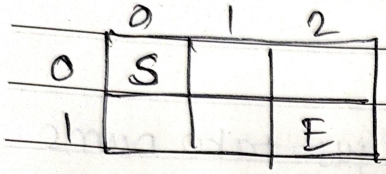
26/6/2023

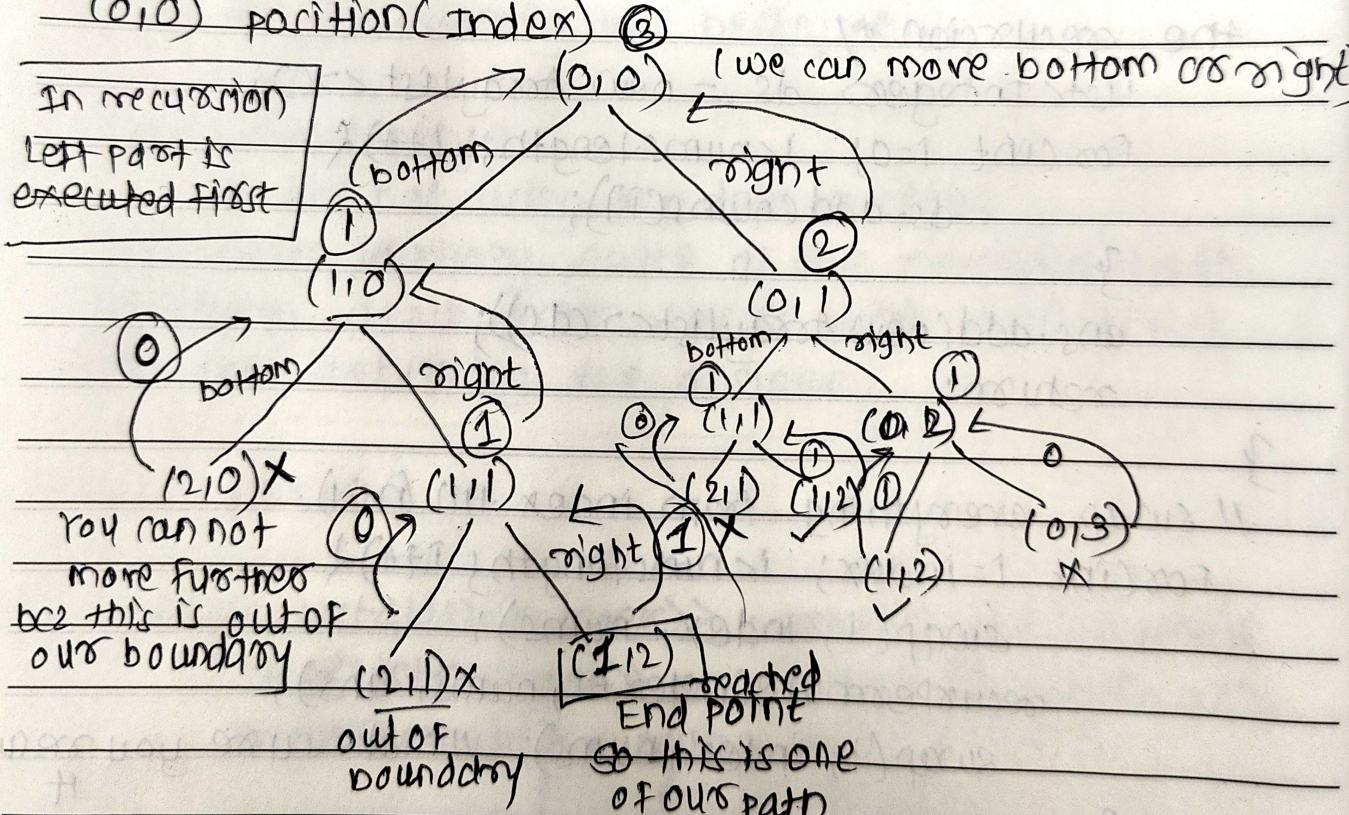
Grid Unique Paths →

problem states that we can move only right & bottom direction.



In recursion we can traverse all possible combinations

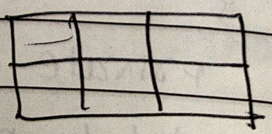
At first we are at (0,0) position (Index)



initially, i=0, j=0

n=1, m=2

For (2×3) matrix



Base condn → i>=n || j>=m

return 0;

when i==n-1 && j==m-1
return 1.

```
int countPaths(int i, int j, int n, int m){
    if(i==(n-1) && j==(m-1)) return 1;
    if(i>=n || j>=m) return 0;
    else return countPaths(i+1,j) + countPaths(i,j+1);
```

Time complexity →

Its going to be exponecial in nature as we are going to try all possible combinations

space complexity also will be exponsensial bcz we are going to use stack space in recursion.

optimal solution (DP)

recursive to DP

exponesial time complexity to quadratic T.C.

2 states (o/i) [i] [j] at max value of i can be n & value of j can be m.

Total no. of comb^n of states will be (n x m)

This where DP soln comes in.

Create Hash Table of (n x m)

initialize everything as -1

	0	1	2
0	-1	-1	-1
1	-1	-1	-1

We can see that we already fig. answer out somewhere else & answer you got 1. Instead of calling recursive fun^n as bottom & the right, we can use stored val. this what dp is: repeating the past without actually computing it.

Suboverlapping problems ⇒ You encounter the same state in some other part of recursion tree & you don't call recursion for that instead you use the val. stored in D.P. hashtable.

To convert recursive code to D.P. code
for that we need to carry 1 Hash table

Base condn & IF condn is same

```
if (i==(n-1) && j==(m-1)) return 1
if (i>=n || j>=m) return 0;

if(dp[i][j]!=-1) return dp[i][j]
else return dp[i][j] = countPath(i+1, j, dp)
        +              countPath(i, j+1, dp);
}
```

T.C. $O(N \times M)$
S.C $O(N \times M)$