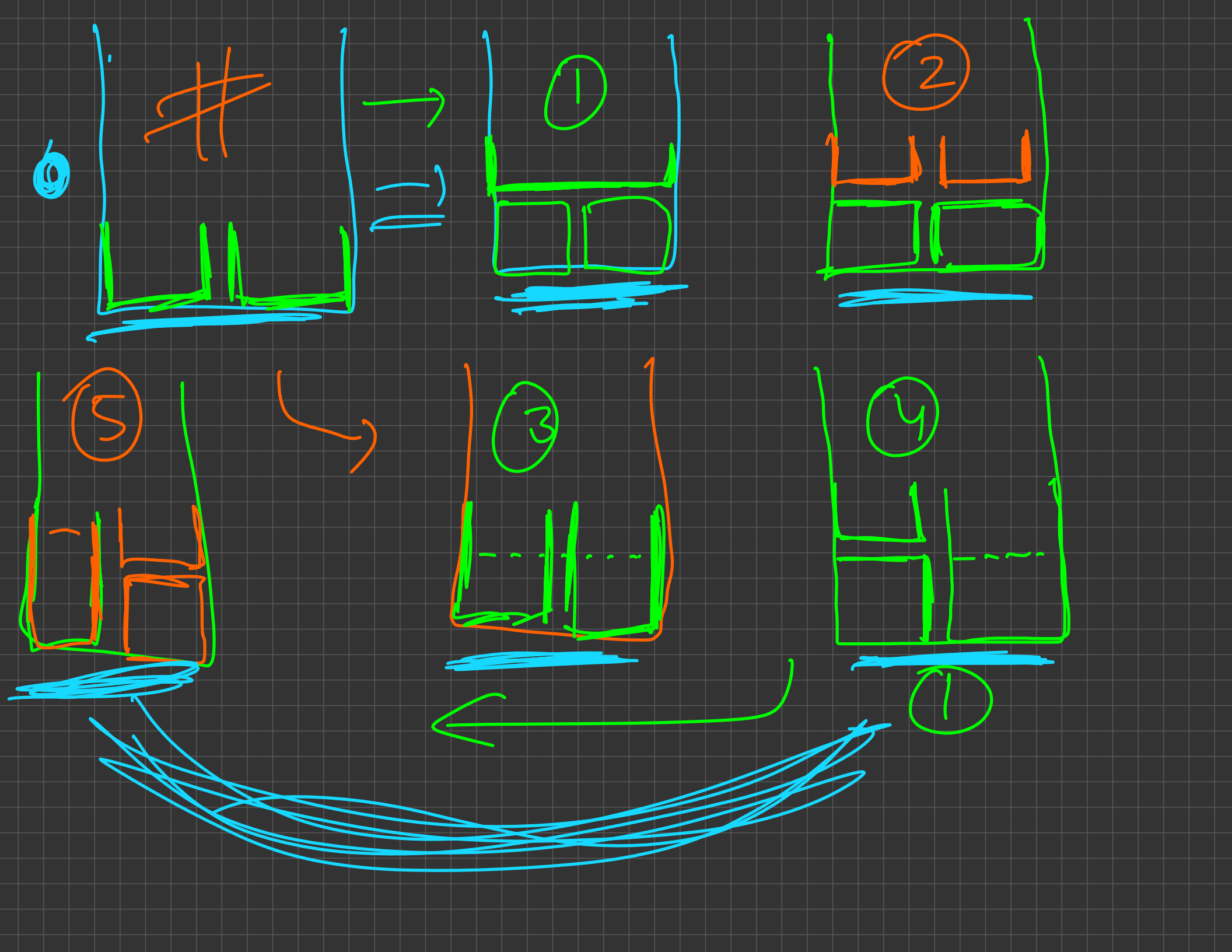
evaluate

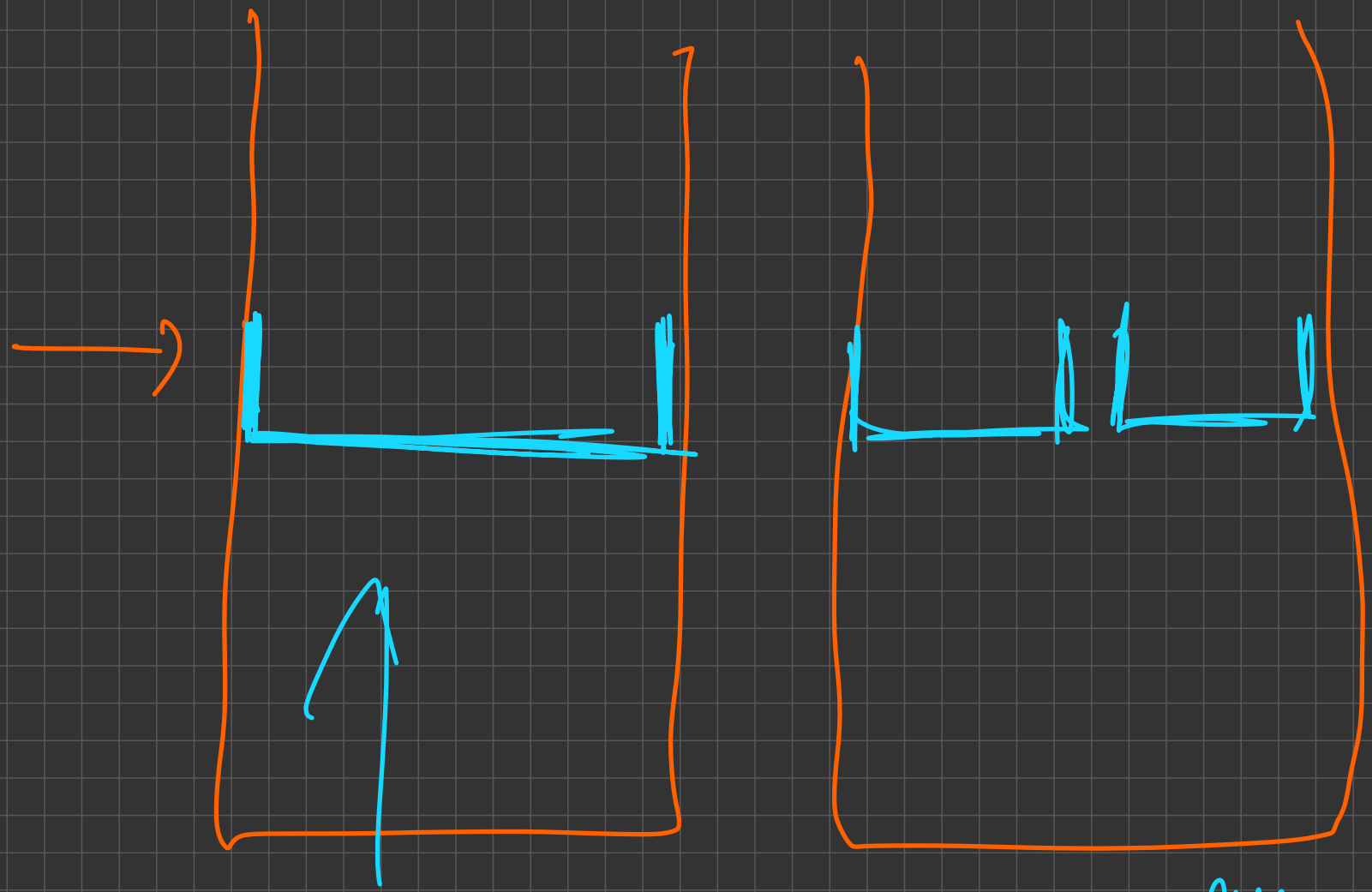
backtrack

Problem 1: Link




#
1
2
3

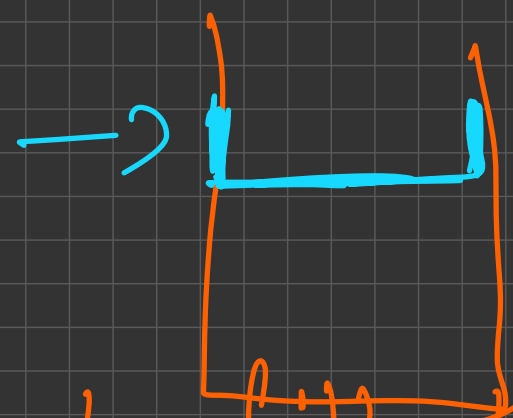

1
2
3
4
5



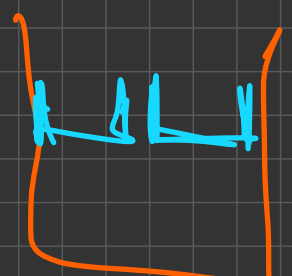
$dp[i][1]$

$dp[i][0]$ = no. of way to fill up the tower from $(i+1)$th index such that you have a 1×2 cell extending

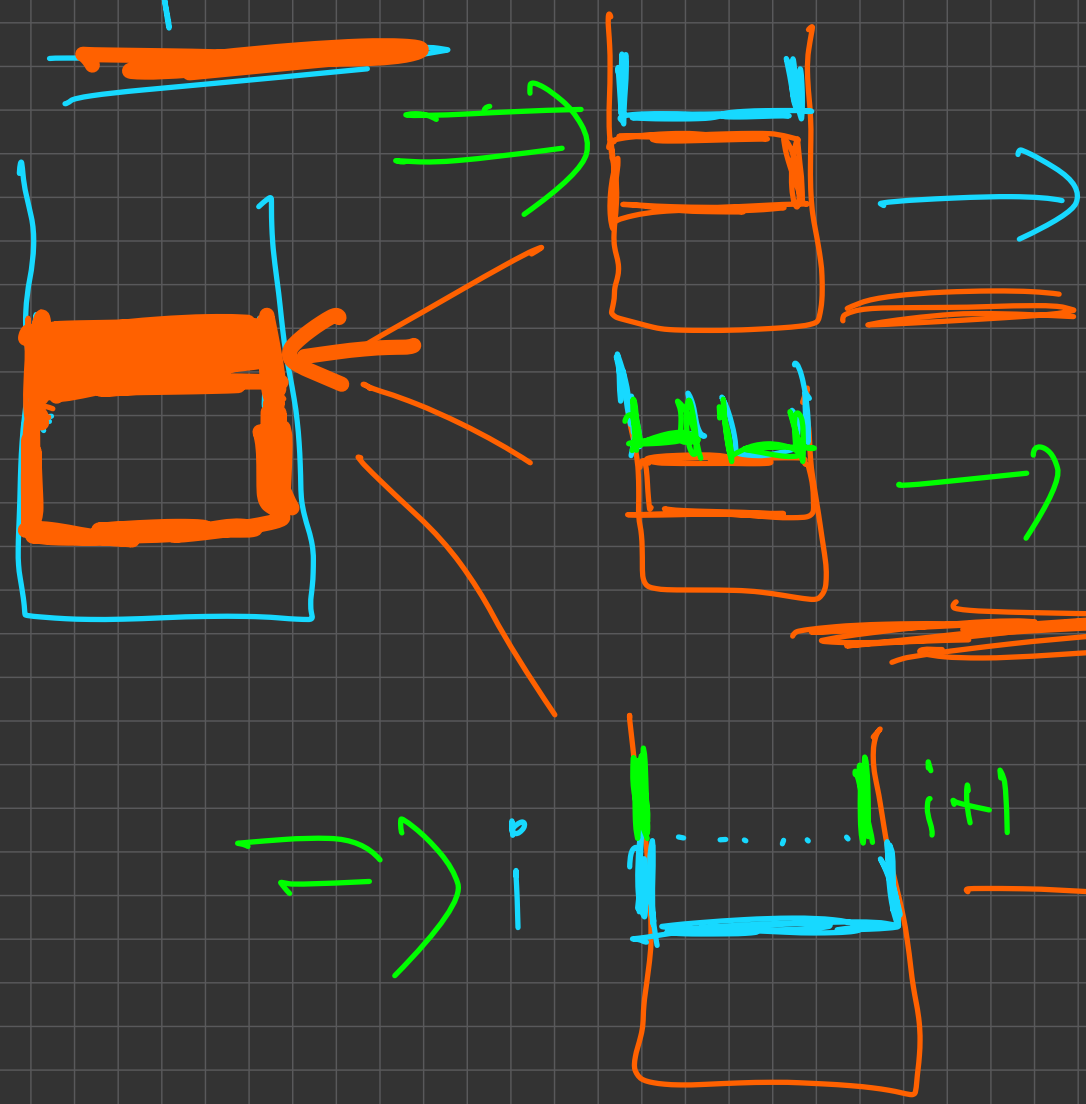
$dp[i][0]$ = no of ways to fill the tower from $(i+1)$th row to the top set you have a 1×2 cell empty



$dp[i][1]$ = no of ways to fill the tower from $(i+1)$th row to the top set you have 2 (1×1) cells empty



$dp[i][0]$



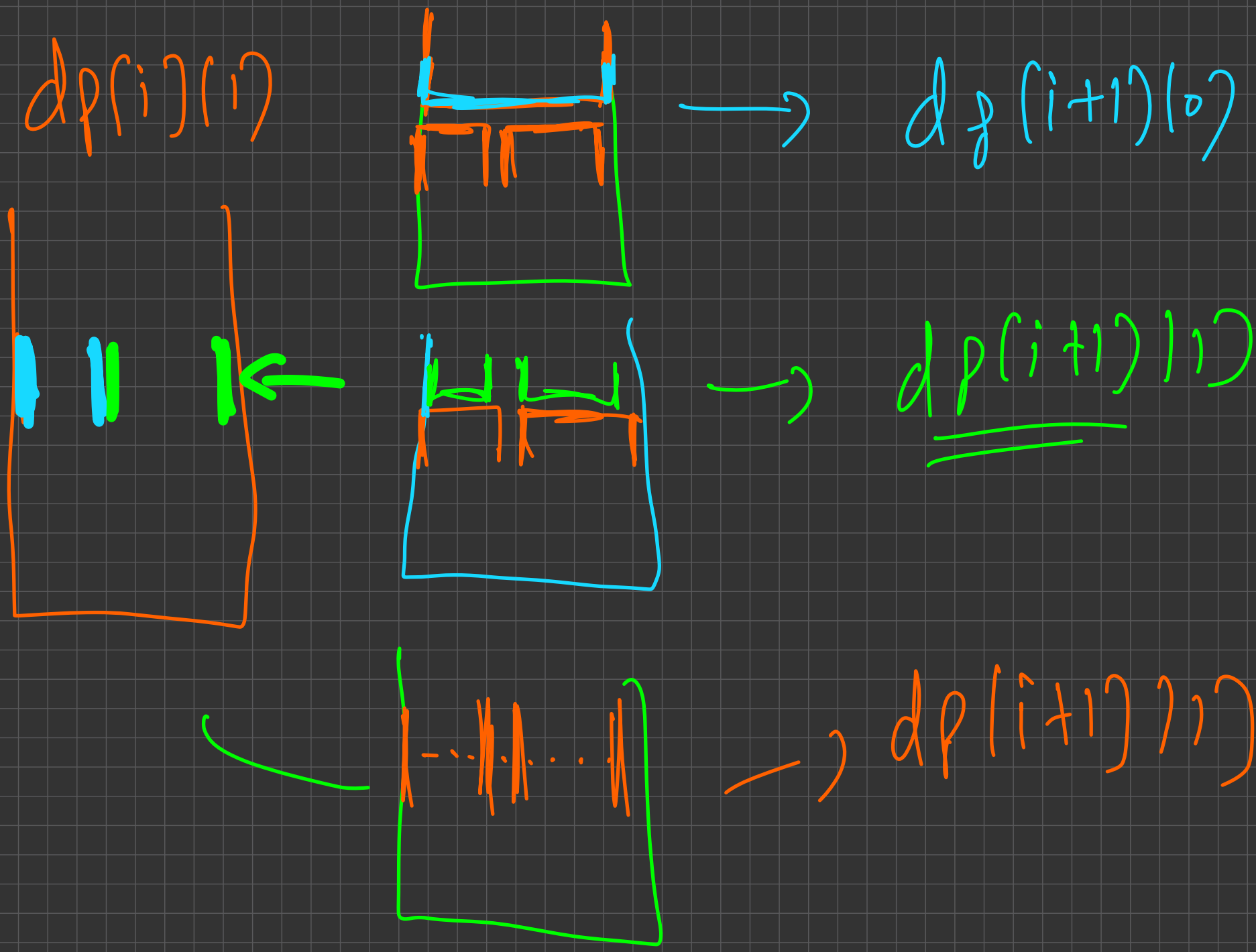
$\rightarrow dp[i+1][0]$

$\rightarrow dp[i+1][1]$

$\rightarrow dp[i+1][0]$

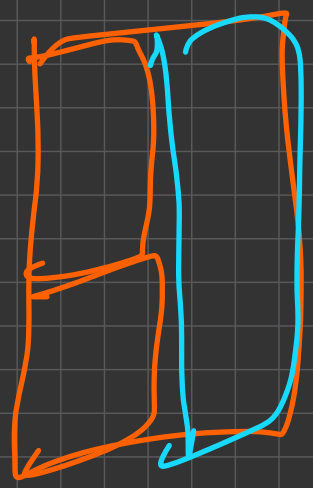
$$dp[i][0] = 2\,dp[i+1][0] + dp[i+1][1]$$

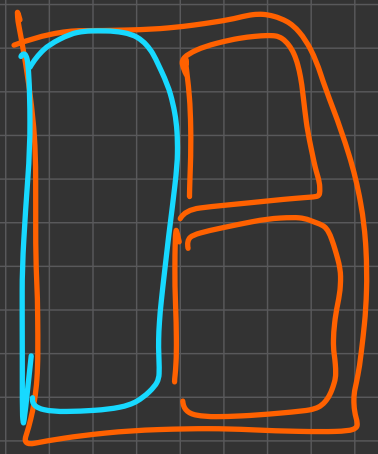
$dp[i][1]$



$\rightarrow dp(i+1)(1)$

$= dp(i+1)(1)$





$$dp[i][1] = dp[i+1][0]$$

$$+ 4 \cdot \underline{\underline{dp[i+1][1]}}$$

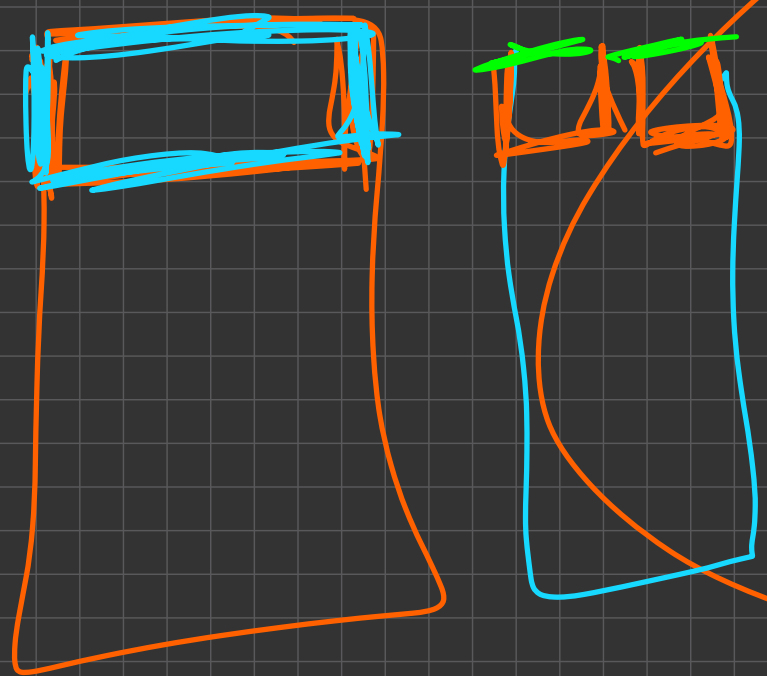
$\underline{\underline{O(1)}}$ $\underline{\underline{S.C}}$

$O(\underline{\underline{n}})$ $\overline{T.C}$

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

$$dp[i][0] = 2 \cdot dp[i+1][0] + dp[i+1][1]$$

$$dp[i][1] = 4 \cdot dp[i+1][1] + dp[i+1][0]$$



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

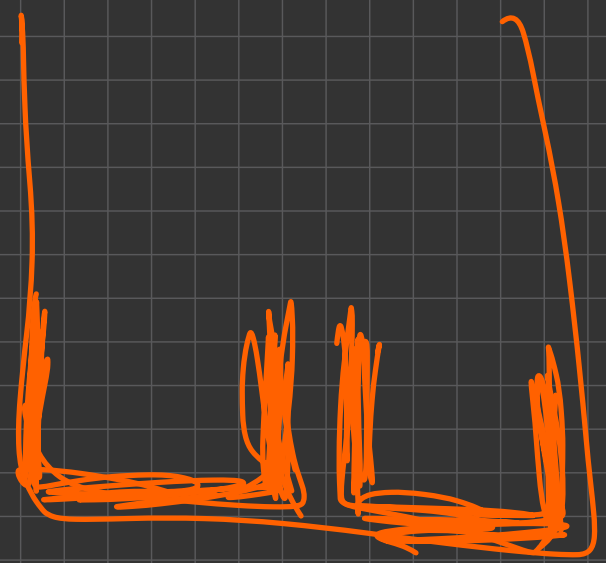
$$dp[n-1][1] = 1$$

$\sqrt{5}$ ' C

$$\left(\begin{array}{c} dp(i)(0) \\ \\ dp(i)(1) \end{array}\right) = \left\{\begin{array}{l} dp(i+1)(i) \\ dp(i+1)(0) \end{array}\right.$$

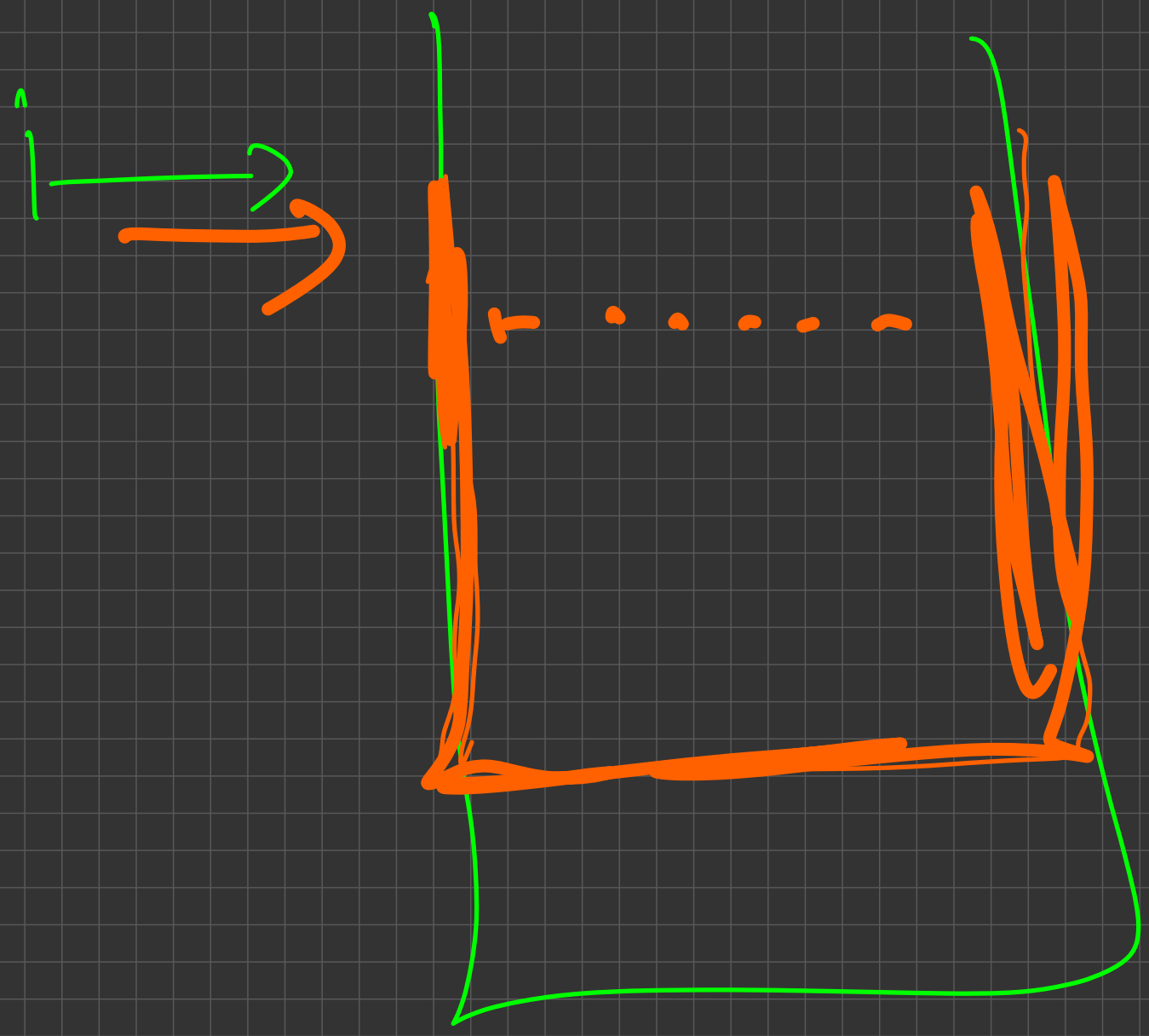
1×2

$df(0)(0)$



$df(0)(1)$

$$f \cdot s = df(0)(0) + df(0)(1)$$



n^2

Space Optimization

- What other state does our current state depend on?
- Do we need answers to all subproblems at all times?
- Well, let’s store only relevant states then!
- But wait, does this always work?
  - What if the final subproblem requires all the states?
  - What if we need to backtrack? [more on this in later lectures]

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

$$fib(n-3)$$

$O(n^2)$ $n \le 10000$

state Δ $O(r)$

Trie

10^8 10^7 spare

256 MB

256 MB

512 MB

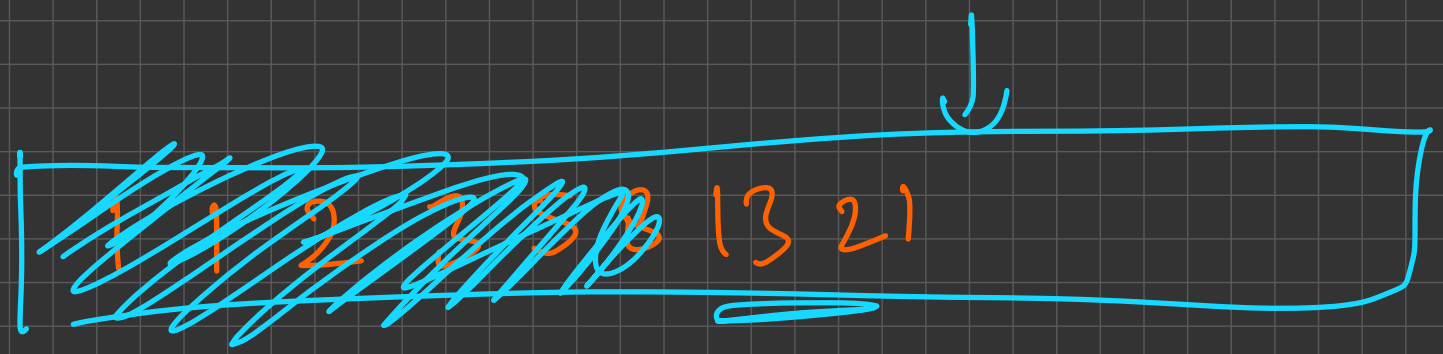
$O(10^8)$ time ✓

$O(10^8)$ space ✗

$O(n^2)$ state & $O(1)$ T-T

OR $n \leq 10^4$

$O(n)$ state & $O(n)$ T-T

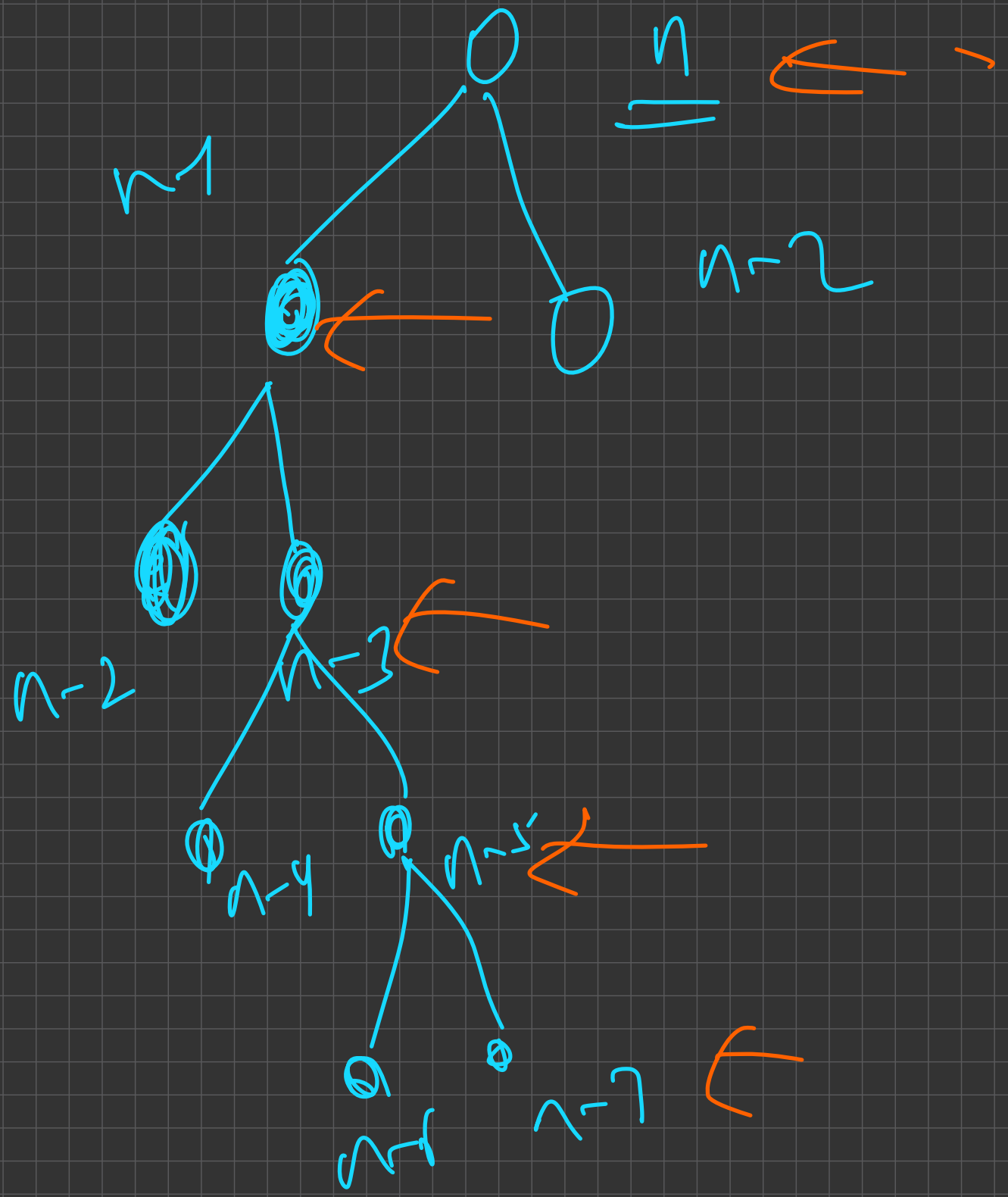


$O(n)$

prev - prev

prev

current = prev + prev - prev


n
n-1
n-2
n-2
n-3
n-4
n-5
n-6
n-7

fib(8)

fib(7) fib(6)

0 → 1 → 2 → 3

fib(8)

fib(7)

fib(6)

$a = 1 \longrightarrow$ fib(1)

$b = 1 \longrightarrow$ fib(2)

$c = ? \longrightarrow$ fib(3)

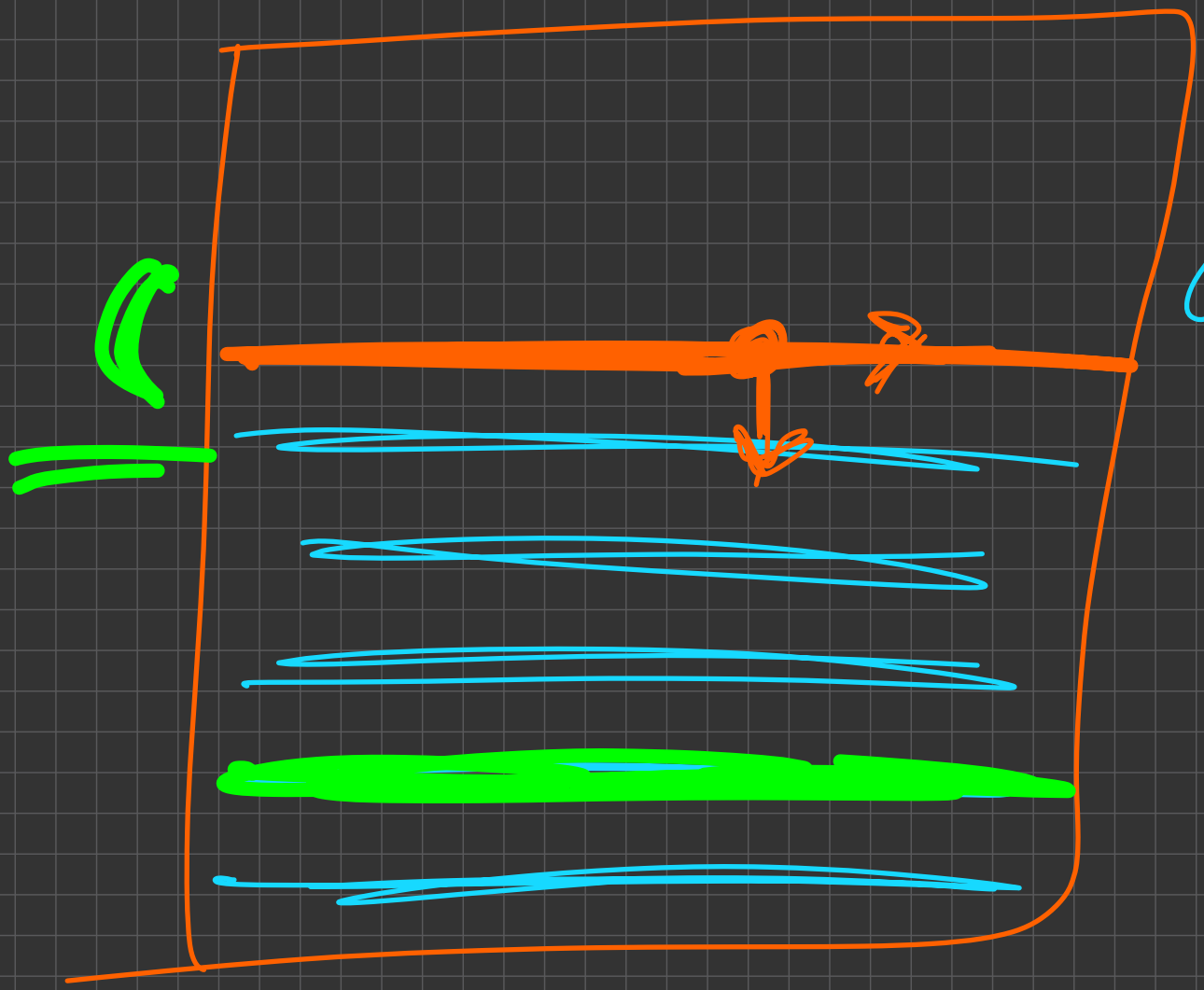
for (int i = 3 ; i ≤ n ; i++) {

$c = a + b$

$a = b$

$b = c$ }

$\underline{\underline{c}}$

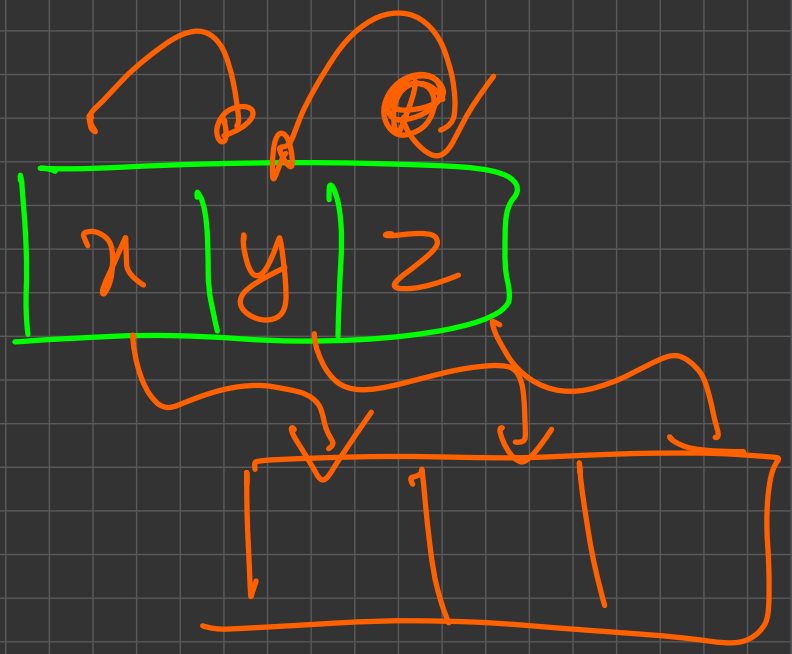


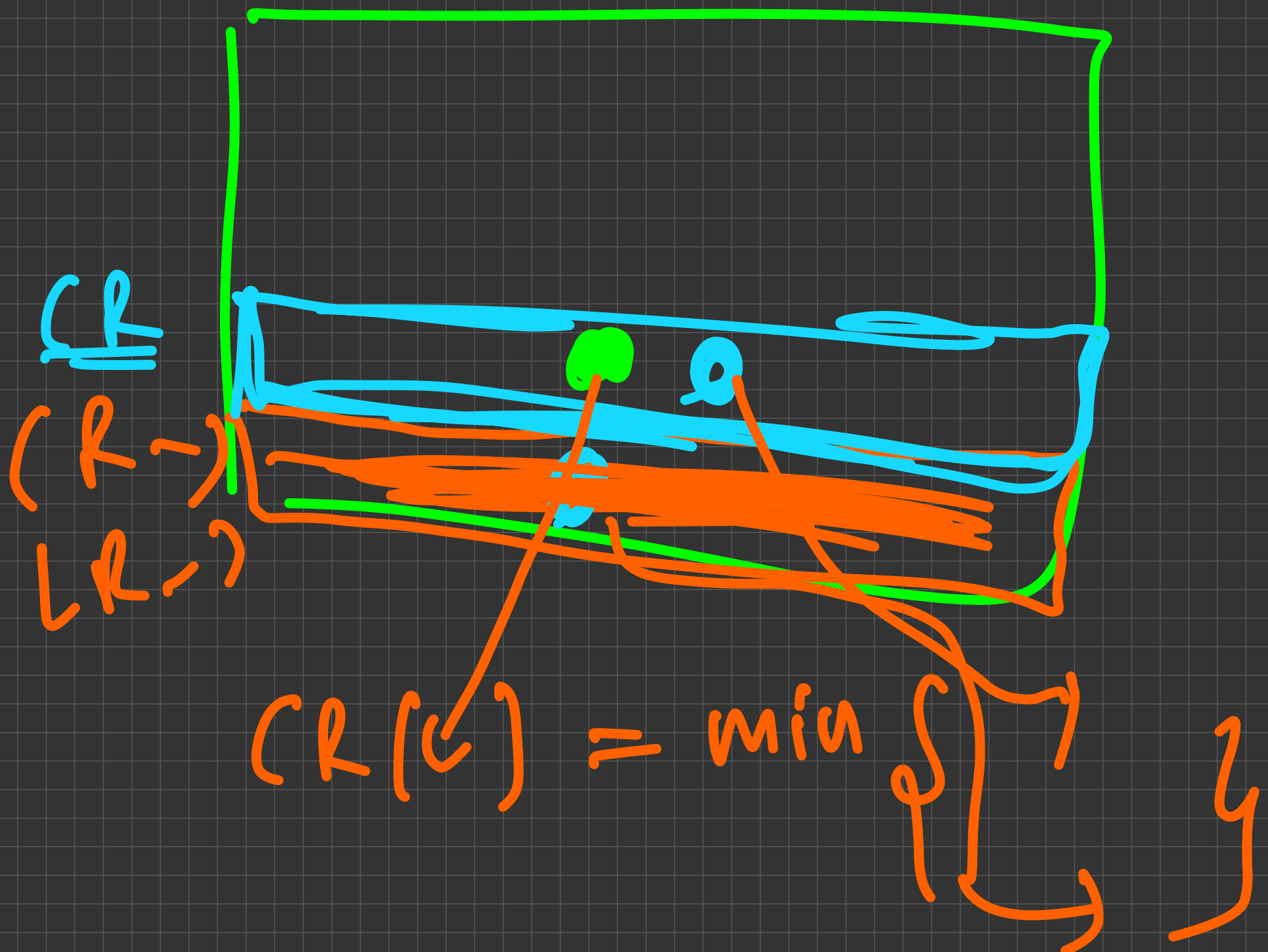
$df(i)(j) =$

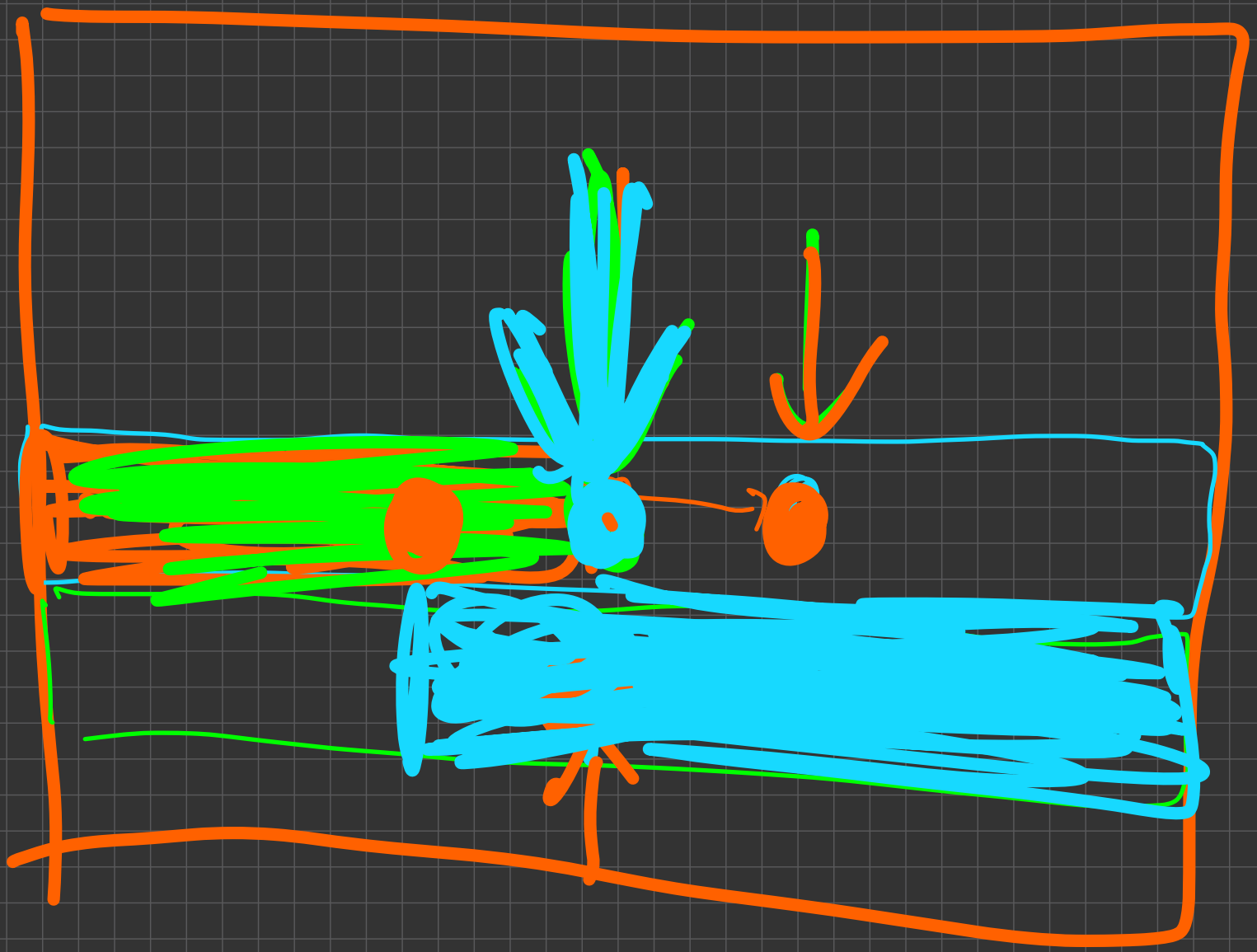
min sum path

from (i, j)

to $(n-1, m-1)$


x
y
z


CR
(R-)
LR-)
CR(C) = min {



- Fibonacci Problem
  - dp[i] depends on dp[i - 1], dp[i - 2]
- Grid Problem
  - dp[i][j] depends on dp[i + 1][j], dp[i][j + 1]

State Optimization

- Ask yourself do you need all the parameters in the dp state?
- If you have dp[a][b][c], and a + b = c, do you need to store c as a parameter or can you just compute it on spot?
- If you can compute a parameter in dp state from other parameters, no need to store it.
- Which parameters should you remove? Highest

Transition Optimization

- Observe the transition equation.
- Can you do some pre-computation to evaluate the equation faster?
- Using clever observations.
- Using range query data structures