Lecture :- Dyamic programming -2

Agenda

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

Qu Given arr[n], find max subsequence sum

0	1	2	3	4	5
2	-4	5	3	-8	1

—— all +ve el

-4	-2	-3	-10

—— max el.

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

9	4	3

ans = 12

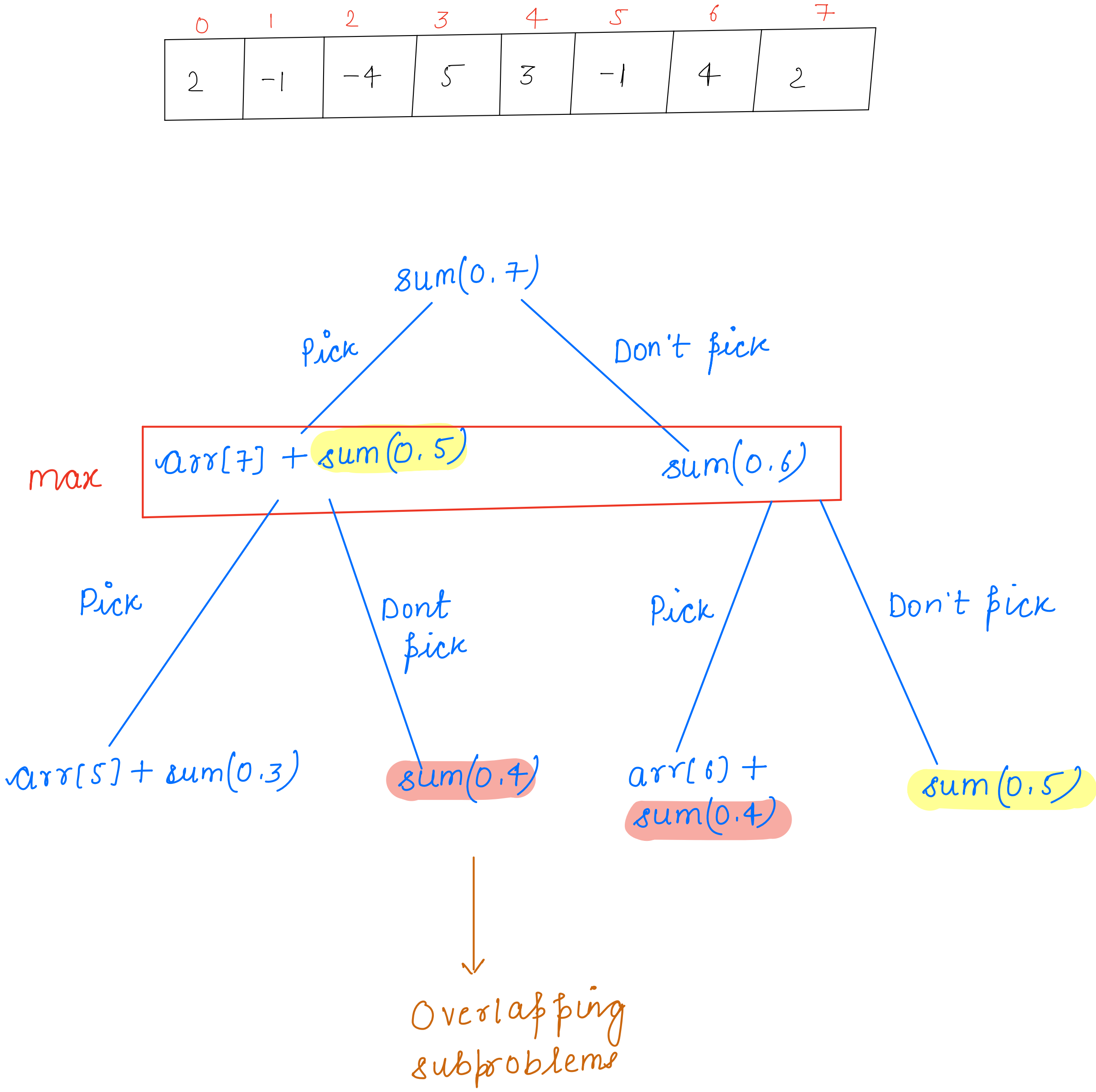
9	4	13	24

ans = 33

13	14	2

9	14	2

10	20	30	40



0 1 2 3 4 5 6 7
2 -1 -4 5 3 -1 4 2
sum(0,7)
Pick
Don't pick
max
arr[7] + sum(0,5)
sum(0,6)
Pick
Dont pick
Pick
Don't pick
arr[5] + sum(0,3)
sum(0,4)
arr[6] + sum(0,4)
sum(0,5)
Overlapping subproblems

Recursive code

```
int maxsum(int[] arr, end) {
        if(e == 0) {
            return arr[0];
        }
        if (end < 0) {
            return 0;
        }
        pick = arr[end] + maxsum(arr, end-2);
        dontPick = maxsum(arr, end-1);
        return max(pick, dontPick);
}
```

TC: $O(2^n)$
SC: $O(n)$

Memoised code

```
                                   dp[]
int maxsum(int[] arr, end,)  {

    if( e == 0 ) {
        dp[end] = arr[0];
        return arr[0];
    }

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

    dontPick = maxsum(arr, end-1);
    dp[end] = max(pick. dontPick);
    return max(pick, dontPick);
}
```

TC:

SC: $O(n) + O(n)$

$\uparrow$ arr $\uparrow$ stack

Tabulative approach

arr =

0	1	2	3	4	5	6	7
2	-1	-4	5	3	-1	4	2

dp =

0	1	2	3	4	5	6	7

$\boxed{dp[i] = \text{max sub sum from } (0, i) + \text{cond}^n.}$

dp[5]	dp[3]
max (arr[5] + dp[3] , dp[4])	dp[3] max sum(0,3)

dp[3]

max sum(0,3)

1. arr[3] + max sum(0, 1)
   dp[1]
2. max sum (0,2)
   dp[2]

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

Generalisation

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

Edge cases :

$i = 0, 1$

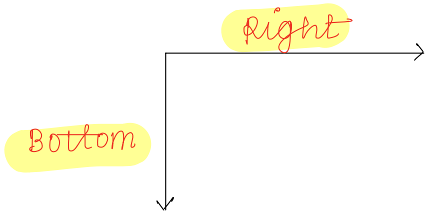
Tabulative code

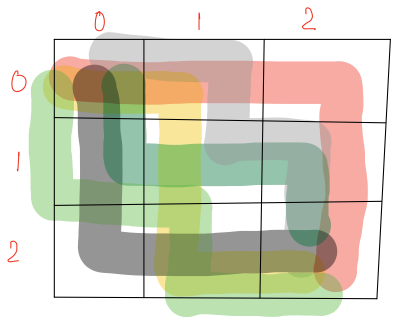
```
int maxsum(int[] arr) {
    n = arr.length;
    dp[n];
    dp[0] = arr[0];
    dp[1] = max(arr[0], arr[1]);
    for(i=2; i<n; i++) {
        inc = arr[i] + dp[i-2];
        exc = dp[i-1];
        dp[i] = max(inc, exc);
    }
    return dp[n-1];
}
```

TC: O(n)
SC: O(n)

Qu count no. of ways to go from (0.0) to $(n-1, m-1)$ cell.

Allowed directions





ans = 6

Idea

dp[] =

	0	1	2
0	1	1	1
1	1	dp[0,1] + dp[1,0] 1 + 1 = 2	1 + 2 = 3
2	1	2+1=3	3+3=6

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

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

i != 0
&
j != 0

Algorithm

```
int countWays( int n, int m) {
        dp[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? [Yes]

$n = 4$ and $m = 4$

	0	1	2	3
0				
1				
2				
3				

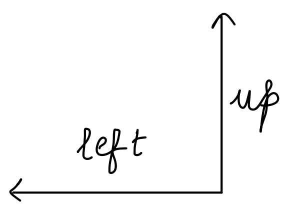
Observation

ith row only depends on $(i-1)$ row.

Dry run

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

Allowed directions



1 — non-blocked

0 – blocked

Break: 8:22 – 8:32

Qu **Dungeon princess** [Hard]

You're prince (0,0)

↓

reach princess (n-1, m-1)

↓

You have to tell min health you need to save princess

Allowed directions

Right →

Bottom ↓

Note: If your health <=0, you die

	0	1	2
0	−2 (1)	−3 (−2)	3
1	−5 (−4)	−10	1
2	10	30	−5

health = 3 ✗

	0	1	2
0	−2 (2)	−3 (−1)	3
1	−5 (−3)	−10	1
2	10	30	−5

health = 4 ✗

	0	1	2
0	−2 (3)	−3 (0)	3
1	−5 (−2)	−10	1
2	10	30	−5

health = 5 ✗

	0	1	2
0	−2 (4)	−3 (1)	3 (4)
1	−5	−10	1 (5)
2	10	30	−5 (0)

health = 6 ✗

	0	1	2
0	−2 (5)	−3 (2)	3 (5)
1	−5	−10	1 (6)
2	10	30	−5 (1)

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		11	5
2	1	1	6

$x + 3 = 5$

$x + 1 = 6 = 5$

$x + 10 = 1$
$x = -9$

$x + 30 = 6 = -24$

$dp[i][j] \Rightarrow$ min health to enter (i, j)

Dry run

$dp[2][2] \longrightarrow x + (-5) = 1$
$x = 6$

$dp[1][1]$

Right $\rightarrow$ $x - 10 = 5$
$x = 15$

Bottom $\rightarrow$ $x - 10 = 1$
$x = 11$

$$arr[\,][\,] = \begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & -2 & -3 & 3 \\ 1 & -5 & -10 & 1 \\ 2 & 10 & 30 & -5 \end{array}$$

$x-2=5=7$

$x-2=6=8$

$x-3=2=5$

$x-3=11=14$

$$dp[\,][\,] = \begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & \boxed{7} & 5 & 2 \\ 1 & 6 & 11 & 5 \\ 2 & 1 & 1 & 6 \end{array}$$

$x+3=5$

$x+1=6=5$

$x+10=1$

$x=-9$

$x+30=6=-24$

$x-5=11$

$x=16$

$x-5=1$

$x=6$

Try this

arr[][] ⇒

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

dp[][] ⇒

	0	1	2
0			
1			
2			

Expression

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

$i \,!= n-1$

$j \,!= m-1$

Algorithm

```
int calculateMinHealth( int[][] 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 =1 - arr[i][j];
                    dp[i][j] = x <=0 ? 1 : x;
                }
                else if( i==n-1) {
                    x = dp[i][j+1] - arr[i][j];
                    dp[i][j] = x <=0 ? 1 : x;
                } else if( j==m-1) {
                    x = dp[i+1][j] - arr[i][j];
                    dp[i][j] = x <=0 ? 1 : x;
                } else{
                    right = dp[i][j+1] - arr[i][j];
                    bottom = dp[i+1][j] - arr[i][j];
                    x = min (right, bottom)

                    dp[i][j] = x <=0 ? 1 : x;
                }
            }
        }
    return dp[0][0];
}
```

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

Thankyou :)