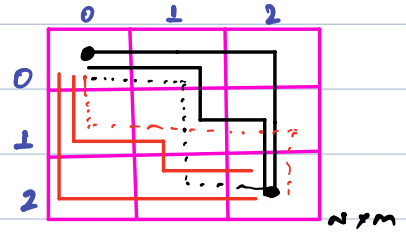
Today's agenda

- unique paths in grid #1
- minimum Path sum
- Cherry Pickup

Q) Number of ways to go from (0,0) → (n-1, m-1) cell.

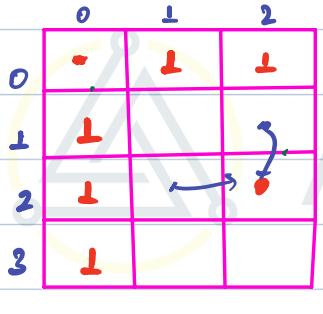
Movement allowed: cell → 1 step right or ↓ 1 step bottom

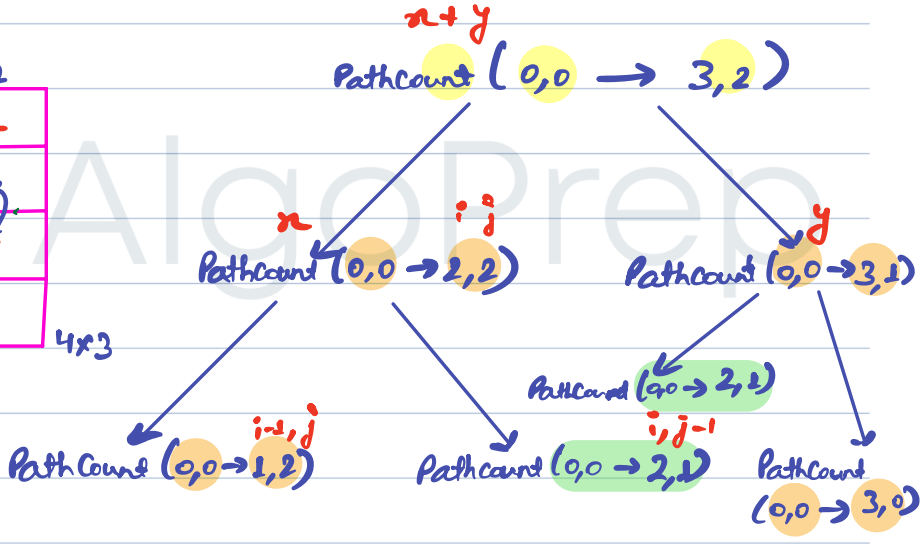
Ex:



ans = 6

RRR





//Pseudo code

```
int dP[N][M] = {-1}

int PathCount (int i, int j) {
    if (i==0 || j==0){ return 1; }
    if (dP[i][j] != -1){ return dP[i][j]; }

    int x = PathCount (i-1, j);
    int y = PathCount (i, j-1);

    dP[i][j] = x+y;
    return x+y;
}
```

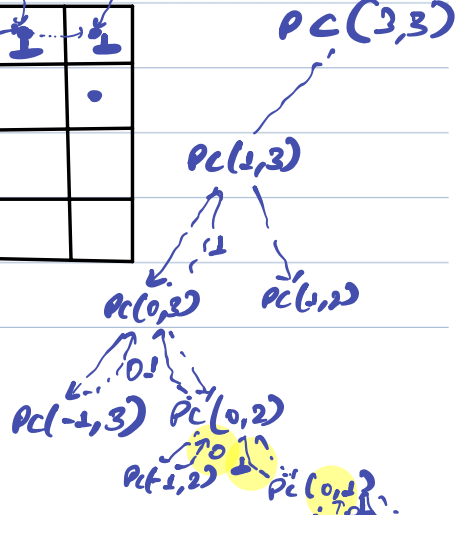
T.C: O(N*M)

S.C: O(N*M)

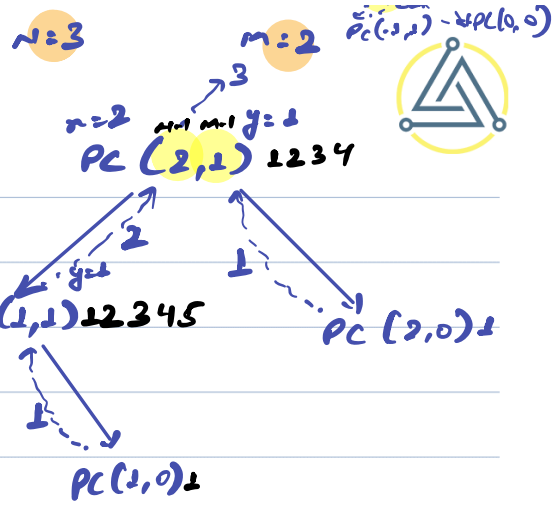
```
{ if (i==0 || j==0){ return 1; } }  ≈  { if (i<0 || j<0){ return 0; }
                                        if (i==0 && j==0){ return 1; } }
```

	1	1	1
1			
1			
1			

	0	1	2	3
0	1	1	1	1
1				1
2				
3				



```
int dp[n][m] = {-1};
int PathCount (int i, int j) {
2  if (i==0 || j==0){ return 1; }
2  if (dp[i][j] != -1){ return dp[i][j]; }
3  int x = PathCount (i-1,j);
4  int y = PathCount (i,j-1);
5  dp[i][j] = x+y;
6  return x+y;
}
```



3 dp

	0	1
0	-1	-1
1	-1	~~-1~~ 2
2	-1	~~-1~~ 3

3×2

Q) Number of ways to go from (0,0) → (n-1, m-1) cell.

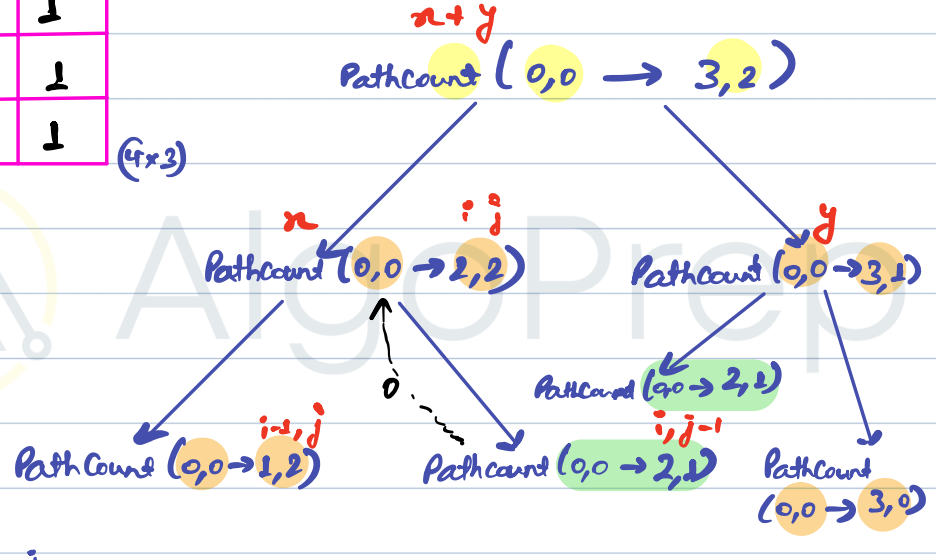
Movement allowed: cell → 1 step right or ↓ 1 step bottom

↳ mat[i][j] == 0 blocked cell, mat[i][j] == 1 unblocked cell

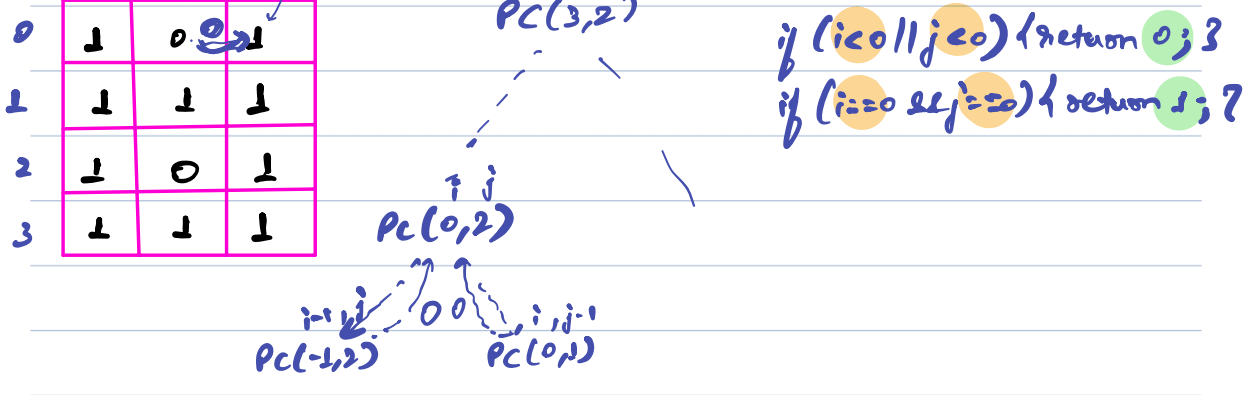
↳ Path can't go via blocked cell

	0	1	2
0	1	0	1
1	1	1	1
2	1	0	1
3	1	1	1

(4×3)



	0	1	2
0	1	0	1
1	1	1	1
2	1	0	1
3	1	1	1




```
if (i<0 || j<0) { return 0; }
if (i==0 && j==0) { return 1; }
```

//Pseudo code

```
int dp[n][m] = {-1}

int PathCount (int[][]mat, int i, int j) {
    if (i<0 || j<0) {return 0;}
    if (i==0 && j==0) {return 1;}
    if (mat[i][j] == 0) { return 0;}
    if (dp[i][j] != -1) { return dp[i][j];}
    int x = PathCount (i-1, j);
    int y = PathCount (i, j-1);

    dp[i][j] = x+y;
    return x+y;

}
```

T.C: O(n*m)

S.C: O(n*m)

Q) Minimum Path Sum

↳ Given a 2d array, filled with non-negative numbers find a Path from $(0,0) \rightarrow (N-1, M-1)$ which minimizes the total cost Path.

Movement allowed: cell → 1 step right or ↓ 1 step bottom

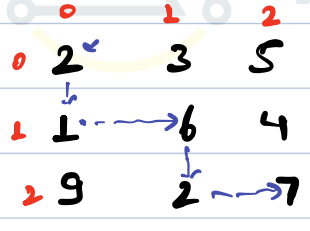
Ex:

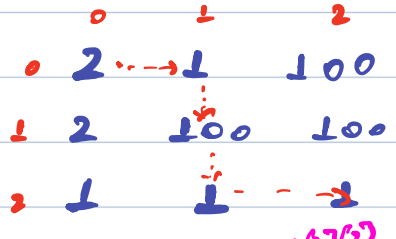
	0	1	2
0	2	3	5
1	1	6	4
2	9	2	7

→ ans: 18

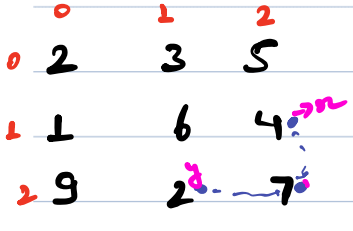
Idea 1 → wrong idea

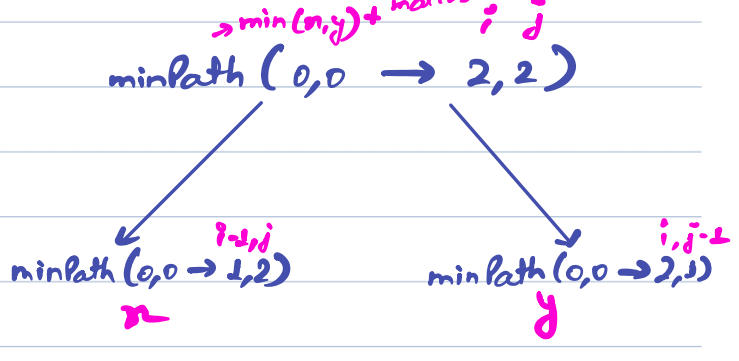
→ Choose the min cost out of 2 available options.





Idea 2





//Pseudo code

```
int dp[n][m]:{-1}

int minCostPath (int mat[n][m], int i, int j){
    if (i<0 || j<0) {return Integer.MAX_VALUE;}
    if (i==0 && j==0) {return mat[0][0]; }
    if (dp[i][j] != -1){return dp[i][j]; }

    int x = minCostPath (mat, i-1, j);
    int y = minCostPath (mat, i, j-1);

    dp[i][j] = min(x,y) + mat[i][j];
    return min(x,y) + mat[i][j];
}
```

T.C: O(n*m)

S.C: O(n*m)

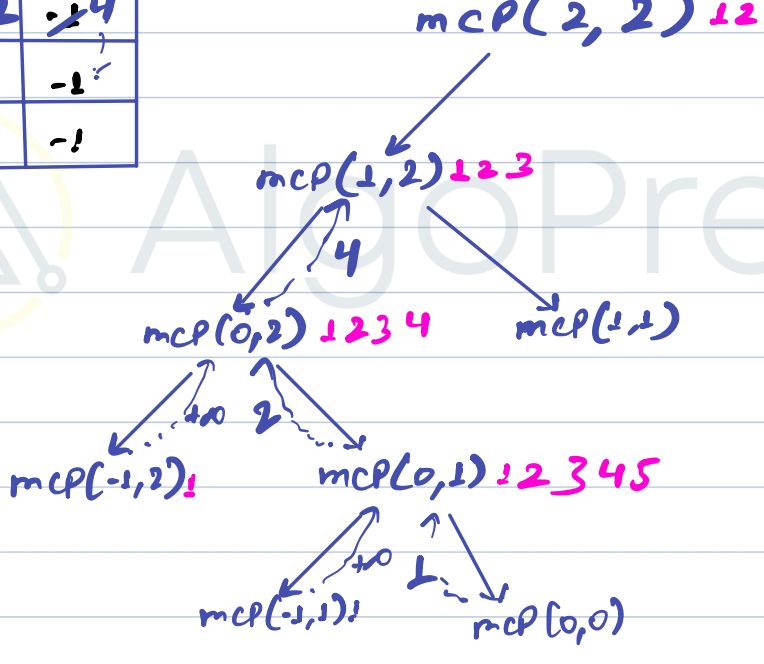
mat[3][3]:

	0	1	2
0	1	1	2
1	200	100	100
2	1	2	7

dp[3][3]

	0	1	2
0	-1	~~-1~~ 2	~~-1~~ 4
1	-1	-1	-1
2	-1	-1	-1

```
int dp[n][m]:{-1}
int minCostPath(int mat[n][m], int i, int j){
1   if(i<0||j<0){return Integer.MAX_VALUE;}
    if(i==0 && j==0){return mat[0][0];}
    if(dp[i][j]!=-1){return dp[i][j];}
2   int x = minCostPath(mat, i-1, j);
3   int y = minCostPath(mat, i, j-1);
4   dp[i][j] = min(x,y) + mat[i][j];
5   return min(x,y) + mat[i][j];
}
```



Break till 9:45 PM

Q) Cherry Pickup

↳ Given 2D array each cell contains 0, 1, -1.

→ 0 means cell is empty

→ 1 means cell is having a cherry

→ -1 means blockade

→ Return the maximum number of cherries you can collect by moving right or down and doing a full cycle.

↳ while coming back up or left

(0,0 → N-1, M-1) & (N-1, M-1 → 0,0)

Ex:

	0	1	2
0	0	1	-1
1	1	0	-1
2	1	1	1

→ ans = 5

//ideal → incorrect idea

	0	1	2
0	0	1	-1
1	1	0	-1
2	1	1	1

→ ans = 5

Ex:

	1	2	3	4	5	6
0	1 (P1, P2)	1	1	1	0	0
1	0	0	0	1	0	1
2	1	0	0	1	0	0
3	0	0	0	1	0	0
4	0	0	0	1	1	1

→ ans = 12

// Correct idea

(0,0)

Ex:

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

P1 → P2 ↓ (5, 6)

P1, P2 (n-1, m-1)

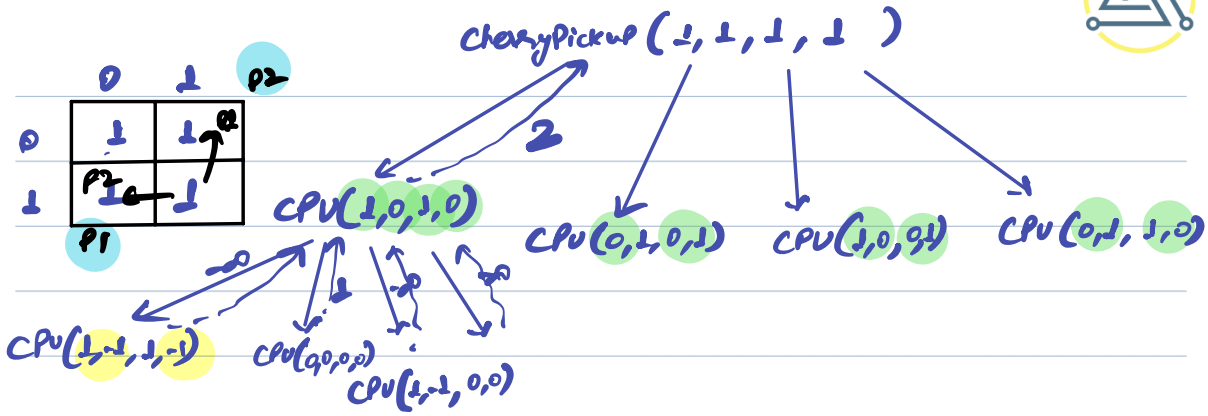
obs1: instead of doing 1 TL-BR and 1 BR-TL, Think to maximize in terms of 2 TL-BR.

obs2:

P1 (→ row1, ↳ col1)		P2 (→ row2, ↳ col2)		
R	+	R	→	x
D		D	→	y
R		D	→	z
D		R	→	A

max(x, y, z, A) + mat[i][j]

```
//Pseudo code
int dp[n][n][n][n] = {-1}
int cherryPickup (int[][]mat, int row1, int col1, int row2, int col2){
    int col2 = row1 + col1 - row2;
    if(row1<0 || row2<0 || col1<0 || col2<0 || mat[row1][col1]==-1 ||
       mat[row2][col2]==-1){ return -∞; }
    if ( row1==0 && row2==0 && col1==0 && col2==0){
        return mat[row1][col1];
    }
    if(dp[row1][col1][row2][col2]!=-1){ return dp[row1][col1][row2][col2];}
T.C:O(n^3)  int temp1 = cherryPickup (mat, row1, col1-1, row2, col2-1);
S.C:O(n^3)  int temp2 = cherryPickup (mat, row1-1, col1, row2-1, col2);
            int temp3 = cherryPickup (mat, row1, col1-1, row2-1, col2);
            int temp4 = cherryPickup (mat, row1-1, col1, row2, col2-1);
            int ans = max(temp1, temp2, temp3, temp4) +
                      mat[row1][col1] + mat[row2][col2];
            int conti=0;
            if ( row1 == row2 && col1 == col2 ){
                conti = mat[row1][col1];
            }
            else{
                conti = mat[row1][col1] + mat[row2][col2];
            }
            dp[row1][col1][row2][col2] = ans+conti;
            return ans + conti;
}
```




```
cherryPickup (mat, row1, col1-1, row2, col2-1);
cherryPickup (mat, row1-1, col1, row2-1, col2);
cherryPickup (mat, row1, col1-1, row2-1, col2);
cherryPickup (mat, row1-1, col1, row2, col2-1);
```


Obs 3: No. of steps for both the Path are same.

Ex:

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

(0,0) (3,4) (4,5) (4-1, 4-1)

row1 + col1 = = row2 + col2

⇒ col2 = row1 + col1 - row2