Current PSP : 54.1 → 57 %

Nov23_PSP_22Apr
Gobika K
Piyush Kumar
kameswarreddy Yeddula
Manjunatha I
Sai Sharath
Harshil Dabhoya
Rajeev
Yash Malviya
Vijay V A
Mohammad Mateen
Kevin Theodore E
Robin Dhiman
Suraj Devraye

Nov23_PSP_22Apr
SABBAVARAPU KARTHIK
manikandan m
Prashant Kumar Soni
Mayur Hadawale
MD JASHIMUDDIN
Vigneshwaran K
Pradeep Kumar Chandra
Shaurya Srivastava
barani r
SIJU SAMSON
Mohammed Arshad
Sarat Patel
Pushkar Deshpande

Agenda:

Max subsequence sum w/o adjacent elements

No. of paths

No. of paths with obstacle

Dungeons & Princess.

Q → Given an array find max subsequence sum

* you are not allowed to pick adjacent elements

Subarray vs Subsequence

arr =

9	4	13	3	1	9
0	1	2	3	4	5

4	13	3	1
0	1	2	3

Subarray

contiguous

follow input order

9	13	3	9
0	1	2	3

Subsequence

follow input order

example

arr =

9	4	13
0	1	2

ans = 9 + 13 = 22

arr =

9	4	13	24
0	1	2	3

ans = 9 + 24

Quiz 1

arr =

10	20	30	40
0	1	2	3

ans = 20 + 40 = 60

Observation

arr =

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

take

don't take

the last stage will be we have considered all subsequences from [0, 7] to get maximum sum

ss (0, 7)

take (7) → a[7] + ss (0,5)

don't (7) → ss (0, 6)

a[7] + ss (0,5): take → a[5] + ss (0,3); don't → ss (0,4)

ss (0, 6): take → ss(0, 4) + a[6]; don't → ss (0, 5)

Optimal Substructure

→ Bigger Problem can be solved using smaller problem

Overlapping Subproblem

→ Repeating subproblem

Generalizing

ss (0, N-1)

take → a[N-1] + ss (0, N-3)

don't → ss (0, N-2)

redundant

$ss[i] \rightarrow$ Max Subsequence sum from $(0, i)$

Relation : $ss[i] = \max(\underbrace{ss[i-2] + a[i]}_{\text{take}}, \underbrace{ss[i-1]}_{\text{dont}})$

Brute force

```
int subsequence (A, i) {
    if (i<0) return 0; // Base condition
    take = A[i] + subsequence (A, i-2);
    dont = subsequence (A, i-1);
    return max (take, dont)
}
```

$T.C = O(2^n) \quad S.C = O(n)$

Memoize

DP = [] ∀i mark as 0

```
int subsequence (A, i) {
    if (i<0) return 0; // Base condition
    if (DP[i] != 0) return DP[i] // reuse
    take = A[i] + subsequence (A, i-2);
    dont = subsequence (A, i-1);
    DP[i] = max (take, dont) // store
    return DP[i]
}
```

$\boxed{T.C = O(n) \qquad S.C = O(n)}$

Iterative Code

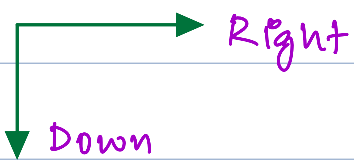
Go from smallest problem to bigger problem

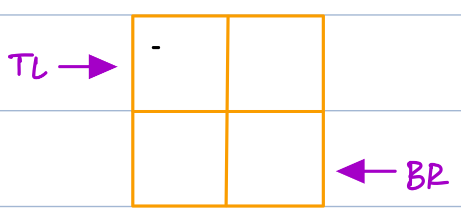
```
dp[0] = max(A[i], 0)
dp[1] = max(A[1], A[0], 0)
for (i=2; i<n; i++){
    take = A[i] + DP[i-2]
    dont = DP[i-1]
    DP[i] = max(take, dont)
}
```

$\boxed{\begin{array}{l} T.C = O(n) \\ S.C = O(n) \end{array}}$

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

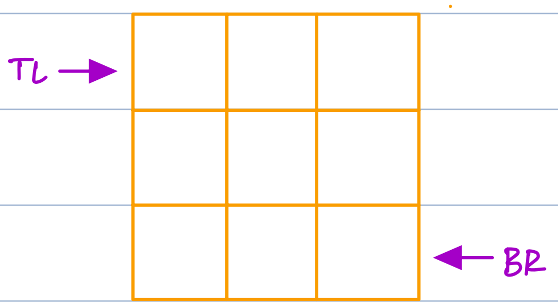
movements allowed





RD

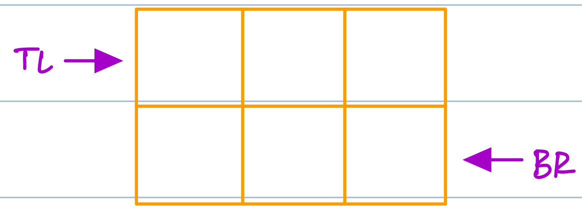
DR



RRDD , RDRD , RDDR

DDRR , DRDR , DRRD

Quiz



RRD, DRR, RDR

Last step (observation)

(R-1, C-1)

(Top) (Left)

x ways (R-1, C-2) (R-2, C-1) y ways

$(0,0) \xrightarrow{x \text{ ways}} (R-1, C-2) \xrightarrow{1 \text{ way}} (R-1, C-1)$

$(0,0) \xrightarrow{y \text{ ways}} (R-2, C-1) \xrightarrow{1 \text{ way}} (R-1, C-1)$

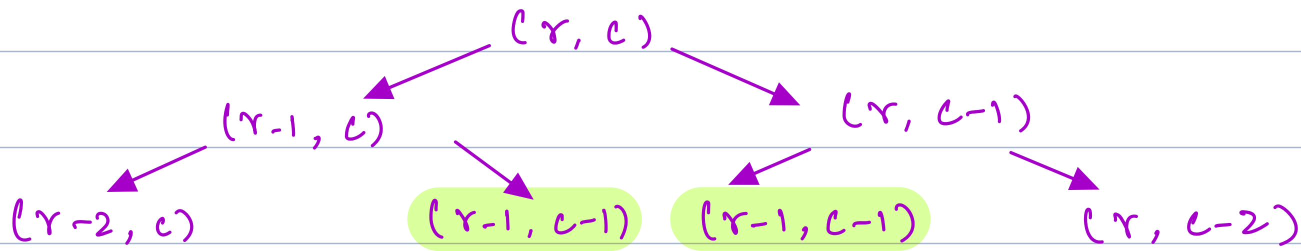
total ways (x+y)

Brute force

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

$\boxed{\begin{array}{l} T.C = 2^{RC} \\ S.C = O(RC) \end{array}}$

Dynamic Programming



Both optimal substructure & Overlapping subproblem is met.

memoization

```
DP [R][C] // initialize to -1
int  ways (r, c) {
    if (r==0 || c==0)  return 1;
    if (DP [r][c] != -1)  return DP[r][c] //reuse
    top = ways (r-1, c)
    left = ways (r, c-1)
    DP[r][c] = top + left;    // store
    return  DP [r][c]
}
```

T.C = O(R * C)

S.C = O(R * C)

Iterative approach // smallest to bigger problem

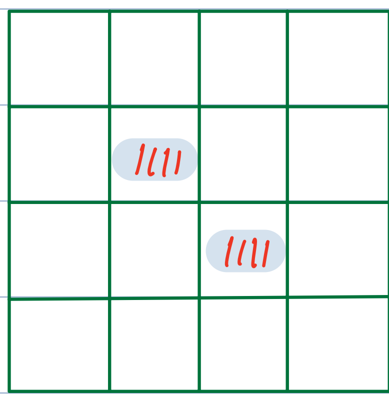
```
DP [R] [C] ;
DP [0] [0] = 1;
for ( i= 0; i< R; i++) {
    for ( j= 0; j< C; j++) {
        if ( i==0 || j ==0)      DP [i] [j] = 1
        else
            DP[i] [j] = dp [i-1] [j] + dp [i] [j-1];
    }
}
```

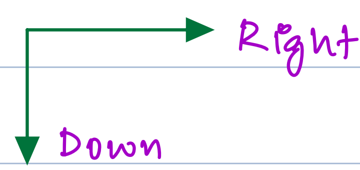
	0	1	2	
TL → 0	1	1	1	
1	1	2	3	
2	1	3	6	← BR

Q → Find total no. of ways to reach BR from TL with obstacles (0 represent obstacle , 1 represent free)

1	1	1	1
1	0	1	1
1	1	0	1
1	1	1	1

→ visuali-ze





Observation

1	1	1	1
1	///	1	2
1	1	///	2
1	2	2	4

if (mat[i][j] == 0)

DP[i][j] = 0

// rest of the code remains the same

Quiz 3

TL →

////	////	////

← BR

1 represent free

0 represent obstacle

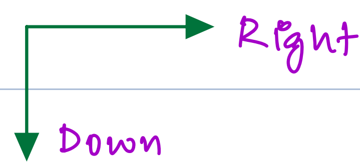
ans = 0

Q → **Dungeons and Princess**

Find the minimum health level of the prince to start with to save the princess, where the negative numbers denote a dragon and positive numbers denote red bull

Redbull will increase the health whereas the dragon will decrease health.

health <= 0 , it means prince is dead



	0	1	2	3
0	-3	2	4	-5
1	-6	5	-4	6
2	-15	-7	5	-2
3	2	10	-3	-4

4

	0	1	2	3
0	1	3		
1		8	4	10
2				8
3				4

ans = 4

Observation

Greedy approach

	0	1	2	3
0	1	2	-100	-11
1	-1	-100	-11	-2
2	100	2	-9	-1

In greedy approach we choose (0, 1) over (1, 0) which is a bad move

Brute force Approach

	0	1	2	3
0	-3	2	4	-5
1	-6	5	-4	6
2	-15	-7	5	-2
3	2	10	-3	-4

smallest problem to solve

mat[0][0] ✗

Dont have entire view

mat[3][3]

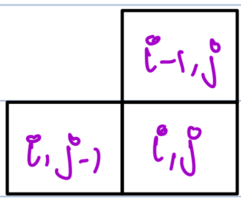
smallest problem

if arr[N-1][M-1] = -4 min Health needed to enter this cell

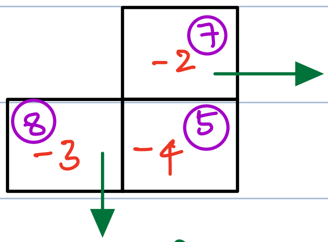
$x + (-4) = 1$

$x = 5$

Next Steps

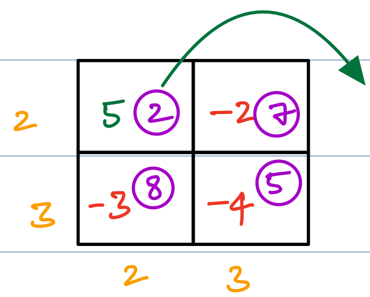


To enter (i, j) I should enter from $(i-1, j)$ or $(i, j-1)$



min Health to enter $x + (-2) = 5$

min Health to enter $y + (-3) = 5$



min health needed $x + 5 = 7 \rightarrow 2$

$x + 5 = 8 \rightarrow 3$

Generalizing

min Health $[i][j] \rightarrow x$

$$x + arr[i][j] = \min \begin{cases} x \text{ at } [i+1][j] \\ x \text{ at } [i][j+1] \end{cases}$$

$$
\begin{array}{c|cccc}
 & 0 & 1 & 2 & 3 \\
\hline
0 & -3 & 2 & 4 & -5 \\
1 & -6 & 5 & -4 & 6 \\
2 & -15 & -7 & 5 & -2 \\
3 & 2 & 10 & -3 & -4
\end{array}
\quad \xrightarrow[\text{DP}]{\text{min Health}} \quad
\begin{array}{c|cccc}
 & 0 & 1 & 2 & 3 \\
\hline
0 & 4 & 1 & 1 & 6 \\
1 & 7 & 1 & 5 & 1 \\
2 & 16 & 8 & 2 & 7 \\
3 & 1 & 1 & 8 & 5
\end{array}
$$

Quiz 4

T.C to perform above solution $= O(R * C)$

Algorithm (index (i, j))

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

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

since $x > 1$

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

 # pseudo code

```
dp [N][M]   // initialize as 0
  if (arr [N-1] [M-1] > 0)
      dp [N-1] [M-1] = 1
  else
      dp [N-1] [M-1] = 1 + abs ( arr [N-1] [M-1])
```

// Fill the last column & last row

```
for (i = n-2; i >= 0; i--) {
    for (j = n-2; j >= 0; j--) {
            dp[i][j] = max(1, min(dp[i+1][j],
                       dp[i][j+1]) - arr[i][j])
    }
}
```

// DP [0][0] has answer

$\boxed{T.C = O(R*C) \quad S.C = O(R*C)}$

Catalan Numbers

Numerous application in combinatorial mathematics.

→ no. of distinct binary search tree with N nodes

→ no. of correct combination of N pairs of paranthesis

Sequence

$C_0 = 1 \quad , \quad C_1 = 1$

$C_2 = C_0 C_1 + C_1 C_0 = 2$

$C_3 = C_0 C_2 + C_1 C_1 + C_2 C_0 = 5$

$C_4 = C_0 C_3 + C_1 C_2 + C_2 C_1 + C_3 C_0 = 14$

Generalize

$$C_N = C_0 * C_{N-1} + C_1 * C_{N-2} + \cdots + C_{N-1} * C_0$$

$$C_n = \sum_{i=0}^{i<n} C_i * C_{n-i-1}$$

pseudo code

```
C[0] = 1    C[1] = 1;
for (i=2; i<=n; i++) {
    for (j=0; j<i; j++) {
        C[i] += C[j] * C[n-1-j]
    }
}
```