Todays Content:

1. Man SubSeq Sum
2. Ways TL → BR

1Q) Given arr[N] calculate max subseq sum

Note1: In a subseq 2 adj elements cannot be picked

Note2: Empty sequence is also valid

arr[3] = 9 14 3 : ans = 14

arr[4] = 9 4 13 24 : ans = 33

arr[3] = 13 14 2 : ans = 15

arr[4] = -4 -3 -2 -3 : ans = 0

Idea: max{sum of even index, sum of odd index}

* won't work

Idea2: Generate all subseq

a) Neglect seq with adj elements

b) Out of all valid subseq get max sum

TC: $2^n * n$ // n → to check valid subseq or not

Idea:

	0	1	2	3	4	5	6	7
arr[8] =	2	-1	-4	5	3	-1	4	2

// max subseq sum from [0-7] without adj elements

1. subproblems
2. overalpping subpro

- ms(0 - 7) : $\underbrace{\text{(0 - 7)}}_{dp(7)}$
  - Skip 7th ind → ms(0 - 6) : dp(6)
    - skip 6th → ms(0-5) : dp(5)
    - Pick 6th → ms(0-4) + 4 (arr[6])
  - Pick 7th index → ms(0-5) + 2 (arr[7])
    - skip 5th → ms(0-4) : dp(4)
    - Pick it 5th → ms(0-3) + -1 (arr[5])

Dp Steps:

dp state : dp(i) = max subseq sum from [0, i] without adj elements

dp exp : // Calculate state with subproblems

0 1 2 ... i-2 i-1 i

- leave it → ms(0, i-1)
- Pick ith → ms(0, i-2) + arr[i]

$$dp(i) = \max(dp(i-1),\ dp(i-2) + arr[i])$$

Final ans: We want max subsum from {0, n-1} : dp[n-1]

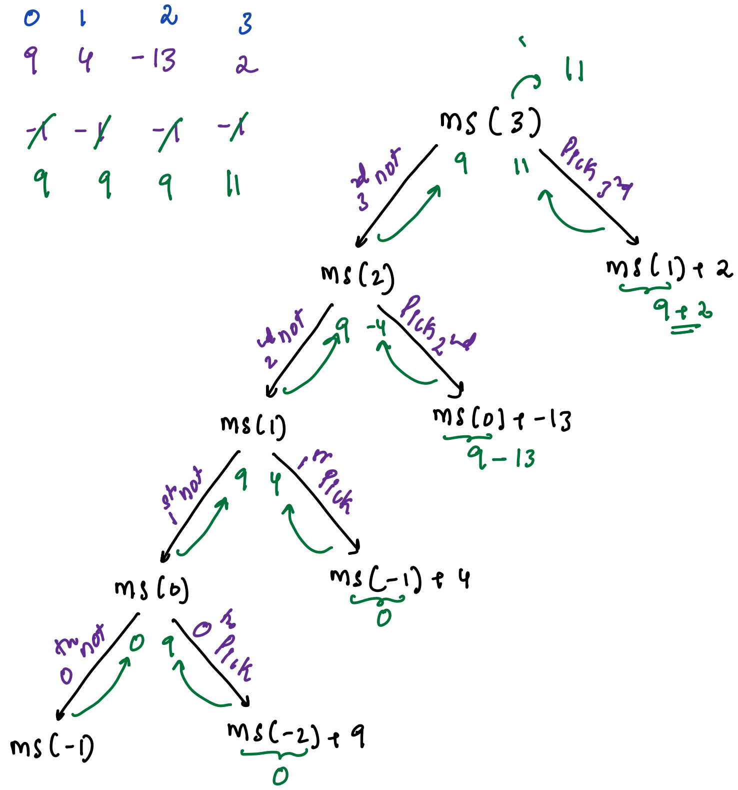
Table size: int dp[n]   TC: #states * TC for each state   SC: O(N+N)

↳ #n*1 : O(N)

<u>Code</u>:

```
int dp[n] = INVALID/-1/...
int maxSub( int A[], int i ){ // max subseq from [0, i]
    if( i<0 ){ // negative index, no subseq return 0}
    if( dp[i]==-1){ // Calculating 1st time
        dp[i] = max(maxSub( A, i-1), maxSub( A, i-2) + A[i])
    }
    return dp[i]
}
int main(){
    int A[n]
    maxSub( A, n-1) // max sub from 0, n-1
}
```

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



2-way

	0	1	2	3	4	5	6	7
ar[8] =	2	-1	-4	5	3	-1	4	2

ms[0 - 7] : dp(0)

0th not → ms[1-7] : dp(1)

0th Pick → ms[2-7] + ar[0]

ms[1-7]: 1st not → ms[2-7] : dp(2)

ms[1-7]: 1st Pick → ms[3-7] + ar(1)

ms[2-7] + ar[0]: 2nd not → ms[3-7] : dp(3)

ms[2-7] + ar[0]: 2nd Pick → ms[4-7] + ar[2]

dpState : dp(i) = manSubSum from {i, n-1}

dpExpression : dp(i) = i i+1 i+2 ..... n-1

leave / Pick

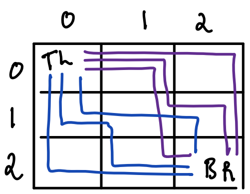
dp(i) = man (dp(i+1) ar[i] + dp(i+2))

final ans : man sub sum from [0, n-1] : dp(0) Table size : dp[n]

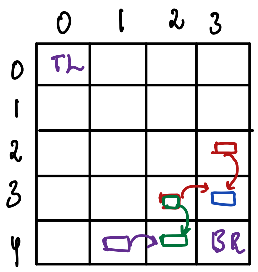
```
int dp[n] = -1
int manSub ( int A[], int i ) { // man subseq , [i, n-1]
    if( i >= n ) { // Invalid return 0 }
    if( dp[i] == -1 ) {
        dp[i] = man ( manSub ( i+1 ), ar[i] + manSub ( i+2 ))
    }
    return dp[i]
}
```

2Q) Number of ways to go from $(0,0) \rightarrow$ BR cells, mat[N][M]

Note: Cell $\longrightarrow$ right or $\downarrow$ bottom



\# ways = 6



TODO: Fix BR & Try

• subproblems
• overlapping

TL BR

$\text{all}(\underbrace{0,0 \longrightarrow 4,3}_{dp(4,3)})$

$\text{all}(0,0 \longrightarrow 3,3) : dp(3,3)$ $\qquad$ $\text{all}(0,0 \longrightarrow 4,2) : dp(4,2)$

$\text{all}(0,0 \longrightarrow 3,2)$ $\quad$ $\text{all}(0,0 \longrightarrow 2,3)$ $\quad$ $\text{all}(0,0 \longrightarrow 4,1)$ $\quad$ $\text{all}(0,0 \longrightarrow 3,2)$

Dp Steps:

Dp State: $dp(i,j)$ = \# ways to go from $(0,0) - (i,j)$

Dp Expression: $dp(i,j) = dp(i-1,j) + dp(i,j-1)$

$$\begin{tikzcd}
 & (i-1,j) \arrow[dr, "1"] & \\
(0,0) \arrow[ur, "x"] \arrow[dr, "y"'] & & (i,j) \\
 & (i,j-1) \arrow[ur, "1"'] &
\end{tikzcd}$$

c

Total ways : $(0,0) \rightarrow (i,j) = x+y$

x = ways $(0,0) \rightarrow (i-1,j) = dp(i-1,j)$

y = ways $(0,0) \rightarrow (i,j-1) = dp(i,j-1)$

Final ans = \#ways from $\overset{TL}{(0,0)} \rightarrow \overset{BR}{(n-1,m-1)}$: $dp[n-1][m-1]$

Dp Table = int dp[n][m] $\qquad$ TC = $\# N^{*}M * 1$

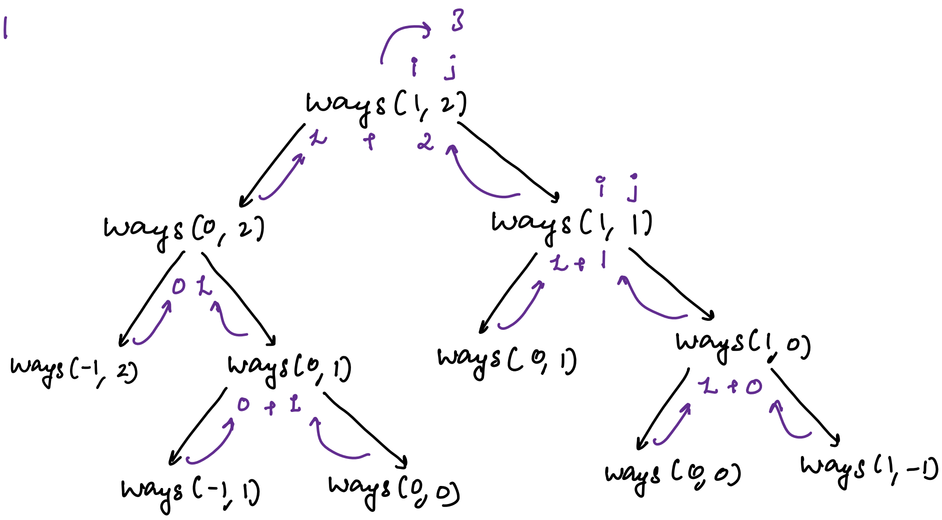
```
int dp[n][m] = INVALID/-1/..

int ways( int i, int j){
    if(i<0 || j<0){return 0}
    if(i==0 && j==0) {return 1} //VIMP Edge Case
    if( dp[i][j]==-1){ // Calculate 1st time
        dp[i][j] = ways( i-1,j) + ways(i, j-1)
    }
    return dp[i][j]
}
```

Q: ways we can go from (0, 0) → (1, 2) in mat[][] → ↓

int dp[2][3] = -1

	0	1	2
0	1	1	1
1	1	2	3

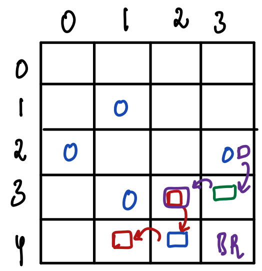


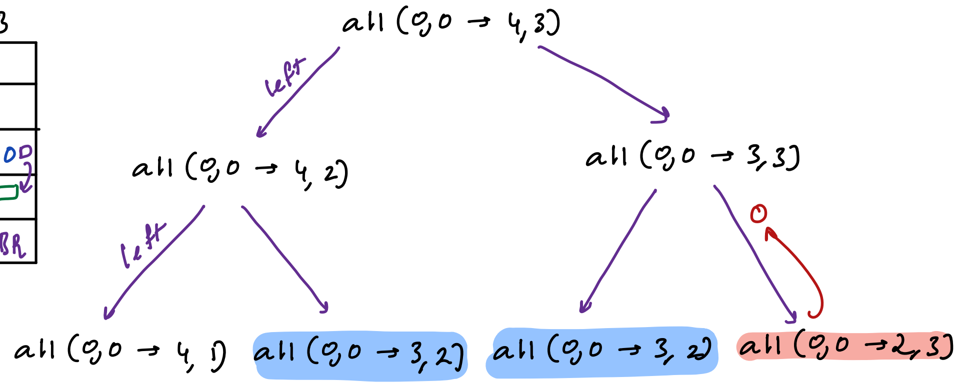
2Q) Number of ways to go from (0, 0) → BR cell

Note1: Cell → right, ↓ bottom

Note2: mat[i, j] = 0, blocked cell mat[i, j] = 1 unblocked cell

Note3: A path cannot go from blocked cell





Note: If a cell blocked, it should return 0.

dpState:

dpexp:

```
int dp[n][m] = INVALID/-1/..
int ways( int i, int j, int mat[][]){
    if( i<0 || j<0) {return 0}
    if( mat[i][j] ==0) {return 0} // only changes
    if( i==0 && j==0) { return 1}
    if( dp[i][j] ==-1){ // Calculate 1st time
        dp[i][j] = ways( i-1, j, mat) + ways( i, j-1, mat)
    }
    return dp[i][j]
}
```