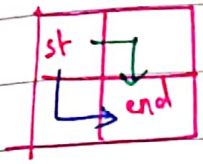
DP ON GRIDS

* Total Unique Paths (M x N Matrix :

start - [0][0]

end - [m-1][n-1]

Find unique paths from [0][0] - [m-1][n-1]



⇒ 2 Ways

⇒ Figuring out all possible ways (Recursion)

① Express in terms of ind ⇒ $f(i, j)$

② Do all shuffs

③ Sum all ways / Max / Min. (Based on ques)

Recurrence Equation : $f(i, j)$ → no. of ~~ways~~ unique ways from (0, 0) → (i, j)

(m-1, n-1)

(m-1, n-1)

① BASE CASES

$f(i, j)$ {

```
if (i==0 && j==0)
    return 1;
```

→ You reached dest, so count that path

```
if (i<0 || j<0)
    return 0;
```

→ Went out of boundary so return 0

② Explore other shifts: → more right / more downward (if starting from [0][0])

Since we perform top-down approach, use up or left

up = f(i-1, j)
left = f(i, j-1)

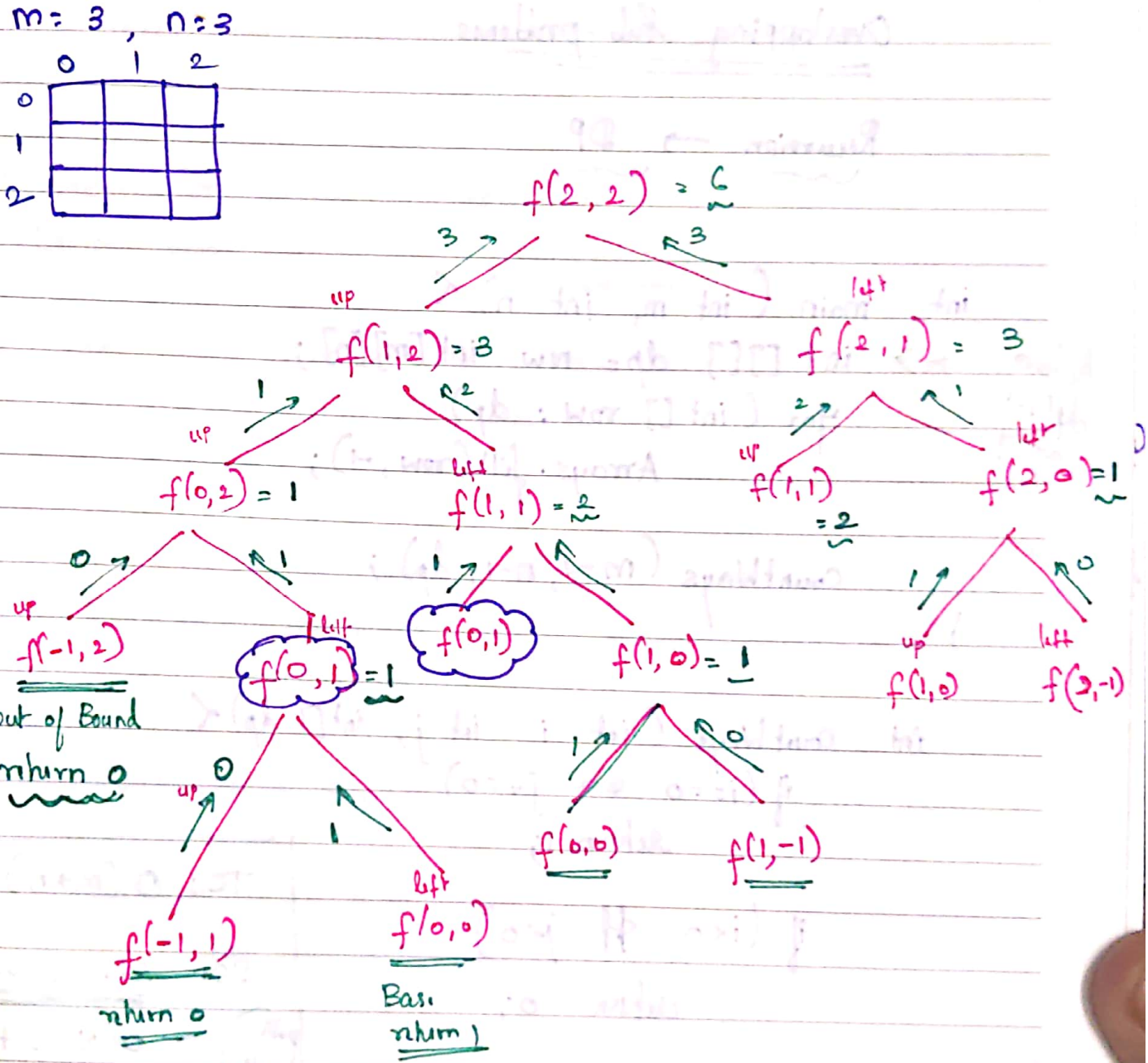
③ Sum all ways:

return up + left;

```
(m-1) (n-1)
f(i, j) {
    if (i==0 && j==0)
        return 1;
    if (i<0 || j<0)
        return 0;
    up = f(i-1, j);
    left = f(i, j-1);
    return up+left;
}
```

TC: $(2^{m \times n})$ → for each box two options

SC: O(path length)



i) f(0,1) ; up = 0 left = 1
up + left = 1

ii) f(0,2) = 1

iii) f(1,0) = 1

iv) f(1,1) = 2

v) f(1,2) = 3 vi) f(2,0) = 1 vii) f(2,1) = 3

Overlapping Sub-problems

Recursion → DP

```
int main (int m, int n) {
    int [][] dp = new int [m][n];
    for (int [] row : dp)
        Arrays.fill(row, -1);

    CountWays (m-1, n-1, dp);
}

int CountWays (int i, int j, int[] dp) {
    if (i==0 && j==0)
        return 1;

    if (i<0 || j<0)
        return 0;

    if (dp[i][j] != -1)
        return dp[i][j];

    int up = CountWays (i-1, j, dp);
    int left = CountWays (i, j-1, dp);
    return dp[i][j] = up + left;
}
```

declare dp[][] array

check if already found

save before return

TC: O(m+n)

SC: Stack space:
path length ← O((N-1) + M-1))
+
DP Size

Memo → TABULATION (Bottom-up (0-n))

i) Declare Dp Array: dp[n][m]

ii) Declare Base Cases: dp[0][0] = 1

iii) Here 2D Array: → row first, col next
(for loops) => 2

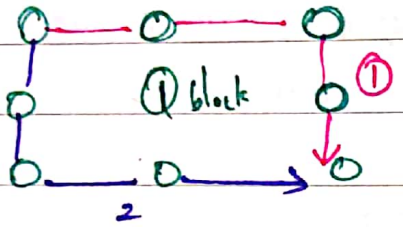
```
int[][] dp = new int[m][n];
for (int[] row : dp)
    Arrays.fill(row, -1);

dp[0][0] = 1;

for (int i=0; i<m; i++) {
    for (int j=0; j<n; j++) {
        if (i==0 && j==0)
            continue;

        int up = 0;
        int left = 0;
        if (i>0)
            up = dp[i-1][j];
        if (j>0)
            left = dp[i][j-1];
        dp[i][j] = up + left;
    }
}
return dp[m-1][n-1];
```

TC: O(m*n)
SC: O(m*n)

★ Total Unique Paths - 2

→ Same like Unique Paths, with one Blocker

→ If you encounter a blocker, that way won't be consider



⇒ 2 Ways

Same like Unique Paths, with one extra Base Case,

if within boundary, & you encounter a blocker ①, return 0

MEMOIZATION

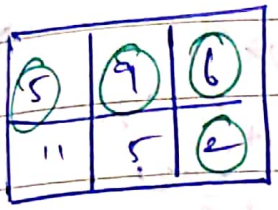
```
int dp[][] = new int[m][n];
Arrays.fill(dp, -1);
path(m-1, n-1, Arr, dp);
return dp[m-1][n-1];

public int path(int i, int j, int[][] arr, int[][] dp)
{
    if(i>=0 && j>=0 && arr[i][j]==1)
        return dp[i][j]=0;

    if(i==0 && j==0)
        return dp[i][j]=1;

    if(i<0 || j<0)
        return 0;

    if(dp[i][j]!=-1)
        return dp[i][j];

    int up = path(i-1, j, arr, dp);
    int left = path(i, j-1, arr, dp);
    return dp[i][j] = up + left;
}
```

Have this first, if not, returns error for [i]

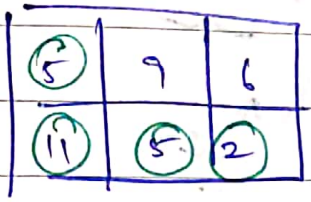
* Minimum Path Sum

Find path from [0][0] to [M-1][n-1] with Min Cost

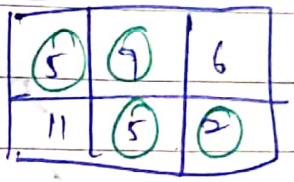
dir. ↓ down, → right



22



23



21 → min path

Greedy method won't work for all cases ⇒ Go with DP (try out all paths) & find min cost path

⇓

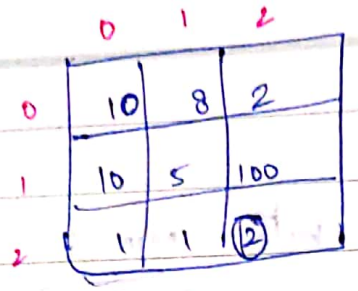
Recursion.

f(i, j) → min cost to reach from (0,0) to (M-1)(n-1)

```
{
  if (i==0 && j==0)
      return arr[i][j];
  if (i<0 || j<0)
      return INT_MAX;
```

→ Went out of Bound that path not needed so return a MAX

```
  up = a[i][j] + f(i-1, j);
  lft = a[i][j] + f(i, j-1);
  return min(up, lft);
```



	0	1	2
0	10	8	2
1	10	5	100
2	1	1	(2)

i) $f(0,1) = 18$

ii) $f(0,2) = 20$

iii) $f(1,0) = 20$

iv) $f(1,1) = 23$

v) $f(1,2) = 120$

vi) $f(2,0) = 21$

vii) $f(2,1) = 22$

MEMOIZATION → SAME as we did for Unique Grids

i) int dp[][] array = new int[m][n];

ii) fill with -1

iii) check if already calculated, if dp[i][j] != -1 return dp[i][j]

iv) Save in dp before returning.

TABULATION

```
int[][]dp = new int[m][n];
dp[0][0] = arr[0][0];

for (int i=0; i<m; i++) {
    for (int j=0; j<n; j++) {
        if (i==0 && j==0)
            continue;
        int up = INT_MAX; matrix[i][j];
        int left = INT_MAX; Matrix[i][j];
        if (i>0)
            up += dp[i-1][j]
        else
            up += INT_MAX;
        if (j>0)
            left += dp[i][j-1];
        else
            left += INT_MAX;
dp[i][j] = Math.min(up,left);
```

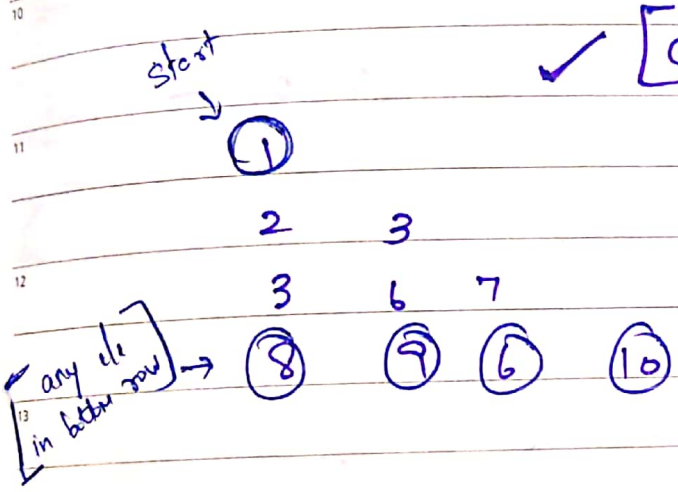
TC: O(m*n)
SC: O(m*n)

075-290 Week 11

* (LII) TRIANGLE GRID (fixed starting point, Varying end point)

Gn: Triangle Array: Find min path sum to reach from top to Bottom row

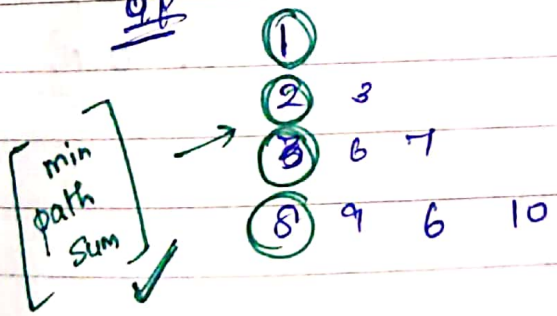
✓ [Can reach any element in bottom row]



Dirs: ↓ (down) ↘ (diag)

✗ [GREEDY WON'T WORK for all Cases]

dp



✓ Try all possible paths:
↳ RECURSION

✓ Find Min

RECURSION STEPS

i) Express in terms of idx $f(i,j)$

ii) Explore all paths. (down / diagonal)

iii) Find min among all paths

17 Sunday

076-289 Week 11

⇒ Usually we start from Bottom (m-1, n-1)

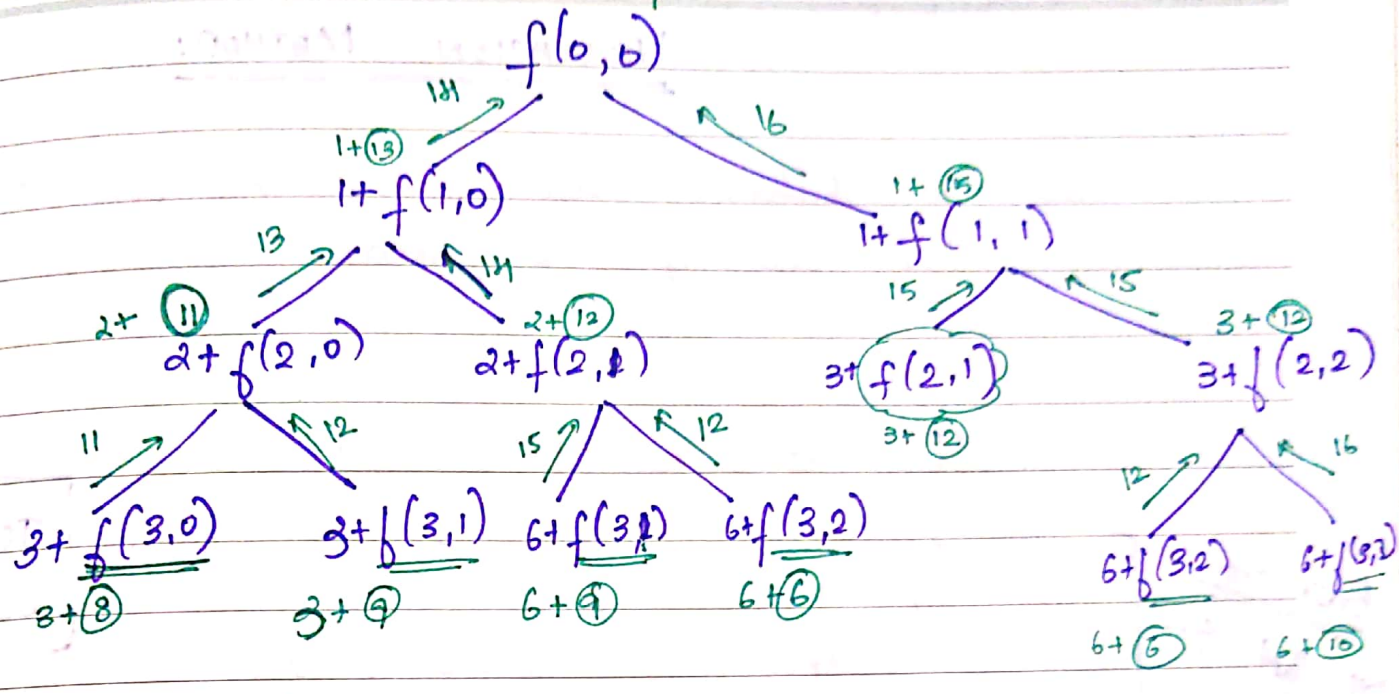
But here **end is not Fixed** → so can't start from [M-1][n-1]

So, y(0,0) ⇒ (start from [0,0] → last row (any ele))

```
f(i,j) {
    if (i == n-1)
        return a[n-1][j];
    d = a[i][j] = f(i+1, j);
    dg = a[i][j] = f(i+1, j+1);
    return min(d, dg)
}
```

Only one Base Case

TC: exponential
SC: O(N)



f(2,0) = 11 f(1,0) = 13

f(2,1) = 12 f(2,2) = 12

MEMOIZATION

TC: O(n×n) ≈

SC: O(n*n) + O(n)

(recursion space)

```
int[][] dp = new int[n][n];
Arrays.fill(dp, -1);
find(0, 0, n, dp)
return dp[0][0];
public int find(int i, int j, int[][] dp) {
    if (i == n-1)
        return dp[i][j] = arr[i][j];
    if (dp[i][j] != -1)
        return dp[i][j];
    int down = arr[i][j] + find(i+1, j);
    int dg = arr[i][j] + find(i+1, j+1);
    return min(down, dg);
```

1
2 3
3 6 7
8 4 6 10

TABULATION METHOD:

Here 4 different Base cases, So fill the last row of dp array

```
int[][] dp = new int[n][n];

for (j=0; j<n; j++) {
    dp[n-1][j] = arr[n-1][j];
}

for (int i=n-2; i>=0; i--) {
    for (int j=0; j<=i; j++) {
        int down = arr[i][j] + dp[i+1][j];
        int diag = arr[i][j] + dp[i+1][j+1];
        dp[i][j] = Math.min(down, diag);
    }
}
return dp[0][0]
```

Base case ← fill last row

fills from Backwards, ← Goal base case is last row

TC: O(n*n)
SC: O(n*n)

[In Tabulation, always start filling from Base Case]

* L12

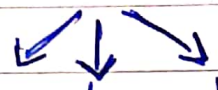
Min/MAX falling path Sum

→ Variable Starting pt

→ Variable Ending pt

→ n * m matrix

→ start from any cell in first row → find Max path end in any cell in last row

→ Directions: 

1	2	10	4
100	3	2	1
1	1	20	2
1	2	2	1

o/p 105

GREEDY WON'T Work

→ As there is no uniformity, you'll lose (We don't know how values are split)

REQUIRED

MAX Path Sum → from (any) cell = 1st row
to (any) cell = last row

TRY ALL PATHS & PICK MAX ⇒ RECURSION

i) Express in index (i,j) as Base Cases

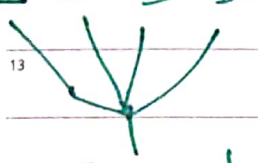
ii) Explore all paths 

iii) find Max among all

TRICK : No exact starting pt / no exact end pt

What to Do ? Consider either starting / ending points as explore all

	0	1	2	3
0	1	2	10	4
1	100	3	2	1
2	4	1	20	2
3	1	2	2	1



⇒ Here, Consider ending row & try to find max path for all ele's

We have 4 recursions : $f(3,0)$ $f(3,1)$ $f(3,2)$ $f(3,3)$ — till 0th row

For each, find max path,

& max of all 4 will be ans :

$$\max(f(3,0), f(3,1), f(3,2), f(3,3)) = \text{Ans}$$

Recursion func Snippet :

can be $f(3,0)$ / $f(3,1)$ / $f(3,2)$ / $f(3,3)$

```
f(i, j) {
    if (i == 0)
        return arr[i][j];
    if (j<0 || j>=m)
        return MIN_VAL;
    int up = a[i][j] + f(i-1, j);
    int left = a[i][j] + f(i-1, j-1);
    int right = a[i][j] + f(i-1, j+1);
    return max(up, max(left, right));
```

Comes before (i==0)

In Recursion: moves are opposite

i<0 (not req)

TC : each call : 3 options 3^n (exponential)

SC : O(N) → Recursion stack

Tc: O(n*m)
Sc: O(n*m) + O(n)

27 Wednesday
MARCH

Week 13 086-279

RECURSION → MEMOIZATION (to store res of overlapping sub-problems)

```
int getMaxPath (int[][] arr) {
    int m = arr.length;
    int n = arr[0].length;
    int[][] dp = new int[m][n];
    for (int[] i : dp)
        Arrays.fill(i, -1);
    int maxi = MIN_VAL;
    for (int j=0; j<n; j++) {
        ans = getpath(m-1, j, n, arr, dp);
        maxi = Math.max(maxi, ans);
    }
    return maxi
}
```

indiv recursion calls for each ele in last row as col wise maxi each time

```
int getpath (int i, int j, int n, int[][] arr, int[][] dp) {
    if (j<0 || j>=n)
        return -1e9;
    if (i==0)
        return dp[i][j] = arr[i][j];
    if (dp[i][j] != -1)
        return dp[i][j];
    int up = arr[i][j] + getpath(i-1, j, n, arr, dp);
    int left = arr[i][j] + getpath(i-1, j-1, n, arr, dp);
    int right = arr[i][j] + getpath(i-1, j+1, n, arr, dp);
    return dp[i][j] = Math.max(up, left, right);
}
```

Base Case (Out of Bound)

Base Case if hit first row

check if already in dp

3 directions

max of all 3

	0	1	2
0	1	1	100
1	2	2	3
2	3	10	2

Solved sub-problems

$f(1,0) = 3$
$f(1,1) = 102$
$f(2,0) = 205$

① $f(2,0)$ → 105

min (left), 6 (up), 105 (right)

$3 + f(1,-1)$ return min

$3 + f(1,0)$ $3+3$

$3 + f(1,1)$ $3+102$

$f(1,-1)$: min (3), $2+f(0,-1)$ min

$f(1,0)$: 3, 3; $2+f(0,0)$ Base $2+1$; $2+f(0,1)$ Base $2+1$

$f(1,1)$: 3, 3, 102; $2+f(0,0)$ $2+①$; $2+f(0,1)$ $2+①$; $2+f(0,2)$ $2+100$

② $f(2,1)$ → 113

13 (left), 112 (up), 113 (right)

left: $10+f(1,0)$ already solved $10+3$

up: $10+f(1,1)$ $10+102$

right: $10+103$ $10+f(1,2)$

$f(1,2)$: 4, 103, min

$3+f(0,1)$ base $3+①$

$3+f(0,2)$ base $3+100$

$3+f(0,3)$ out of Board

③ Similarly do for $f(2,2)$ & return max of all 3

TABULATION

(Soln from base cases)

i) declare dp[m][n]

ii) fill out the Base Cases: if (i==0)
return arr[i][j]
↳ all ele's

* for first row, fill all elements

```
for (int j=0 → n-1)
    dp[0][j] = arr[0][j]
```

iii) Populate all other indexes of dp Arrays:

```
for (int i=1 → m-1) {
    for (int j=0 → n-1) {
        int up = arr[i][j] + dp[i-1][j];
        int left = arr[i][j];
        int right = arr[i][j];
        if (j>0)
            left += dp[i-1][j-1];
        else
            left += -1e9;
        if (j< n-1)
            right += dp[i-1][j+1];
        else
            right += -1e9;
        dp[i][j] = max(up, left, right);
    }
}
```

089-276 Week 13

iv) Find MAX of ALL RECURSIONS
(Bottom row has 4 recursions)

```
int Maxi = MIN_VAL;
for (int j=0; j<n; j++) {
    Maxi = Math.max(Maxi, dp[M-1][j]);
}
return Maxi;
```

Maxi → ans

TC: O(N*m)
SC: O(n*m)

] lesser than prev 2

090-275 Week 13

★ 113 CHERRY PICKUP → 3D DP

Gn: Grid of size `R x c', each grid has chocolates. Wanted to collect max of choc with help of friends.

Initialy Alice → top-left corner (0, 0)
Bob → ~~bottom~~ top-right corner (0, n-1) } same row

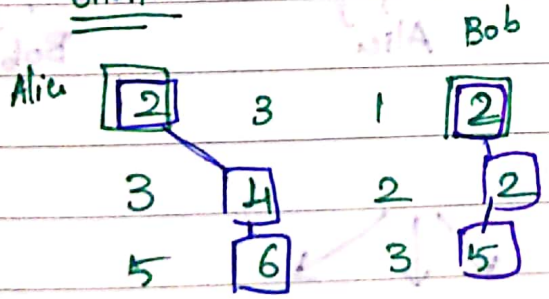
Directions they can move:

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

↓ ↙ ↘

When anyone passes from any cell, he'll pick all choc ~~int~~ in it, no. of choc in that cell become zero

GRID



12 + 9 = 21

Alice (0,0) Bob (0, m-1)

✓ fixed starting pt
✓ Variable ending pt.

Greedy won't work because of uniformity.

What is Required?

(All paths by Alice) + (All paths by Bob)

↓ Recursion (1) ↓ Recursion (2)

Is it possible to done individually ?
Recursion for Alice & Recursion for Bob ?

⇒ It is a tedious process ⇒ bcoz you have to keep track of path & change it to Zero, because two can't pick the same thing

Recursion for Two : Alice & BoB simultaneously

Recursion Rules

i) Express in terms of idx (i_1, j_1) & (i_2, j_2)
Alice Bob

ii) Perform Base Cases

iii) Explore all paths : ↙ ↓ ↘

iv) Get max of all

Fixed starting point & Variable ending pt

:Soo Start Recursion from start

one i is enough, bcoz both will be in same row

094-271 Week 14 4 Thursday APRIL

Alice Bob

0 0 0 m-1

Base Cases : → Destination case
→ Out of Bound case
(always write first)

f(i1, j1, ~~i2~~, j2) {

out of Bound Case:

```
if (j1<0 || j1>=m || j2<0 || j2>m)
    return -1e8;
```

Base Case Destination ←

```
if (i1==m-1 || i2==m-1) {
    if (j1==j2)
        return arr[i1][j1];
    else
        return arr[i1][j1] + arr[i][j2];
}
```

→ If both reaches same ele, pick only one

→ else pick both

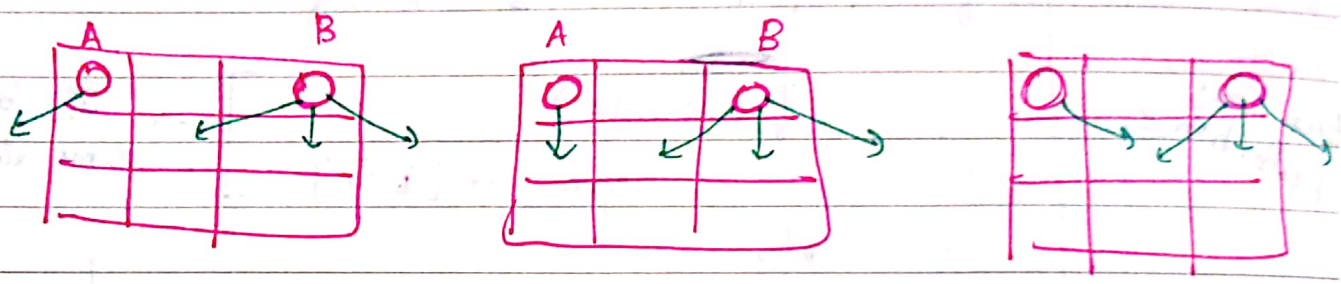
```
// Explore all paths: Alice & Bob go together
Maxi = INT_MIN;
for (Alice → -1 → +1) {
    for (Bob → -1 → +1) {
        if (j1==j2)
            ans = arr[i][j1] + f(i+1, j1+Alice, j2+Bob);
        else
            ans = arr[i][j1] + arr[i][j2] + f(i+1, j1+Alice, j2+Bob);
        Maxi = Math.max(maxi, ans);
    }
}
return Maxi;
}
```

TC: $\left(3^{n} \times 3^{n}\right)$ (Alice, Bob)

SC: O(N) → Auxiliary

Trying out all possible choices: {Different from prev pbm}

At each cell ○ ↙ ↓ ↘ (3 options)

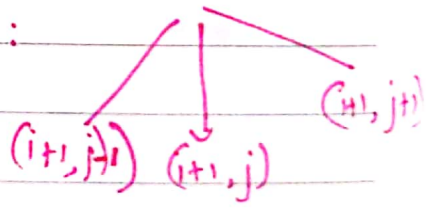


For each moment of Alice, 3 options for Bob

(9) different options at every $f(i, j_1, j_2)$

Why for loop from (-1 → 1)

⇒ because directions are:



(i+1) → remains common

(j) → (0, -1, 1) so for loop from -1 → 1

MEMOIZATION

dp array required :

$f(i, j1, j2)$ → (3 parameters)

→ 3D DP :

```
int dp[][][] = new int[m][n][n];
```

```
int main ( int m, int n, int[][] grid) {
    int dp[][][] = new int[m][n][n];
        for (int row1[][]: dp) {
            for (int row2[]: row1) {
                Arrays.fill (row2, -1);
            }
        }
    return maxchoc (0,0, n-1, m, n, grid, dp);
}
int maxchoc (int i, int j1, int j2, int m, int n, grid, dp) {
    if (j1<0 || j1>=n || j2<0 || j2>=n)
        return -1e9;
    if (i == m-1) {
        if (j1==j2) return grid[i][j1];
        else return grid[i][j1] + grid[i][j2];
    }
    if (dp[i][j1][j2] != -1)
        return dp[i][j1][j2];
```

```
int maxi = INT_MIN_VAL;
for (int alice = -1; alice <= 1; alice++) {
    for (int bob = -1; bob <= 1; bob++) {
        int ans;
        if (j1 == j2)
            ans = grid[i][j1] + maxchoc(i+1, j1+alice,
                                        j2+bob, m, n, grid, dp);
        else
            ans = grid[i][j1] + grid[i][j2] +
                  maxchoc(i+1, j1+alice, j2+bob,
                          m, n, grid, dp);
        maxi = Math.max(maxi, ans);
    }
}
return dp[i][j1][j2] = maxi;
}
```

TC: $O(m \times n \times n) \times 9$

↳ every state, 3x3 loop is ran

SC: $O(m \times n \times n) + O(m)$

↳ Auxiliary space