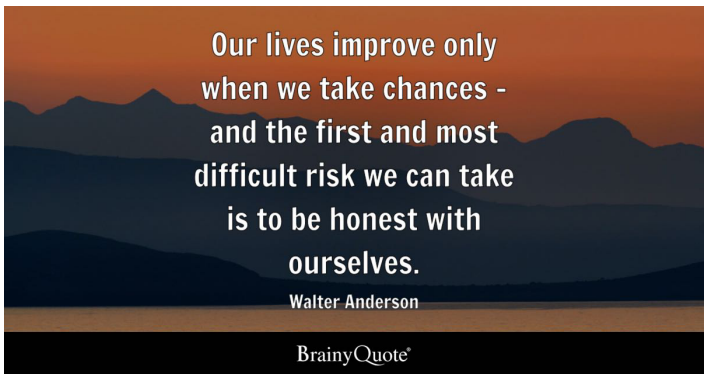
DP 2



Good Evening

Content

(a) Maximum sum without adjacent element

(b) No. of ways to go from (0,0) → last cell

(c) No. of ways to go from (0,0) → last cell (cells blocked)

Amazon
Vmware

(d) Dungeon princess

Microsoft
Google
Amazon

01. Given ar[N] calculate **max subsequence sum**

Note 1 → In a subseq 2 adj ele can't be picked

Note 2 → Empty subseq is also valid.

ar[3] = {9 14 3} ⇒ {14} = 14

ar[4] = {9 4 13 24} ⇒ {9, 24} = 33
(indices: 9 → 0, 4 → 1, 13 → 2, 24 → 3)

ar[3] = {13 14 2} Ans = 15

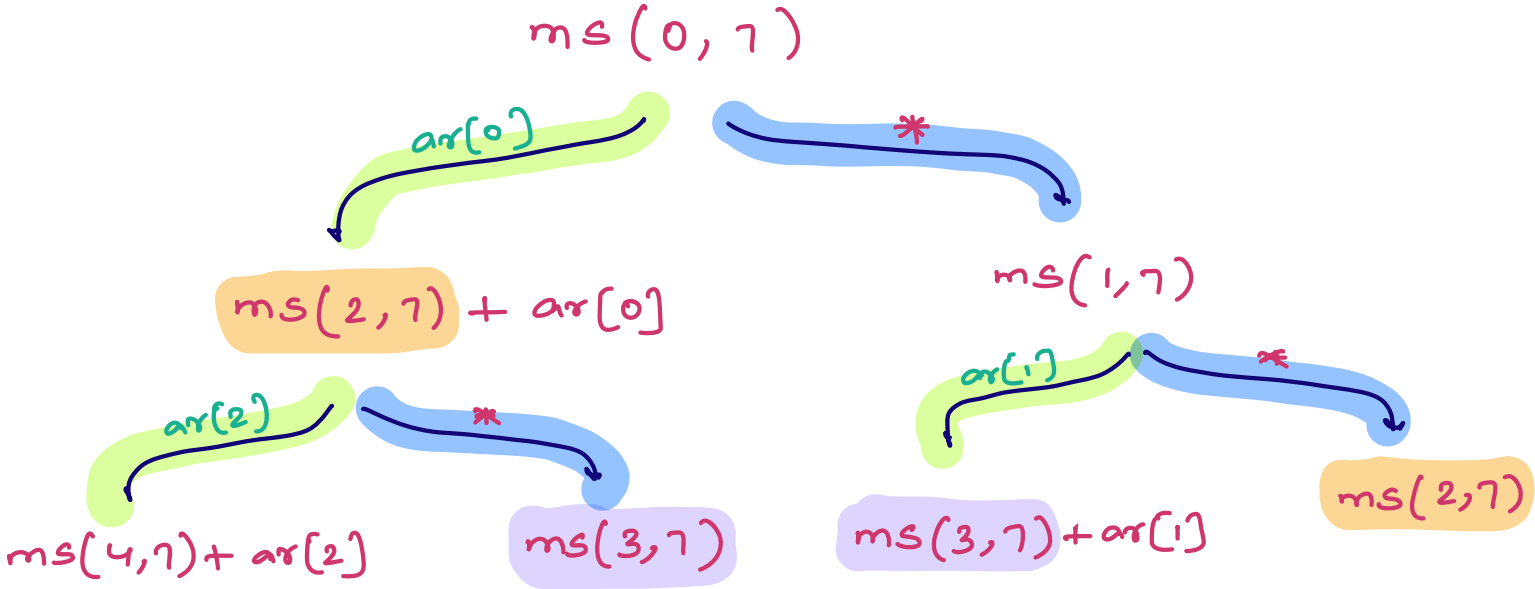
ar[4] = {-4 -3 -2 -3} Ans = 0 → Empty subseq.

Brute force → Generate all valid subsequences & you get the sum & update overall sum accordingly

arr[] = {2 -1 -4 5 3 -1 4 2}
(indices: 0 1 2 3 4 5 6 7)

→ Optimal substructure

→ Overlapping subproblems



// maxsub $\{ i, i+1, i+2 \ldots n-1 \}$

✓ ✗

$ar[i] + \{ i+2 \ldots n-1 \}$ $\qquad$ $0 + \{ i+1, i+2 \ldots n-1 \}$

```
int [] dp = -1 ;                                  → 0
int   maxsubsum ( int[] A, int i)
{
    if ( i ≥ n) return 0;

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

     dp[i] = max ( ar[i] + maxsubsum(A, i+2),
                                0 + maxsubsum(A, i+1));

    return dp[i];
}
```

* Bottom up Approach

arr[] = {9, 4, 13, 24}

dp[] = {33, 28, 24, 24}

dp[i] = max subseq sum [i-n-1] such that there are no two adj elements

```
int dp[N];
for (i=n-1; i≥0; i--)
{
    dp[i] = max(0 + dp[i+1], arr[i]+dp[i+2])
}
return dp[0];
```

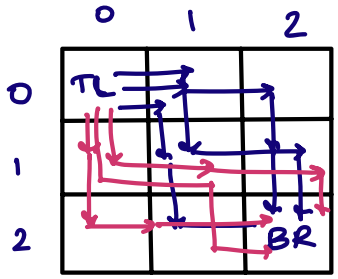
TC: O(n)

SC: O(n)

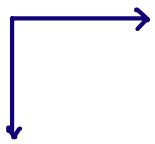
Q Number of ways to go from (0,0) → BR cell in mat[N][M]

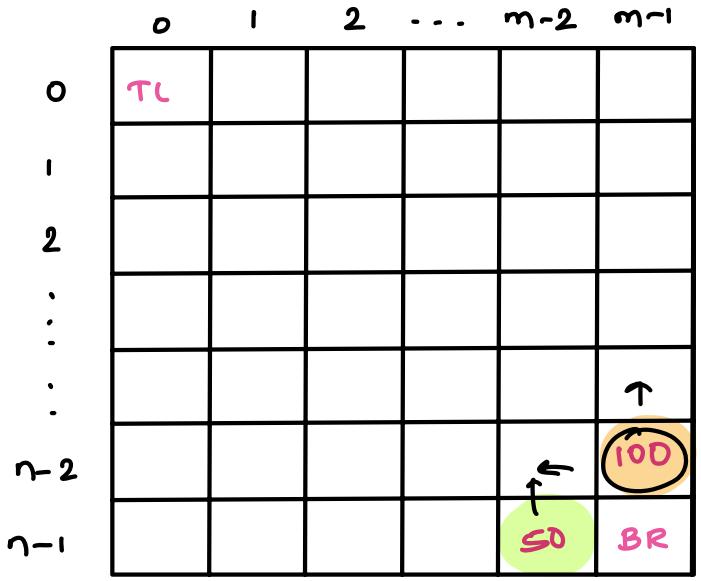
Note :- From cell we can go to right or down

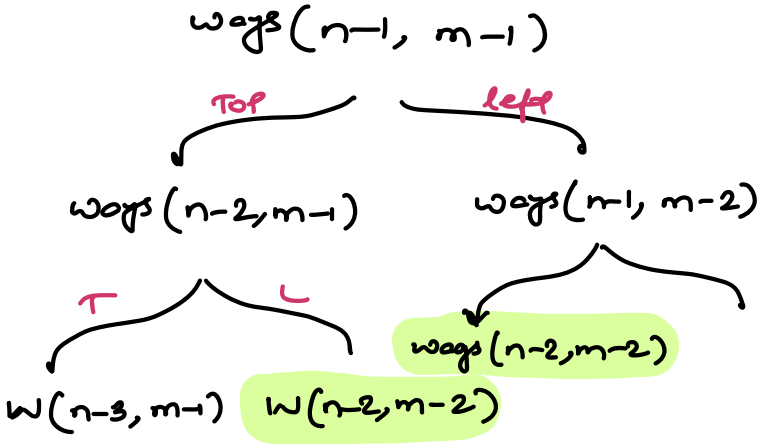
Eg :-



h h v v
h v h v
h v v h
v v h h
v h h v
v h v h





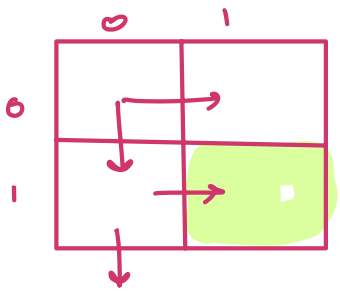


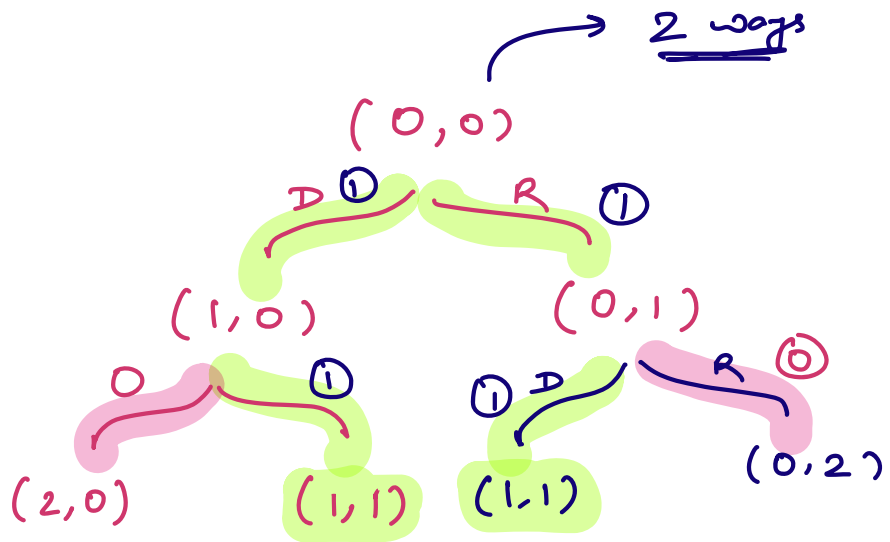
ans = 100+50

= 150

i = n-1 j = m-1

```
int ways(int i, int j)
{
    if (i==0 && j==0) return 1;
    if (i<0 || j<0) return 0;
    return ways(i-1, j) + ways(i, j-1)
}
```





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

* Bottom up Approach

	0	1	2	3	4
0	1	1	1	1	1
1	1	2	3	4	5
2	1	3	6	10	15
3	1	4	10	20	35
4	1	5	15	35	70
5	1	6	21	56	126

dp[N][M]

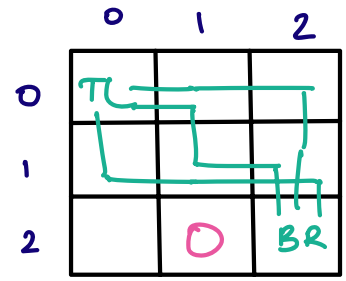
```
// initialise 0th row & 0th col with 1
for(i=1; i<N; i++){
    for(j=1; j<M; j++){
        dp[i][j] = dp[i-1][j] +
                   dp[i][j-1];
    }
}
return dp[N-1][M-1];
```

Q Number of ways to go from (0,0) → BR cell in mat[N][M]

Note :- From cell we can go to right or down

Note :- Certain cells of contains 0 , indicates blocked cell , we can't travel through that path

Eg :-



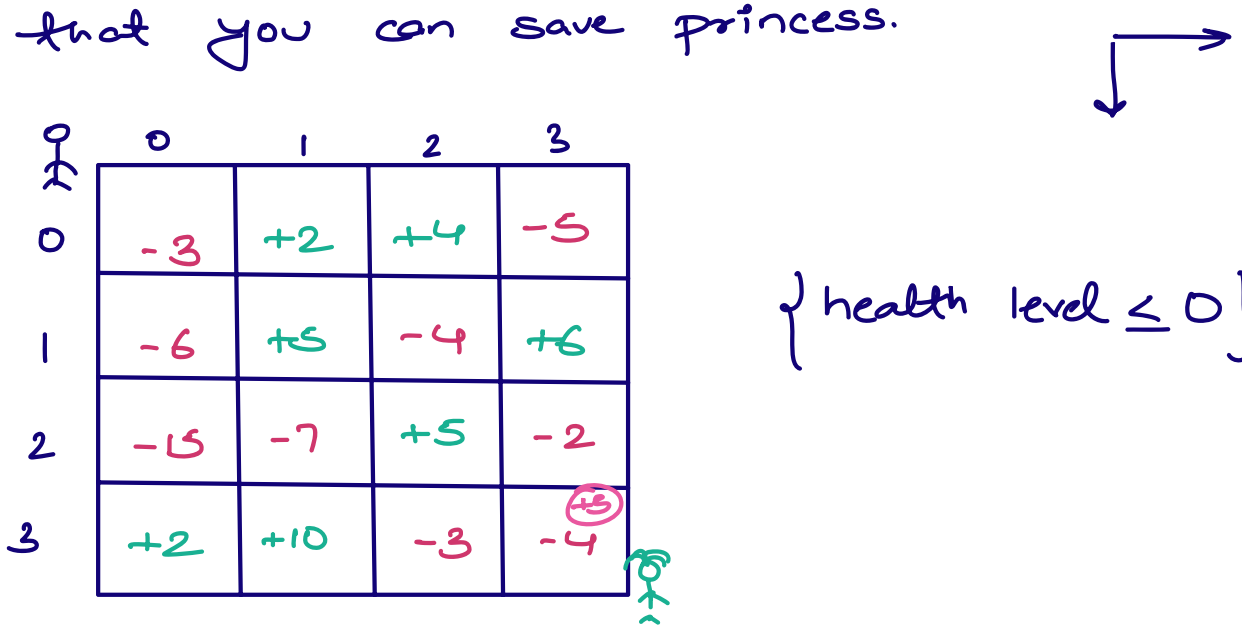
H H V V
H V H V
V H H V

```
int ways(int i, int j)
{
    if(i==0 && j==0) return 1;
    if(i<0 || j<0 || ar[i][j]==0) return 0;
    if(dp[i][j] != -1) return dp[i][j]

    dp[i][j] = ways(i-1, j) + ways(i, j-1);

    return dp[i][j]
}
```

10:24 pm

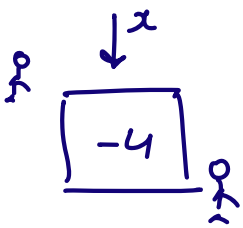
* Dungeon princess

Find the minimum health level to start with so that you can save princess.



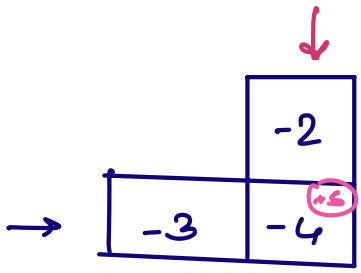
	0	1	2	3
0	-3	+2	+4	-5
1	-6	+5	-4	+6
2	-15	-7	+5	-2
3	+2	+10	-3	-4 (+5)

{health level ≤ 0} → Die



$x + (-4) = 1$

$x = 1 + 4 = 5$



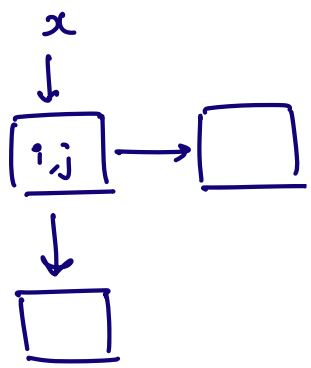
what min health with which I should enter in −2 such that I leave this cell with health as 5

$x + (-2) = 5$

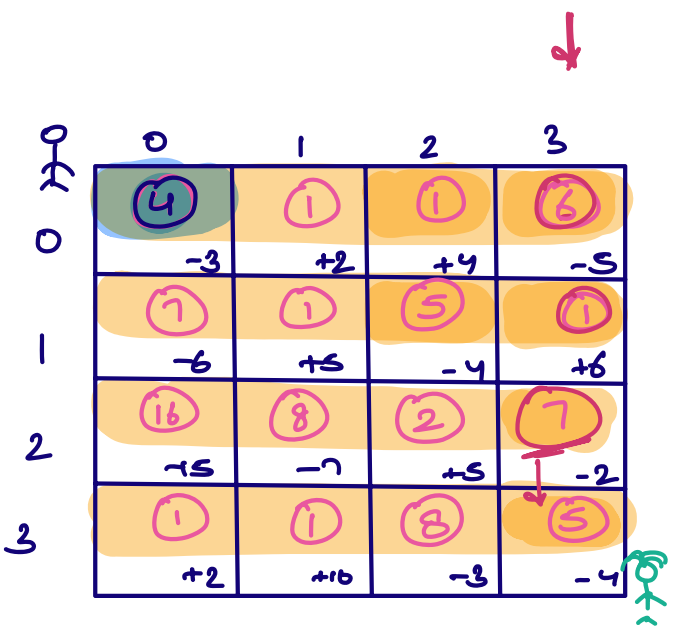
$x = 7$

$y + (-3) = 5$

$y = 8$



$x + arr[i][j] = \min$ (min energy req to reach $(i+1, j)$, min energy req to reach $(i, j+1)$)



$i = n-2 ; i \geq 0 ; i--$

$x + arr[i][m-1] = dp[i+1][m-1]$

$x = dp[i+1][m-1] - arr[i][m-1]$

$x = Max(1, Min(dp[i+1][j], dp[i][j+1]) - arr[i][j])$

dp[N][M]

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

```
if (arr[N-1][M-1] ≥ 0)   dp[N-1][M-1] = 1
else
        dp[N-1][M-1] = |arr[N-1][M-1]| + 1

// fill the last col
for ( i = N-2; i ≥ 0; i--) {
    dp[i][M-1] = Max( 1, dp[i+1][M-1] - arr[i][M-1]);
}
// fill the last row
for ( j = M-2; j ≥ 0; j--) {
    dp[N-1][j] = Max( 1, dp[n-1][j+1] - arr[n-1][j])
}
for ( i = n-2; i ≥ 0; i--)
    for (j = n-2; j ≥ 0; j--) {
        dp[i][j] = Max( 1, min(dp[i+1][j], dp[i][j+1]) -
                                                arr[i][j] );
    }
return dp[0][0];
```

// top down approach

// handle last row & last col.

```
int solve ( i, j , dp[N][M], ar[N][M])
{
    if ( i== n-1 && j==m-1){
    | return Max( 1, |ar[n-1][m-1]| +1 ),.
    }

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

  f1 = solve ( i+1, j, dp, ar)

  f2 = solve ( i, j+1, dp, ar)

  ans= Max( 1, min(f1,f2) -ar[i][j]);

  dp[i][j] =ans;

  return ans;
}
```

TC : O(N*M)

SC : O(N*M)