Lectures : 2D DP

Agenda

- Maximum sum without adjacent elements
- Unique paths
- Dungeon princess

Qu1 Given arr[n], find max subsequence sum.

Eg:

$$\begin{array}{rcccccc} & 0 & 1 & 2 & 3 & 4 & 5 \\ arr[] = \{ & 2 & -4 & 5 & 3 & -8 & 1 \; \} \end{array}$$

↳ all positive els.

$$arr[] = \{ -4 \quad -2 \quad -3 \quad -10 \}$$

Qu Given arr[n], find max subsequence sum such that no 2 adjacent elements are selected.

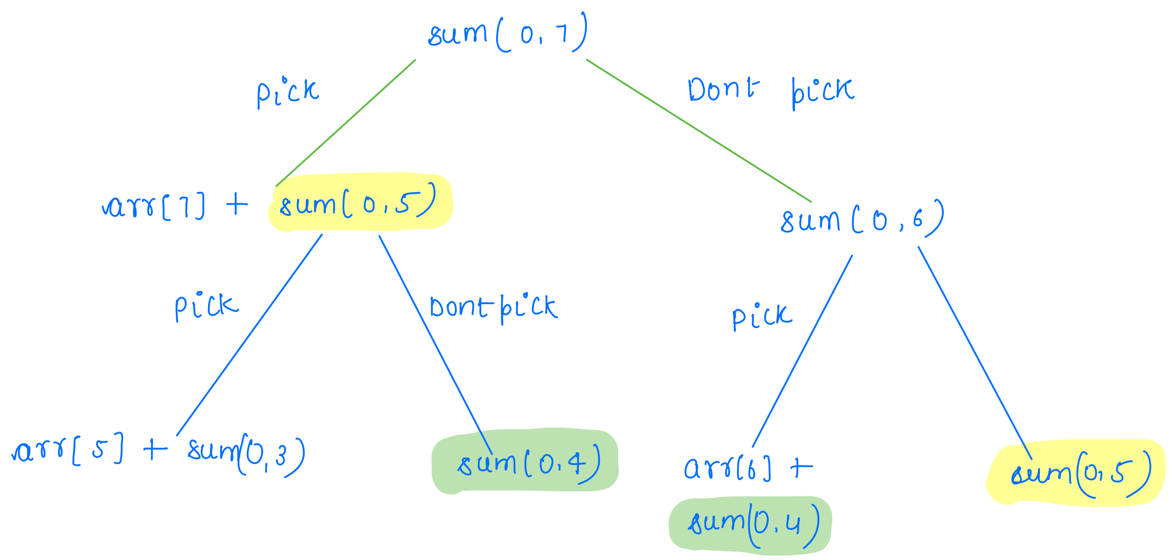
Eg: arr[] = [9 4 3] ans = 12

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

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

arr[] = [9 14 2] ans = 14

$$\begin{array}{lcccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ arr[\,] = & \big[\; 2 & -1 & -4 & 5 & 3 & -1 & 4 & 2 \;\big] \end{array}$$



1. Overlapping subproblems

Recursive code

```
int  maxsum( arr[] , end ) {
        if (end == 0) {
            return arr[0];
        }
        if( end < 0) {
            return 0;
        }
        include =  arr[end] +  maxsum( arr, end-2);

        exclude =  maxsum( arr, end-1);

        return  max( include,  exclude);
}
```

Memoised code

```
int maxsum( arr[] , end , dp[]) {
        if (end == 0) {
            dp[end] = arr[0];
            return arr[0];
        }

        if(end < 0) {
            return 0;
        }
        if(dp [end] != -1) {
            return dp[end];
        }
        include = arr[end] + maxsum(arr, end-2);

        exclude = maxsum ( arr, end-1);

        dp[end] = max( include, exclude);

        return max( include, exclude);
}
```

Tabulative approach

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

dp[i] = max subsequence sum from (0, i)

1. dp[2] = max subsequence from (0, 2)

sum(0,2)

Include: arr[2] + sum(0,0) ↓ dp[0]

exclude: sum(0,1) ↑ dp[1]

max(arr[2] + dp[0], dp[1])

max(-4 + 2 , 2)

max(-2, 2) = 2

2. dp[3] :

sum(0,3)

Include: arr[3] + sum(0,1) ↑ dp[1]

exclude: sum(0,2) ↑ dp[2]

max(arr[3] + dp[1] , dp[2])

max(5 + 2, 2)

max(7, 2) = 7

⋮

Generalisation ith idx —

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

Tabulative code

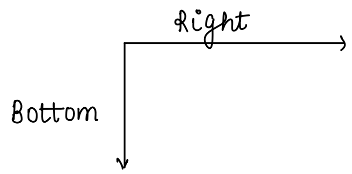
```
int maxSubsequenceSum(arr[]) {
    n = arr.length;
    int dp[] = new int[n];
    dp[0] = arr[0];
    dp[1] = max(arr[0], arr[1]);

    for(i=2; i<n; i++) {
        include = arr[i] + dp[i-2];
        exclude = dp[i-1];
        dp[i] = max(include, exclude);
    }
    return dp[n-1];
}
```

TC: O(n)

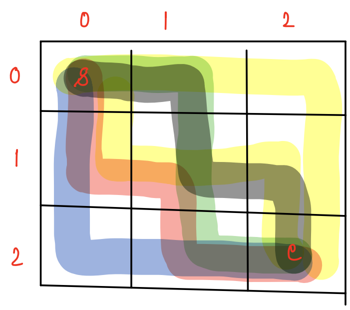
SC: O(n)

__Qu__ Count no. of ways to go from $(0,0)$ to $(n-1, m-1)$ cell.

Allowed directions:



__Example:__



ans = 6 ways

__ip:__ n and m

Idea:

dp[][] =

	0	1	2
0	1	1	1
1	1	(0,1) (1,0) 1 + 1 = 2	1+2 = 3
2	1	1+2 = 3	3+3 = 6

dp[i][j] = count no. of ways from (0,0) to (i,j) idx

dp[2][2] = 6

dp[1][2] (3) + dp[2][1] (3)

(0,0) → (1,2)

r r b b

b r r b

r b r b

(0,0) → (2,1)

b b r r

r b b r

b r b r

$$dp[i][j] = dp[i-1][j] + dp[i][j-1]$$

i != 0

j != 0

Code:

```
int ways(n, m) {
    dp[][] = new int[n][m];
    for(i=0; i<n; i++) {
        for(j=0; j<m; j++) {
            if( i==0 || j==0) {
                dp[i][j] = 1;
            } else {
                dp[i][j] = dp[i-1][j] + dp[i][j-1];
            }
        }
    }
    return dp[n-1][m-1];
}
```

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

Qu: can we optimise the space complexity?

$n=4$ and $m=4$

1	1	1	1
1	2	3	4
1	3	6	10
1	4	10	20

$O(n*m)$

Observation: ith row only depends on $(i-1)$ row

prev[]

curr[]

Dry run: prev[] = [1, 1, 1, 1] ⟶ 0th row — O(m)

1st row:

prev = [1, 1, 1, 1]

curr[] = [1, 2, 3, 4]

curr[i] = curr[i-1] + prev[i]

2nd row:

prev = curr

prev = 1 2 3 4

curr = 1 3 6 10

3rd row

prev = curr

prev = 1 3 6 10

curr = 1 4 10 20

$\boxed{\text{return curr[m-1].}}$

Break: 8:24: 8:37

Qu: Dungeon princess

You're prince (0, 0)

↓

reach princess (n-1, m-1)

↓

You have to tell min health you need to save princess.

[LC Hard]

Allowed directions:

Right →

Bottom ↓

Note: If your health <= 0, you'll die.

Ex:

	0	1	2
0	-2 1	-3 -2	3
1	-5 -4	-10	1
2	10	30	-5

health=3 (x)

	0	1	2
0	-2 2	-3 -1	3
1	-5 -3	-10	1
2	10	30	-5

health=4 (x)

	0	1	2
0	-2 3	-3 0	3
1	-5 -2	-10	1
2	10	30	-5

health=5 (x)

	0	1	2
0	-2 4	-3 1	3 4
1	-5	-10	1 5
2	10	30	-5 0

health=6 (x)

	0	1	2
0	-2 5	-3 2	3 5
1	-5	-10	1 6
2	10	30	-5 7

health=7 (✓)

ans = 7

↑

Idea:

arr[][] =

	0	1	2
0	2	-3	3
1	-5	-10	1
2	10	30	-5

dp[][] =

	0	1	2
0			2
1	6	11	5
2	1	1	6

$\rightarrow x + 3 = 5 = 2$

$\rightarrow x + 1 = 6 = 5$

$x + 10 = 1 = -9$ | $x + 30 = 6$ $\Rightarrow x = -24$

dp[i][j] = min health required to enter (i,j)

Dry run:

dp[1][1]

$x + (-10) = 5$

$x = 15$

$x - 10 = 1$

$x = 11$

dp[1][0]

$x - 5 = 1$

$x = 6$

$x - 5 = 11$

$x = 16$

arr[][] =

	0	1	2
0	2	-3	3
1	-5	-10	1
2	10	30	-5

dp[][] =

	0	1	2
0			2
1	6	11	5
2	1	1	6

$\rightarrow x + 3 = 5 = 2$

$\rightarrow x + 1 = 6 = 5$

$x + 10 = 1 = -9$ | $x + 30 = 6$ $\Rightarrow x = -24$

dp[0][1]

$x - 3 = 2$

$x = 5$

$x - 3 = 11$

$x = 14$

dp[0][0]

$x - 2 = 5$

$x = 7$

$x - 2 = 6$

$x = 8$

arr[][] =

	0	1	2
0	-2	-3	3
1	-5	-10	1
2	10	30	-5

dp[][] =

	0	1	2	
0	7	5	2	$\longrightarrow x + 3 = 5 = 2$
1	6	11	5	$\longrightarrow x + 1 = 6 = 5$
2	1	1	6	

$x + 10 = 1 = -9$ | $x + 30 = 6 \Rightarrow x = -24$

return dp[0][0]

Try this?

arr[][] =

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

ans

dp[][] =

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

$x - 1 = 2$

$x - 1 = 1$

Expression

$$dp[i][j] = \min\left(dp[i+1][j],\ dp[i][j+1]\right) - arr[i][j]$$

$i \,!= n-1$

$j \,!= m-1$

Code:

```
int   calculateMinHealth(arr[][]) {
        n = arr.length;
        m = arr[0].length;
        dp[n][m];
        for(i = n-1; i>=0; i--){
            for(j=m-1; j>=0; j--){
                if( i== n-1  &&  j==m-1)  {
                    //  x + arr[n-1][m-1] = 1
                    //  health = x
                  x = 1- arr[i][j];
                  if( x <= 0)  {
                      dp[i][j] = 1;
                  }  else {
                      dp[i][j] = x;
                  }
            }
        else if( i == n-1 ) {
                  x = dp[i][j+1] - arr[i][j];
                  if( x <= 0)  {
                      dp[i][j] = 1;
                  }  else {
                      dp[i][j] = x;
                  }
        }
```

```
→ else if( j == m-1) {
        x = dp[i+1][j] - arr[i][j];
        if( x <= 0)  {
            dp[i][j] = 1;
        } else {
            dp[i][j] = x;
        }
  }
→ else {
      h1 = dp[i+1][j] - arr[i][j] > 0  ?
           dp[i+1][j] - arr[i][j]  :
           1;
      h2 = dp[i][j+1] - arr[i][j] > 0  ?
           dp[i][j+1] - arr[i][j]  :
           1;
      dp[i][j] = min(h1, h2);
    }
  }
}

return dp[0][0];
}
```

TC: $O(n*m)$

SC: $O(n*m)$

H/W: Optimise space complexity? [Yes]

Thankyou