DP – 2

Madhan Kumar M S
Abhishek Sharma
Akansh Nirmal
amit khandelwal
Bhaveshkumar
Burhan
Divya P
Gagan Kumar S
Gowtham
Hemant Kumar
KULDEEP PATIDAR
Murali Mudigonda
Nikhil Pandey
Purusharth A
Rajat Sharma
Rajendra
Sanket Giri
Saurabh Ruikar
Shani Jaiswal
shilpa mamillapalli
Shradha Srivastava
Sridhar Hissaria
Subhashini
Sumit Adwani
Suyash Gupta
Vasanth
Vimal Kumar

AGENDA:

- Max subsequence sum w/o adj ele.
- No. of paths
- No. of paths with obstacles.
- Dungeon & Princess

contest 5 ⟶

Trees, Heaps, Greedy

26th April

Hypothetical Real World Scenario

In the **Scaler educational program**, students are encouraged to balance between attending structured lectures and dedicating time to self-study, rest, or personal projects. Scaler understands the importance of self-directed learning and well-being, hence introduces a **flexible attendance policy**.

Hypothetical

According to this policy :

- A student is required not to miss lectures on **two consecutive days**, ensuring a balanced engagement with the curriculum while also providing students time with their own learning.

Given the number of hours allocated for classes each day, the task is to determine the **MAX NUMBER OF HOURS** a student can allocate to self-study, rest, or personal projects by being absent, without violating the policy of missing two consecutive lectures.

max hours of self study

A = { 5 4 8 } (0 1 2) → 13

A = { 9 4 10 12 } (0 1 2 3) → 21

A = 1 2 3 4 5

subarray ⟶ Any part of array ⟵ 2 3 4

subsequence ⟶ Obtained by deleting 0 or more indexes

~~5 4~~ not subsequence

1 3 4

may or may not be cont"

Q> Given an array find max subsequence sum.

* you are not allowed to pick adjacent elements.

{ 9 4 13 } ans = 22

{ 9 4 13 24 } ans = 33

QUIZ

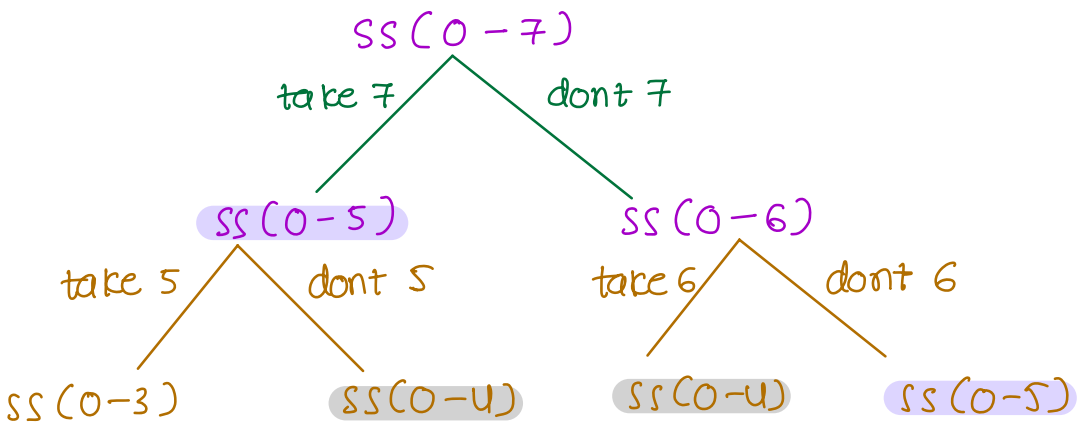
{ 10 20 30 40 } ans = 60

last step

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

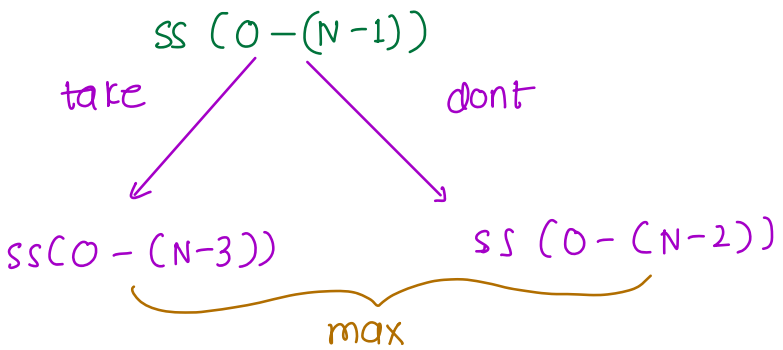
SS(0-7) ⟶ max subsequence sum from 0 to 7

dp state



① Optimal substructure ✓

② Overlapping subproblem ✓



SS (i) = max subsequence sum from 0 to i

dp relation { SS(i) = max { A[i] + SS(i-2) // take
SS(i-1) // dont

Pseudocode

A[]

```
int SS ( index ) {
    if ( index < 0 ) { return 0 }

    take = A[index] + SS ( index - 2 )
    dont = SS ( index - 1 )

    return max ( take , dont )
}
```

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

Memoize ⟶ DP array OR Hashmap

```
dp = new Hashmap<Int, Int>()

                A[]

int ss(index) {
    if(index < 0) { return 0 }
    if(dp.containskey(index)) return dp.get(index)
    take = A[index] + ss(index-2)
    dont = ss(index - 1)
    dp.put(index, max(take, dont))
    return max(take, dont)
}
```

TC for any memoized dp code

Unique no. of dp states * TC per state

N * O(1)

TC: O(N)

SC: O(N)

Iterative code

smallest subproblem we have sol" for

```
dp[0] = max(A[0], 0)
dp[1] = max(A[0], A[1], 0)
```

```
for i ⟶ 2 to N-1 {
    take = A[i] + dp[i-2]
    dont = dp[i-1]
    dp[i] = max(take, dont)
}
```

TC : O(N)

SC : O(N)

```
print(dp[N-1])
```

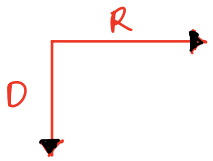
A = 5 -2 4 1 -10

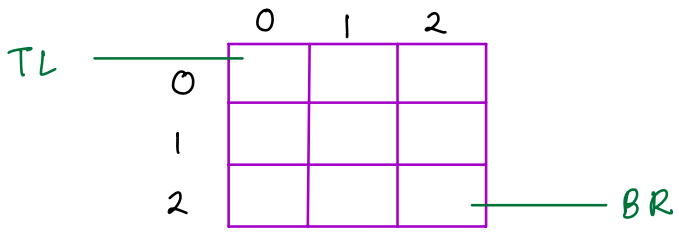
dp	5	5	9	9	9
	0	1	2	3	4

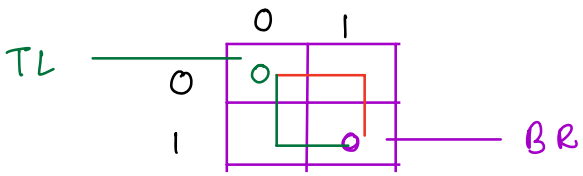
HW solve the above in O(1) space complexity

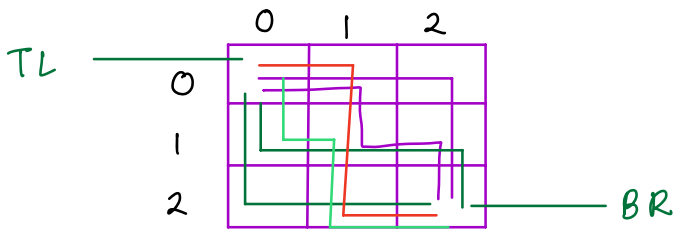
Q> Find total no. of ways to reach BR from TL

movements allowed



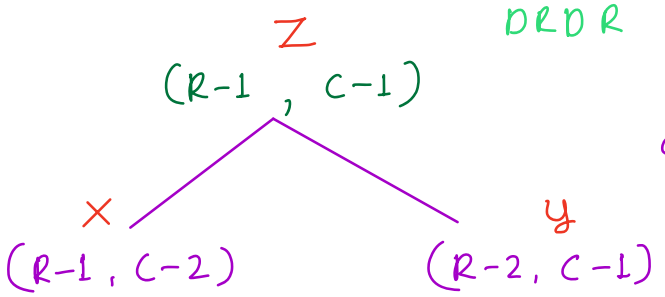






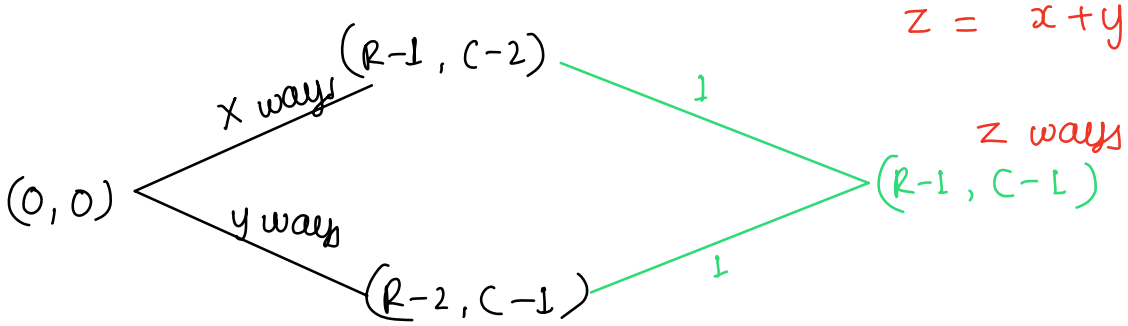
R R D D
R D R D
D R R D
D D R R
R D D R
D R D R

last step



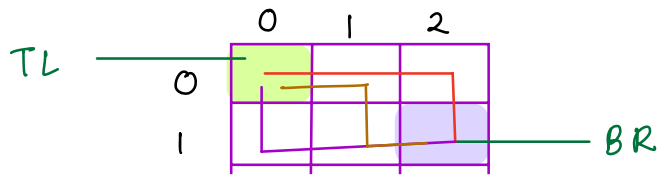
optimal substructure

$z = x + y$

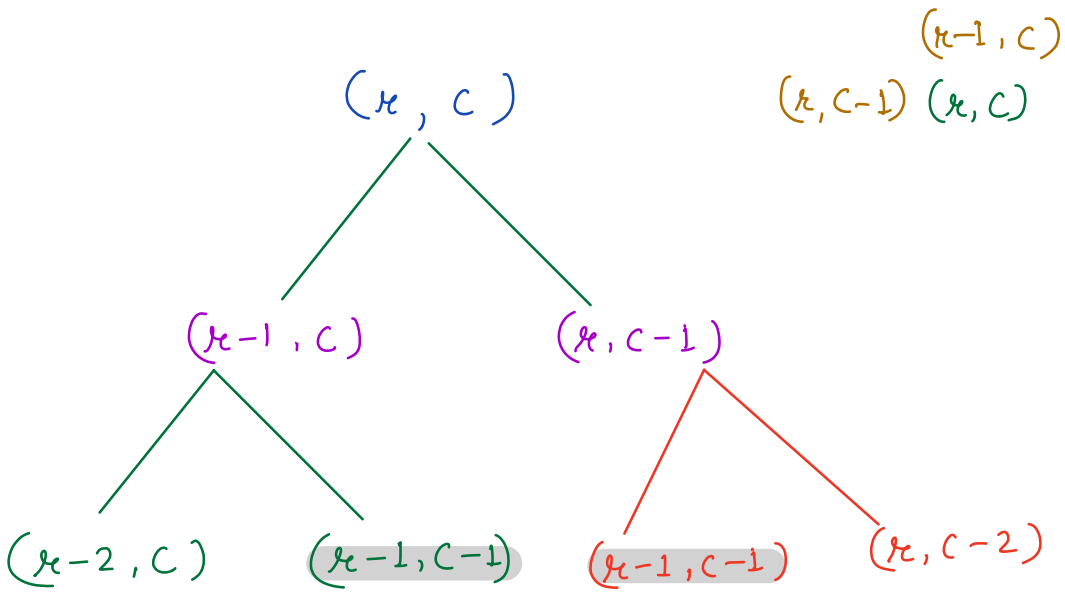


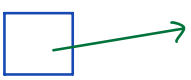
$z = x + y$

QUIZ:



3 ways





No. of ways to reach from 0,0 to 0,0

1

Pseudocode

R rows
C cols

```
int ways ( r , c ) {
    if ( r==0 || c==0 ) { return 1 }

    top  = ways ( r-1 , c )
    left = ways ( r , c-1 )

    return top + left
}
```

TC : $O(2^{RC})$

SC : $O(R+C)$

```
dp[R][C]   // -1
```

Pseudocode

R rows
C cols

```
int ways ( r , c ) {
    if ( r==0 || c==0 ) { return 1 }
    if (dp[r][c] != -1) return dp[r][c]
    top  = ways ( r-1 , c )
    left = ways ( r , c-1 )
    dp[r][c] = top + left
    return top + left
}
```

TC: $O(R*C)$

SC: $O(R*C)$

Iterative

smallest sub problem

dp[R][C]

dp[0][0] = 1

```
for r ⟶ 0 to R {
    for c ⟶ 0 to C {
        if (r==0 || c==0) {
            dp[r][c] = 1
        }
        else {
            top = dp[r-1][c]
            left = dp[r][c-1]
            dp[r][c] = top + left
        }
    }
}
```

TC: O(RC)

SC: O(RC)

print(dp[R-1][C-1])

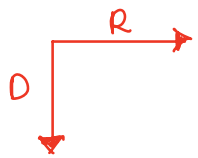
	0	1	2
0	1	1	1
1	1	2	3
2	1	3	6

prev row, curr row } O(C)

Break : 22:42

Q> Find total no. of ways to reach BR from TL with obstacles

movement allowed



A[r][c] == 1 Obstacle

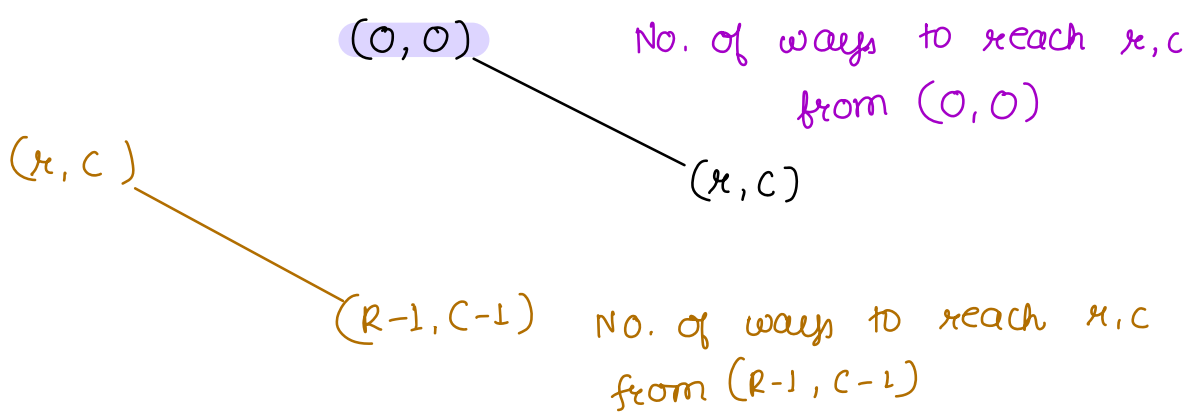
==0 Free

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

	0	1	2	3
0	1	1	1	1
1	1	///	1	2
2	1	1	///	2
3	1	2	2	4

whenever current cell is an obstacle

dp[r][c] = 0



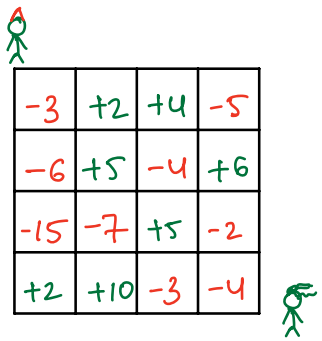
For any 2D dp problem try to think from first step or last step

Dungeon & Princess

movement allowed

R

D

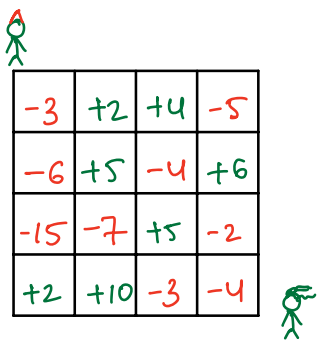


-3	+2	+4	-5
-6	+5	-4	+6
-15	-7	+5	-2
+2	+10	-3	-4

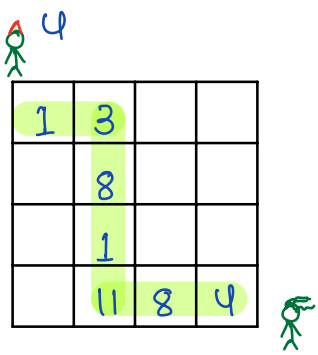
If health <= 0 ⟶ death

Find the **minimum health** level to start with, so that you can save the princess.

ie Find the min health to start such that there exists one path where you donot die in the future

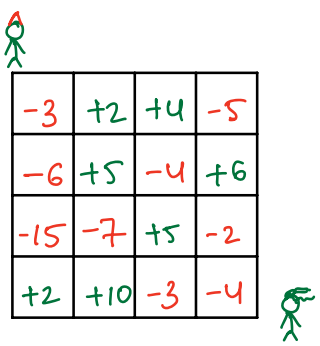


-3	+2	+4	-5
-6	+5	-4	+6
-15	-7	+5	-2
+2	+10	-3	-4



1	3		
	8		
	1		
	11	8	4

ans = 4



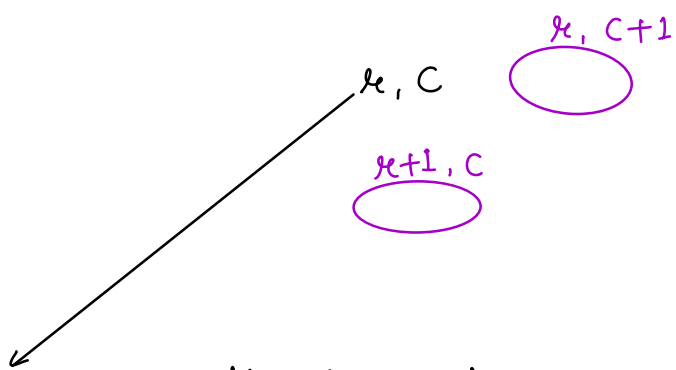
-3	+2	+4	-5
-6	+5	-4	+6
-15	-7	+5	-2
+2	+10	-3	-4

Smallest subproblem ⟶ `r-1 c-1`

min health with which I should reach last cell `-4` cell such that I survive 5

min health with which I should reach last cell `4` cell such that I survive 1

`max ( 1 − A[r][c] , 1 )`



min health to reach r,c cell such that I save the princes in future.

$$dp[r][c] = \max\begin{cases} \min\begin{cases} dp[r+1][c] \\ dp[r][c+1] \end{cases} \\ 1 \end{cases} - A[r][c]$$

if dp[r][c] <= 0 reset to 1

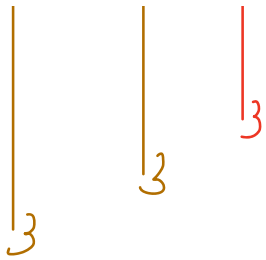
4 -3	1 +2	1 +4	6 -5
7 -6	1 +5	5 -4	1 +6
16 -15	8 -7	2 +5	7 -2
1 +2	1 +10	8 -3	5 -4

Health

Pseudocode

```
dp [R][C]    // init

for  r ⟶ R-1 to 0 {
    for  c ⟶ C-1 to 0 {
        if ( r == R-1 && c == C-1 ) {
            dp[r][c] = max(1, 1 - A[r][c])
        }
        else if ( r == R-1) {
            dp[r][c] = max( 1, dp[r][c+1] - A[r][c])
        }
        else if ( c == C-1) {
            dp[r][c] = max(1, dp[r+1][c] - A[r][c]
        }
        else {
            mhealth = min ( dp[r+1][c] , dp[r][c+1]
```



$dp[r][c] = \max(1, mhealth - A[r][c])$

TC: $O(RC)$

SC: $O(RC)$

print (dp [0][0])