Optimised brute force

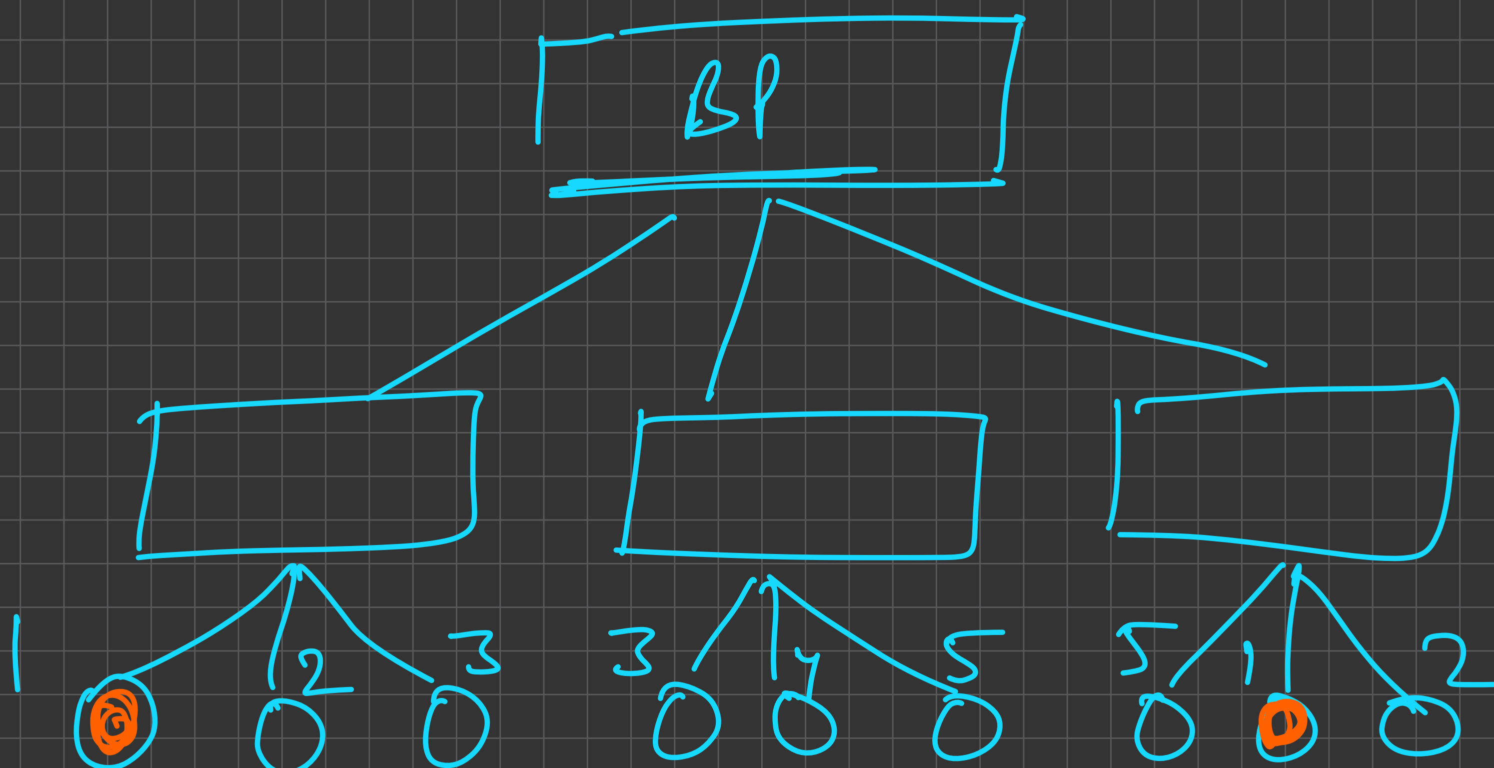
Dynamic Programming 1

- Priyansh Agarwal

input = 10s — function — 10

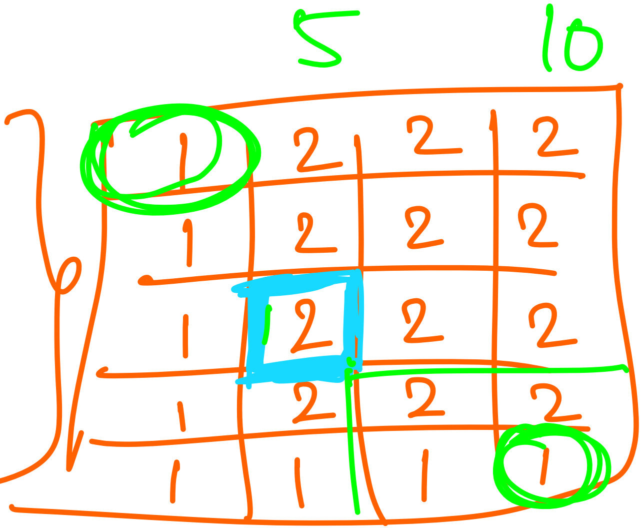
10 seconds

10s



Why Dynamic Programming?

- Overlapping subproblems
- Maximize/Minimize some value
- Finding number of ways
- Covering all cases (DP vs Greedy)
- Check for possibility

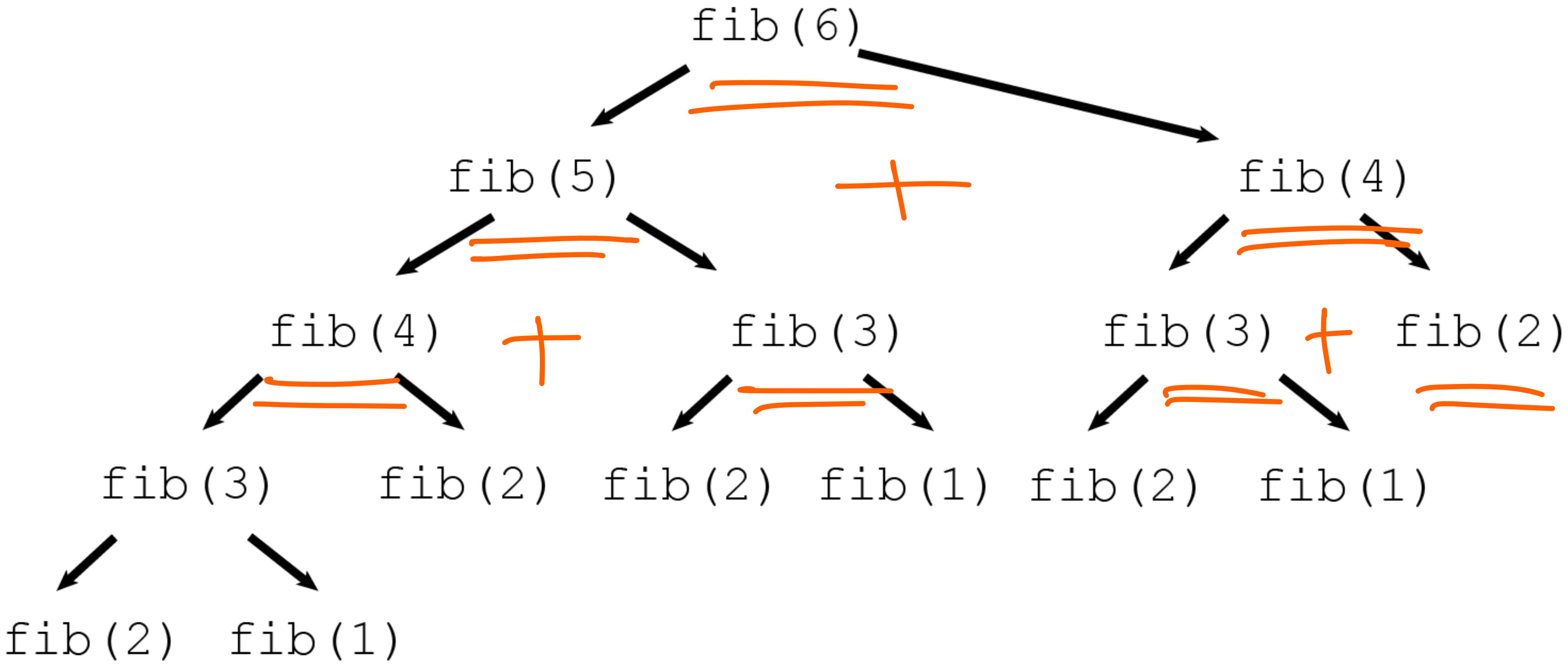


$n = 1000$, $m = 1000$

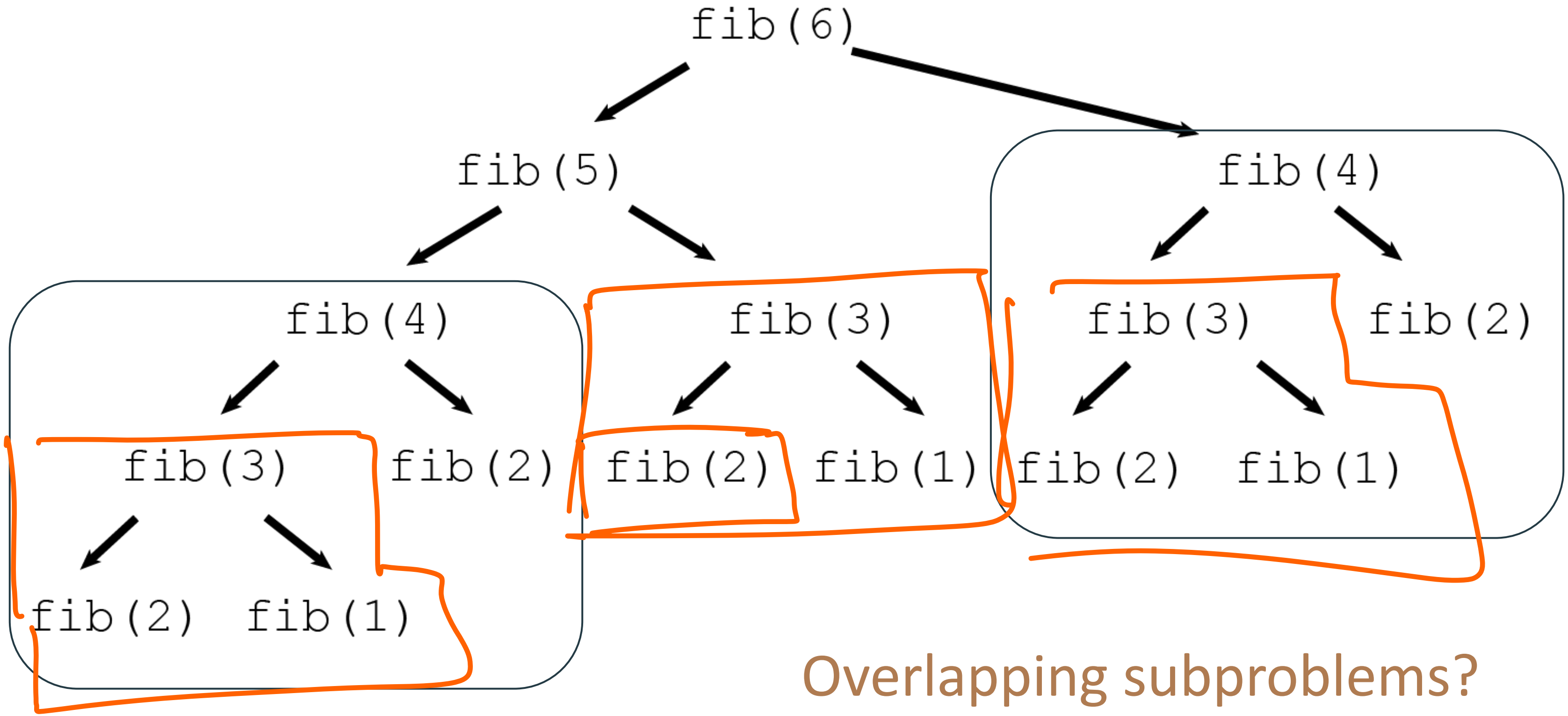
$n \times m$ grid

Need of DP

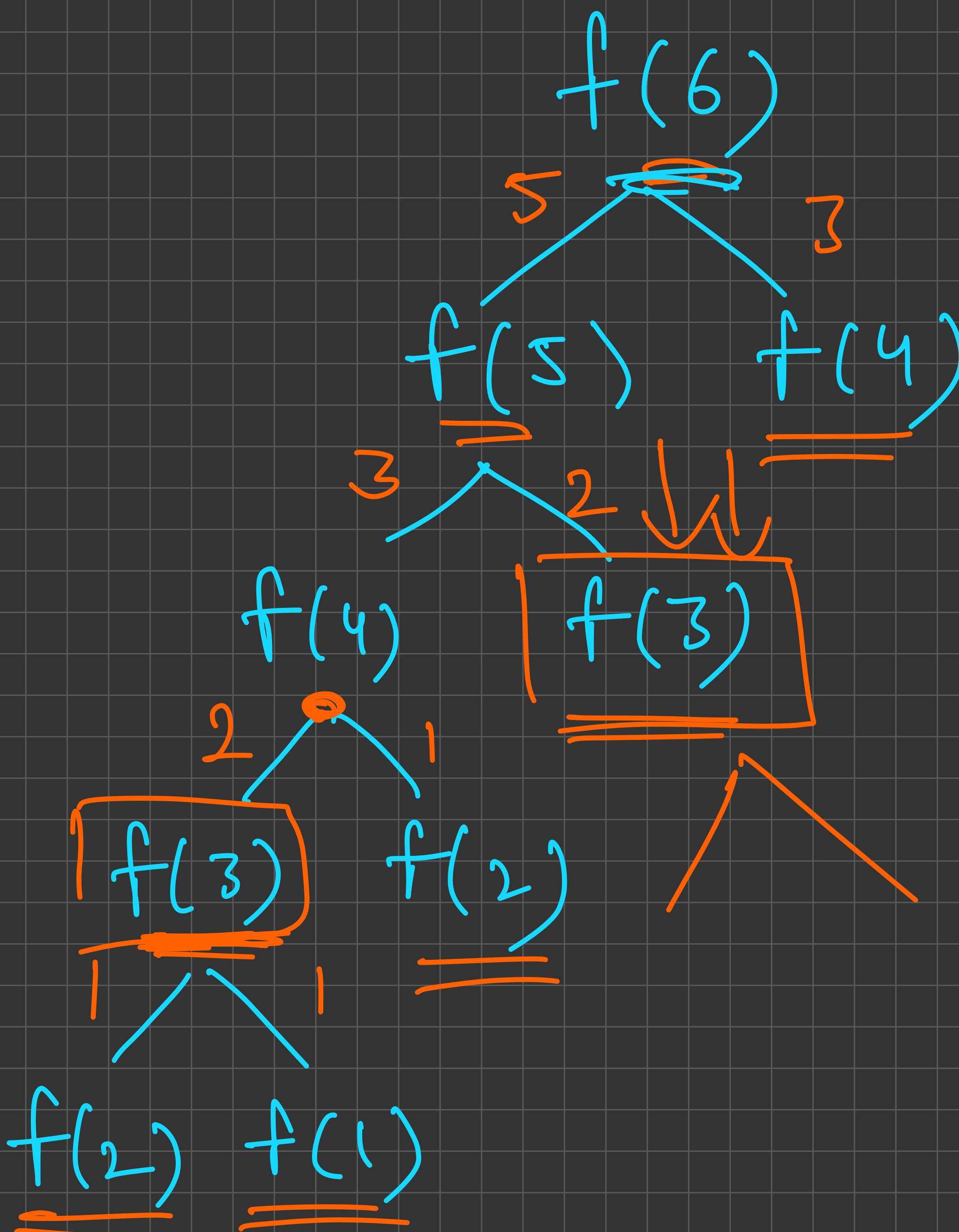
- Let's understand this from a problem
    - Find n^{th} fibonacci number
    - $F(n) = F(n - 1) + F(n - 2)$
    - $F(1) = F(2) = 1$



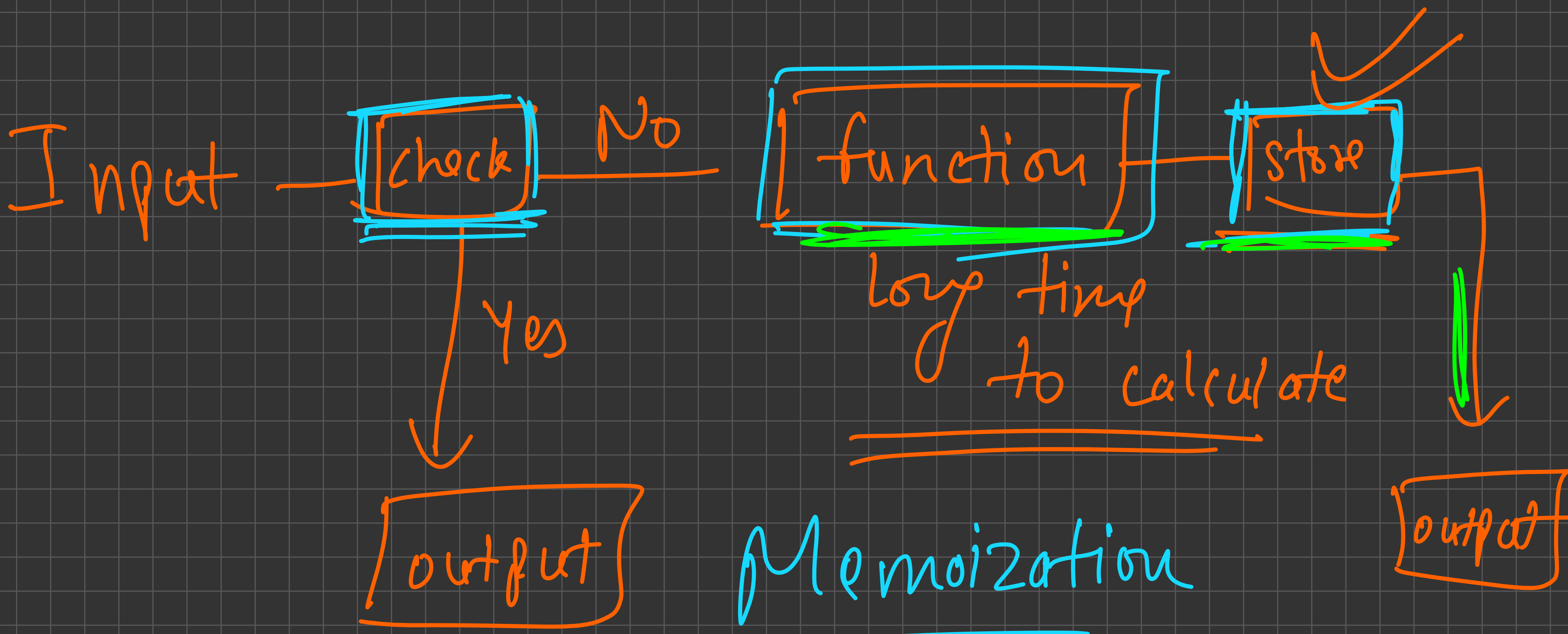
Any problem here?



Overlapping subproblems?



1	1
2	1
3	2
4	3
5	5
6	8



Memoization

map < input , output > cache

treemap , hashmap

map   unordered_map

-10	-10	-10	-10	50	-10	-10
0	1	2	3	4	5	6

int fib[100] → (-10)

input → output {-10, 20}

dp [i][j][k]

{i, j, k} $\longrightarrow$ output

dp[i][j][k]

worst case TC

unordered map — map

$O(n)$ — $O(1)$ — $O(\log n)$

Array[index] $\rightarrow$ value

$arr[10^9]$

$\rightarrow$) as default value

Memoization

- Why calculate F(x) again and again when we can calculate it once and use it every time it is required?
  - Check if F(x) has been calculated
    - If No, calculate it and store it somewhere
    - If Yes, return the value without calculating again

Without DP

```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;
}
```

functionEntered = 1664079
with n = 30

With DP

```
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;
}
```

if nth fibo no. was calculated

storing

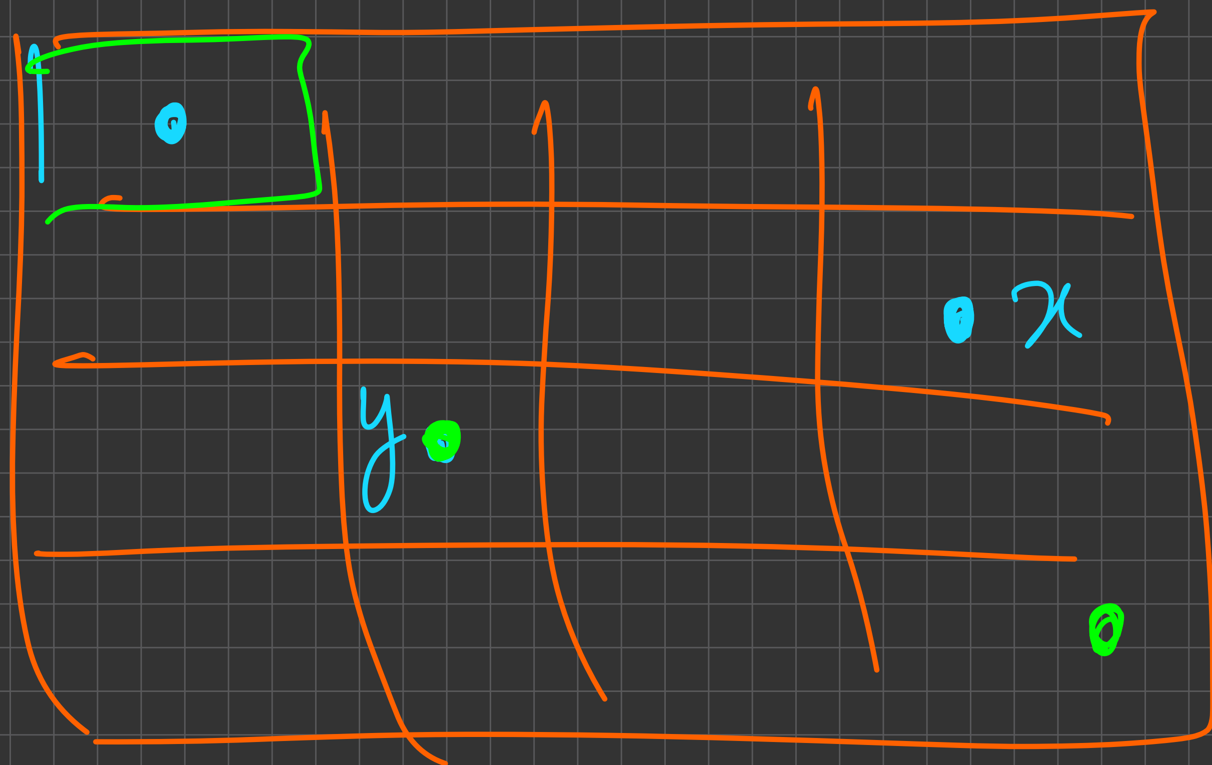
functionEntered = 57
with n = 30

Let's solve another problem!

Given a 2D grid (N X M) with numbers written in each cell, find the path from top left (0, 0) to bottom right (n - 1, m - 1) with minimum sum of values on the path

1	5	8
6	2	7
9	3	4

↓ → allowed only



$2^{n \cdot m}$

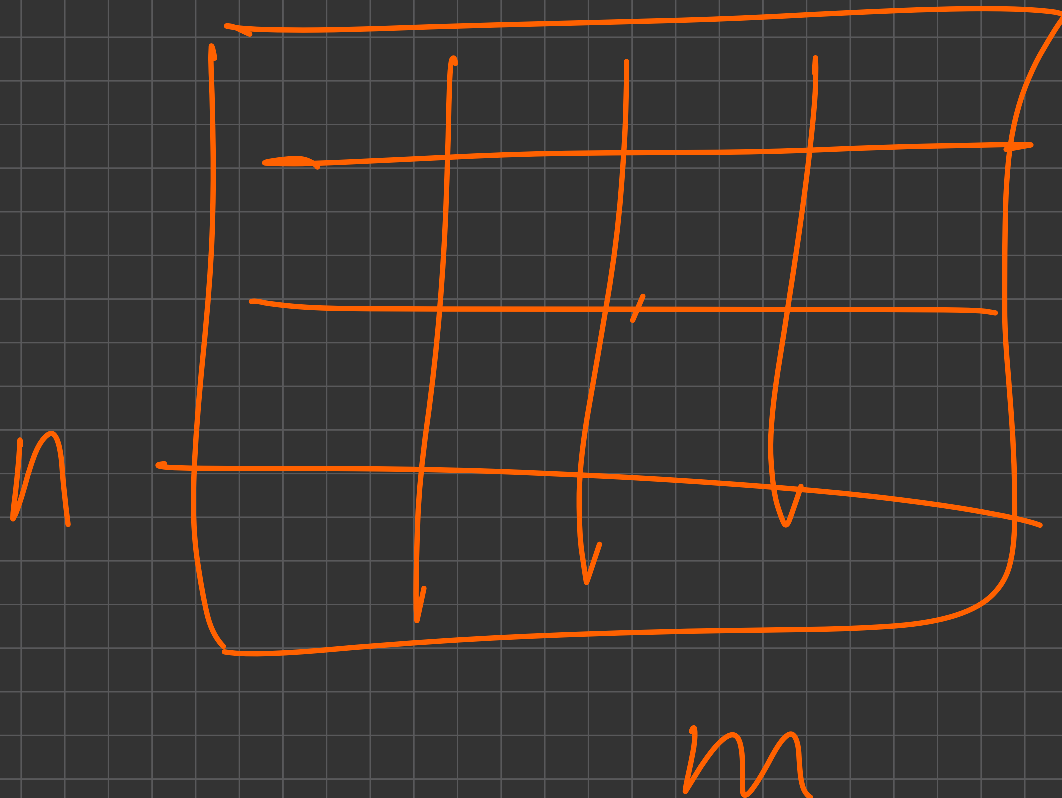
1×5

1

2^5

2^{nm}

2^{n+m}



$n-1$ times $=$

$m-1$ times

$(n-1)$ D steps

$(m-1)$ R steps

$$\boxed{\begin{matrix} n=1 \\ m=5 \end{matrix}}$$

$$\boxed{\dfrac{(D+R)!}{D!\cdot R!} = \dfrac{(n+m-2)!}{(n-1)!\,(m-1)!}}$$

$$\frac{(n+m-2)!}{(n-1)!\,(m-1)!} = \frac{(1+5-2)!}{0!\;4!}$$

$$\boxed{C^{n+m-2}_{n-1}} = \frac{4!}{4!} = 1$$

Naive Way

Explore all paths. Standing at (i, j) try both possibilities (i + 1, j), (i, j + 1)

Every cell has two choices

Time complexity: $O(2^{m*n})$?

Actual Time complexity: $O(C(n + m - 2, m - 1))$

$$\underline{\underline{\text{int } f}}\ (\underline{\underline{\text{int } i}},\ \underline{\underline{\text{int } j}})$$

$$\left.\begin{array}{l} f(i+1, j) \\ f(i, j+1) \end{array}\right\}$$

$f(i, j)$ = shortest sum path from (i, j) to $(n-1, m-1)$

Efficient Way

Overlapping subproblems

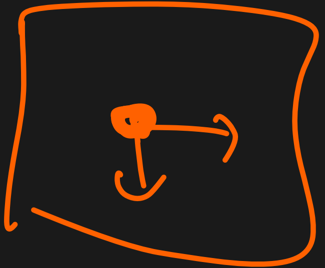
Memoization

Time complexity: O(n * m)

Space complexity: O(n * m)

```cpp
int grid[n][m]; // input matrix

int dp[n][m]; // every value here is -1

// subproblem: f(i, j) represents minimum sum path from (i, j) to (n - 1, m - 1)
int f(int i, int j){ //
    if(i >= n || j >= m){ // moving outside the grid // not allowed
        return INF;
    }
    if(i == n - 1 && j == m - 1) // reached the destination
        return grid[n - 1][m - 1];

    if(dp[i][j] != -1) // this state has been calculated before
        return dp[i][j];

    // state never calculated before
    dp[i][j] = grid[i][j] + min(f(i, j + 1), f(i + 1, j));
    return dp[i][j];
}

void solve(){
    cout << f(0, 0) << nline;
}
```

(i, j)



Important Terminology

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. Represented with some parameters.

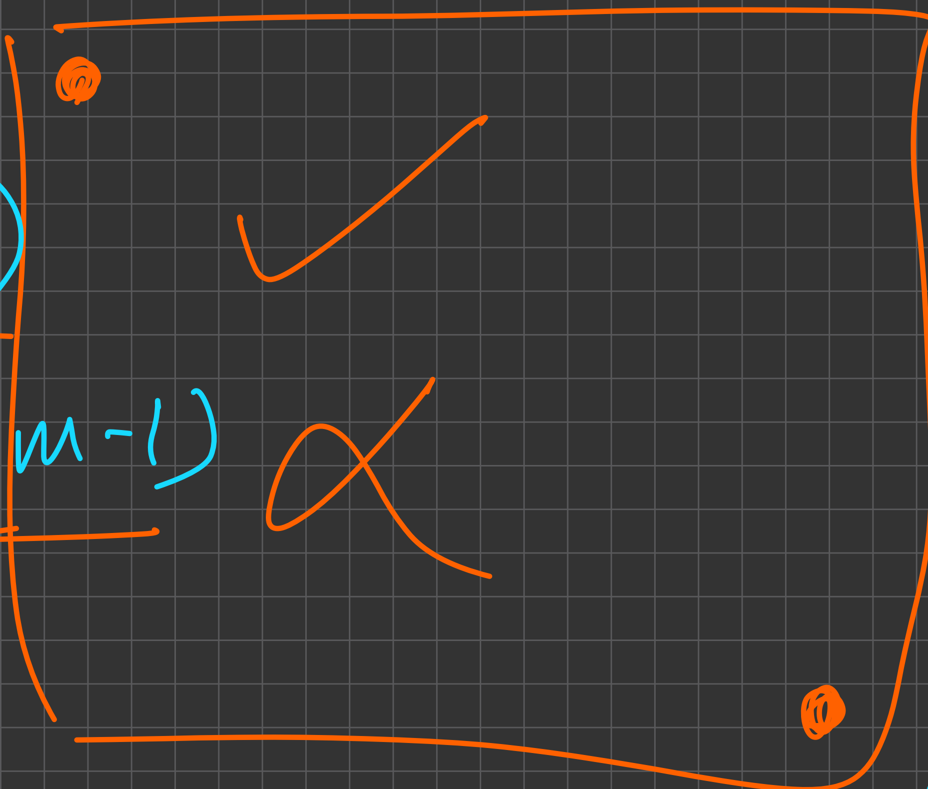
$f(6)$ $f(5)$ $f(4)$

Transition: Calculating the answer for a state (subproblem) by using the answers of other smaller states (subproblems). Represented as a relation b/w states.

$$\boxed{f(6)} = \boxed{f(5) + f(4)}$$

$f(0,0)$

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

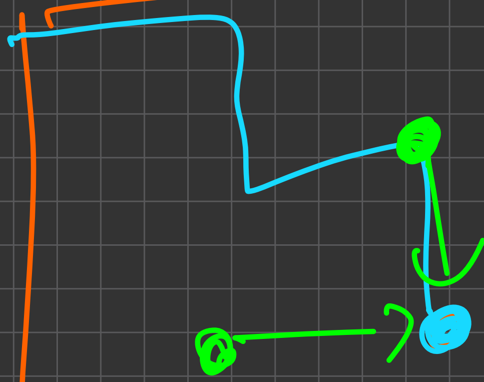


$f(i,j)$ state

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

$$f(i,j) = \underline{grid[i][j]} + \min \begin{cases} f(i+1, j) \\ f(i, j+1) \end{cases}$$

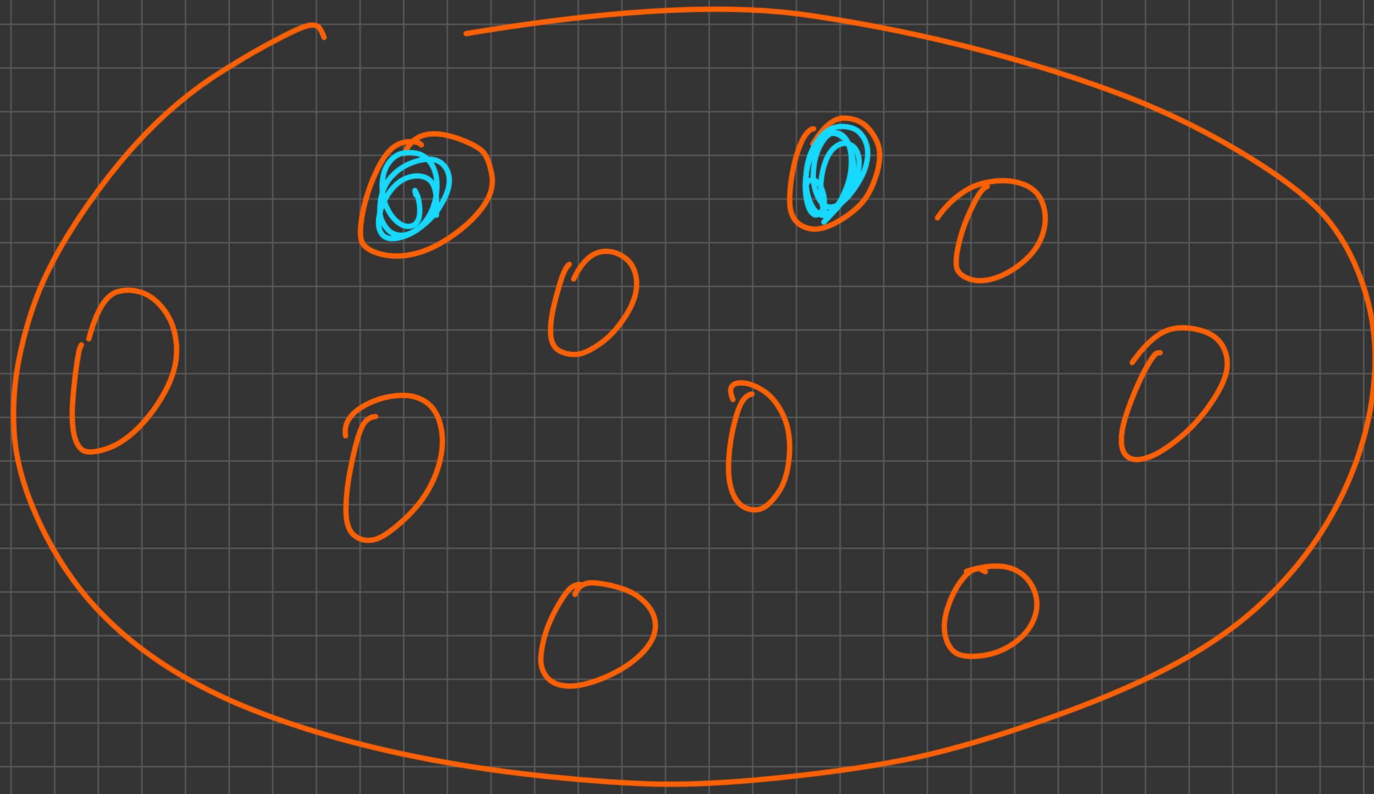
transition equation



$$f(n-1,\ m-1)$$

$$f(i,j) = grid[i][j] + \min\begin{cases} f(i-1,j) \\ f(i,j-1) \end{cases}$$

$$f(i,j) = \text{min sum path from } (0,0) \text{ to } (i,j)$$



Exercise

Fibonacci Problem:

- State
  - dp[i] or f(i) meaning i^{th} fibonacci number
- Transition
  - dp[i] = dp[i - 1] + dp[i - 2]

Exercise

Matrix Problem:

- State
  - dp[i][j] = shortest sum path from (i, j) to (n - 1, m - 1)
- Transition
  - dp[i][j] = grid[i][j] + min(dp[i + 1][j], dp[i][j + 1])

Let’s solve another problem

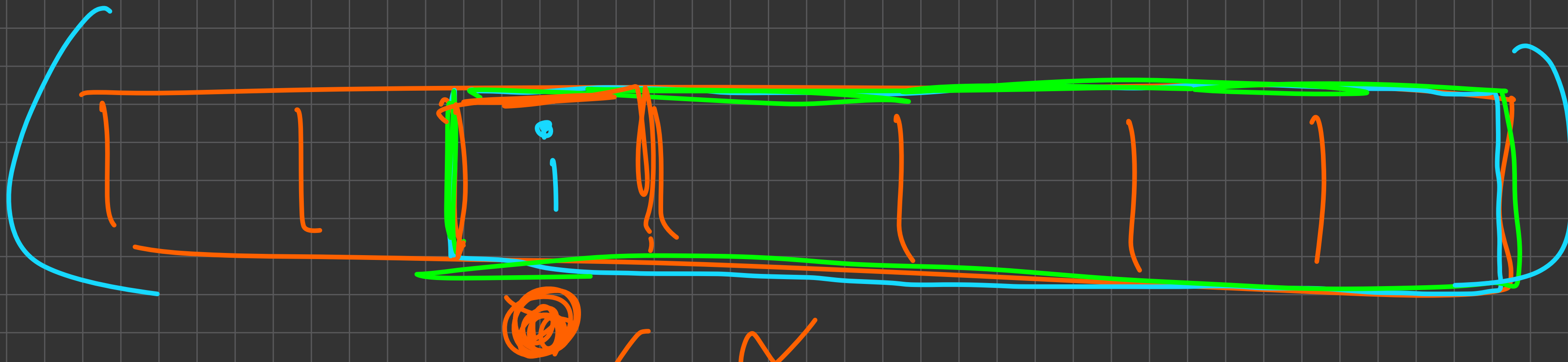
Given an array of integers (both positive and negative). Pick a subsequence of elements from it such that no 2 adjacent elements are picked and the sum of picked elements is maximized.

1	4	2	-10	10	5

Sum = 14

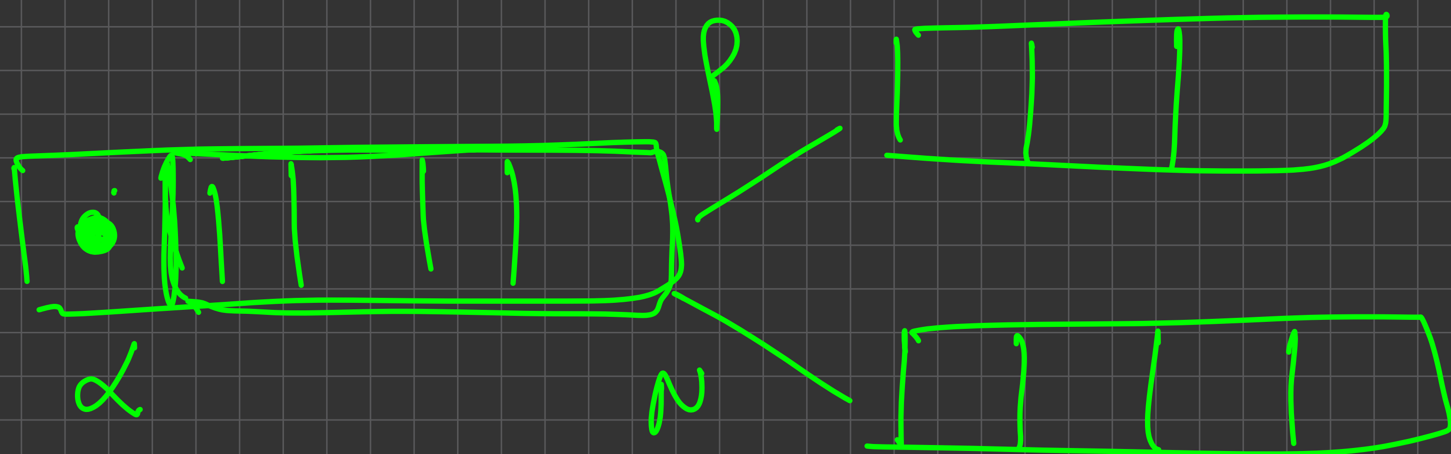
1	4	2	-10	10	5

Sum = 13



α

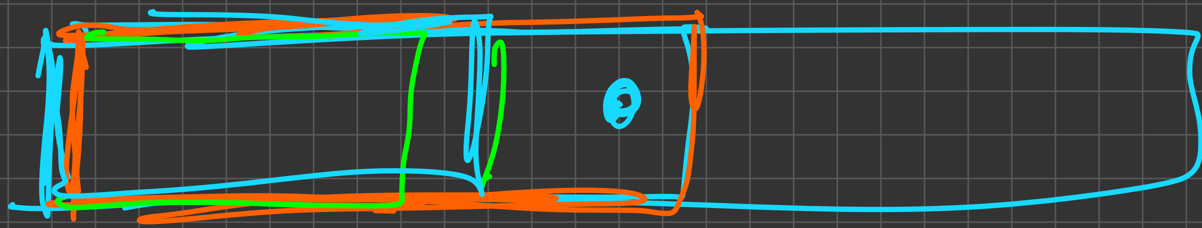
$dp[i]$ = max sum from ith index to $(n-1)$th index



$+ \; arr[i]$

$$dp[i] = \max \begin{cases} dp[i+1] \\ arr[i] + dp[i+2] \end{cases}$$

$dp[0]$ final subproblem



$dp[i]$ = max sum from 0th index to ith index

$$dp[i] = \begin{cases} dp[i-1] \\ \underline{arr[i]} + \underline{dp[i-2]} \end{cases}$$

$\underline{dp[n-1]}$

$0 \quad | \; 2 \; | \; 4 \; | \; -1 \; |$

$$\boxed{dp[i] = \text{max sum from } 0 \text{ to } i}$$

$dp[0]$

$dp[0] = 2$

$$dp[1] \; \Big\langle \begin{matrix} dp[0] \\ 4 + 0 \end{matrix} \Big\} \; \begin{matrix} 2 \\ 4 \end{matrix} \Big\} \; 4$$

$$dp[2] \; \Big\langle \begin{matrix} dp[1] \\ dp[0] + -1 \end{matrix} \Big\} \; \begin{matrix} 4 \\ 2-1 \end{matrix} \Big\} \; \begin{matrix} 4 \\ 1 \end{matrix} \Big\} \; 4$$

$\underline{4 \quad 2} \quad -1$

$dp[0] = 4$

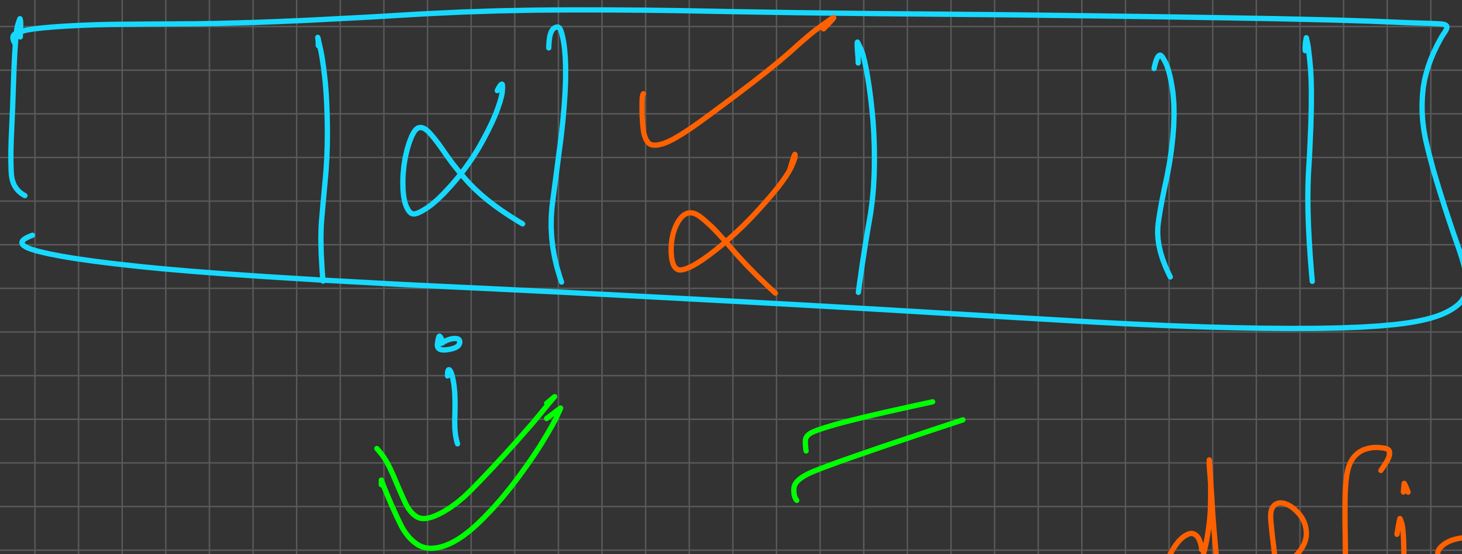
$dp[1] = 4$

$dp[2] = 4$

$dp[i][0] \rightarrow$ don't include ith

$dp[i][1] \rightarrow$ include the ith

max answer from i to n-1



$$dp[i][0] = \max\begin{cases} dp[i+1][1] \\ dp[i+1][0] \end{cases}$$

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

$$f.A = \max(dp[0][0],\ dp[0][1])$$

Some ways to solve the problem

1. Having 2 parameters to represent the state

State:

dp[i][0] = maximum sum in (0 to i) if we don’t pick i^{th} element

dp[i][1] = maximum sum in (0 to i) if we pick i^{th} element

Transition:

dp[i][0] = max(dp[i - 1][1], dp[i - 1][0])

dp[i][1] = arr[i] + dp[i - 1][0]

Final Answer:

max(dp[n - 1][0], dp[n - 1][1])

Some ways to solve the problem

2. Having only 1 parameter to represent the state

State:

$dp[i]$ = max sum in (0 to i) not caring if we picked i^{th} element or not

Transition: 2 cases

- pick i^{th} element: cannot pick the last element : $arr[i] + dp[i - 2]$
- leave i^{th} element: can pick the last element : $dp[i - 1]$

$$dp[i] = \max(arr[i] + dp[i - 2], dp[i - 1])$$

Final Answer:

$dp[n - 1]$

```cpp
int a[n]; // input array

int dp[n]; // filled with -INF to represent uncalculated state

// f(i) = max sum till index i
int f(int index){
    if(index < 0) // reached outside the array
        return 0;
    if(dp[index] != -INF) // state already calculated
        return dp[index];

    // try both cases and store the answer
    dp[index] = max(a[index] + f(index - 2), f(index - 1));
    return dp[index];
}

void solve(){
    cout << f(n - 1) << nline;
}
```

Time and Space Complexity in DP

Time Complexity:

Estimate: Number of States * Transition time for each state

Exact: Total transition time for all states

Space Complexity:

Number of States * Space required for each state

fibo → O(n)

grid → O(n . m)

$dp(1) = n/1$ iterations

$dp(2) = n/2$ iteration

$\vdots$

$dp(n) = n/n$ iterations

$$\# \text{ states} = \underline{\underline{O(n)}}$$

$$\underline{\underline{W \cdot T \cdot T}} = \underline{\underline{O(n)}}$$

$$= \underline{\underline{O(n^2)}}$$

$$n + \frac{n}{2} + \frac{n}{3} \cdots \cdots \frac{n}{n}$$

$$n\left(1 + 1/2 + 1/3 - - - 1/n\right)$$

$$\leq \log n$$

$$n \cdot \log n$$

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

$O(1)$

$$dp[i] = dp[i-1] + dp[i-2] \cdots \cdots dp[0]$$

$dp(n)$ $O(i)$ $O(n)$

fibonacci problem

states . T.T

$= O(n) \cdot O(1) = \underline{\underline{O(n)}}$

Grid Problem

states $= n \cdot m$ T.T $= O(1)$

$= \underline{\underline{O(n \cdot m)}} \leftarrow \left(C^{\,n+m-2}_{\,n-1} \right)$