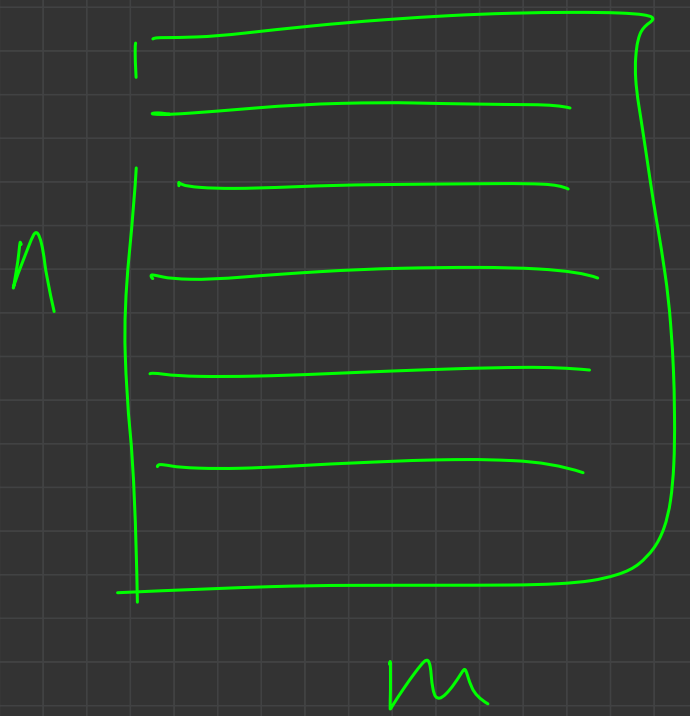
Problems from last class

Problem 1: Link

Problem 2: Link

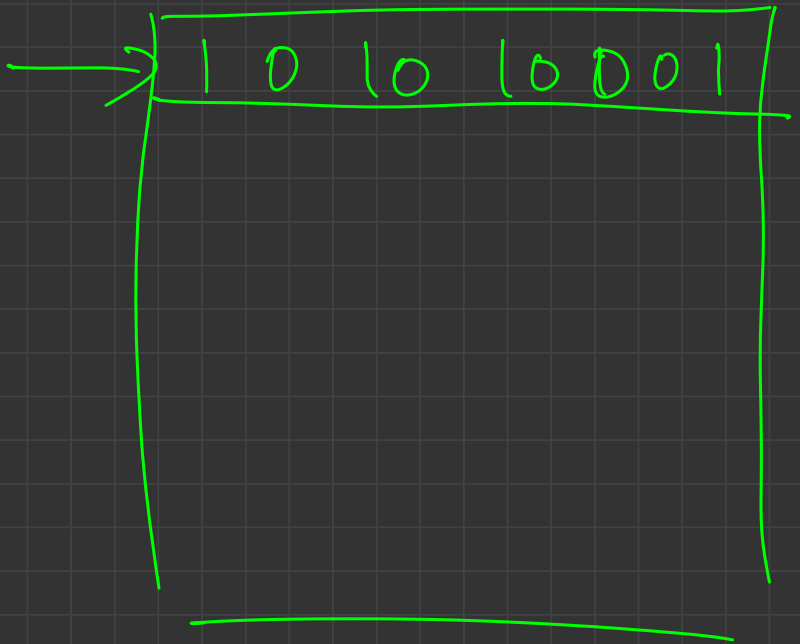


At least one of the bits comes odd number of times

0 —— 1023

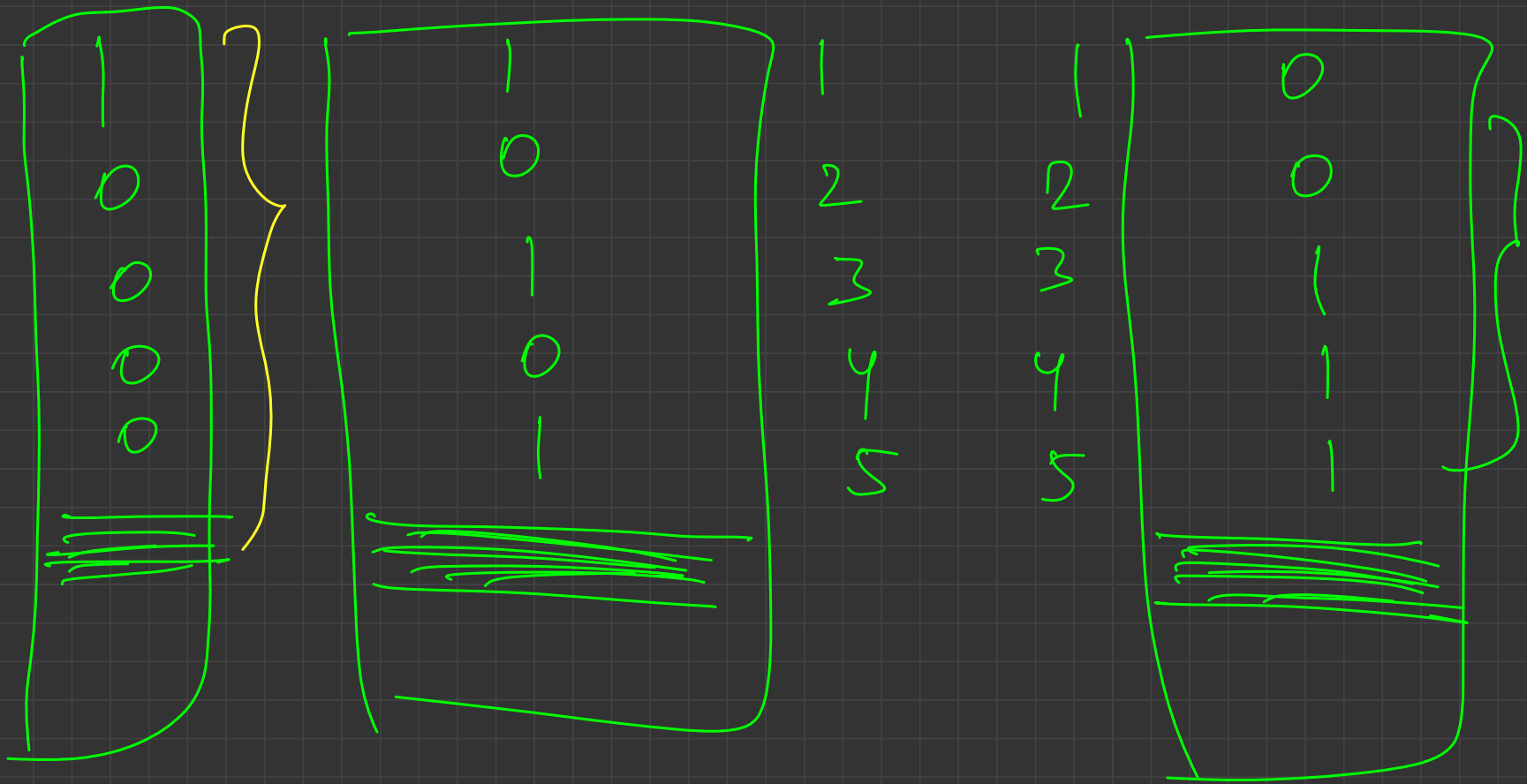
1 1 1 1 1 1 1 1 1 1

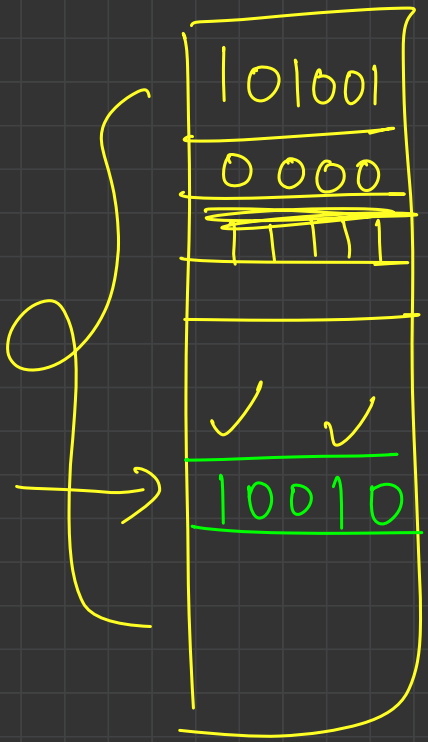
1 0 0 0 0 0 0 0 0 0 0



look at the kth bit

make every number a 1 or a 0 depending on whether the kth bit is set in it


1
0
0
0
0
1
0
1
0
1
1
2
3
4
5
1
2
3
4
5
0
0
1
1
1



State

dp[i][0] = is it possible to pick up odd number of 1s from (i to n-1)

dp[i][1] = is it possible to pick up even number of 1s from (i to n-1)

$dp[i][0]$ $\begin{cases} 1 \\ 0 \end{cases}$ $dp[i+1][1]$ to be true

$dp[i+1][0]$ to be true

$dp[i][1]$ $\begin{cases} 1 \\ 0 \end{cases}$ $dp[i+1][0]$ to be true

$dp[i+1][1]$ to be true

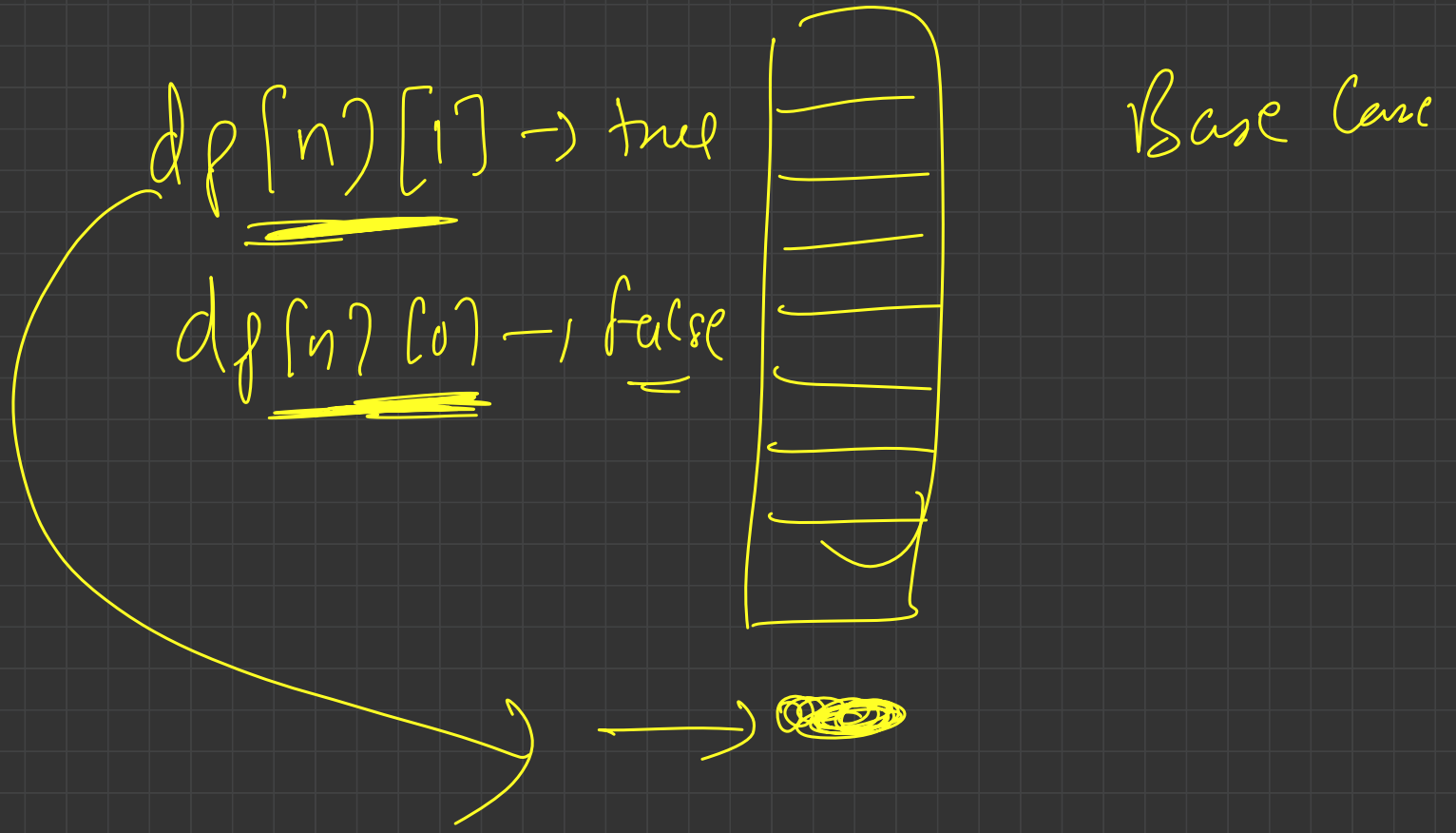
Transition $dp[n-1] \rightarrow dp[n]$

final subproblem for 10th bit

dp[0][0] is true or not — No / Yes

dp[0][0] → 10 times

- 0th → True
- 1st → False
- ⋮
- 10th bit


dp[n][1] -> true
dp[n][0] -> false
Base Case

Time Complexity : $[(n \cdot 2) \cdot 1] \cdot 10$

$O(bits \cdot n \cdot m)$ $O(bits \cdot n)$

Space Complexity : $O(n)$

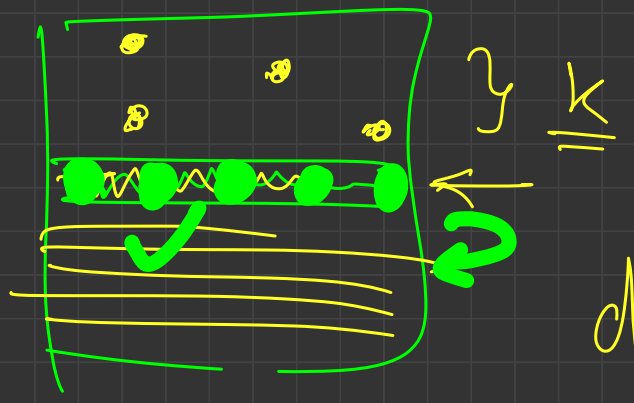
$O(n \cdot m)$

auxilliary space

dp[n][parity][bit]

$dp(i)[k]$ = is there a choice of columns from row i to n such that the $\wedge$ of all numbers > 0 provided my current is k

$k \leq 1023$



$dp(i+1)[k \wedge grid[i][j]]$ is true for any j from 0 to $m-1$ then $dp(i)[k]$ is true

time complexity : $(n \times 1023) \times m$

$\approx n \times m \times 10 + n \times 2 \times 10$

space complexity : $n \times 1023$

$\approx n \times 2$

Problem 2 :

dp[n][m] = min. cuts req. to convert a n x m rectangle into squares

1 x m , (n-1) x m

1 + dp[1][m] + dp[n-1][m]

1 + dp[2][m] + dp[n-2][m]

$\min($ overall k from (1 to n-1) $(1 + dp[k][m] + dp[n-k][m])$ horizontal cut

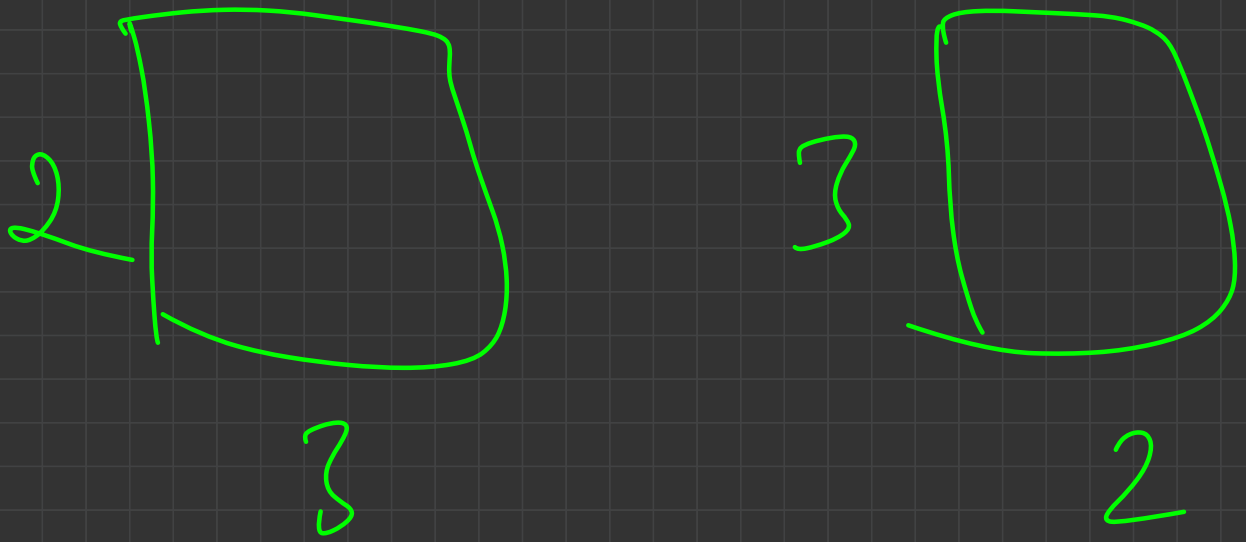
$\min$ overall k from (1 to m-1) $(1 + dp[n][k] + dp[n][m-k])$ vertical cut

State Tran final Subproblem

$dp(a)(b)$ $dp(a)(b)$

$== dp(b)(a)$

Base case $dp(x)(x) == 0$

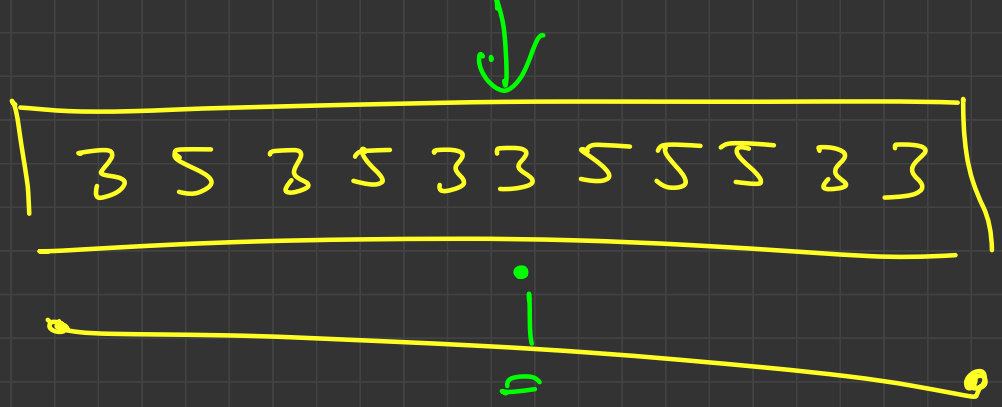

2
3
3
2

Dynamic Programming 2.1

- Priyansh Agarwal

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



$p+q=i$

$dp[i][p][q]$

$dp[i][p]$

$q = i-p$

Parameters are required to differentiate b/w states

$dp(5)(2)(3)$

$dp(5)(4)(1)$

$dp(x_1)(y_1)(z_1)$

$dp(x_2)(y_2)(z_2)$

if there is a linear relation b/w x, y, & z

if $x_1 == x_2$ && $y_1 == y_2$

then $z_1 == z_2$

$2x + 3y = 2$

$2x_1 + 3y_1$

$2x_2 + 3y_2$

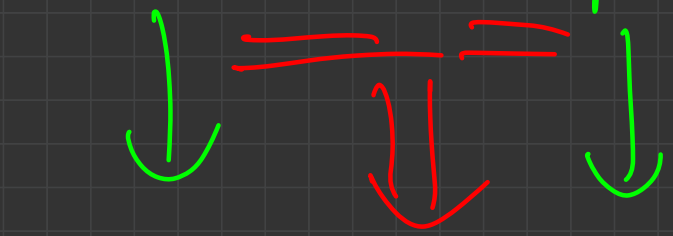
5 parameters in a df state with relation

$$x + 2y + 5z + 6a + 2b = 1000$$

S_1 S_2

if 4 parameters match, then so will the 5th

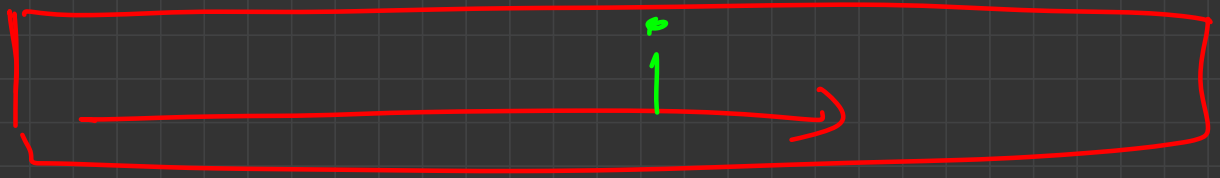
Time & Space



dp[i][j]

states x T.T

states



3 types of characters

$dp(n)(m)(k)$ $\quad an + bm = ck$

$n \leq 100$, $m \leq 2000$,

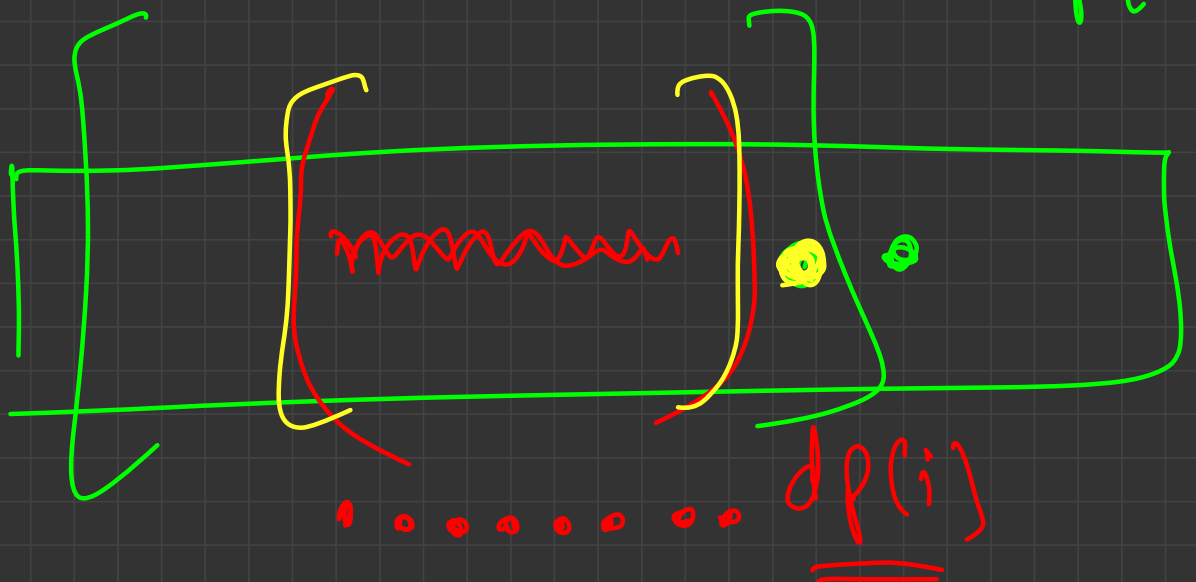
Tc sc $\quad k \leq 200$

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

prefix sum in our case

$dp(i) = \min$ among $dp(i-1), dp(i-2) \ldots dp(i-k;)$



Time Complexity : # states x T·T

Space : # states

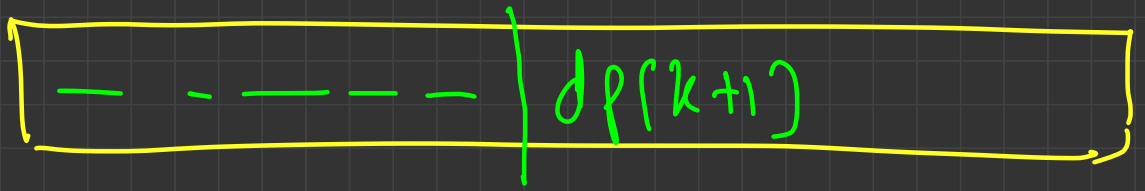
$dp[i] = dp[i-1] + dp[i-2] + dp[i-3]$

$\vdots$

$dp[0]$

$dp[n]$ is

$dp[0] = 1$ B.C



$dp[0] - \cdots - dp[k]$

✓ $sum[k] = (dp[0] + dp[1] - \cdots - dp[k])$

✓ $dp[k+1] = sum[k]$

✓ $sum[k+1] = (sum[k] + dp[k+1])$

✓ $dp(k+2) = sum(k+1)$

✓ $sum(k+2) = sum(k+1) + dp(k+2)$

T.C = $O(n)$

Space: $O(n)$ dp
$O(n)$ sum

$\rightarrow O(n)$

$$dp[k] = dp[k-1] + dp[k-2] + dp[k-3] \dots dp[0]$$

$$dp[k-1] = dp[k-2] + dp[k-3] \dots\dots dp[0]$$

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

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

T.C $\rightarrow$ $O(n)$ $\qquad$ Space $\rightarrow$ $O(n)$

$dp[0] = 1$, $dp[1] = 1$, $dp[2] = 2$

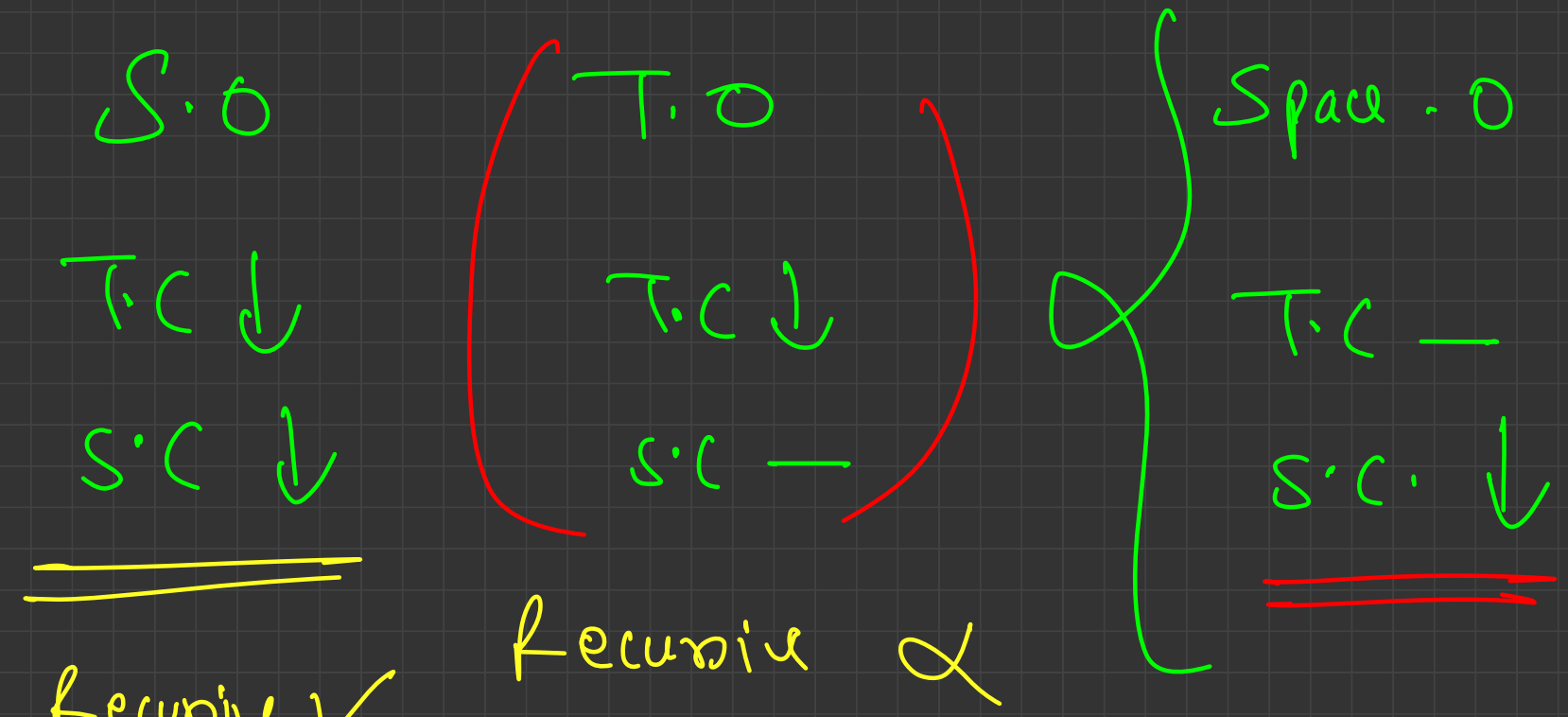
$dp[3] = 4$, $dp[4] = 8$, $dp[5] = 16$

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

$dp[0] = 1$

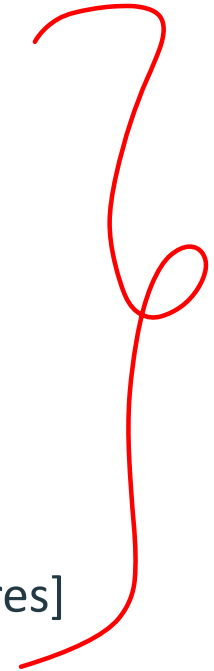
T.C → $O(\log n)$ $O(n^2)$

S.C → $O(1)$ $O(n)$


S.O
T.C ↓
S.C ↓
Recursive ✓
T.O
T.C ↓
S.C —
Recursive ✗
Space.O
T.C —
S.C. ↓

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]



$$dp(n) = dp(n-1) + dp(n-2) + dp(n-3)$$

$O(n)$

dp[4], dp[3], dp[2]) dp[1]

dp(0) = 10, dp(2) = 20,

dp(3) = 40,

dp(5)

dp(4) = 70

O(1)

[10, 20, 40] O(1)

dp[4] = 70

[10, 20, 40, 70)

(70, 20, 40)

(70, 130, 40)

[20, 40, 70)

[20, 40, 70, 130) → [40, 70, 130)

O(1)

O(3) → O(1)

$a = 10, \quad b = 20, \quad c = 40$

$c_1 = a + b + c = 70$

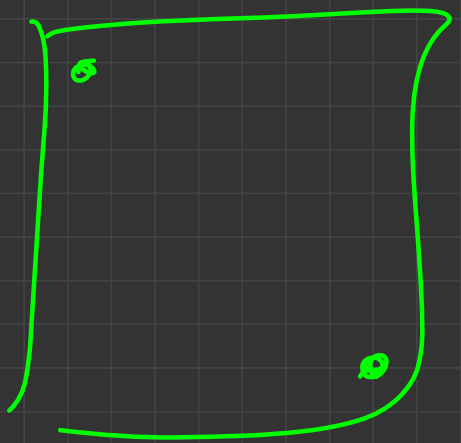
$b_1 = c$

$a_1 = b$

$a = a_1, \quad b = b_1, \quad c = c_1$

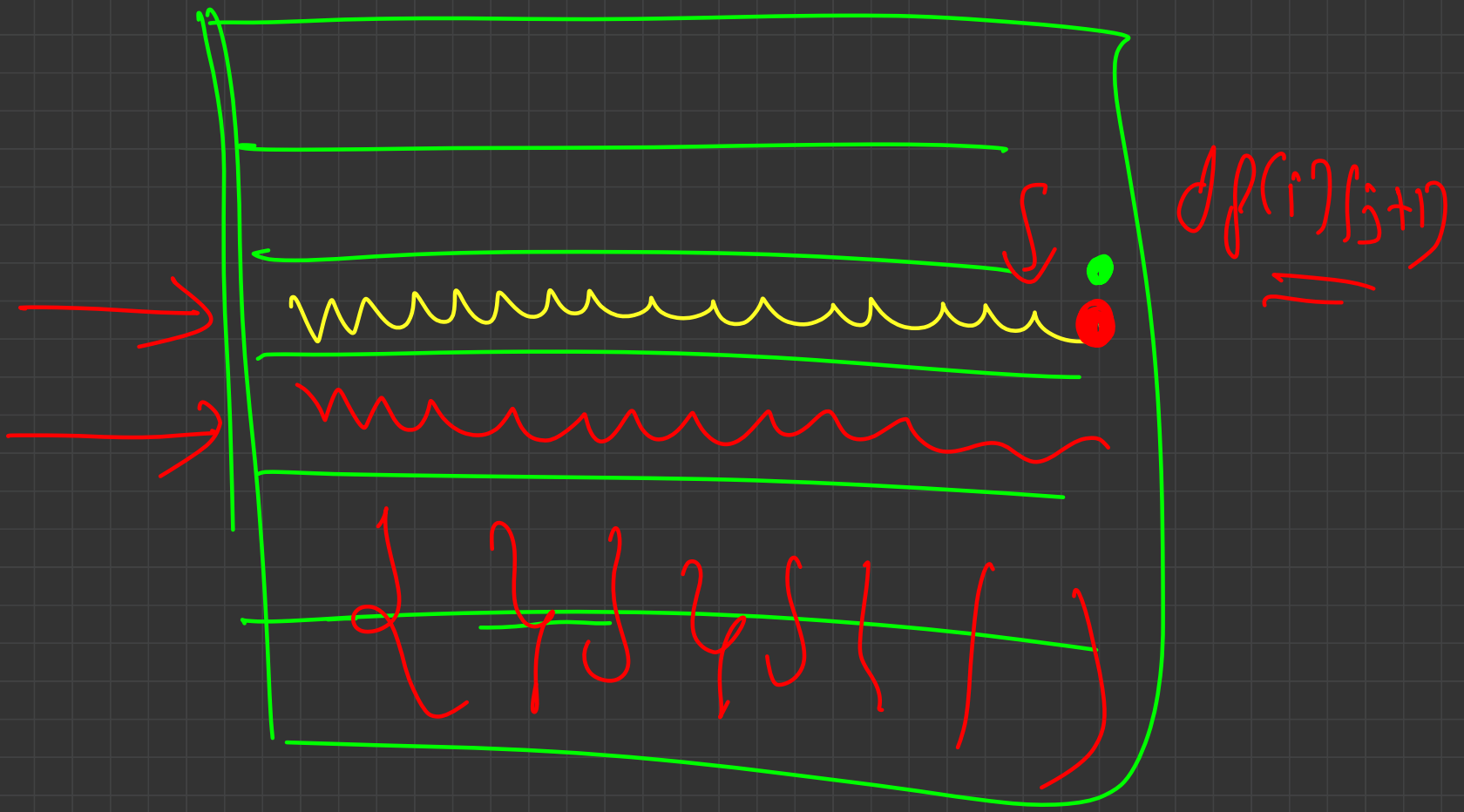
$dp[i][j] = grid[i][j] + \min$

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



$O(n \cdot m)$ Time

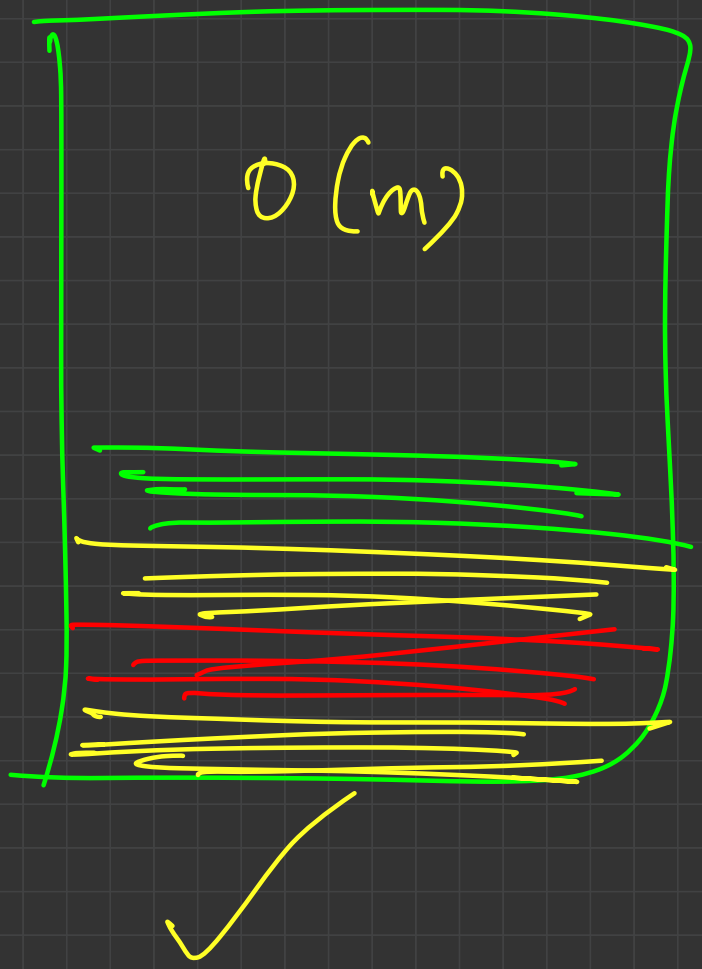
$O(n \cdot m)$ space

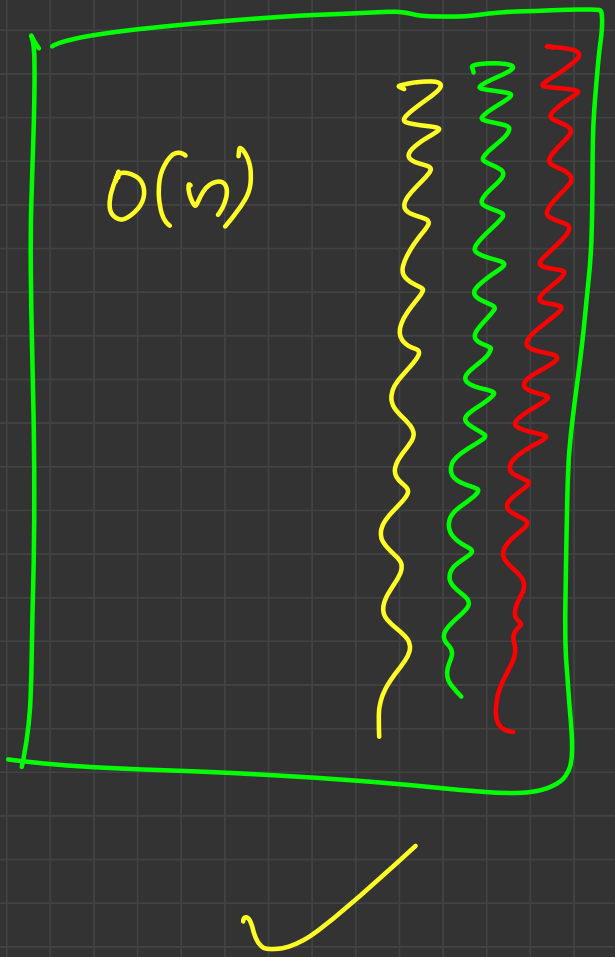

dp[i][j+1]

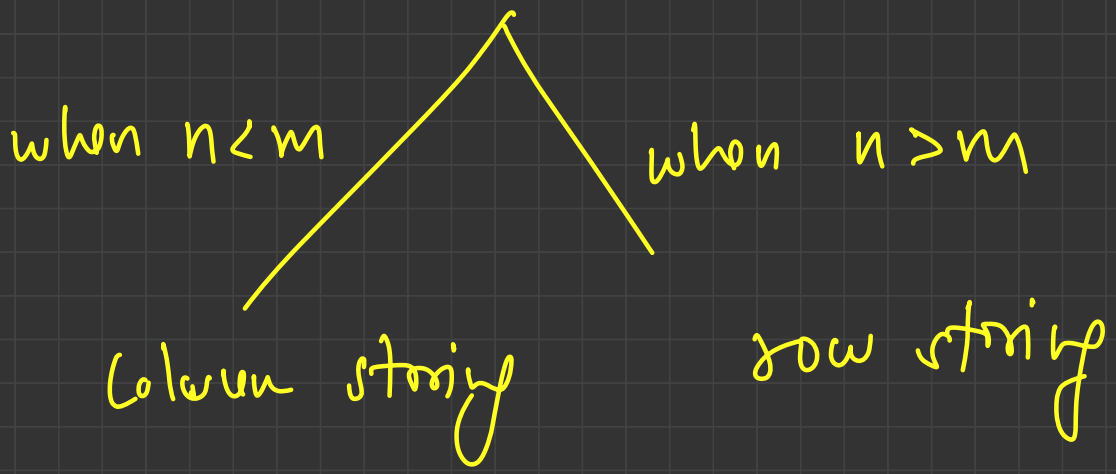
$dp[i][j]$

$dp[i+1][j]$

$dp[i][j+1]$


O(m)


O(s)


when $n < m$
when $n > m$
Column string
row string

$100 \times 20 \longrightarrow 20$

$20 \times 100 \longrightarrow O(m) \rightarrow O(100)$

$O(n)$

State

Recursive

Time Complexity ↓

Space Complexity ↓

which parameter to choose

Transition

↓

↓

Data struct

Clever obs

Space

↓

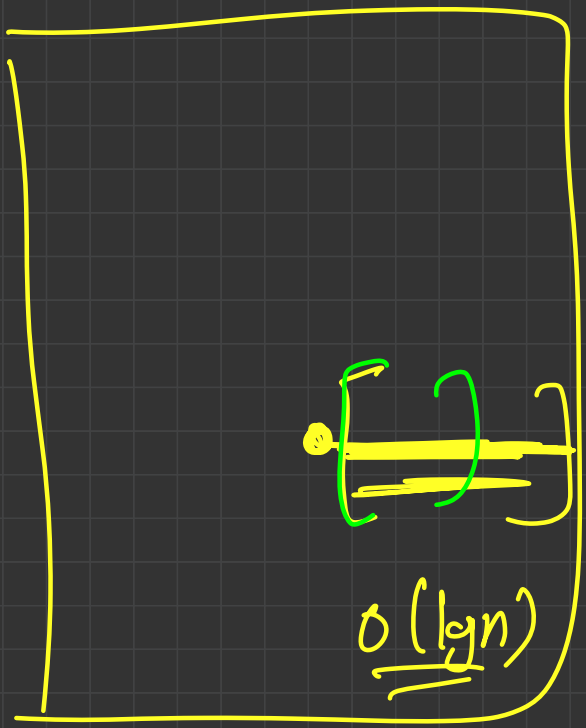
↓

row, column

Amritapuri

$O(n^4 \log n)$

$O(n^4)$



suffix min

O(1)

suffix array

Answer Construction

- Grid problem: Find the actual path with the minimum sum.
- Minimizing coins problem: Find the actual choice of coins.
- 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.

Answer Construction - Grid Problem

```cpp
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);
}
```

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;
}
```