Todays Content

a. Maximum sum without adjacent elements

b. No: of ways to go from (0,0) → another cells

c. No: of ways to go from (0,0) → another cells blocked cells

d. Min path sum in a 2d Matrix

Quick Revisim:

Step: 1 Try it with Recursion

Step: 2 Overlapping SubProblems

Step: Dp

a) dp table : Initialize with invalid val, // not possible to be output.

b) If subproblem solve it time : Solve it, store it, return it

c) If subproblem already solved : retrive & return it

d) TC : (# No: of SubProblems) * (TC for each subproblem)

Q) Given ar[N] calculate max subsequence sum. {any ele, order index}

Note1: In a subseq 2 adj elements cannot be picked

Note2: Empty SubSequence is also valid = {0}

ar[3] = { 9 14 3 } = ans: 14

ar[4] = { 9 4 13 24 } (0 1 2 3) = ans: 33

ar[3] = { 13 14 2 } (0 1 2) = ans: 15

ar[4] = { -4 -3 -2 -3 } = ans: 0

Idea:

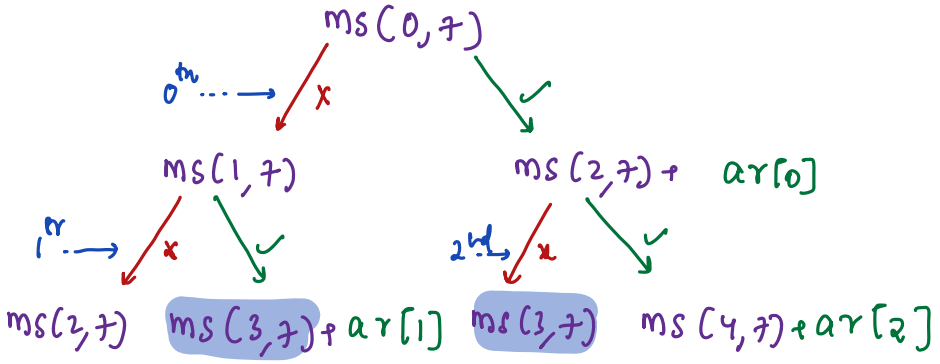
1. sum of all odd indices } * fail
   Sum of all even indices }

2. Generate all subsequences:
   Consider only non: adjacent Subseq sum
   & get overal max.
   TC: $O(2^N * N)$

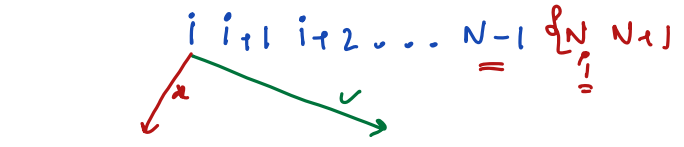
Idea:

ar[8] = { 2 -1 -4 5 3 -1 4 2 } (0 1 2 3 4 5 6 7)

1. Recursion
2. Overlapping Sub Problems




```
int manSub(int ar[], int i){ // max non-adjacent subseq sum from [i ... N-1]
    if(i >= N){ return 0 }
    return max{ manSub(ar, i+1)
                ar[i] + manSub(ar, i+2) }
}
```



i+1 i+2 ... N-1 ar[i] + i+2 ... N-1

manSub(ar, i+1) ar[i] + manSub(ar, i+2)

Note: In Recursive code, overlapping apply dp.

dp[i] = max non subsequence sum from {i .. N-1}

dp[N] = max sum {N .. (N-1)} *

dp[N-1] = max sum {N-1 ... (N-1)} ✓

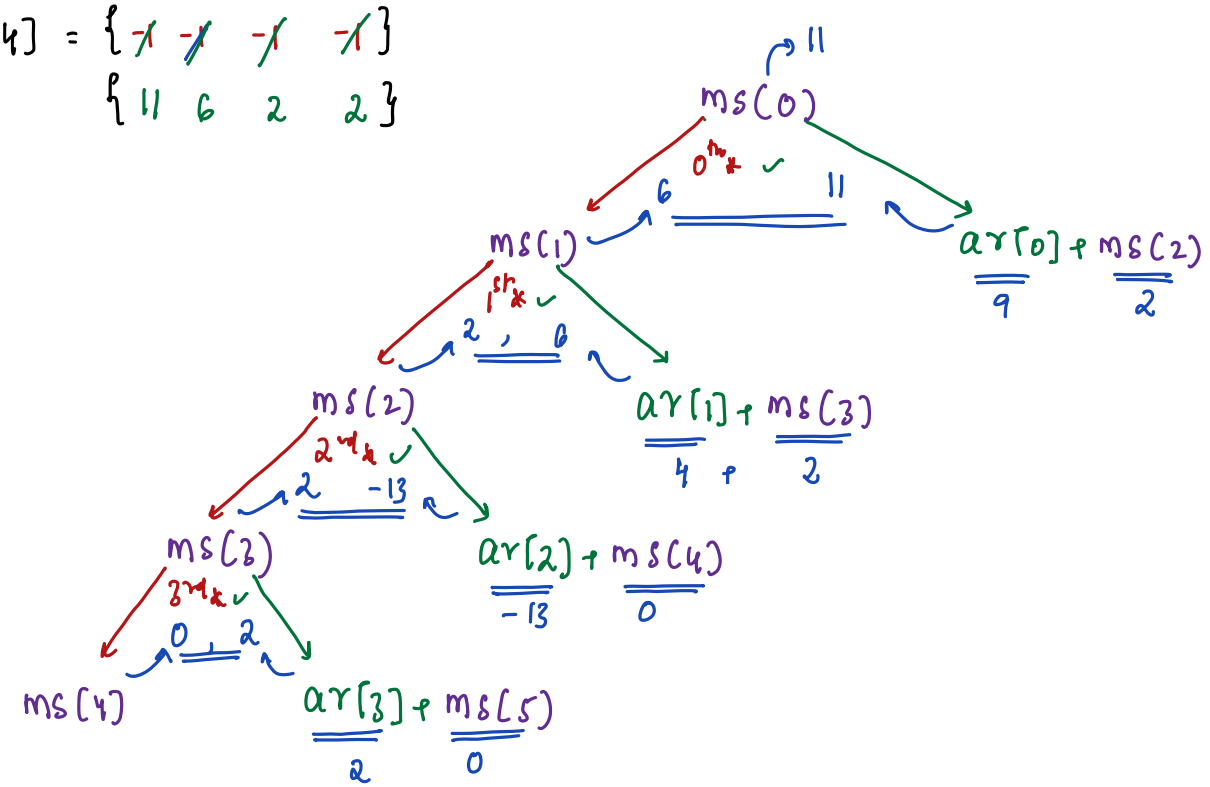
```
int dp[N] = -1; // Invalid
int maxSub(int A[], i){
    if( i >= A.length) { return 0}
    if( dp[i] == -1) {
        dp[i] = max { maxSub(ar, i+1)  ar[i] + maxSub(ar, i+2) }
    return dp[i];
}
```

10:15 break

TC: O(N) * 2 = O(N) SC: O(N)

Dry Run:

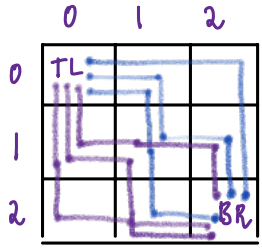
	0	1	2	3
ar[4] =	9	4	-13	2
dp[4] =	~~-1~~	~~-1~~	~~-1~~	~~-1~~
	11	6	2	2



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

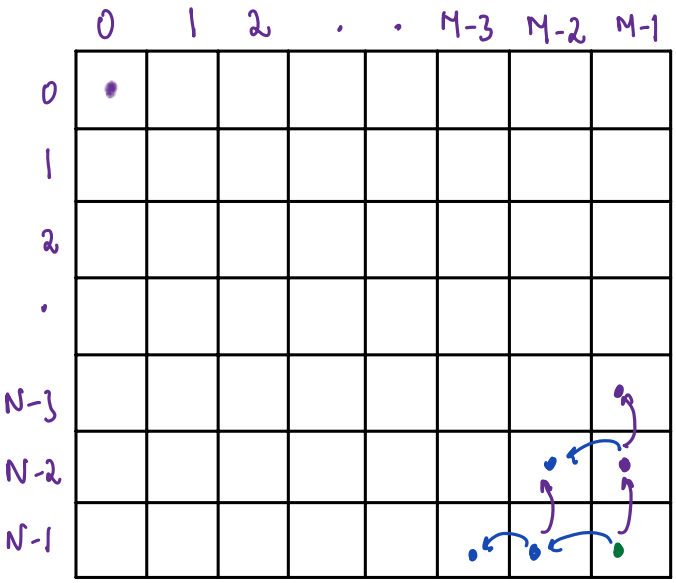
Note: From cell we can goto right or down

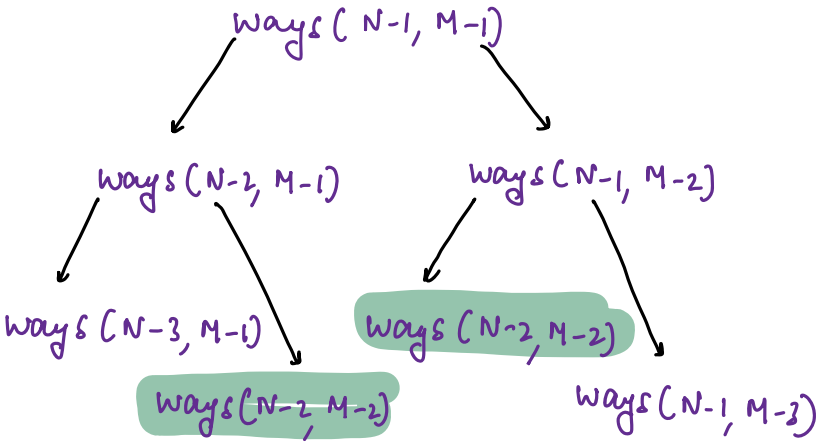
Eg:



ans = 6

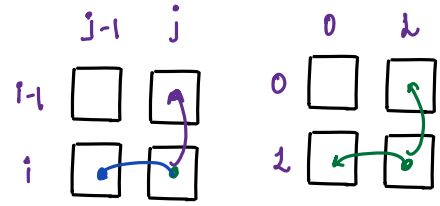
Eg: mat[N][M]





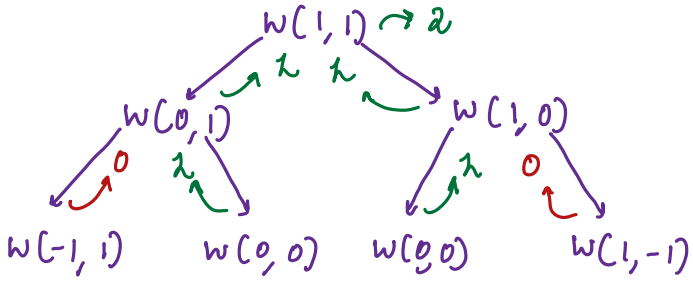
1. Recursion
2. Overlapping Sub Problems.

```
int ways(int i, int j){// Number of ways to reach i j from 0,0
    if(i<0 || j<0){return 0}
    if(i==0 && j==0){return 1}
    return {ways(i-1,j) + ways(i,j-1)}
}
```



Note: ways(0,0): No: of ways to reach (0,0) from (0,0): 1 way: Stay there

Similar to stair case problem.



Note2: Optimise with dp.

dp[i][j] = no: of ways to reach i, j from 0, 0

dp[N][M] = ways to reach N, M from 0,0 *

dp[N-1][M-1] = ways to reach N-1, M-1 from 0,0: ✓

```
int dp[N][M] = -1; //Invalid.
int ways(int i, int j){
    if(i<0 || j<0) {return 0}
    if(i==0 && j==0) {return 1}
    if( dp[i][j]==-1){
    } dp[i][j] = ways(i-1,j) + ways(i,j-1)
    return dp[i][j]
}
```

TC: (N*M)*(1) = O(N*M)

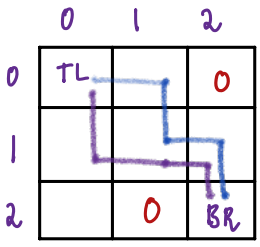
SC: O(N*M)

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

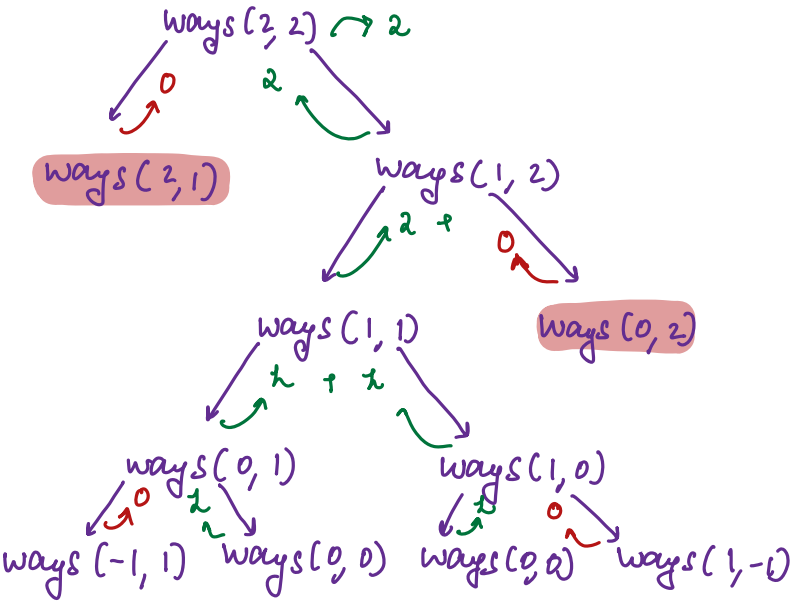
Note: From cell we can goto right or down

Note: Certain cell contain 0, indicates blocked cells, we cannot travel via blocked cells

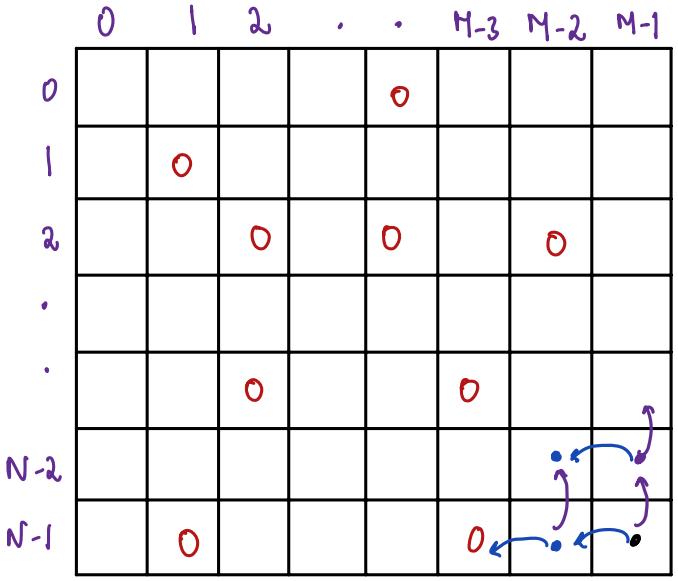
Eg:

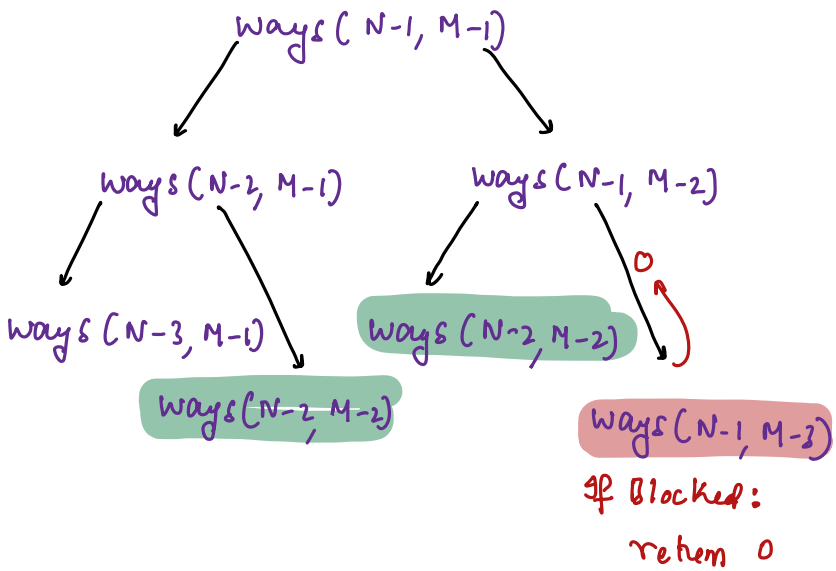


ways = 2



Eg: mat[N][M]






```
int dp[N][M] = -1; //Invalid.
int ways(int i, int j, int mat[][]){
    if(i<0 || j<0) {return 0}
    if(mat[i][j]==0) {return 0} //condition should come above below condition?
    if(i==0 && j==0) {return 1}
    if(dp[i][j]==-1){
    } dp[i][j] = ways(i-1,j) + ways(i,j-1)
    return dp[i][j]
}
```

TC: (N*M)*(1) = O(N*M) SC: O(N*M)

38) Min cost path from (0,0) → (BR)

Note: from cell we can goto right or down

Note: When we cross/land on a cell, cost associated it

Calculated min cost required to from (0,0) → BR

Eg:

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

: TODO

Eg: mat[N][M]

	0	1	2	.	.	.	M-2	M-1
0								
1								
2								
.					.	.	.	2
.					.	3	9	3
N-2					.	7	8	6
N-1					.	4	3	10