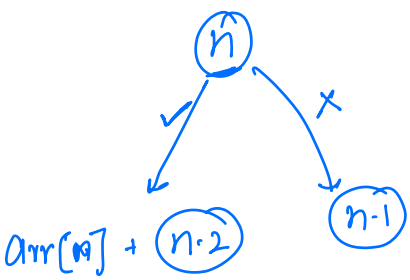
Q) Find 'an' arr . Find max subsequence sum.

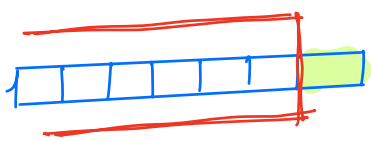
* - You are not allowed to pick adjacent elements.

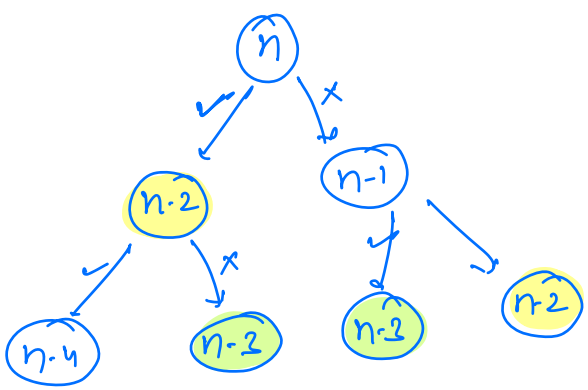
arr - [9 4 13] ans : 22

arr - [9 4 13 24] ans = 33.

B.f. → Consider all the valid subsequence ⇒ Backtracking.

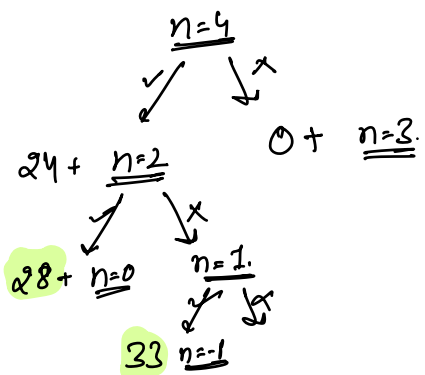






overlapping sup-problems.

arr - [9 4 13 24]
1 2 3 4



$$maxSum(i) = max\begin{cases} arr[i] + maxSum(i-2) \\ 0 + maxSum(i-1) \end{cases}$$

dp[N] → initialize INT.MIN

```
int maxSum( int[] arr, int i, int[] dp){
    if (i<0) { return 0 }
    if( dp[i] != INT.MIN) { return dp[i] ; }
    f1 = arr[i] + maxSum (i-2,dp); // include
    f2 = 0 + maxSum (i-1,dp); //exclude.
    ans = max ( f1, f2)
    dp[i] = ans ;
    return ans ;
}
```

[T.C → O(N)
S.C → O(N)]

Bottom-up.

dp[i] → max subsequence sum (0--i)

```
for ( i → 0 to N-1. ){
    dp[i] = max(arr[i] + dp[i-2] , 0 + dp[i-1])
}
```

T.C → O(N)

S.C → O(1)

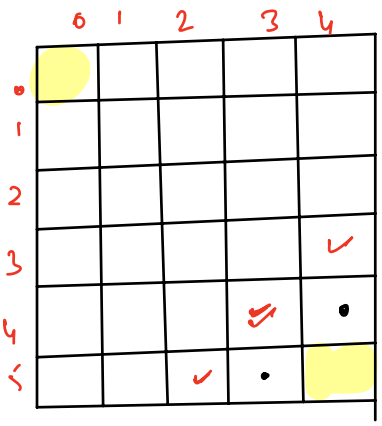
{ similar to fibonacci problem }

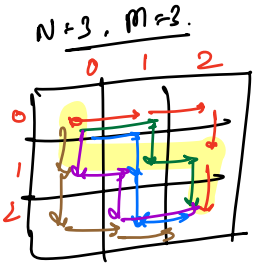
Houserobber.

Htodo.

Q2) Imagine there is matrix of N * m dimensions. Initially you are at (0,0). You can move in horizontal & vertical directions with 1 step at a time.

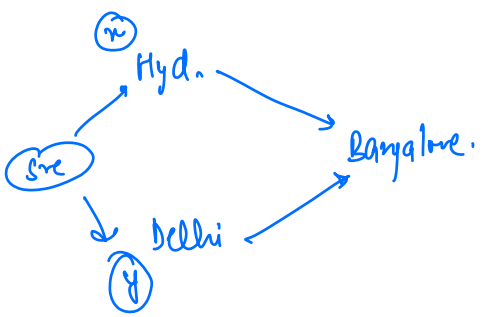
Total no. of ways from (0,0) to (N-1, m-1).

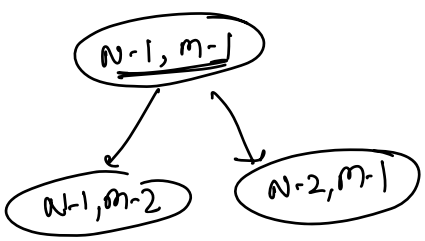


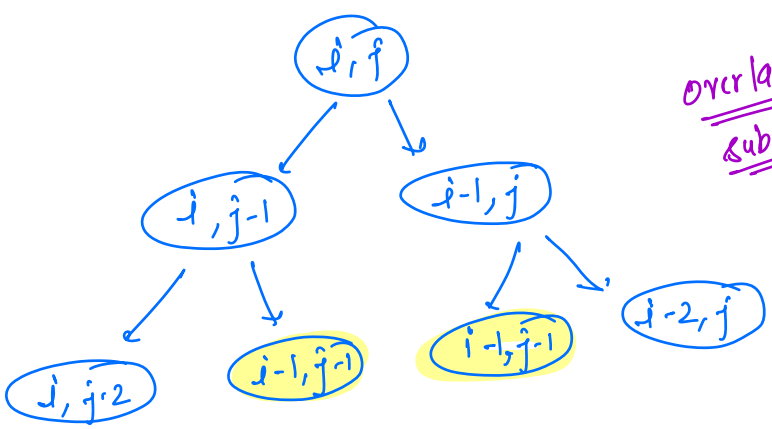


RRDD
RDRD
RDDR
DDRR
DRDR
DRRD

ans = 6.







overlapping sub-problems

Ways (i, j) = Ways (i-1, j) + ways (i, j-1)

```
int dp[N][M]; // initialize with -1

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

(N-1) (M-1)

{T.C → O(N*m)
S.C → O(N*m)}

Bottom-up:

	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

total no. of ways to reach that spot → 0.

⇓

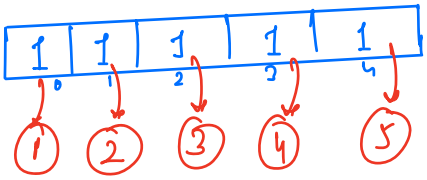
that spot is unreachable.

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

{S.C → O(N*m) T.C → O(N*m)}

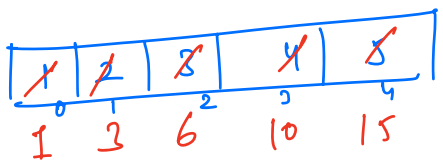
N = 6, m = 5

dp → | 1 | 1 | 1 | 1 | 1 |

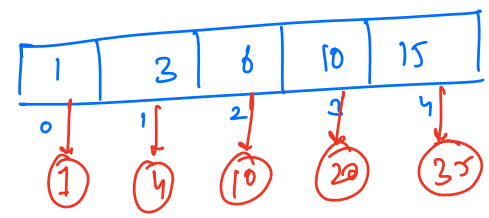


idea → similar to pfsum.

dp- | ~~1~~ | ~~2~~ | ~~3~~ | ~~4~~ | ~~5~~ |



dp- | 1 | 3 | 6 | 10 | 15 |



todo.

[SC- O(N)
T.C- O(N*m)]

Q:

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

1 → non-blocked cell

0 → blocked cell

Total no. of ways from (0,0) to (n-1, m-1)

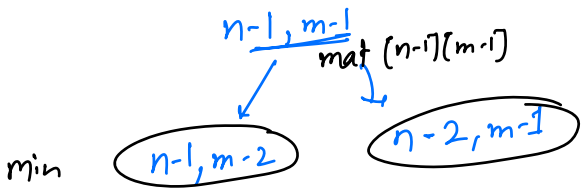
→ right

↓ down.

```
if (mat[i][j] != 0) {
    ways(i,j) = ways(i,j-1) + ways(i-1,j)
}
else {
    ways(i,j) = 0
}
```

Q: Min cost from (0,0) to (N-1, M-1). → right. ↓ down.

	0	1	2	3	4
0	2	1	3	7	3
1	3	3	4	8	3
2	0	5	1	2	1
3	0	2	3	1	3
4	4	1	5	2	3



$$minlost(i,j) = cost[i][j] + Min\left(minlost(i, j-1),\ minlost(i-1, j) \right)$$

Base-case.

top-down.

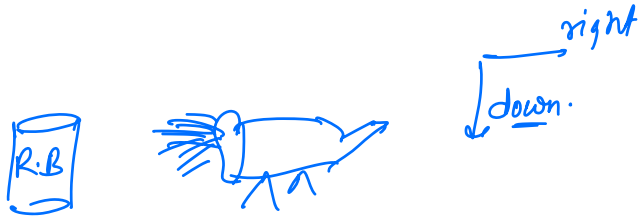
```
int dp[N][M];   → initialise with INFMAX

int minlost( int[][] arr, int i (N-1), int j (M-1), int[][] dp){
    if(i==0 && j==0){ return arr[0][0] }
    if(i<0 || j<0){ return INT_MAX }
    if(dp[i][j] != INT_MAX){ return dp[i][j] }
    dp[i][j] = arr[i][j] + min( minlost(arr, i, j-1, dp),
                                minlost(arr, i-1, j, dp) );
    return dp[i][j];
}
```

T.C → O(N*M), S.C → O(N*M)

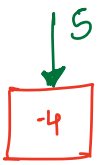
Dungeons & Princess

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



{ health-level ≤ 0 } $\Rightarrow$ dic.

Find the minimum health level to start with?



$x + (-4) = 1$

$x - 4 = 1$

$x = 5$.

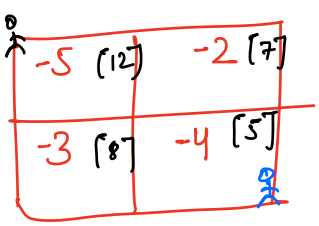
$x + (-3) = 5$

$x \cdot 3 = 5$

$x = 8$.

$x - 2 = 5$

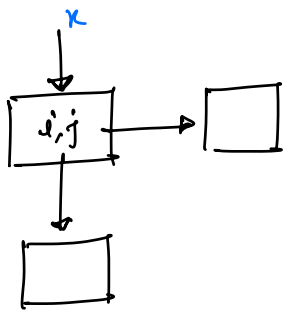
$x = 7$.



$x + (-5) = \min(7, 8)$

$x - 5 = 7$

$x = 12$.



$$x + arr[i][j] = \min \begin{cases} \text{minEnergy required to enter } (i, j+1), \\ \text{minEnergy required to enter } (i+1, j) \end{cases}$$

$$\text{minEnergy}(i,j) = \max\left(1, \min\left[\text{minEnergy}(i, j+1), \text{minEnergy}(i+1, j)\right] - arr[i][j]\right)$$

< 0, then minEnergy $(i,j) = 1$.

ans.

	0	1	2	3
0	(4) -3	(1) +2	(7) +4	(6) -5
1	(3) -6	(1) +5	(5) -4	(1) +6
2	(16) -15	(8) -7	(2) +5	(7) -2
3	(1) +2	(1) +10	(8) -3	(5) -4

$x + 6 = 7$.

$x = 7 - 6 = 1$.

0, 0

```
int solve( mat, i, j, int[][] dp){
{
    if( i == n-1 && j == m-1 ){
    {
        return max( 1, 1 - mat[i][j] );
    }
    if( dp[i][j] != -1 ){ return dp[i][j] };

        //Solve it by recurrive call
}
```

[T.C → O(N*m), S.C → O(N*m)]