30/8/2023

2D- DP

Q Robber is stealing gold from houses in a village where all houses are in a straight line.

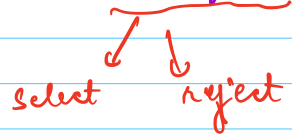
In order to avoid suspicion, the robber doesn't steal gold from 2 adjacent houses

Find max gold the robber can steal.

$A = [\overset{1}{9} \quad \overset{2}{1} \quad \overset{3}{8} \quad \overset{4}{10}]$ Ans = 19

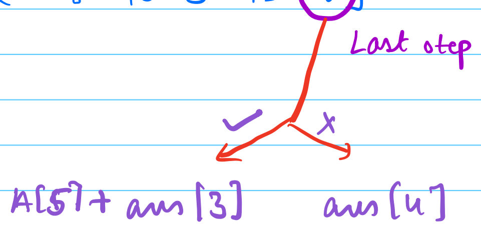
$A = [\overset{1}{2} \quad \overset{2}{8} \quad \overset{3}{10} \quad \overset{4}{3} \quad \overset{5}{15} \quad \overset{6}{6}]$, Ans = 27

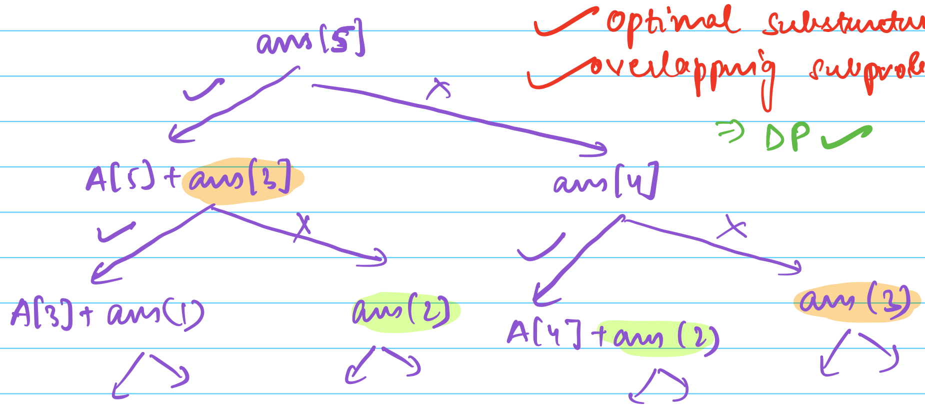
Bruteforce:- ∀ valid subsequence, check & store answer.



$A = [\overset{0}{2} \quad \overset{1}{8} \quad \overset{2}{10} \quad \overset{3}{3} \quad \overset{4}{15} \quad \overset{5}{6}]$

ans[i] → max sum from index 0 to i

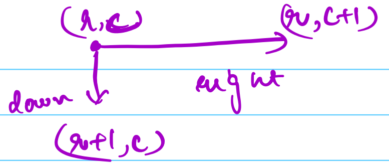




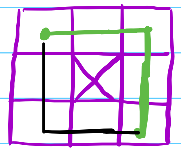
✓ optimal substructure.
✓ overlapping subprob.
⇒ DP ✓

```
ans[0] = max(A[0], 0)                              TC: O(N)
ans[1] = max(ans[0], A[1])                         SC: O(N)

for i→2 to N-1 {
    ans[i] = max(A[i] + ans[i-2] ,  ans[i-1]);
                  accept             reject
}
return ans[N-1];


                                                   TC: O(N)
    a  = max(A[0], 0)                              SC: O(1)
    b  = max(ans[0], A[1])

for i→2 to N-1 {
        c = max(A[i] +   a   ,    b    );
                  accept       reject

        a = b
        b = c                                      a   b   c
                                                       a   b   c
}
return c;
```

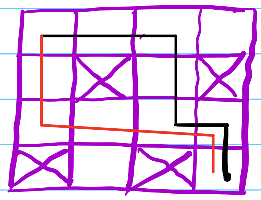
Q Given a 2D matrix
Start → Top left (0,0)
End → Bottom right



Find the no. of ways to move from top left to bottom right cell, given there are some blocked cells.



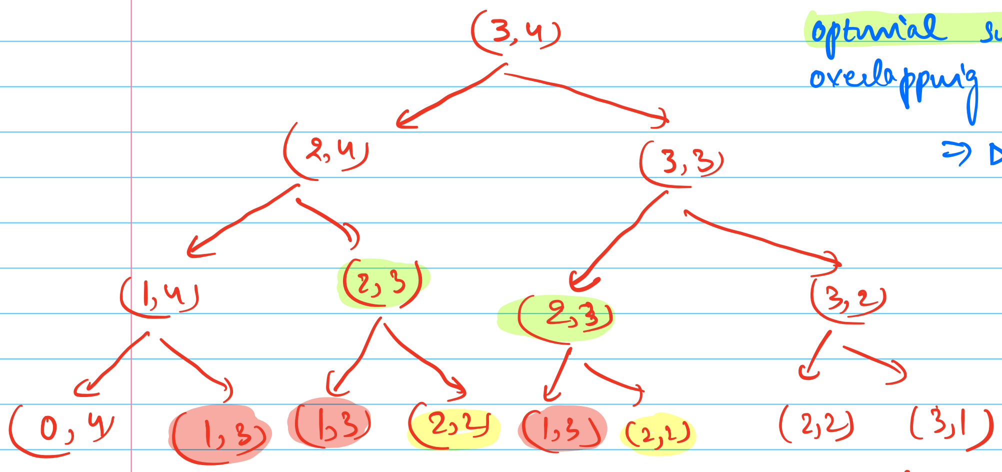
Ans = 2



A[i][j] = 1
blocked

Ans = 2

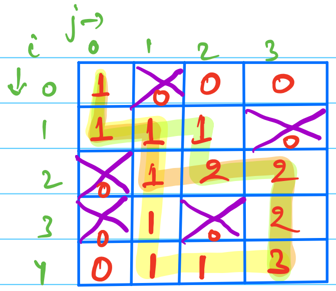
Last step.
$$\begin{tikzcd} & (i-1,j) \arrow[d] \\ (i,j-1) \arrow[r, "OR"] & (i,j) \end{tikzcd}$$



optimal substructure.
overlapping subproblem.
⇒ DP.

ways[i][j] ⇒ # ways to reach (i,j)

```
for i→0 to (N-1) {
    for j→0 to (M-1) {

        if (A[i][j]==1) dp[i][j]=0

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

        else if (i==0) { // first row
            dp[i][j] = dp[i][j-1]
        }
        else if (j==0) { // first column.
            dp[i][j] = dp[i-1][j]
        }
        else {
            dp[i][j] = dp[i-1][j] + dp[i][j-1];
        }
    }
}
return dp[N-1][M-1];
}
```



i ↓ \ j →	0	1	2	3
0	1	~~0~~	0	0
1	1	1	1	~~0~~
2	~~0~~	1	2	2
3	~~0~~	1	~~0~~	2
4	0	1	1	3

Ans = 3

TC: O(N*M)
SC: O(N*M)

only store curr & prev row

SC: O(M+M) ≃ O(M)

```
int cA = i>0 ? dp[i-1][j] : 0;
int cL = j>0 ? dp[i][j-1] : 0;

dp[i][j] = cA + cL;
```

Meet at 8:42 am IST

Q. Dungeon & Princess

Given a matrix where cell A[i][j] represents a room s.t →

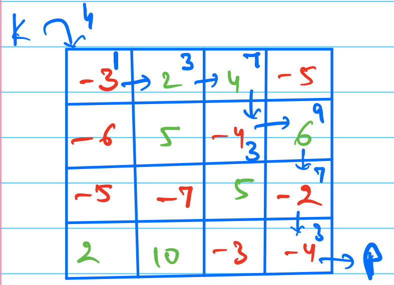
A[i][j] < 0 → There is a guard in the room that can reduce health by |A[i][j]|

A[i][j] = 0 → empty room.

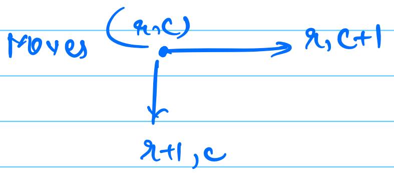
A[i][j] > 0 → There are magic herbs in the room that can increase health by A[i][j].

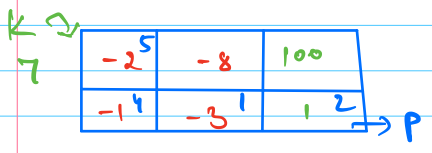
Find min initial health of knight s.t. the knight can reach the princess alive

If at any point, the health <= 0, the knight is dead.

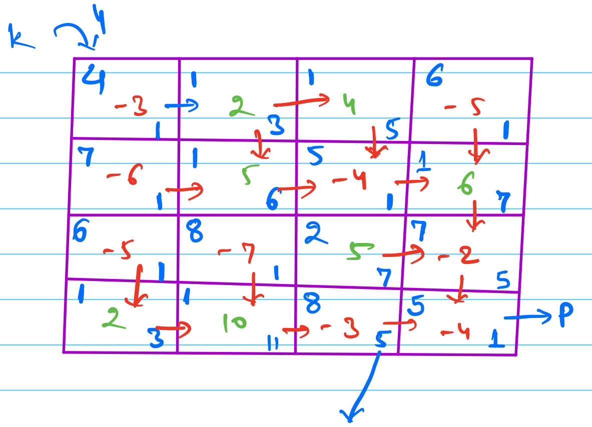


-3	2	4	-5
-6	5	-4	6
-5	-7	5	-2
2	10	-3	-4





-2	-8	100
-1	-3	1



Min health to enter = 5

Last step

ans[i][j] = min health req. to enter (i, j) to reach princess.

$x = \min(ans[i+1][j], ans[i][j+1])$

$ans[i][j] = \max(x - A[i][j], 1)$

```
for i → (N-1) to 0 {
    for j → (M-1) to 0 {
        if ( i == N-1 && j == M-1) {
            x = 1;
        }

        else if (i == N-1) {   // last row
            x = ans[i][j+1]
        }
        else if (j == M-1) {   // last column
            x = ans[i+1][j]
        }
        else {
            x = min(ans[i+1][j], ans[i][j+1]);
        }
        ans[i][j] = max( x - A[i][j], 1);
    }
}
} return ans[0][0];
```

TC: O(N×M)
SC: O(N×M)

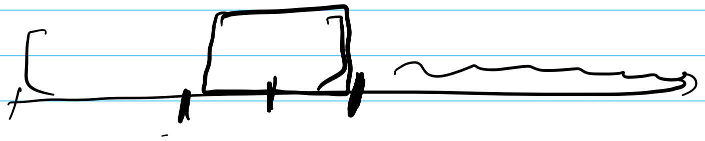
$\downarrow$

O(M)

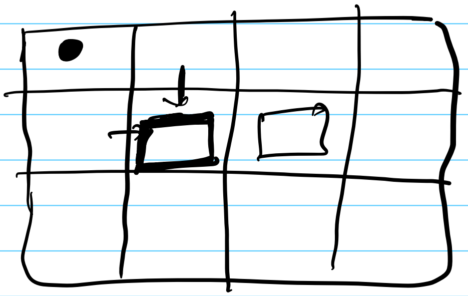
How to find path?

```
if ( ans[i+1][j] < ans[i][j+1] ) { go down }
else go right
```

```
boolean check( ----) {

}

i>0 to ∞ {
    check ( ) ...
}
```



1 — log



(N×M)

log ( )

$10^6 - 10^7 - 10^9$

10^{12}

10^4

O(N*M log(range))

```
int  solve (int[] , int currIndex) {
      if (currIndex <0)  return 0;
       if (dp[currIndex] != -1) return  dp[currIndex];
      // take
        ans = A[currIndex] + solve (A, currIndex-2);

     // skip

       ans = max(ans, solve(A, currInd-1);

       dp[currIndex]  =ans;

      return   ans;

   }
```